МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
Навчально-науковий інститут Інформаційних технологій
Кафедра Інженерії програмного забезпечення

Методичний посібник

СПЕЦІАЛЬНІ МОВИ ПРОГРАМУВАННЯ

Київ 2021

УДК 681.3.07

Схвалено Вченою радою Державного університету телекомунікацій
(протокол № 7 від 28.01.2021 року)

Трінтіна Н.А., Негоденко О.В., Бондарчук А.П., Терещенко О.І.

Методичні рекомендації до викладання дисципліни «Спеціальні мови програмування»:– К.: ДУТ, 2021. - 311 с.

Методичні рекомендації до викладання дисципліни «Спеціальні мови програмування» розроблені відповідно до програми курсу «Спеціальні мови програмування». Предметом вивчення навчальної дисципліни є принцип програмування на мові Python, відповідно до якого студент, користуючись знаннями та навичками у програмуванні на мові Python, пише програму за будь-яким завданням, отримує результати графічно або чисельно, аналізує код та результати роботи програми. У методичному посібнику до викладання дисципліни «Спеціальні мови програмування» достатньо практичних прикладів для засвоєння теоретичних знань та практичних вмінь для відпрацювання навиків розробки програмного забезпечення мовою Python.

Зміст

Вступ

Python - мова програмування, яка інтенсивно застосовується IT-гігантамі, такими як, Google і Yandex. Python практично нічим не обмежений, тому також може використовуватися у великих проектах.

Python може: працювати з xml / html файлами, працювати з http запитами, GUI (графічний інтерфейс), створювати веб-сценарії, працювати з FTP, працювати з зображеннями, аудіо та відео файлами, робототехнікою, програмувати математичні і наукові обчислення, і багато іншого.

У посібнику можна познайомитеся з основними поняттями мови, такими як методи, рядки, функції, списки, словники, кортежі, одномірні масиви, класи і цикли, і навчитися писати чистий і код, що добре читається. Навчитися методам складної обробки двовимірних масивів. Ознайомитесь з розширеними можливостями функцій. Ознайомитесь з створення RESTful додатку, роботою з асинхронними функціями в Python, з управлінням даними за допомогою Python, SQLite та SQLAlchemy, створенням telebot у Python, ознайомитесь з системою управління баз даних з відкритим вихідним кодом MongoDB.

У процесі виконання робот можна навчитися програмувати на Python на прикладі розробки ігор. У роботах розглядається створення двовимірних ігор за допомогою бібліотеки Pygame. Посібник допомагає вивчити основні модулі бібліотеки Pygame та навчитись: використовувати цикли для розробки гри, змінні і логічні вирази, використовувати такі структури даних, як списки, словники і кортежі, відлагоджувати програми і шукати помилки, писати простий ШІ для ігор, створювати просту графіку і анімації для ігор.

Практична робота №1

Тема. Робота з Python. Завантаження програми. Робота в інтерактивному режимі.

Мета роботи. Отримати навички роботи в інтерактивному режимі.

Зміст.

1. Вивчення відомостей про роботу в інтерактивному режимі.
2. Виконання роботи.
3. Отримання результату.

Ключові положення.

Що вміє робити python:

• Робота с xml/html файлами
• Робота с http запитами
• GUI (графічний інтерфейс)
• Створення веб-сценарієв
• Робота с FTP
• Робота з зображеннями, аудіо та відео файлами
• Робототехніка
• Програмування математичних та наукових обчислень

Python можно загрузити з веб-сайту https://python.org/downloads/windows/, обираємо python 3.

Після завантаження та установки **python 3** відкриваємо IDLE Рис.1

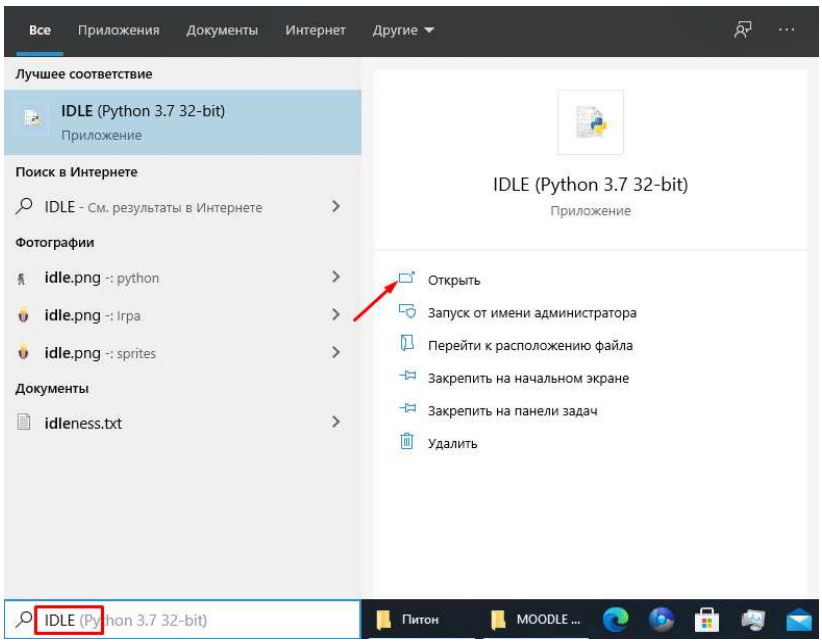


Рис.1

Щоб написати **“hello world”** на python достатньо усього однієї строки print("Hello world") та натиснемо **Enter**, Рис.2.

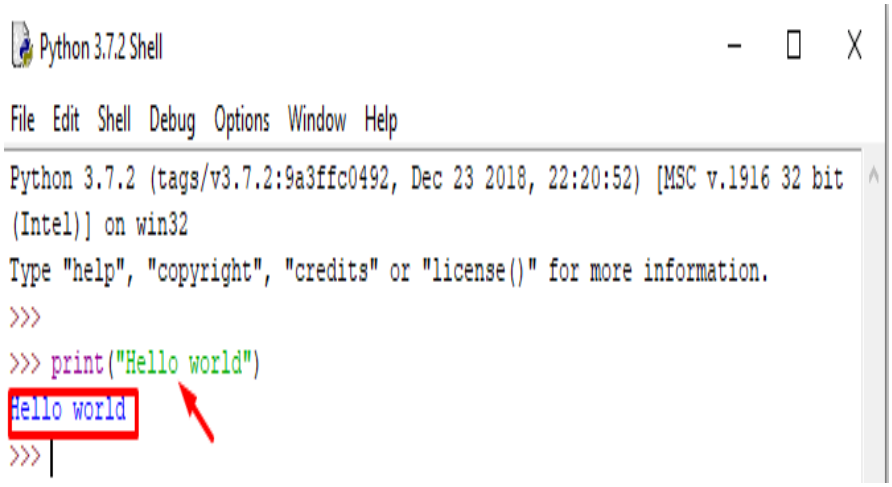


Рис.2

Інтерактивний режим не буде основним. В основному ви будете зберігати програмний код в файл і запускати вже файл. Для того, щоб створити нове вікно в інтерактивному режимі IDLE, виберіть File → New File (або натисніть Ctrl + N). Рис.3

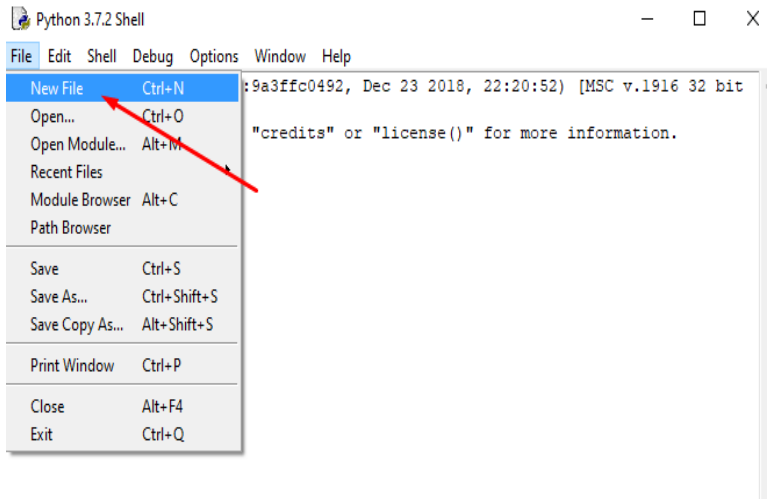


Рис.3

У вікні введіть наступний код: name = input ("Як Вас звати?") Print ("Привіт,", name) Рис.4

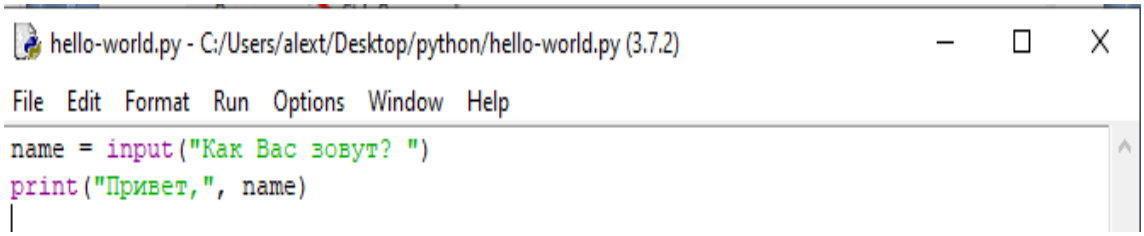


Рис.4

Тепер натиснемо F5 (або виберемо в меню IDLE Run → Run Module) і переконаємося, що програма, що ми написали, працює. Перед запуском IDLE запропонує нам зберегти файл. Збережемо туди, куди вам буде зручно, після чого програма запуститься.Рис.5,6

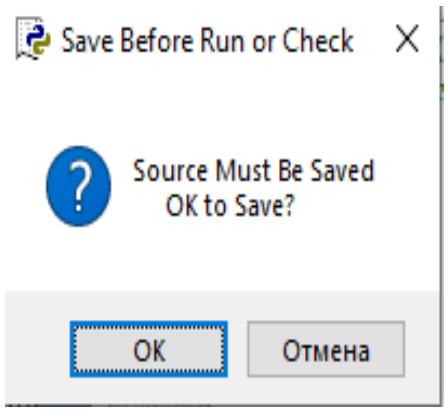


Рис.5

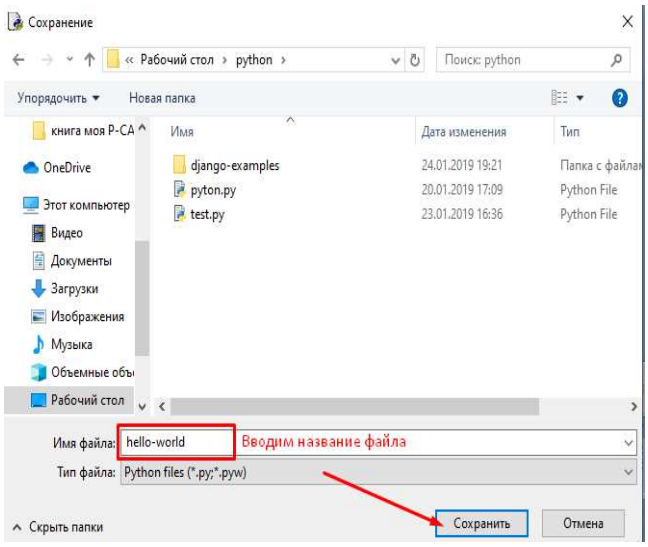


Рис.6

Перший рядок друкує питання ("Як Вас звати?") і очікує, поки ви не надрукуєте що-небудь і не натиснете Enter, і зберігає введене значення в змінній name. У другому рядку ми використовуємо функцію print для виведення тексту на екран, в даному випадку для виведення "Привіт," і того, що зберігається в змінної "name". Рис.7

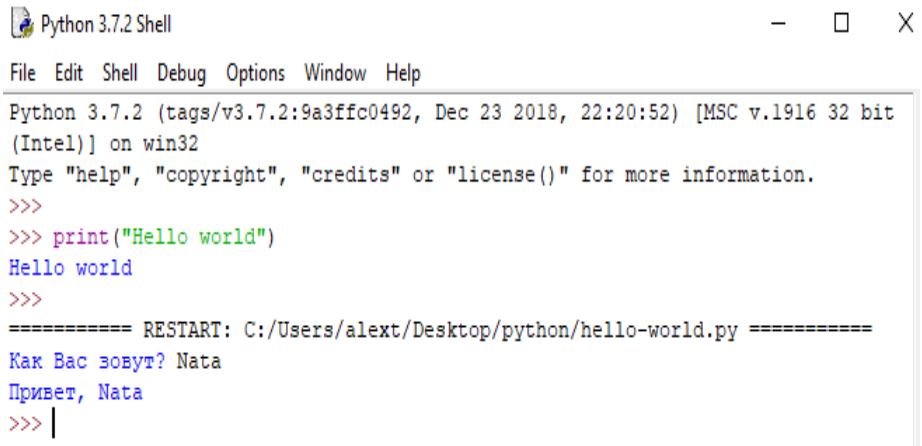


Рис.7

Синтаксис

1. Кінець рядка є кінцем інструкції (крапка з комою не потрібна).

2. Вкладені інструкції об'єднуються в блоки по величині відступів. Відступ може бути будь-яким, головне, щоб в межах одного

вкладеного блоку відступ був однаковий. І про читабельність коду не забувайте. Відступ в 1 пробіл, наприклад, не найкраще рішення. Використовуйте 4 пробілу (або знак табуляції, на худий кінець).

3. Вкладені інструкції в Python записуються відповідно до одним і тим же шаблоном, коли основна інструкція завершується двокрапкою, слідом за яким розташовується вкладений блок коду, зазвичай з відступом під рядком основний інструкції.

Введення і виведення даних

Введення даних здійснюється за допомогою команди input (список введення):

```
a = input ()
print (a)
```

У дужках функції можна вказати повідомлення - коментар до даних які вводяться:

```
a = input ( "Введіть кількість:")
```

Команда input () за замовчуванням сприймає вхідні дані як рядок символів. Тому, щоб ввести цілочисельне значення, слід вказати тип даних int ():

```
a = int (input ())
```

Для введення дійсних чисел застосовується команда

```
a = float (input ())
```

Вивід даних здійснюється за допомогою команди print (список виведення):

```
a = 1
b = 2
print (a)
print (a + b)
print ( 'сума =', a + b)
```

Існує можливість запису команд в один рядок, розділяючи їх через **;** . Однак не слід часто використовувати такий спосіб, це знижує читабельність:

```
a = 1; b = 2; print (a)
print (a + b)
print ( 'сума =', a + b)
```

Для команди print може додаватися так званий сепаратор - роздільник між елементами виведення:

```
x = 2
y = 5
print (x, "+", y, "=", x + y, sep = "")
```

Результат відобразиться з пробілами між елементами: 2 + 5 = 7

Для форматованого виведення використовується **format**:

Строковий метод format () повертає відформатовану версію рядка, замінюючи ідентифікатори в фігурних дужках {}. Ідентифікатори можуть бути позиційними, числовими індексами, ключами словників, іменами змінних.

Синтаксис команди format:

```
поле замены     ::= "{" [имя поля] ["!" преобразование] [":" спецификация] "}"
имя поля        ::= arg_name ("." имя атрибута | "[" индекс "]")*
преобразование  ::= "r" (внутреннее представление) | "s" (человеческое
↪представление)
спецификация    ::= см. ниже

спецификация ::= [[fill]align][sign][#][0][width][,][.precision][type]
заполнитель  ::= символ кроме '{' или '}'
выравнивание ::= "<" | ">" | "=" | "^"
знак         ::= "+" | "-" | " "
ширина       ::= integer
точность     ::= integer
тип          ::= "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" |
                 "n" | "o" | "s" | "x" | "X" | "%"
```

Аргументів на format () може бути більше, ніж ідентифікаторів в рядку. В такому випадку ті які залишилися - ігноруються.

Ідентифікатори можуть бути або індексами аргументів, або ключами:

Наприклад: вивід числа 11 без форматування. Рис.8

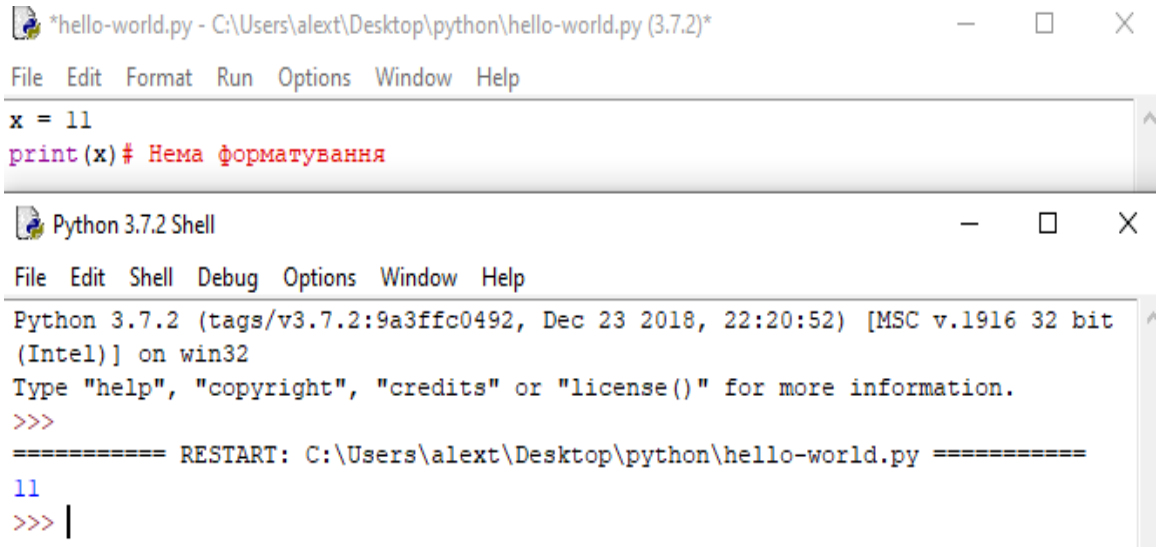


Рис.8

Наприклад: вивід числа 11 з форматуванням. Рис.9

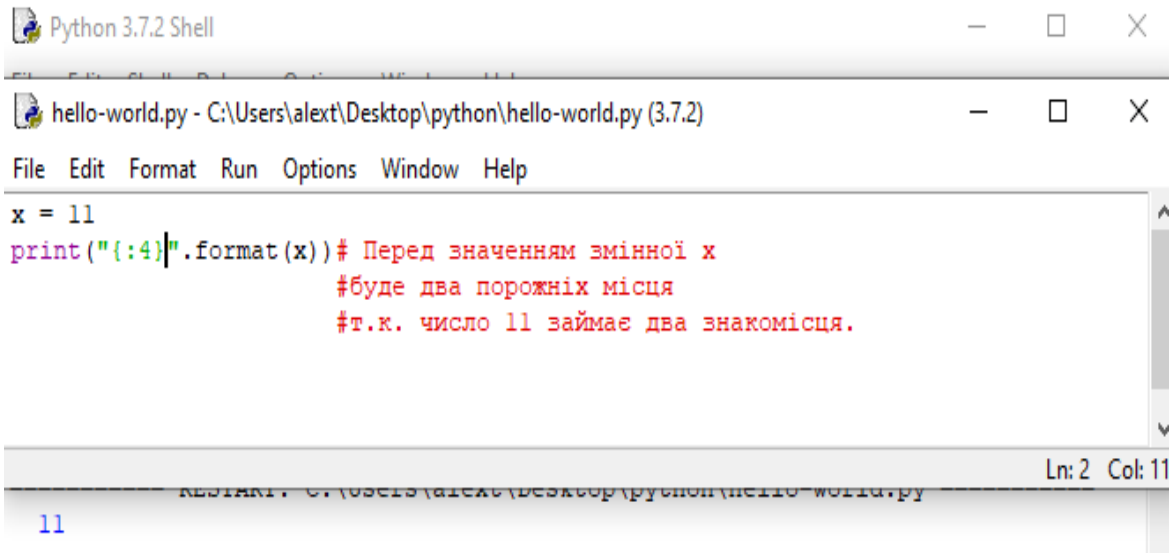


Рис.9

В результаті виведеться число 11, а перед ним два пробілу, так як було вказано використовувати для виведення чотири знакомісця.

Доступні наступні варіанті вирівнювання

'<' Символи-наповнювачі будуть праворуч (вирівнювання об'єкта по лівому краю) (за замовчуванням).

'>' Вирівнювання об'єкта по правому краю.

'=' Наповнювач буде після знака, але перед цифрами. Працює тільки з числовими типами.

'^' Вирівнювання по центру.

Наприклад:

```
>>> '{:>30}'.format('Nata')
'                          Nata'
>>> '{:<30}'.format('Nata')
'Nata                          '
>>> '{:*^30}'.format('Nata')
'**************Nata**************'
```

Опція «Знак»

'+' -Знак має бути надрукований для всіх чисел.

'-' -Минус для негативних, нічого для позитивних.

'**Пропуск**' - Минус для негативних, пробіл для позитивних.

Наприклад:

```
>>> '{:^30}'.format('Nata')
'             Nata             '
>>> '{:+f}; {:+f} '.format(3.14, -3.14)
'+3.140000; -3.140000 '
>>> '{: f}; {: f} '.format(3.14, -3.14)
' 3.140000; -3.140000 '
>>> '{:-f}; {:-f} '.format(3.14, -3.14)
'3.140000; -3.140000 '
>>>
```

Поле «Тип» може приймати наступні значення:

'd','i', 'u' -Десяткове число.

'o' -Число в вісімковій системі числення.

'x' -Число в шістнадцятковій системі числення (букви в нижньому регістрі).

'X' -Число в шістнадцятковій системі числення (літери у верхньому регістрі).

'e' -Число з плаваючою точкою з експонентою (експонента в нижньому регістрі).

'E' -Число з плаваючою точкою з експонентою (експонента в верхньому регістрі).

'f', 'F' -Число з плаваючою крапкою (звичайний формат).

'g' -Число з плаваючою крапкою з експонентою (експонента в нижньому регістрі), якщо вона менше, ніж -4 або точності, інакше звичайний формат.

'G' -Число з плаваючою крапкою. з експонентою (експонента в верхньому регістрі), якщо вона менше, ніж -4 або точності, інакше звичайний формат.

'c' -Символ (рядок з одного символу або число - код символу).

's' -Строка.

'%' -Число множиться на 100, відображається число з плаваючою точкою, а за ним знак%.

Для форматування дійсних чисел з плаваючою точкою використовується наступна команда:

print ('{0: .2f}'. format (дійсне число))

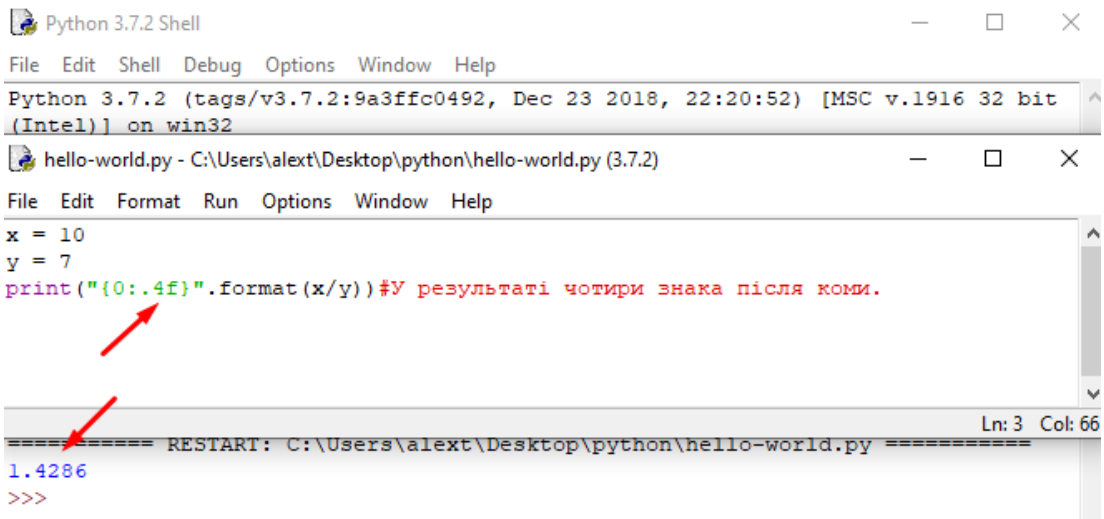


Рис.10

Типи даних

1. Цілі числа (тип int) - позитивні і негативні цілі числа, а також 0 (наприклад, 4, 687, -45, 0).

2. Числа з плаваючою точкою (тип float) - дробові, вони ж речові, числа (наприклад, 1.45, -3.789654, 0.00453). Примітка: для поділу цілої та дробової частин тут використовується точка, а не кома.

3. Строки (тип str) - набір символів, укладених в лапки (наприклад, "ball", "What is your name?", 'dkfjUUv', '6589'). Примітка: лапки в Python можуть бути одинарними або подвійними; одиночний символ в лапках також є рядком, окремого символьного типу в Пітоні немає.

Дії над цілими числами

Табл.1

^	Зведення у ступінь
*	Множення
/	Ділення
+	Додавання
-	Віднімання
//	Отримання цілої частини від ділення
%	Остача від ділення
-x	Зміна знаку числа
abs	Модуль числа
Divmod(x,y)	Пара(x//y,x%y)
Pow(x,y[,z])	X^y по модулю (якщо модуль надано)

Операції в програмуванні

Операція - це виконання будь-яких дій над даними, які в даному випадку називають операндами. Саму дію виконує оператор - спеціальний інструмент.

Так в математиці і програмуванні символ плюса є оператором операції додавання по відношенню до чисел. У разі рядків цей же оператор виконує операцію конкатенації - з'єднання.

Наприклад:

```
>>> 10.25 + 98.36
108.61
 >>> 'Hello' + 'World'
'HelloWorld'
```

Бітові операції

Над цілими числами також можна робити бітові операції табл.2.

Табл.2

|	Побітове але
^	Побітове виключаючи але
&	Побітове і
<<	Побітовий зсув вліво
>>	Побітовий зсув вправо
~	Інверсія бітів

Системи числення

• int ([object], [підставу системи числення]) - перетворення до цілого числа в десятковій системі числення. За замовчуванням система числення десяткова, але можна задати будь-яку підставу від 2 до 36 включно.

• bin (x) - перетворення цілого числа в двійкову систему числення.

• hex (x) - перетворення цілого числа в шістнадцятирічну систему числення.

• oct (x) - перетворення цілого числа в восьмирічну систему числення.

Числа (float) – з плаваючою комою

Речові числа підтримують ті ж операції, що і цілі. Однак (через представлення чисел в комп'ютері) речові числа неточні, і це може привести до помилок:

>>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1

Виведе 0.9999999999999999

Для високої точності використовують інші об'єкти (наприклад Decimal і Fraction). Також речові числа не підтримують довгу арифметику.

Наприклад:

```
>>> a = 3 ** 1000
>>> a + 0.1
Traceback (most recent call last):
File "", line 1, in
OverflowError: int too large to convert to float
```

Наприклад:

```
>>> c = 150
>>> d = 12.9
>>> c + d
162.9
>>> p = abs(d - c) # Модуль числа
>>> print(p)
137.1
>>> round(p) # Округлення
137
```

Додаткові методи роботи з цілими числами:

float.as_integer_ratio () - пара цілих чисел, чиє ставлення одно цього числа.

float.is_integer () - чи є значення цілим числом.

float.hex () - переводить float в hex (шістнадцятирічну систему числення).

classmethod **float.fromhex (s)** - float з шістнадцяткового рядка.

Модуль **math** надає більш складні математичні функції

Наприклад:

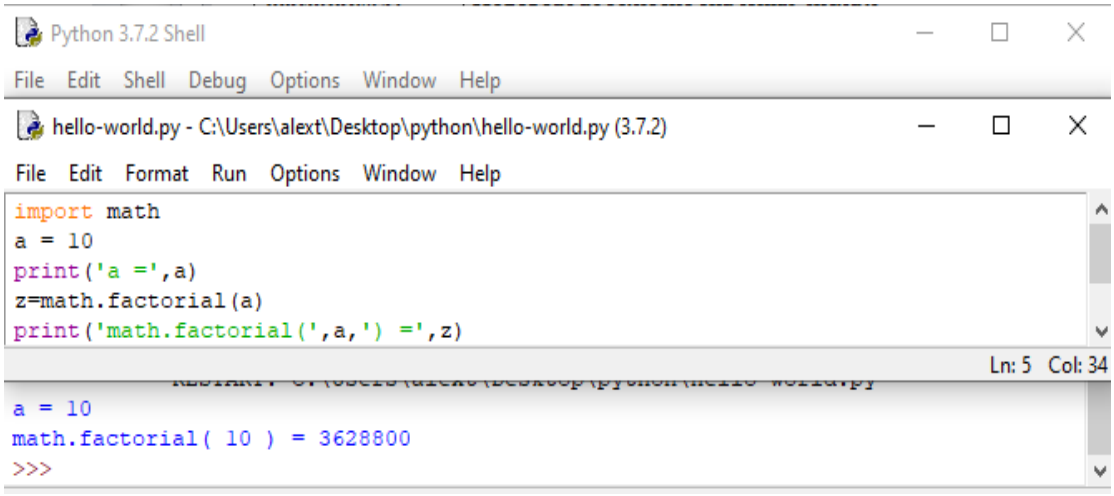
```
>>> import math
>>> math.pi
3.141592653589793
>>> math.sqrt(85)
9.219544457292887
```

Бібліотека модуля math (деякі функції)

Табл.3

math.ceil(x)	Повертає найближче ціле число більше, ніж x
math.fabs(x)	Повертає абсолютне значення числа x
math.factorial(x)	Обчислює факторіал x
math.floor(x)	Повертає найближче ціле число менше, ніж x
math.exp(x)	Обчислює e**x
math.log2(x)	Логарифм по основі 2
math.log10(x)	Логарифм по основі 10
math.log(x[, base])	За замовчуванням обчислює логарифм за основою e, додатково можна вказати основу логарифма
math.pow(x, y)	Обчислює значення x в ступені y
math.sqrt(x)	Корень квадратний від x

Наприклад:



Тригометричні функції модуля math

math.cos(x)	Повертає cos числа x
math.sin(x)	Повертає sin числа x
math.tan(x)	Повертає tan числа x
math.acos(x)	Повертає acos числа x
math.asin(x)	Повертає asin числа x
math.atan(x)	Повертає atan числа x

Модуль **random** реалізує генератор випадкових чисел і функції випадкового вибору.

Наприклад:

```
>>> import random
>>> random.random()
0.15651968855132303
```

Робота з комплексними числами у прикладах:

```
>>> x = complex(1, 2)
>>> print(x)
(1+2j)
>>> y = complex(3, 4)
>>> print(y)
(3+4j)
>>> z = x + y
>>> print(z)
(4+6j)
>>> z = x * y
>>> print(z)
(-5+10j)
>>> z = x / y
>>> print(z)
(0.44+0.08j)
>>>
```

Комплексні числа не можна порівнювати!!!

Зміна типів даних

Для зміни одних типів даних в інші в мові Python передбачений ряд вбудованих в нього функцій.

Ці функції перетворять те, що поміщається в їх дужки відповідно в ціле число, дійсне число або рядок.

Наприклад:

```
>>> str(1) + 'a'
'1a'
>>> int('3') + 4
7
>>> float('3.2') + int('2')
5.2
>>> str(4) + str(1.2)
'41.2'
```

Правила написання імен змінних

1.Бажано давати змінним осмислені імена, що говорять про призначення даних, на які вони посилаються.

2.Ім’я змінної не повинно збігатися з командами мови (зарезервованими ключовими словами).

3.Ім’я змінної має починатися з букви або символу підкреслення (_), але не з цифри.

4.Ім’я змінної не повинно містити пробіли.

Щоб дізнатися значення, на яке посилається змінна, перебуваючи в режимі інтерпретатора, досить її викликати, тобто написати ім'я і натиснути Enter.

Наприклад:

```
>>> sq = 4
>>> sq
4
>>>
```

Приклад роботи зі змінними в інтерактивному режимі:

```
>>> apples = 100
>>> eat_day =5
>>> day = 7
>>> apples = apples - eat_day * day
>>> apples
65
>>>
```

Практичне завдання

1.Змінній var_int надайте значення 10, var_float - значення 8.4, var_str - "No".

2.Змініть значення, збережене в змінної var_int, збільшивши його в 3.5 рази, результат зв'яжіть зі змінною var_big.

3.Змініть значення, збережене в змінної var_float, зменшивши його на одиницю, результат зв'яжіть з тієї ж змінної.

4.Розділіть var_int на var_float, а потім var_big на var_float. 5.Результат даних виразів не прив'язуйте до жодних змінних.

6.Виведіть значення всіх змінних.

7. Напишіть програму, яка запитувала б у користувача:

- ПІБ ("Ваші прізвище, ім'я, по батькові?")

- вік ("Скільки Вам років?")

- місце проживання ("Де Ви живете?")

-де Ви навчаєтесь (“Де Ви навчаєтесь?”)

-номер Вашої групи(“Номер Вашої групи?”)

-порядковий номер по списку у группі(“Який Ваш порядковий номер у списку групи?”)

-питання відповідно до варіанту

Після цього виводила б рядки:

"Ваше ім'я"

"Ваш вік"

"Ви живете в"

“Ви навчаєтесь в”.

“Номер моєї групи -”

“Мій порядковий номер у списку групи-“

«Ваш варіант відповіді»

№ Варіанту - остання цифра у списку групи.

№ Варіанту	Питання
0	Як справи?
1	Як Ви себе почуваєте?
2	Коли будете вдома?
3	Яку оцінку отримав на ЗНО по українській мові?
4	Сьогодні сонячно?
5	Коли нарешті карантин?
6	Як звати Вашого друга?
7	Ви думаєте вступати у магістратуру?
8	Якого кольору Ваш зошит?
9	Який Ваш настрій сьогодні?

8. Написати програму для розрахунку Z

$$Z = \frac{9\pi t + 10\cos(x)}{\sqrt{t} - |\sin(t)|} * e^{x}$$

Для введення даних використовуйте команду input, визначивши тип змінних.

Результат вивести з двома знаками після коми.

x - остання цифра у списку групи, t=1.

Практична робота №2

Тема. Інструкція if-elif-else, перевірка істинності, тримісний вираз if / else. Цикли for і while, оператори break і continue, else.

Мета роботи. Отримати навички роботи з циклами.

Зміст.

1. Вивчення відомостей про роботу з циклами.
2. Виконання роботи.
3. Отримання результату.

Ключові положення.

Умовна інструкція if-elif-else (її ще іноді називають оператором розгалуження) - основний інструмент вибору в Python. Простіше кажучи, вона вибирає, яку дію слід виконати, в залежності від значення змінних в момент перевірки умови.

Синтаксис інструкції if

Спочатку записується частина if з умовним виразом, далі можуть слідувати одна або більше необов'язкових частин elif, і, нарешті, необов'язкова частина else. Загальна форма записи умовної інструкції if виглядає наступним чином:

```
  if test1:
    state1
elif test2:
    state2
else:
    state3
```

Наприклад:

```
a = int(input())
if a < -5:
    print('Low')
elif -5 <= a <= 5:
    print('Mid')
else:
    print('High')
```

Перевірка істинності в Python

• Будь-яке число, не рівне 0, або непорожній об'єкт - істина.

• Числа, рівні 0, порожні об'єкти і значення None - брехня

• Операції порівняння застосовуються до структур даних рекурсивно

• Операції порівняння повертають True або False

• Логічні оператори and і or повертають істинний або помилковий об'єкт-операнд.

Логічні оператори:

X and Y

Істина, якщо обидва значення X і Y істинні.

X or Y

Істина, якщо хоча б одне зі значень X або Y істинно.

not X

Істина, якщо X ложно.

Тримісний вираз if / else

Наступна інструкція:

```
if X:
A = Y
else:
A = Z
```

Досить коротка, але, тим не менше, займає цілих 4 рядки. Спеціально для таких випадків і було придумано вираз if / else:

```
A = Y if X else Z
```

У даній інструкції інтерпретатор виконає вираз Y, якщо X істинно, в іншому випадку виконається вираз Z.

```
>>> A = 't' if 'spam' else 'f'
>>> A
 't'
```

Цикл while

Виконує тіло циклу до тих пір, поки умова циклу істинно.

Наприклад:

```
>>> i = 5
>>> while i < 15:
... print(i)
... i = i + 2
...
5
7
```

9

11

13

Цикл for

Цикл for вже трішки складніший, трохи менш універсальний, але виконується набагато швидше циклу while. Цей цикл проходиться по будь-якому ітеріруемому об'єкту (наприклад рядку або списку), і під час кожного проходу виконує тіло циклу.Рис.1

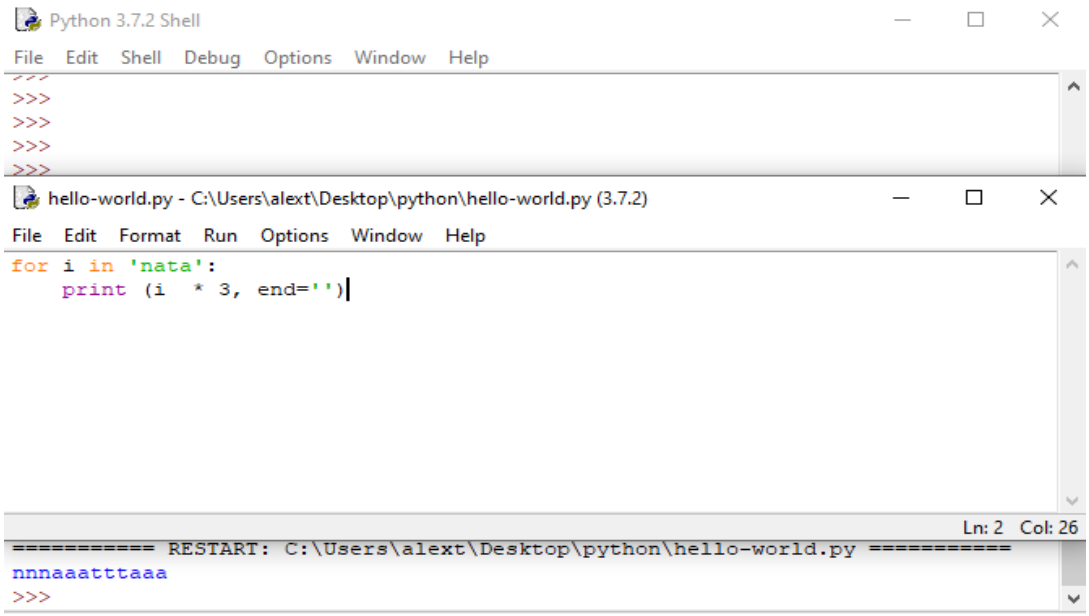


Рис.1

Без end = '' – розташовується у стовпчик. Рис.2

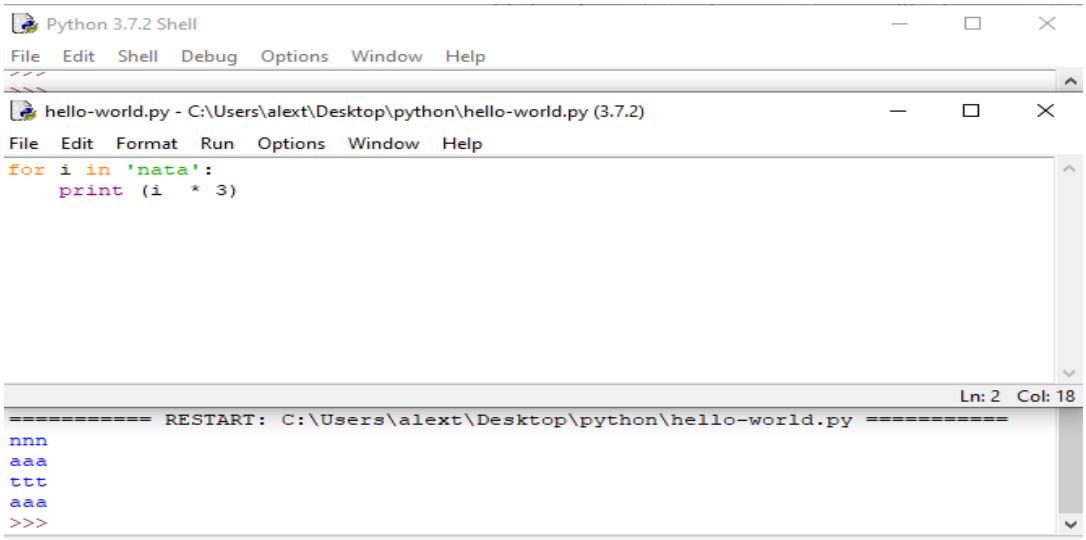


Рис.2

Оператор break

Достроково переривається цикл. Рис.3

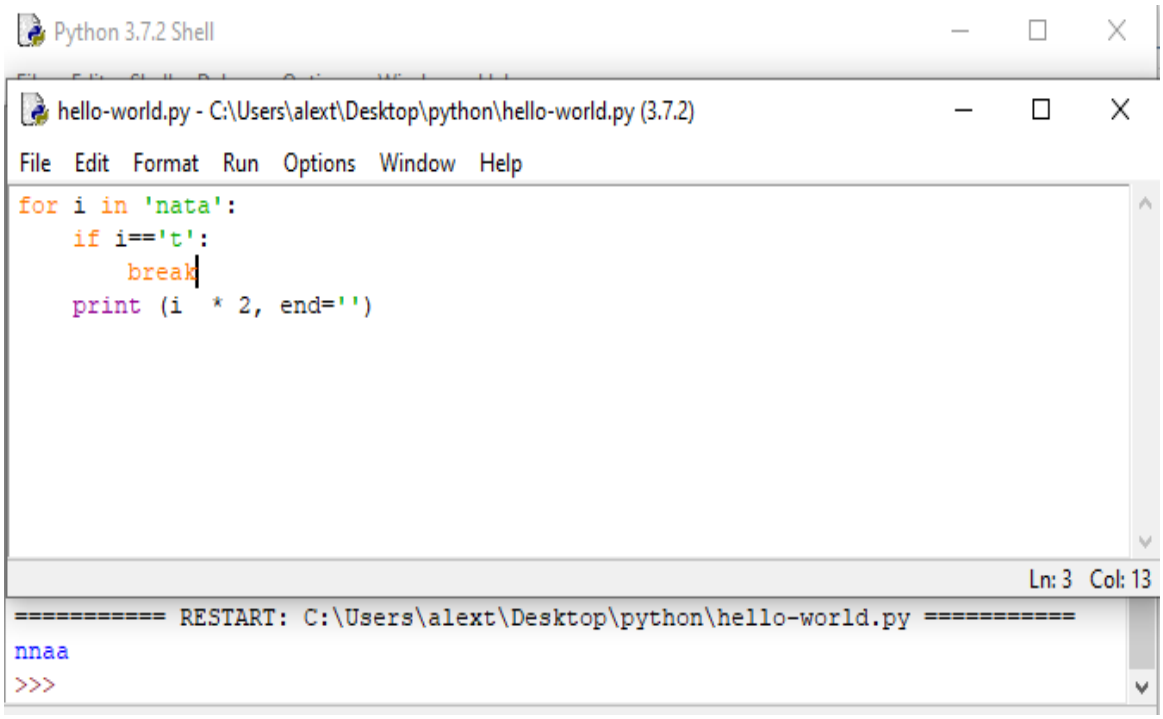


Рис.3

Оператор continue

Починає наступний прохід циклу, минаючи решту тіла циклу (for або while). Рис.4

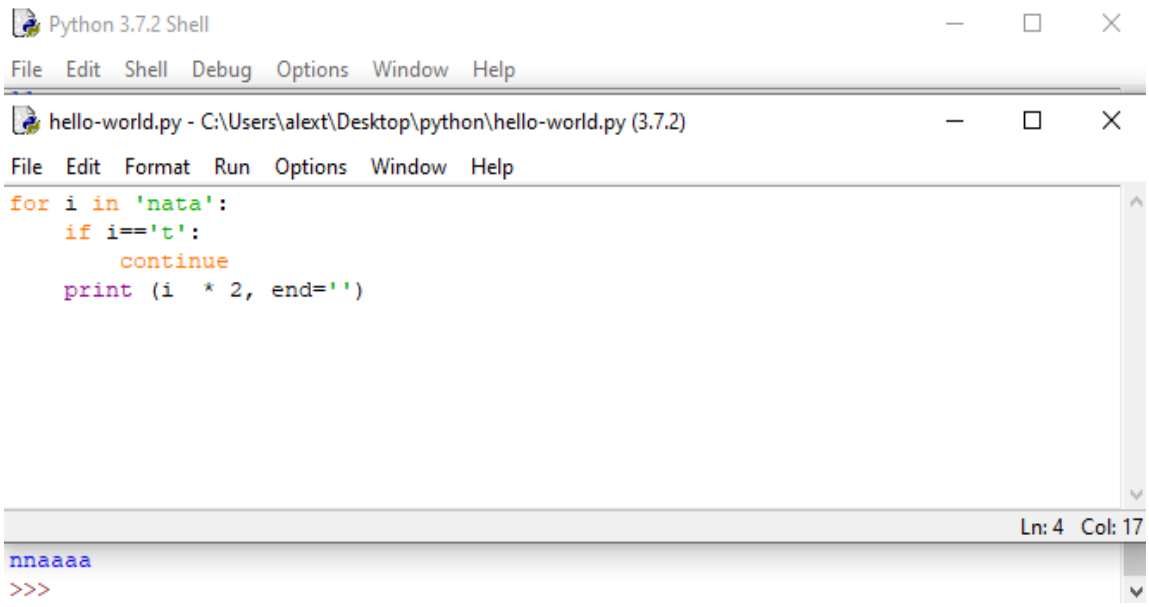


Рис.4

Слово **else**, застосоване в циклі for або while, перевіряє, чи був проведений вихід з циклу інструкцією break, або ж "природним" чином. Рис.5

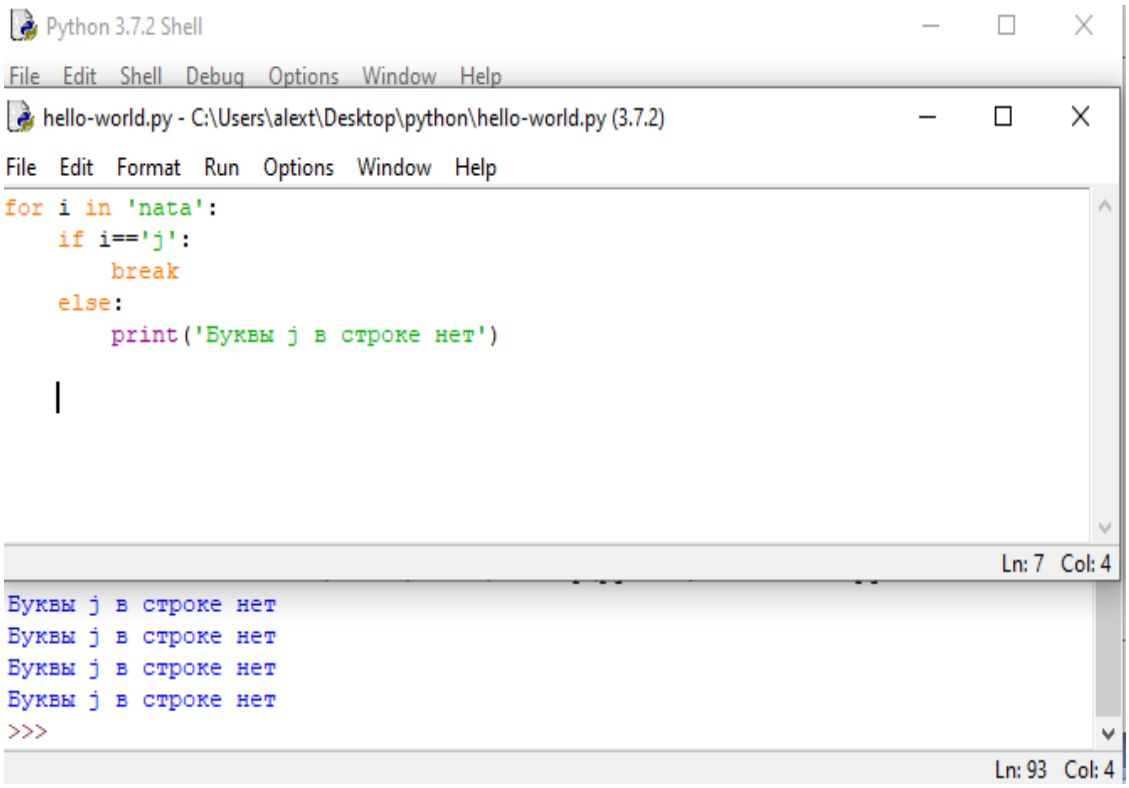


Рис.5

Практичне завдання

1. Знайти суму n елементів наступного ряду чисел: 1 -0.5 0.25 -0.125 ... n. Кількість елементів (n) вводиться з клавіатури. Вивести на екран кожен член ряду і його суму. Вирішити задачу використовуючи циклічну конструкцію for.

2. Дано ціле число, що не менше 2. Виведіть його найменший натуральний дільник, відмінний від 1 Вирішити задачу використовуючи циклічну конструкцію while. (У циклі while в якості логічного виразу використовується команда n% і порівнювана з нулем.)

3. Вичисліть значення функції:

$$f(x)=\begin{cases}0{,}5-4x, & x\ge 0\\ \sin^2 x^2, & x<0\\ x+1, & \end{cases}$$

4. Дано три цілих числа. Вибрати з них ті, які належать інтервалу [1,№]. Де №- остання цифра порядкового номеру у списку групи.

5. Дано 3 будь яких чисел. Знайти мінімальне серед них і вивести на екран. (Для №- парного). Знайти максимальне серед них і вивести на екран. (Для №- непарного).

6. Знайти суму n елементів наступного ряду чисел: 1 -0.5 0.25 -0.125 ... n. Кількість елементів (n) вводиться з клавіатури. Вивести на екран кожен член ряду і його суму. Вирішити задачу використовуючи циклічну конструкцію for.

7. Дано ціле число, що не менше 2. Виведіть його найменший натуральний дільник, відмінний від 1 Вирішити задачу використовуючи циклічну конструкцію while. (У циклі while в якості логічного виразу використовується команда n% і порівнювана з нулем.)

8. Написати програму. Програма має вивести ряд послідовності Фібоначчі. Кожен наступний член дорівнюється суммі двох попередніх.
0, 1, 1, 2, 3, 5, 8, 13……..

9. “Вгадайте число від 1 до 20”
9.1. По запрошенню “ Введіть число від 0 до 20” вводиться число.
9.2. Генератор випадкових чисел генерує послідовність від 0 до 20.
9.3. Якщо запропоноване число менше генерованого виведіть “Запропоноване число менше задуманого”.
9.4. Якщо запропоноване число більше генерованого виведіть “Запропоноване число більше задуманого” .

9.5. Якщо числа співпадають виводиться на якому кроці числа співпали та коментарій “Ви вгадали число з …. разу”.
9.6. Якщо Ви ввели число не в діапазоні 0-20 вивести “Невірне значення!!!”

Практична робота №3

Тема. Функції і методи рядків.

Мета роботи. Отримати навички роботи з функціями та рядками.

Зміст.

1. Вивчення відомостей про роботу з функціями та рядками.
2. Виконання роботи.
3. Отримання результату.

Ключові положення.

Рядки в апострофах і в лапках - одне і те ж. Причина наявності двох варіантів в тому, щоб дозволити вставляти в літерали рядків символи лапок і апострофів, не використовуючи екранування. Екрановані послідовності дозволяють вставити символи, які складно ввести з клавіатури.

У таблиці 1 приведено список функцій та методів рядків.

Таблиця 1

Функція або метод	Призначення
S = ‘str’; S = “str”; S = ‘’‘str’‘’; S = “””str”””	літерали рядків
S = “s\np\ta\nbbb”	екрановані послідовності
S = r”C:\temp\new”	Неформатовані рядки (пригнічують екранування)
S = b”byte”	рядок байтів
S1 + S2	Конкатенація (додавання рядків)

S1 * 3	повторення рядка
S[i]	Звернення за індексом
S[i:j:step]	витяг зрізу
len(S)	довжина рядка
S.find(str, [start],[end])	Пошук підрядка в рядку. Повертає номер першого входження або -1
S.rfind(str, [start],[end])	Пошук підрядка в рядку. Повертає номер останнього входження або -1
S.index(str, [start],[end])	Пошук підрядка в рядку. повертає номер першого входження або викликає ValueError
S.rindex(str, [start],[end])	Пошук підрядка в рядку. Повертає номер останнього входження або викликає ValueError
S.replace(шаблон, заміна)	заміна шаблону
S.split(символ)	Розбиття рядка по розмежувачу
S.isdigit()	Чи полягає рядок з цифр
S.isalpha()	Чи полягає рядок з букв
S.isalnum()	Чи полягає рядок з цифр або букв
S.islower()	Чи полягає рядок із символів в нижньому регістрі
S.isupper()	Чи полягає рядок із символів у верхньому регістрі
S.isspace()	Чи має рядок символи, що не відображаються (пробіл, символ перекладу страниці ('\ f'), "новий рядок" ('\ n'), "переклад каретки" ('\ r'), "горизонтальна табуляція" ('\ t') і "вертикальна табуляція " ('\ v'))
S.istitle()	Чи починаються слова в рядку з великої літери
S.upper()	Перетворення рядка до верхнього регістру

S.lower()	Перетворення рядка до нижнього регістру
S.startswith(str)	Чи починається рядок S з шаблону str
S.endswith(str)	Закінчується рядок S шаблоном str
S.join(список)	Збірка рядка зі списку з роздільником S
ord(символ)	Символ в його код ASCII
chr(число)	Код ASCII в символ
S.capitalize()	Змінює перший символ рядка в верхній регістр, а всі інші в нижній
S.center(width, [fill])	Повертає відцентрований рядок, по краях якого стоїть символ fill (пробіл за замовчуванням)
S.count(str, [start],[end])	Повертає кількість непересічних входжень підрядка в діапазоні [початок, кінець] (0 і довжина рядка за замовчуванням)
S.expandtabs([tabsize])	Повертає копію рядка, в якій всі символи табуляції замінюються одним або декількома пропусками, в залежності від поточного стовпця. Якщо TabSize НЕ вказано, розмір табуляції вважається рівним 8 прогалин
S.lstrip([chars])	Видалення символів пробілів на початку рядка
S.rstrip([chars])	Видалення символів пробілів у кінці рядку
S.strip([chars])	Видалення символів пробілів на початку і в кінці рядка
S.partition(шаблон)	Повертає кортеж, що містить частину перед першим шаблоном, сам шаблон, і частину після

	шаблону. Якщо шаблон не знайдений, повертається кортеж, що містить сам рядок, а потім дві порожніх рядки
S.rpartition(sep)	Повертає кортеж, що містить частину перед останнім шаблоном, сам шаблон, і частину після шаблону. Якщо шаблон не знайдений, повертається кортеж, що містить дві порожні рядки, а потім сам рядок
S.swapcase()	Перекладає символи нижнього регістра в верхній, а верхнього - в нижній
S.title()	Першу букву кожного слова переводить в верхній регістр, а всі інші в нижній
S.zfill(width)	Робить довжину рядку не меншою width, в разі потреби заповнюючи перші символи нулями
S.ljust(width, fillchar=” ”)	Робить довжину рядку не меншою width, в разі потреби заповнюючи останні символи символом fillchar
S.rjust(width, fillchar=” ”)	Робить довжину рядку не меншою width, в разі потреби заповнюючи перші символи символом fillchar
S.format(*args, **kwargs)	форматування рядка

Наприклад:

Конкатенація (додавання)

```
>>> S1 = 'spam'
>>> S2 = 'eggs'
>>> print(S1 + S2)
'spameggs'
```

Наприклад:

Дублювання рядка

```
>>> print('spam' * 3)
spamspamspam
```

Наприклад:

Доступ до індексу

```
>>> S = 'spam'
>>> S[0]
's'
>>> S[2]
'a'
>>> S[-2]
'a'
```

Як видно з прикладу, в Python є можливість доступу по негативному індексу, при цьому відлік йде від кінця рядка.

Витяг зрізу. Оператор вилучення зрізу: [X: Y]. X - початок зрізу, а Y - закінчення; символ з номером Y в зріз не входить. За замовчуванням перший індекс дорівнює 0, а другий - довжині рядка.

Наприклад:

```
>>> s = 'spameggs'
>>> s[3:5]
'me'
>>> s[2:-2]
'ameg'
>>> s[:6]
'spameg'
>>> s[1:]
'pameggs'
>>> s[:]
'spameggs‘
```

Можна задати крок з яким треба витягувати зріз.

Наприклад:

```
>>> s[::-1]
'sggemaps'
>>> s[3:5:-1]
''
>>> s[2::2]
'aeg'
```

Практичне завдання

1. Присвоїть s1= рядок «Зараховано».
2. Присвоїть s2= рядок «Сенсація».
3. Присвоїть s3= рядок «Сенсація* Сенсація* Сенсація».
4. Присвоїть s4= рядок «ОхОхОхАх».
5. За допомогою команди print вивести значення s1= Зараховано, s2= Сенсація, s3= Сенсація* Сенсація* Сенсація, s4=ОхОхОхАх
6. Вивести суму рядків s1 та s2.
7. Повторити рядок s1 чотири рази.
8. Вивести елемент рядку s1 з індексом 3.
9. Витяг зрізу рядку s1 починаючи з індексу 2 та завершуючи індексом 4.
10. Дізнатись кількість входжень підрядка s2 у рядок s3. Результат вивести на екран.
11. Перевести усі символи s1 у верхній регістр. Результат вивести на екран.
12. Перевести усі символи s1 у нижній регістр. Результат вивести на екран.
13. Розбити на розділи по символу * рядок s3. Результат вивести на екран.
14. Перевірити, чи буде рядок читатися однаково справа наліво і зліва направо (тобто чи є він паліндромом). Рядок **«а роза упала на лапу азора»**

15. У рядку таблиці варіантів замінити букву (а) буквою (о). Підрахувати кількість замін. Підрахувати, скільки символів в рядку.

№ Варіант - остання цифра у списку групи.

№	Рядок
1	любите отдыхать на природе
2	палатках не для вас
3	заполненные отели на курортах утомляют
4	украинский стартап предлагает новый формат отдыха
5	речь идет о геодезических куполах
6	альтернатива гостиничным номерам
7	идея стартапа родилась несколько лет назад
8	компактные сферические домики на природе
9	купольные конструкции на склоне горы в окружении гор
0	невероятная близость к природе

Практична робота №4

Тема. Списки (list). Функції і методи списків. Одномірні масиви. Кортежі.

Мета роботи. Отримати навички роботи з списками, кортежами.

Зміст.

1. Вивчення відомостей про роботу з списками, кортежами.
2. Виконання роботи.
3. Отримання результату.

Ключові положення.

Списки в Python - впорядковані змінювані колекції об'єктів довільних типів. (Майже як масив, але типи можуть відрізнятися).

Щоб використовувати списки, їх потрібно створити. Створити список можна декількома способами. Наприклад, можна обробити будь-який ітеріруемий об'єкт (наприклад, рядок) вбудованою функцією **list:**

```
>>> list('список')
['с', 'п', 'и', 'с', 'о', 'к']
>>> s = [] #
>>> l = ['s', 'p', ['isok'], 2]
>>> s
[]
>>> l
['s', 'p', ['isok'], 2]
>>>
```

Як видно з прикладу, список може містити будь-яку кількість будь-яких об'єктів (у тому числі і вкладені списки), чи нічого не містити.

І ще один спосіб створити список - це генератори списків. Генератор списків – спосіб побудувати новий список, застосовуючи вираз до кожного елементу послідовності.

Генератори списків дуже схожі на цикл for.

Наприклад:

```
>>> c = [c * 3 for c in 'list']
>>> c
['lll', 'iii', 'sss', 'ttt']
```

Наприклад:

```
>>> c = [c*3 for c in 'list' if c !='i']
>>> c
['lll', 'sss', 'ttt']
>>> c = [c*3 for c in 'listok' if c !='i']
>>> c
['lll', 'sss', 'ttt', 'ooo', 'kkk']
>>>
```

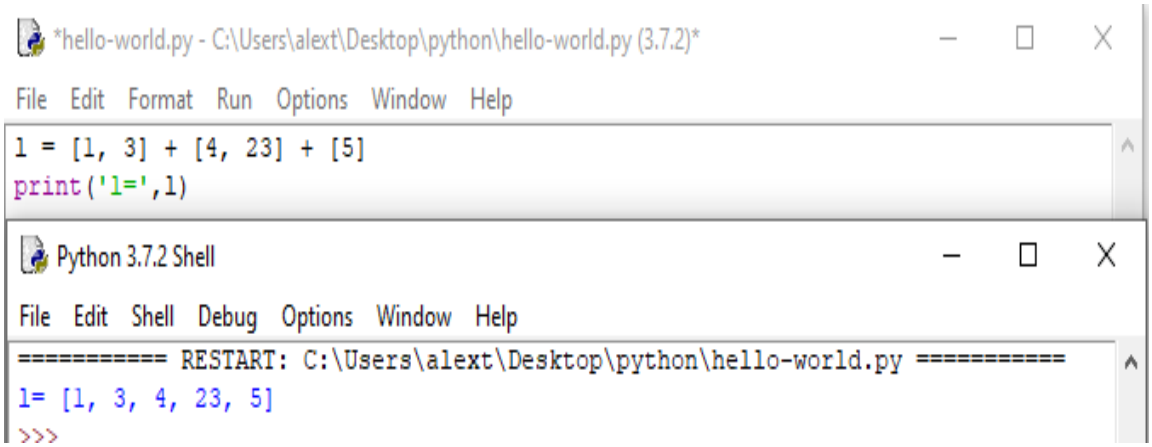
Для списків доступні основні вбудовані функції, а також методи списків.

Табл.1

Метод	Що робить
list.append(x)	Додає елемент в кінець списку
list.extend(L)	Розширює список list, додаючи в кінець все елементи списку L
list.insert(i, x)	Вставляє на і-ий елемент значення x
list.remove(x)	Видаляє перший елемент у списку, який має значення x. ValueError, якщо такого елемента не існує.
list.pop([i])	Видаляє і-ий елемент і повертає його. Якщо індекс не вказано,видаляється останній елемент
list.index(x, [start [, end]])	Повертає положення першого елемента зі значенням x (при цьому пошук ведеться від start до end)
list.count(x)	Повертає кількість елементів зі значенням x
list.sort([key= функція])	Сортує список на основі функції
list.reverse()	Розгортає список
list.copy()	Поверхнева копія списку
list.clear()	Очищає список

Списки можна додавати за допомогою знака **«+»**

Наприклад:



Створення списку за допомогою функції **Split ().**

Використовуючи функцію split в Python можна отримати з рядка список.

stroka = "Привіт, країна"

lst = stroka.split (",")

```
stroka ="Здравствуй, Дедушка Мороз"  #stroka - строка
lst=stroka.split(",") #lst - список
print('stroka = ',stroka)
print('lst=stroka.split(","):',lst)
```

Результат:

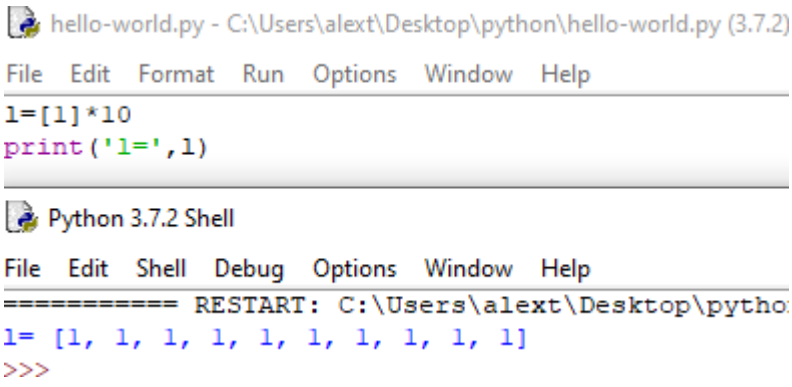
```
============== RESTART: C:/Users/maxim/Desktop/ex_list_
stroka =  Здравствуй, Дедушка Мороз
lst=stroka.split(","): ['Здравствуй', ' Дедушка Мороз']
```

Генератори списків.

1. Спосіб – складення однакових списків замінюється множенням. Список з 10 елементів заповнених одиницями:

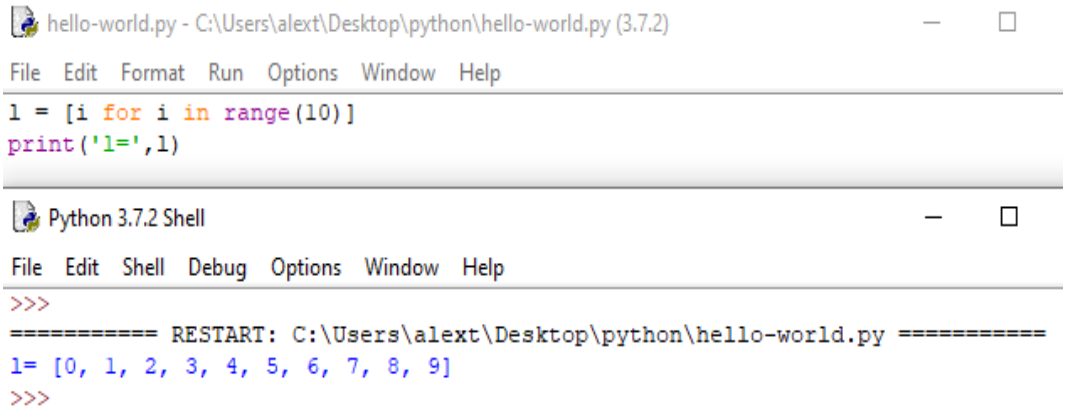
l=[l]*10

print('l=',l)



2. Спосіб

l = [i for i in range(10)]



Модуль random надає функції для генерації випадкових чисел, букв, випадкового вибору елементів послідовності.

random.randint (A, B) - випадкове ціле число N, $A \leq N \leq B$.

random.random () - випадкове число від 0 до 1.

Випадкові числа в списку:

10 чисел, генерованих випадковим чином в діапазоні (10,80)

from random import randint

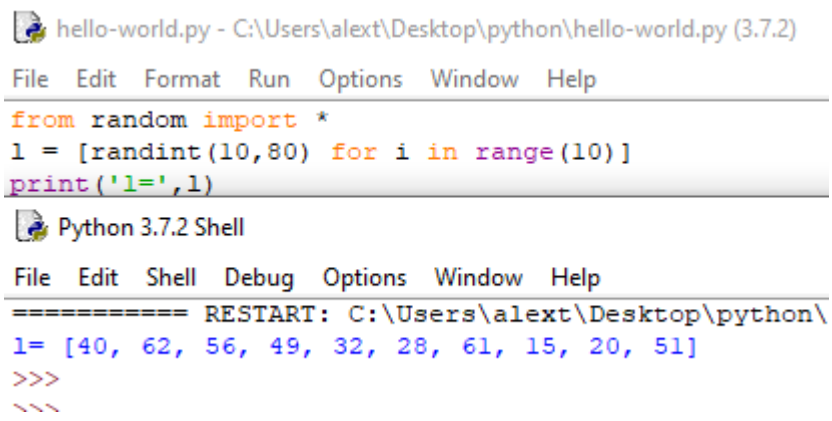
l = [randint (10,80) for x in range (10)]

10 чисел, генерованих випадковим чином в діапазоні (0,1)

l = [random () for i in range (10)]

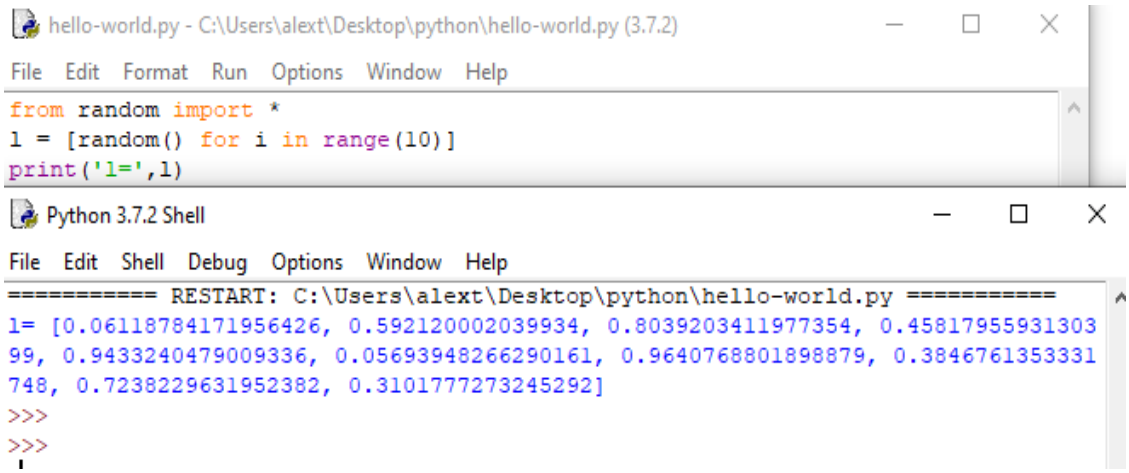
Наприклад:

Десять чисел генерованих випадковим чином у інтервалі 10-80.



Наприклад:

Десять чисел генерованих випадковим чином у інтервалі 0-1.



Наприклад:

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]

>>> a = [66.25, 333, 333, 1, 1234.5]
>>> a.remove(333)
>>> a
[66.25, 333, 1, 1234.5]

>>> a = [66.25, 333, 333, 1, 1234.5]
>>> a.reverse()
>>> a
[1234.5, 1, 333, 333, 66.25]

>>> a = [66.25, 333, 333, 1, 1234.5]
>>> a.sort()
>>> a
[1, 66.25, 333, 333, 1234.5]
```

Введення списку (масиву) в мові Python.

Для введення елементів списку використовується цикл for і команда range ():

for i in range (N):

x [i] = int (input ())

Простіший варіант введення списку:

x = [int (input ()) for i in range (N)]

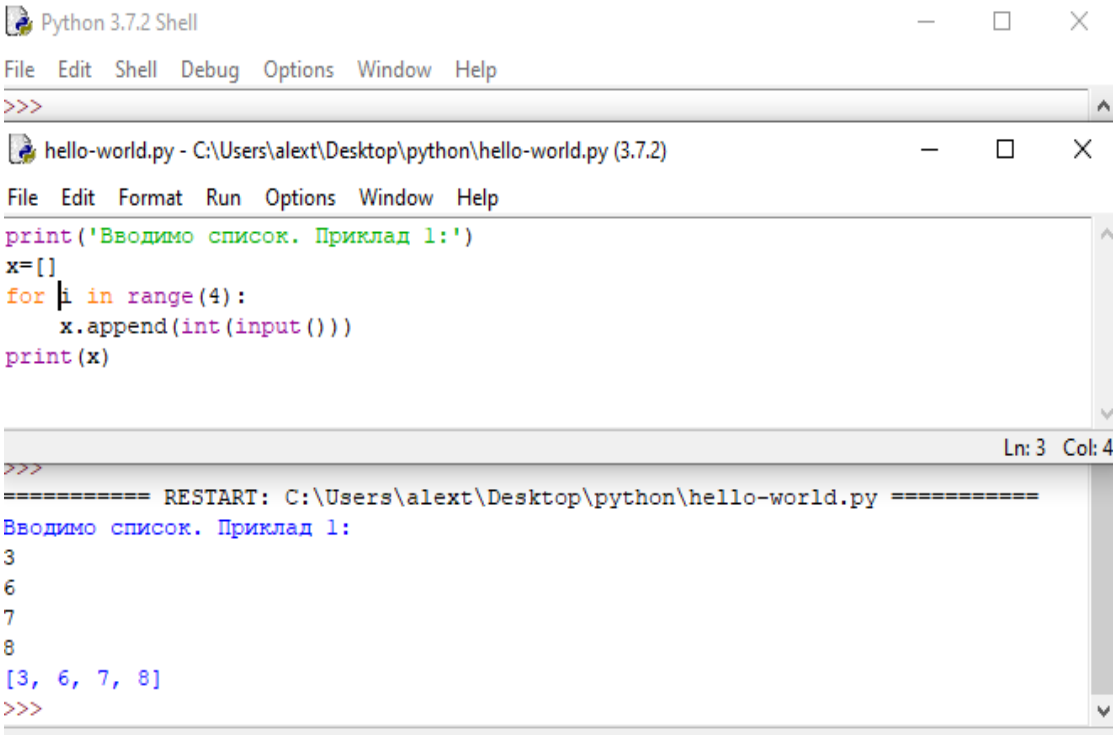
Вивід цілого списку (масиву):

print (L)

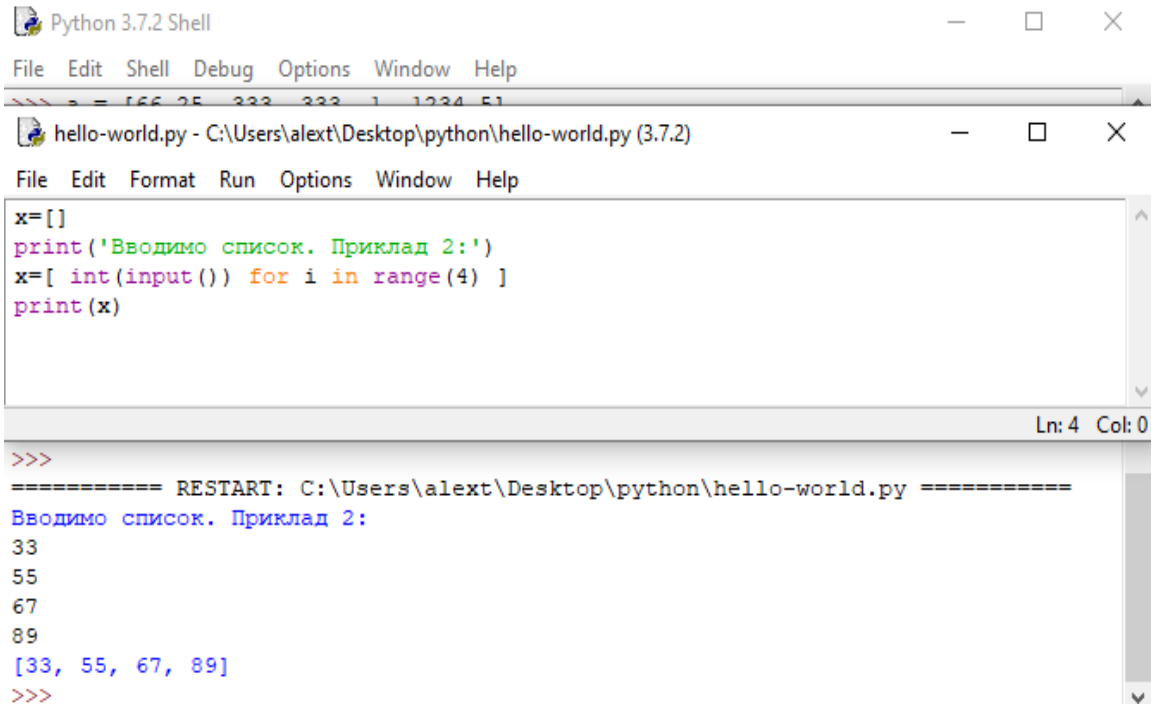
Поелементний вивід списку :

```
for i in range(N):
        print ( L[i], end = " " )
```

Наприклад:



Наприклад:



Взяття елемента за індексом

```
>>> a = [1, 3, 8, 7]
>>> a[0]
1
>>> a[3]
7
>>> a[4]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Як і в багатьох інших мовах, нумерація елементів починається з нуля. При спробі доступу до неіснуючого індексу виникає виняток IndexError. В даному прикладі змінна a була списком, однак взяти елемент за індексом можна і у інших типів: рядків, кортежів.

В Python також підтримуються негативні індекси, при цьому нумерація йде з кінця, наприклад:

```
>>> a = [1, 3, 8, 7]
>>> a[-1]
7
>>> a[-4]
1
>>> a[-5]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Зрізи

В Python, крім індексів, існують ще й зрізи.

item [START: STOP: STEP] - бере зріз від номера START, до STOP (не включаючи його), з кроком STEP.

За замовчуванням START = 0, STOP = довжина об'єкта, STEP = 1. Відповідно, якісь (а можливо, і всі) параметри можуть бути опущені.

Наприклад:

```
>>> a = [1, 3, 8, 7]
>>> a[:]
[1, 3, 8, 7]
>>> a[1:]
[3, 8, 7]
>>> a[:3]
[1, 3, 8]
>>> a[::2]
[1, 8]
```

Усі ці параметри можуть бути негативними.

Наприклад:

```
>>> a = [1, 3, 8, 7]
>>> a[::-1]
[7, 8, 3, 1]
>>> a[:-2]
[1, 3]
>>> a[-2::-1]
[8, 3, 1]
>>> a[1:4:-1]
[]
```

В останньому прикладі вийшов порожній список, так як START <STOP, а STEP негативний. Те ж саме відбудеться, якщо діапазон значень виявиться за межами об'єкта:

Наприклад:

```
>>> a = [1, 3, 8, 7]
>>> a[10:20]
[]
```

Також за допомогою зрізів можна не тільки отримувати елементи, але і додавати і видаляти елементи (зрозуміло, тільки для змінних послідовностей).

Наприклад:

```
>>> a = [1, 3, 8, 7]
>>> a[1:3] = [0, 0, 0]
>>> a
[1, 0, 0, 0, 7]
>>> del a[:-3]
>>> a
[0, 0, 7]
```

Кортеж

Кортеж - не змінний!! Інформація захищена від змін. Має менший розмір. Використовують як словник.

Наприклад:

```
>>> a = (1, 2, 3, 4, 5, 6)
>>> b = [1, 2, 3, 4, 5, 6]
>>> a.__sizeof__()
36
>>> b.__sizeof__()
44
```

Наприклад:

Зробимо пустий кортеж.

```
>>> a = tuple() # За допомогою вбудованої функції tuple()
>>> a
()
>>> a = () # За допомогою літерала кортежу
>>> a
()
>>>
```

Наприклад:

Зробимо кортеж з одного елементу:

```
>>> a = ('s', )
>>> a
('s',)
```

Або

```
>>> a = 's',
>>> a
('s',)
```

Створити кортеж з ітеруемого об'єкта можна за допомогою **функції tuple ()**

Наприклад:

```
>>> a = tuple('hello, world!')
>>> a
('h', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '!')
```

Операції з кортежами це операції над списками, що не змінюють список (додавання, множення на число, методи index () і count () і деякі інші операції). Можна також по-різному змінювати елементи місцями і так далі.

Практичне завдання

1. Згенеруйте 10+№ (де №-остання цифра студента у списку групи) випадкових чисел у діапазоні (30, 90).
2. Заповніть список квадратами чисел від 0 до 9, використовуючи генератор списку.
3. Заповніть список з 10+№ (де №-остання цифра студента у списку групи) числами, де кожне наступне число більше на 2.
4. Створіть будь-який список.

а.) Розширте список , додавши до нього усі елементи списку L[3, 6, 7].

б.) Вставте на другий елемент значення 33333.

в.) Розташуйте список в зворотньому порядку.

г.) У кінець списку додайте 3.

д.) Видаліть перший елемент списку який має значення 3.

е.) Розташуйте список у порядку збільшення.

ж.) Очистить список.

5. Створіть будь-який список з 10 цілих чисел.

а.) Виведіть другий та шостий та четвертий елемент списку.

б.) Зрізати по одному символу з начала та кінцю списку.

6. З масиву **X** довжиною 10+№ (де №-остання цифра студента у списку групи), серед елементів якого є позитивні та негативні та нуль, сформуйте

новий масив **Y**, узявши тільки елементи з **X** які більше по модулю заданого числа **M**. Виведіть на екран число **M** та масиви заданий та отриманий.

7. У масиві цілих чисел, довжиною 10+№ (де №-остання цифра студента у списку групи), усі негативні елементи замініть позитивними. Виведіть заданий та отриманий масиви на екран.
8. Дано одномірний масив, який має 10+№ цілих елементів. Введіть масив з клавіатури. Знайдіть максимальний елемент. Виведіть на екран масив у зворотньому порядку.
9. Нехай журнал по предмету «Інформаційні технології» представлено у вигляді списку: my_len = [['ІТ-31’, ['Акулова Алена', 'Бабушкіна Ксенія ', ...]], [' ІТ-32 ', [... ..]], [' ІТ-33 ', [....]]]. Виведіть список студентів конкретної групи построчно у вигляді:

<Назва групи>

<ПІБ>

<ПІБ>

10. У випадково сгенерованому списку від 1 до 50, який створен з 20 чисел, змініть усі числа, більше шостого числа, на середне арифметичне усіх чисел списку.
11. У випадково сгенерованому списку від -10 до 10, який створен з 20 чисел, створити новий список у котрому спершу йдуть від’ємні числа, потім нулі, потім позитивні числа.

Практична робота №5

Тема. Словники та робота з ними. Методи словників. Множина (set и frozenset). Функції та їх аргументи.

Мета роботи. Отримати навички роботи з словниками, множинами, функціями

Зміст.

1. Вивчення відомостей про роботу з словниками, множинами, функціями.

2. Виконання роботи.

3. Отримання результату.

Ключові положення.

Словники в Python - невпорядковані колекції довільних об'єктів з доступом по ключу. Їх іноді ще називають асоціативними масивами або хеш-таблицями. Щоб працювати зі словником, його потрібно створити. Створити його можна кількома способами.

1. За допомогою литерала:

Наприклад:

```
>>> d = {}
>>> d
{}
>>> d = {'dict': 1, 'dictionary': 2}
>>> d
{'dict': 1, 'dictionary': 2}
```

2. За допомогою функції **dict**:

Наприклад:

```
>>> d = dict(short='dict', long='dictionary')
>>> d
{'short': 'dict', 'long': 'dictionary'}
>>> d = dict([(1, 1), (2, 4)])
>>> d
{1: 1, 2: 4}
```

3.За допомогою методу fromkeys:

Наприклад:

```
>>> d = dict.fromkeys(['a', 'b'])
>>> d
{'a': None, 'b': None}
>>> d = dict.fromkeys(['a', 'b'], 100)
>>> d
{'a': 100, 'b': 100}
```

4.За допомогою генераторів словників, які дуже схожі на генератори списків.

Наприклад:

```
>>> d = {a: a ** 2 for a in range(7)}
>>> d
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

Методи словників

dict.clear () - очищає словник.

dict.copy () - повертає копію словника.

classmethod dict.fromkeys (seq [, value]) - створює словник з ключами з seq і значенням value (за замовчуванням None).

dict.get (key [, default]) - повертає значення ключа, але якщо його немає, не кидає виняток, а повертає default (за замовчуванням None).

dict.items () - повертає пари (ключ, значення).

dict.keys () - повертає ключі в словнику.

dict.pop (key [, default]) - видаляє ключ і повертає значення. Якщо ключа немає, повертає default (за замовчуванням кидає виняток).

dict.popitem () - видаляє і повертає пару (ключ, значення). Якщо словник порожній, кидає виняток KeyError. Пам'ятайте, що словники невпорядковані.

dict.setdefault (key [, default]) - повертає значення ключа, але якщо його немає, не кидає виняток, а створює ключ з значенням default (за замовчуванням None).
dict.update ([other]) - оновлює словник, додаючи пари (ключ, значення) з other. Існуючі ключі перезаписуються. Повертає None (не новий словник!).
dict.values () - повертає значення у словнику.

Що ж можна ще робити зі словниками?

Так то ж саме, що і з іншими об'єктами : вбудовані функції, ключові слова (наприклад, цикли for і while), а також спеціальні методи словників.

Що таке множина?

Множина в python - "контейнер", що містить елементи у випадковому порядку які не повторюються.

Створюємо множину:

Наприклад:

```
>>> a = set('hello')
>>> a
{'h', 'o', 'l', 'e'}
>>> a = {'a', 'b', 'c', 'd'}
>>> a
{'b', 'c', 'a', 'd'}
```

Множину використовують для видалення повторюваних елементів.

Наприклад:

```
>>> words = ['hello', 'daddy', 'hello', 'mum']
>>> set(words)
{'hello', 'daddy', 'mum'}
```

З множинами можна виконувати безліч операцій: знаходити об'єднання, перетин ...

• len (s) - число елементів у множині (розмір множини).

• x in s - чи належить x множині s.

• set.isdisjoint (other) - істина, якщо set і other не мають спільних елементів.

• set == other - все елементи set належать other, все елементи other належать set.

• set.issubset (other) або set <= other - все елементи set належать other.

• set.issuperset (other) або set> = other - аналогічно.

• set.union (other, ...) або set | other | ... - об'єднання декількох множин.

• set.intersection (other, ...) або set & other & ... - перетин.

• set.difference (other, ...) або set - other - ... - безліч з усіх елементів set, які не належать жодному з other.

• set.symmetric_difference (other); set ^ other - безліч з елементів, що зустрічаються в одному безлічі, але не зустрічаються в обох.

• set.copy () - копія безлічі.

Наприклад:

```
>>> a = set('forex')
>>> a
{'x', 'f', 'e', 'o', 'r'}
>>> d = set('name')
>>> d
{'n', 'e', 'a', 'm'}
>>> a|d
{'n', 'x', 'a', 'm', 'f', 'e', 'o', 'r'}
>>> a&d
{'e'}
>>>
```

І операції, безпосередньо змінюють множини:

• set.update (other, ...); set | = other | ... - об'єднання.

• set.intersection_update (other, ...); set & = other & ... - перетин.

• set.difference_update (other, ...); set - = other | ... - віднімання.

• set.symmetric_difference_update (other); set ^ = other - безліч з елементів, що зустрічаються в одному безлічі, але не зустрічаються в обох.

• set.add (elem) - додає елемент в множину.

• set.remove (elem) - видаляє елемент з безлічі. KeyError, якщо такого елемента не існує.

• set.discard (elem) - видаляє елемент, якщо він знаходиться в множині.

• set.pop () - видаляє перший елемент з множини. Так як множина не впорядковані, не можна точно сказати, який елемент буде першим.

• set.clear () - очищення множини.

Наприклад:

```
>>> a = set('forex')
>>> a
{'x', 'f', 'e', 'o', 'r'}
>>> a.add('y')
>>> a
{'x', 'f', 'y', 'e', 'o', 'r'}
>>> a.remove('e')
>>> a
{'x', 'f', 'y', 'o', 'r'}
>>>
```

Єдина відмінність **set** від **frozenset** полягає в тому, що set - змінюваний тип даних, а frozenset - ні.

Наприклад:

```
>>> a = set('qwerty')
>>> b = frozenset('qwerty')
>>> a == b
True
>>> True
True
>>> type(a - b)
<class 'set'>
>>> type(a | b)
<class 'set'>
>>> a.add(1)
>>> b.add(1)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
AttributeError: 'frozenset' object has no attribute 'add'
```

Підпрограма - це іменований фрагмент програми, до якого можна звернутися з іншого місця програми. Підпрограми діляться на дві категорії: процедури і функції.

Процедури. Синтаксис процедури

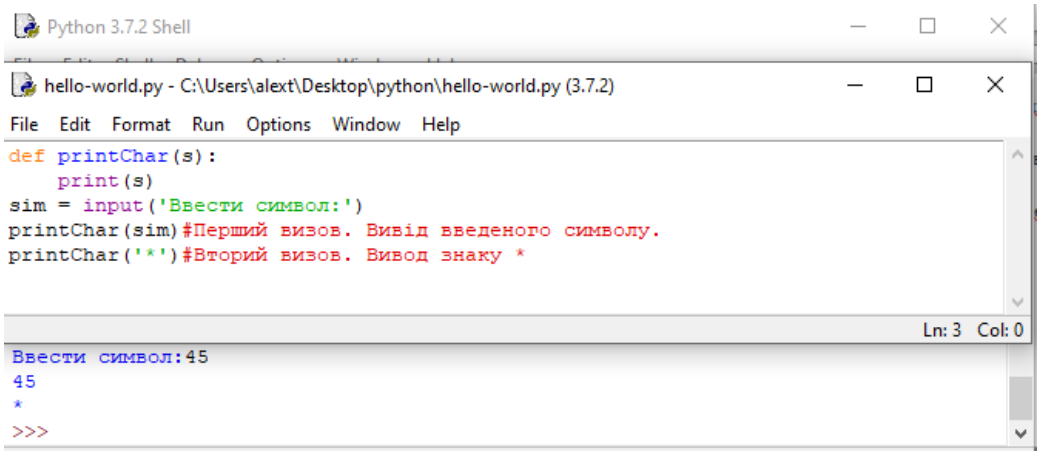
Для визначення процедури використовується ключове слово def, потім вказується ім'я процедури і в дужках її формальні параметри, якщо вони присутні. Після ставиться двокрапка і з наступного рядка з відступом в 4 пробілу вказуються команди. Процедура - допоміжний алгоритм, який виконує деякі дії.

Процедура повинна бути визначена до моменту її виклику. Визначення процедури починається зі службового слова def.

Виклик процедури здійснюється за її імені, за яким слідують круглі дужки, наприклад, Err ().

В одній програмі може бути скільки завгодно багато викликів однієї і тієї ж процедури. Використання процедур скорочує код і підвищує читабельність.

Написати процедуру, яка друкує раз вказаний символ (введений з клавіатури), кожен з нового рядка. Тут важливі відступи.

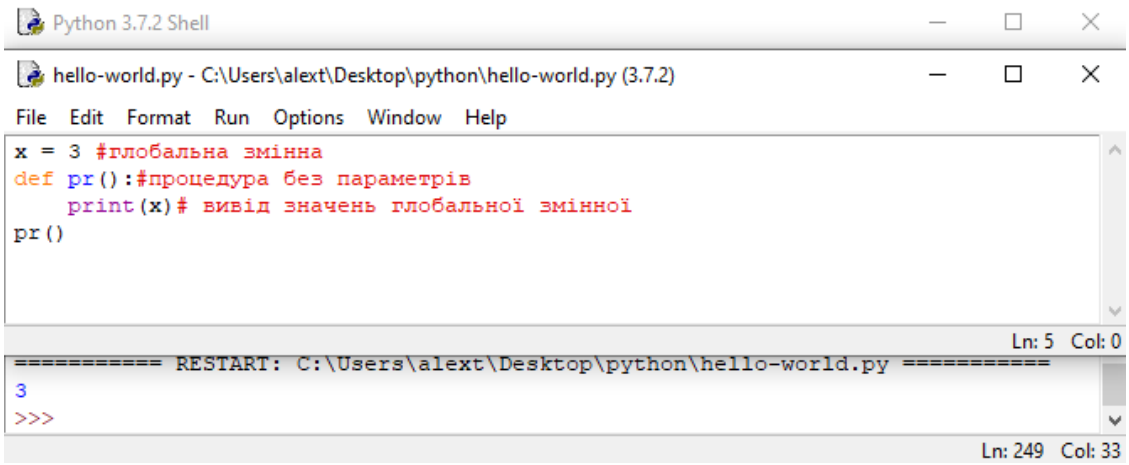


Глобальна змінна - якщо їй присвоєно значення в основній програмі (поза процедурою).

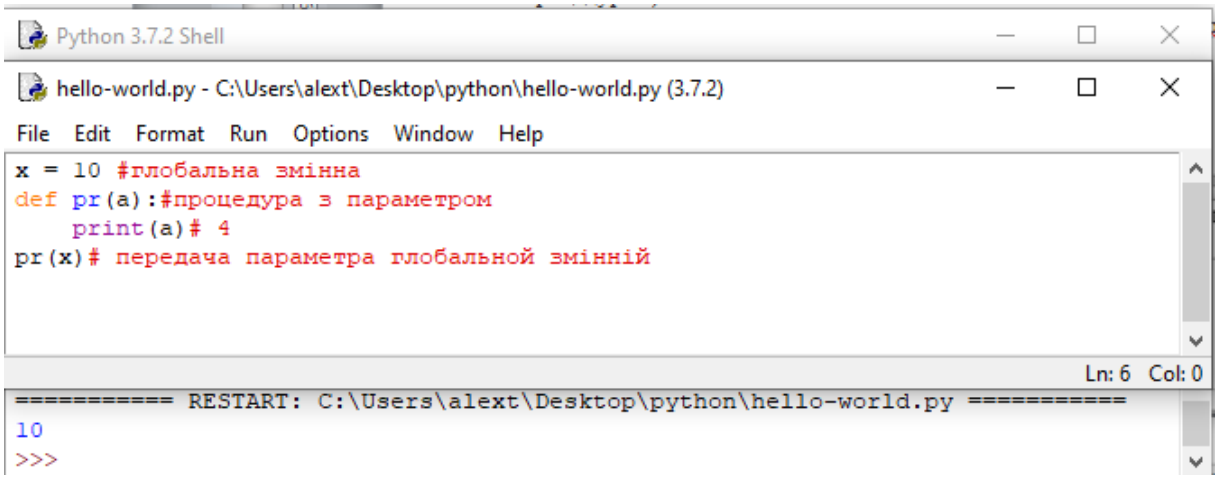
Локальна змінна (внутрішня) відома тільки на рівні процедури, звернутися до неї з основної програми і з інших процедур не можна.

Параметри процедури - локальні змінні.

Наприклад:

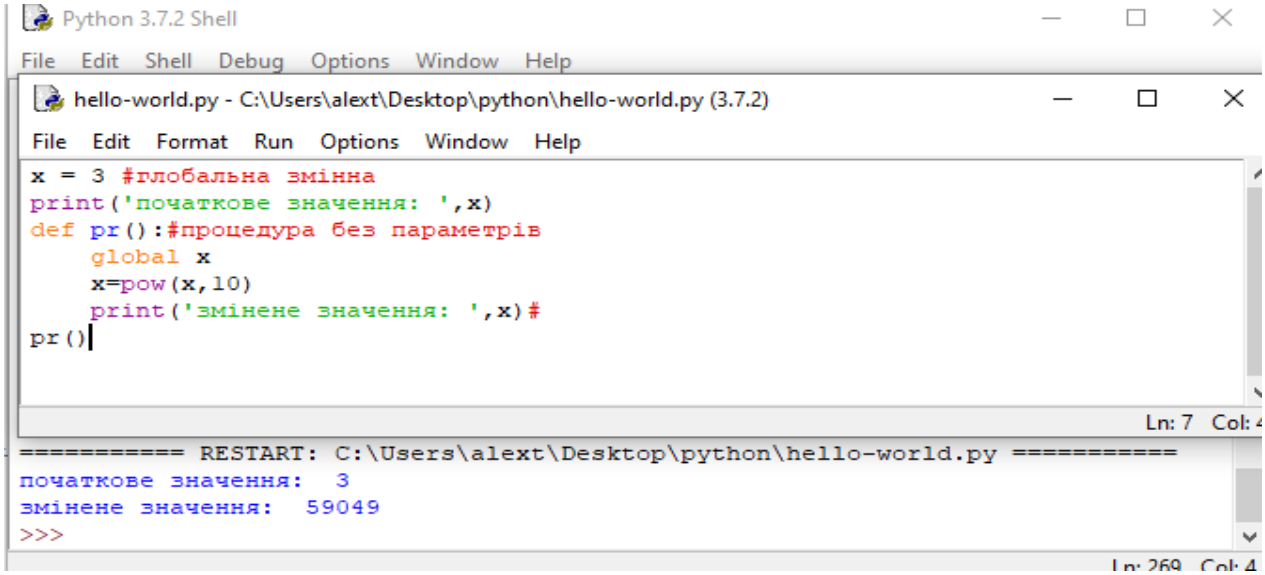


Наприклад:



Існує можливість змінити значення глобальної змінної (не створюючи локальну). У процедурі за допомогою слова **global:**

Наприклад:



Функції та їх аргументи

Функція в python - об'єкт, який приймає аргументи і повертає значення. Зазвичай функція визначається за допомогою інструкції def.

Визначимо найпростішу функцію:

```
def  add (x, y):
    return x + y
```

Інструкція return каже, що потрібно повернути значення. У нашому випадку функція повертає суму x та y.

Наприклад:

```
>>> add(1, 10)
11
>>> add('abc', 'def')
'abcdef'
```

Функція може бути будь-якої складності і повертати будь-які об'єкти (списки, кортежі, і навіть функції!):

Практичне завдання

1. Створити будь яку множину .

2. З v = ['ccc', 'ddd', 'yyy', 'iii', ‘ccc’, ‘dd’] видалити елементи які повторюються.

3. Зробить копія множини v = ['ccc', 'ddd', 'yyy', 'iii', ‘ccc’, ‘dd’]
4. Розрахувати кількість елементів у множині v = ['ccc', 'ddd', 'yyy', 'iii', ‘ccc’, ‘dd’].
5. Створіть дві множини . Об’єднайте їх. Знайдіть їх перетин.
6. Додайте будь який елемент у множину.
7. Виділіть будь який елемент із множини.
8. Видаліть перший елемент із множини.
9. Очистіть усі множини.
10. Створіть процедуру змінення значення глобальної змінної, не створюючи локальної, за допомогою слова **global**.

x= 5- глобальна змінна. Треба возвести її у четверту ступень. Ввести змінене значення глобальної змінної.

11. Створіть функцію, яка розраховує суму цифр числа, яке вводиться з екрану.
12. Визначте, чи є три трикутника рівновеликими. Довжини сторін вводити з клавіатури. Для підрахунку площі трикутника використовувати формулу Герона. Обчислення площі оформити у вигляді функції з трьома параметрами.

Формула Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$
$$\text{где } p = \frac{a+b+c}{2}$$

Введіть одновимірний масив A довжиною m. Поміняйте в ньому місцями перший і останній елементи. Довжину масиву і його елементи введіть з клавіатури. У програмі опишіть процедуру для заміни елементів масиву. Виведіть вихідні та отримані масиви.

13. Дано 3 різних масиву цілих чисел (розмір кожного не перевищує 15). У кожному масиві знайдіть суму елементів і середньо арифметичне значення.
14. Знайдіть кількість елементів сгенерованого масиву, які відмінні від найбільшого елемента не більше ніж на 10%.

Програмування з використанням функцій

1. Дані дійсні числа a0, ..., a6. Отримати для x = 1,3,4 значення p (x + 1) - p (x), де p (y) = a6y6 + a5y5 + ... + a 0.
2. Дано. Дійсні числа s, t, a 0, ... A12. Отримати p (1) -p (t) + p2 (s-t) -pз (t), де p (x) = a12x12 + a11x11 + ... + a 0.
3. ОбчислитиZ = (X1 + Y1) / (X2 - Y2), де X1 і X2 - коріння рівняннях $2x^2$ + 2x + 1 = 0;Y1 і Y2 - коріння рівнянняу$2x^2$ -4y +4 = 0. (Все коріння дійсні).
4. Обчислити Z = (e^{s1} + e^{s2}) / (k1 * k2), де S1 і К1 - сума і кількість позитивних елементів масиву X (10); S2, і К2 - сума і кількість негативних елементів масиву Y (15).
5. Перетворити масиви X (10) і Y (15), розташувавши в них поспіль тільки позитивні елементи.Замість інших елементів записати нулі.
6. Обчислити Z = (Xmax-Ymin) / 2, де Xmax - максимальний елемент масиву X (10): Ymin - мінімальний елемент масиву Y (12): Xmax і Ymin обчислювати в одній підпрограмі.
7. Дано дійсні числа a1, ..., an, b1, ..., bm. У послідовності a1, ..., an і послідовності b1, ..., bm всі члени, які йдуть за членом з найбільшим значенням (за першим по порядку, якщо їх декілька), замінити на 0.5.
8. Дано цілі числа a1, ..., an, b1, ..., bm, К. Якщо в послідовності a1, ..., an є член зі значенням К, тоді усі наступні замінити значенням К. Якщо немає жодного члена зі значенням К, то перший по порядку найбільший член цієї послідовності замінити на значення К та наступні також. За таким же правилом перетворити b1, ..., bm, стосовно до значення 10.

Практична робота №6

Тема. Робота з двовимірними масивами.

Мета роботи. Освоїти роботу з двовимірними масивами.

Зміст.

1. Вивчення відомостей про роботу з двовимірними масивами.

2. Виконання роботи.

3. Отримання результату.

Ключові положення.

Матрицями називаються масиви елементів, які представлені у вигляді прямокутних таблиць. Для матриць визначені правила математичних дій. Елементами матриці можуть бути числа, алгебраїчні символи або математичні функції.

Для роботи з матрицями в Python також використовуються списки. Кожен елемент списку-матриці містить вкладений список. Таким чином, виходить структура з вкладених списків. Кількість вкладених списків визначає кількість стовпців матриці, а число елементів всередині кожного вкладеного списку вказує на кількість рядків у вихідній матриці.

Використаємо введення списку (масиву) в мові Python.

Для введення елементів списку використовується цикл for і команда range ():

```
for i in range (N):
    x [i] = int (input ())
```

Простіший варіант введення списку:

```
x = [int (input ()) for i in range (N)]
```

Функція int тут використовується для того, щоб рядок, введений користувачем, перетворювався у цілі числа.

Вивід цілого списку (масиву):

```
print (L)
```

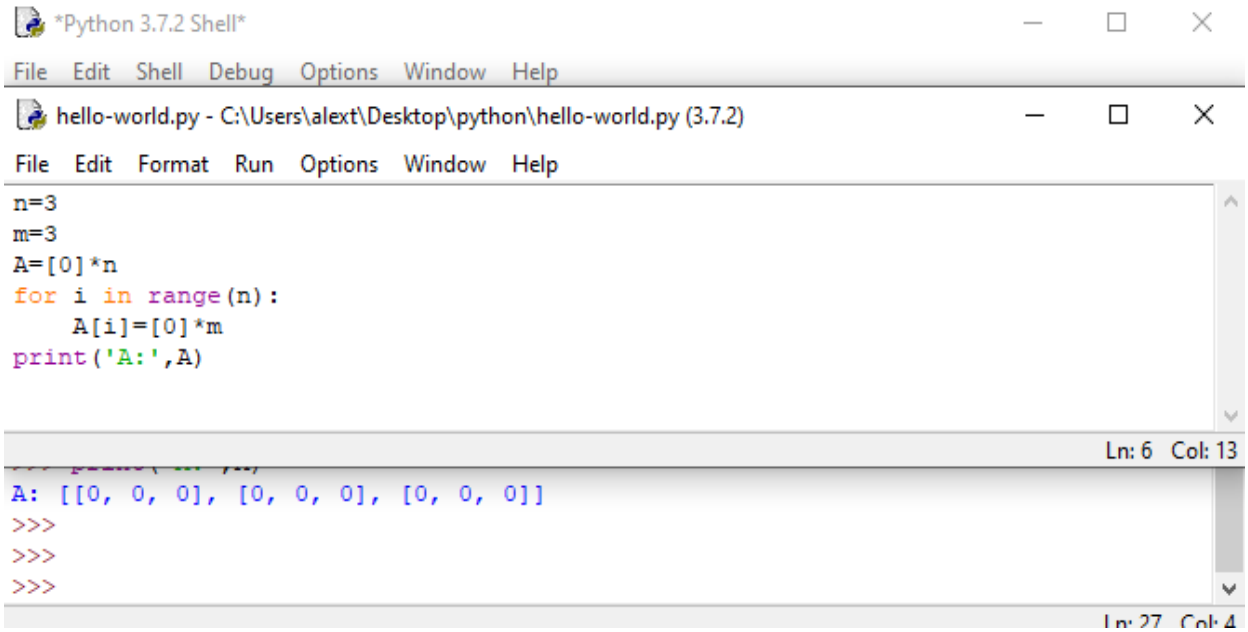
Поелементний вивід списку (масиву):

```
for i in range (N):
```

print (L [i], end = "")

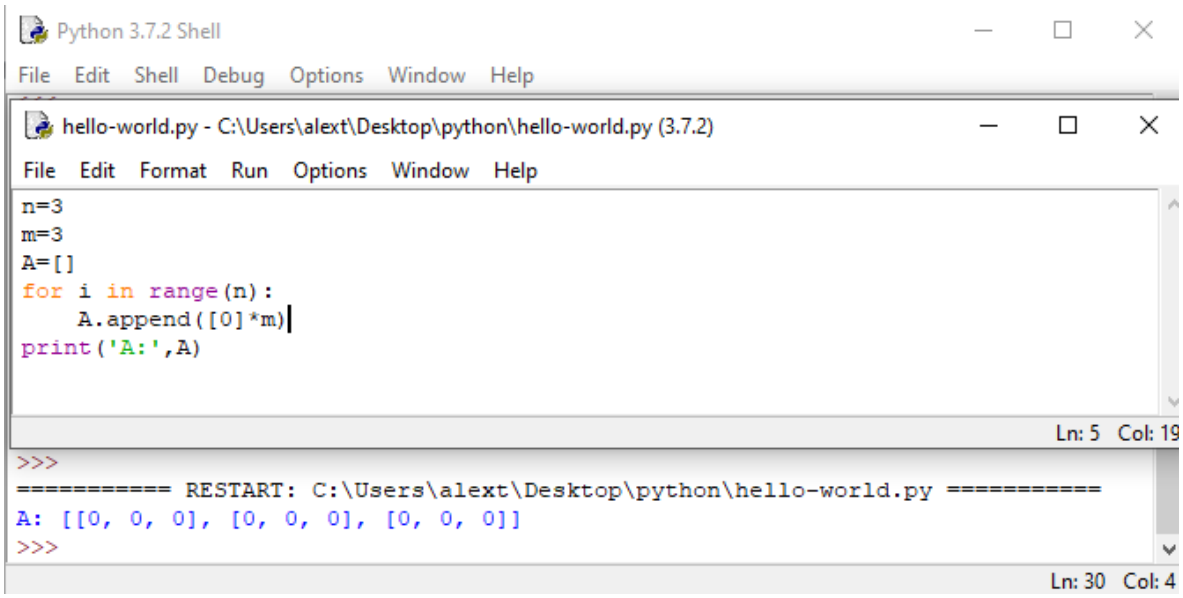
Створення списку. Перший спосіб.

Нехай дано два числа: кількість рядків матриці - **n** і кількість стовпців матриці -**m**.

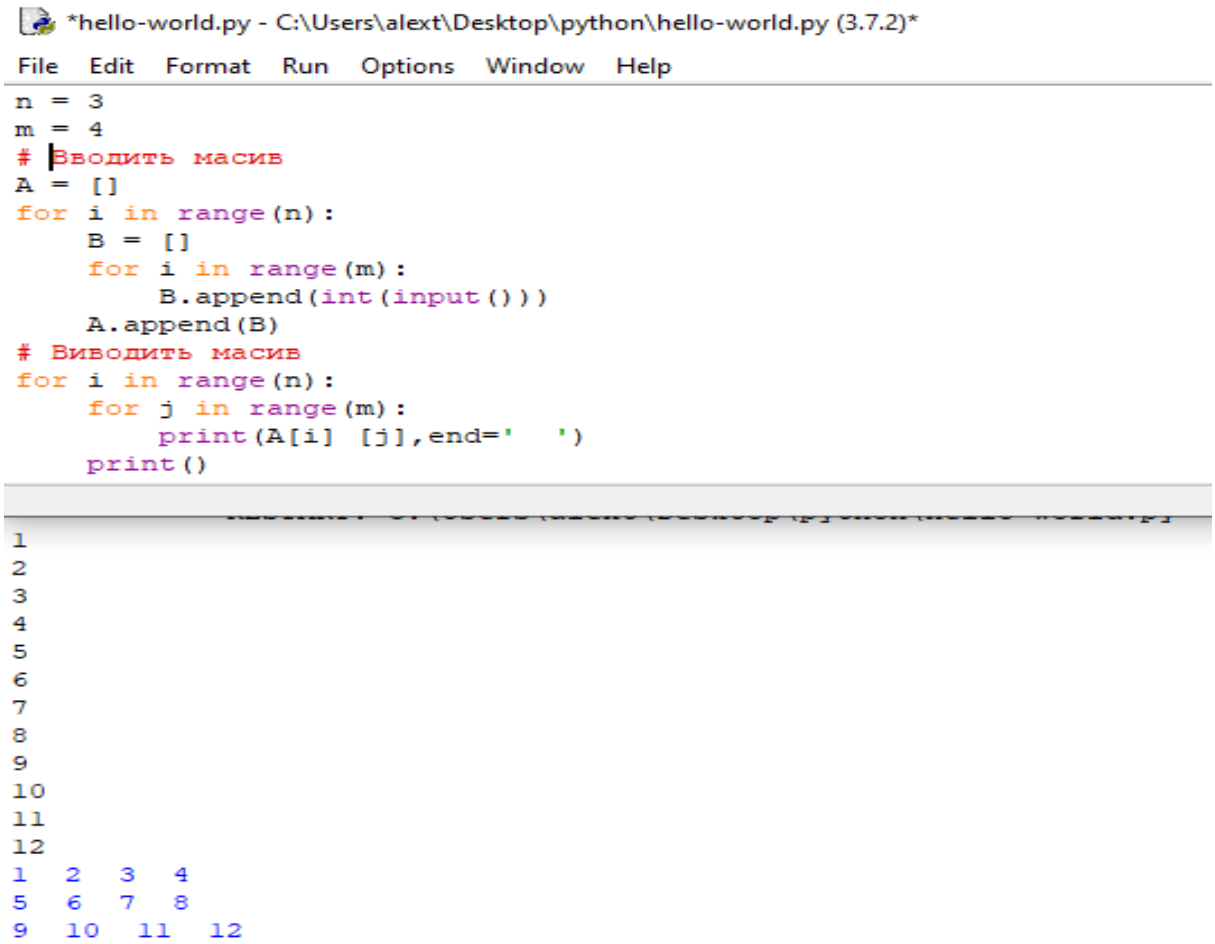


Другий спосіб створення матриці.

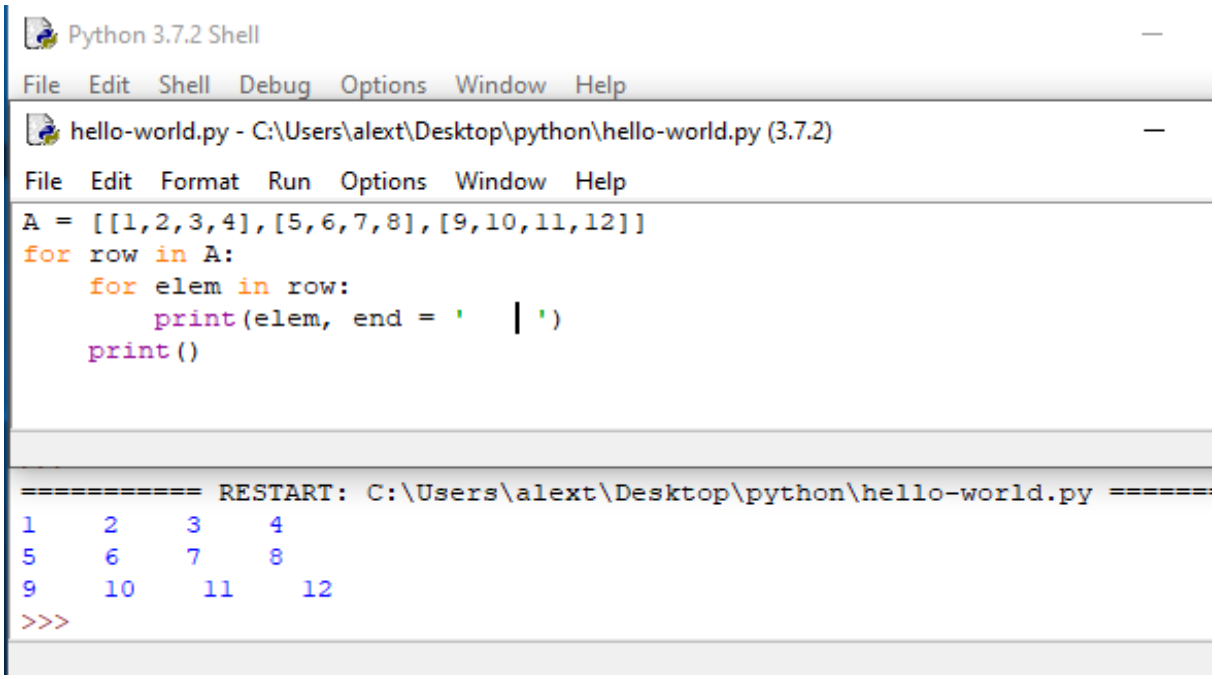
Створити порожній список, потім n раз додати в нього новий елемент, який є списком-рядком за допомогою **list.append**(x) який додає елемент у кінець списку.



Для обробки і виведення списку як правило використовується два вкладених циклу. Перший цикл по номеру рядка, другий цикл по елементах всередині рядка. Наприклад, вивести двовимірний числовий список на екран порядково, розділяючи числа пробілами всередині одного рядка, можна так:



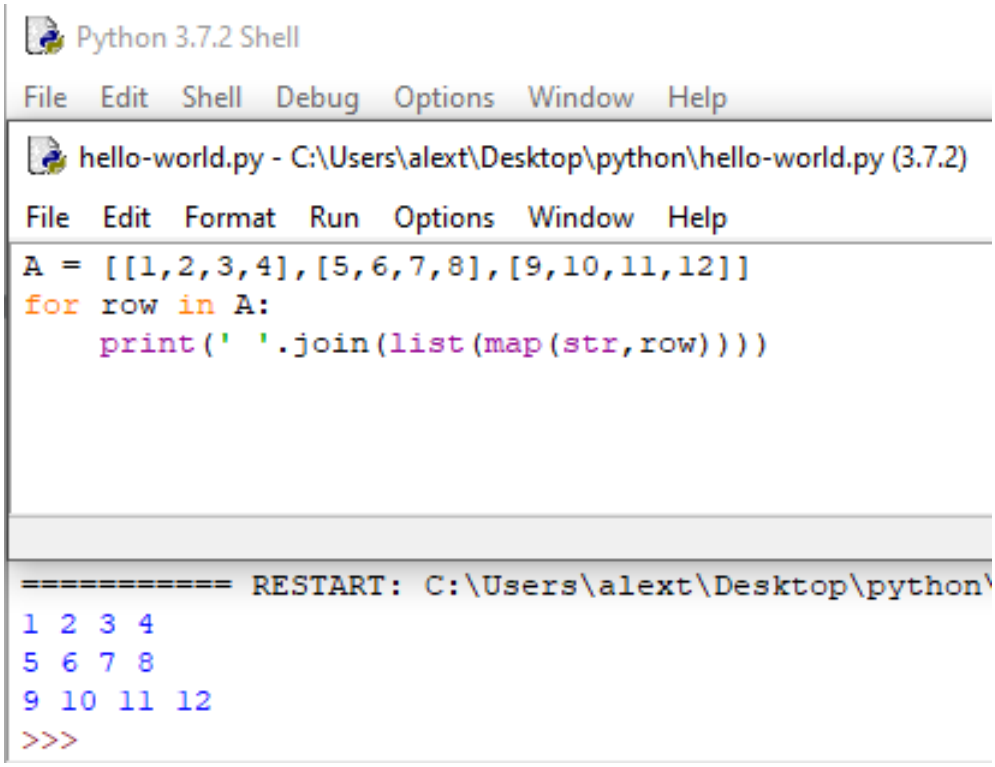
Те ж саме, але цикли не по індексу, а за значеннями списку:



Для виведення матриці можна скористатися методом join.

for row in A:

print (' ' .join (list (map (str, row))))



Обробка та вивід вкладених списків

Часто в задачах доводиться зберігати прямокутні таблиці з даними. Такі таблиці називаються матрицями або двовимірними масивами. У мові програмування Пітон таблицю можна представити у вигляді списку рядків, кожен елемент якого є в свою чергу списком, наприклад, чисел. Наприклад, створити числову таблицю з двох рядків і трьох стовпців можна так:

A = [[1, 2, 3], [4, 5, 6]]

Тут перша строчка списку A[0] є списком [1, 2, 3].

A[0][0]= 1,

A[0][1]= 2,

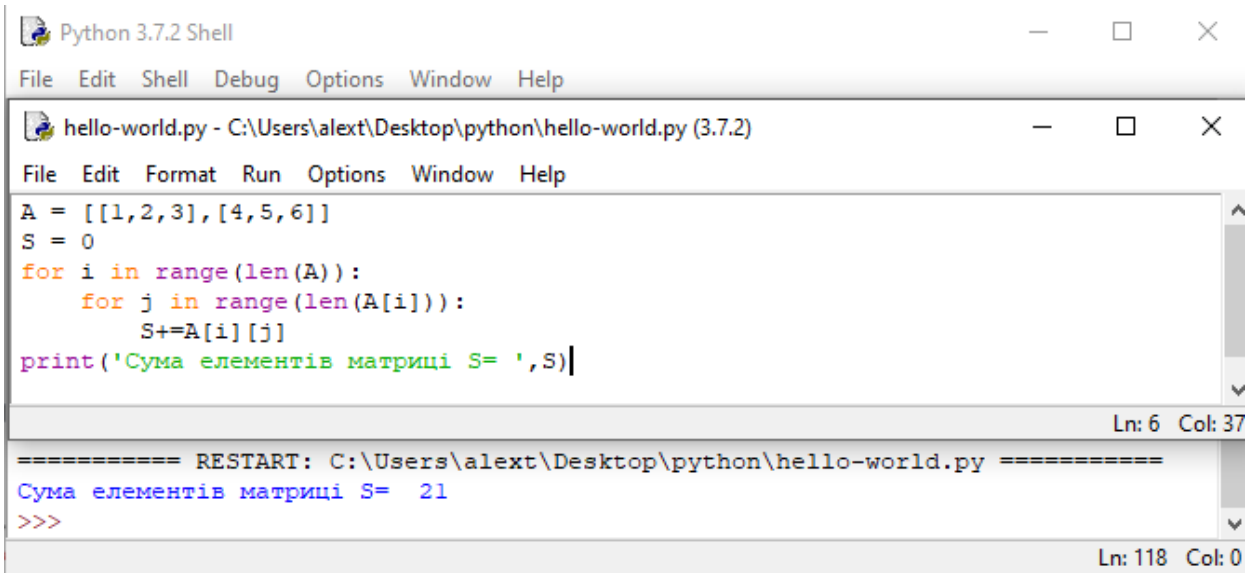
A[0][2]= 3,

A[1][0]=4,

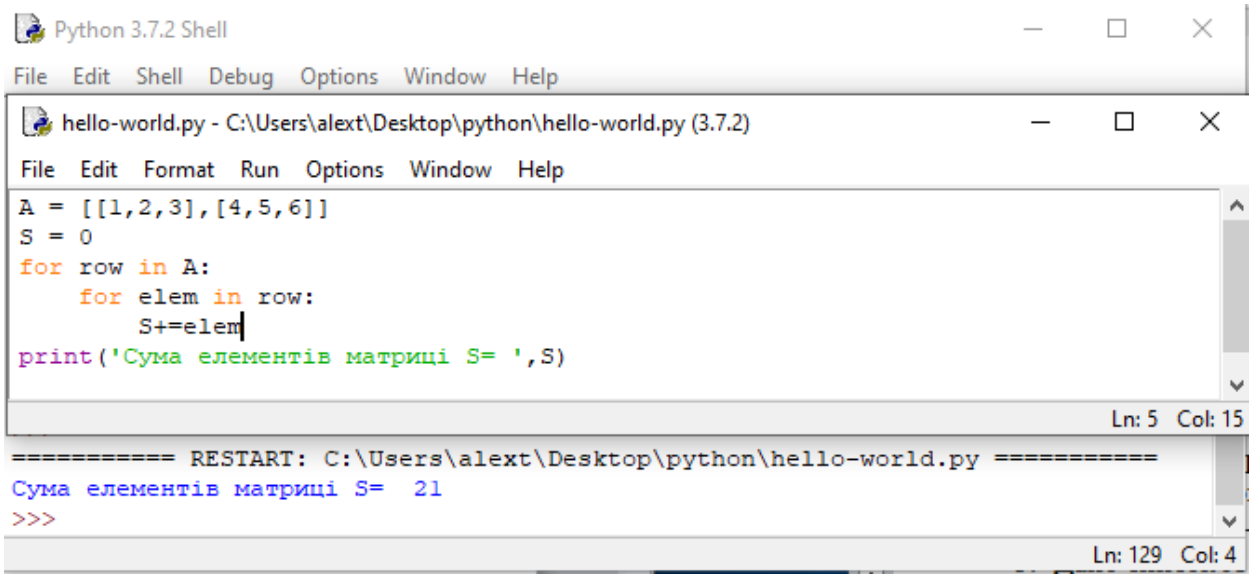
A[1][1]=5,

A[1][2]=6.

Програма використовує два вкладених цикла для розрахунку суми усіх чисел у списку по індексу.



Програма використовує два вкладених цикла для розрахунку суми усіх чисел у списку по значенням строк.



Практичне завдання

1. Створіть матрицю з одних нулів, де кількість рядків матриці - **n** і кількість стовпців матриці - **m**. Де **m** = №+2, **n = №+3,** № - остання цифра у списку групи.
2. Написати програму, яка виводить на екран запит на введення елементів матриці розміром **m** та **n**, а потім виводить матрицю на екран. Де **m** =№+2 та **n = №+3.** №-остання цифра у списку групи.
3. Дано список A = [[1,3,4,5,6,7,№],[8,9,10,11,№,29,30],[№,2,4,7,8,3,5]]. Де № - остання цифра у списку групи. Виведіть матрицю на екран. Ввести три пробілу між елементами матриці. Зробіть теж саме за допомогою метода join.
4. Дано список A = [[1,3,4,5,6,7,№],[8,9,10,11,№,29,30],[№,2,4,7,8,3,5]]. Де № - остання цифра у списку групи. Написати програму, яка розрахує суми усіх чисел у списку по індексу.
5. Дано список A = [[1,3,4,5,6,7,№],[8,9,10,11,№,29,30],[№,2,4,7,8,3,5]]. Де № - остання цифра у списку групи. Написати програму, яка розрахує суми усіх чисел у списку по значенням строк.
6. Дано список A = [[1,3,4,5,6,7,№],[8,9,10,11,№,29,30],[№,2,4,7,8,3,5]]. Де № - остання цифра у списку групи. Написати програму, яка зводить усі елементи матриці у квадрат.

7. Дано список A = [[1,3,4,5,6,7,№],[8,9,10,11,№,29,30],[№,2,4,7,8,3,5]]. Де № - остання цифра у списку групи. Написати програму, яка зводить усі елементи матриці у квадрат які менше за 5.
8. Дано список A = [[1,3,4,5,6,7,№],[8,9,10,11,№,29,30],[№,2,4,7,8,3,5]]. Де № - остання цифра у списку групи. Написати програму, яка буде находити найбільше число матриці.
9. Знайти найменші елементи і номера рядків і стовпців, в яких вони розташовані, для матриць A (4,5) і B (5,3).
10. Обчислити суми позитивних елементів кожного рядка для матриць A (5,6) і B (4,3), використовуючи підпрограму-функцію.
11. Обчислити і запам'ятати кількість негативних елементів кожного стовпця для матриць A (5,5), B (4,5).

Практична робота №7

Тема. Робота з двовимірними масивами. Методи складної обробки двовимірних масивів.

Мета роботи. Освоїти методи складної обробки двовимірних масивів.

Зміст.

1. Вивчення відомостей про роботу з двовимірними масивами.
2. Методи складної обробки масивів.
3. Виконання роботи.
4. Отримання результату.

Ключові положення.

Нехай дана квадратна матриця з n рядків і n стовпців. Необхідно елементам, що знаходяться на головній діагоналі, що проходить з лівого

верхнього кута в правий нижній (тобто тих елементів A [i] [j], для яких i == j) присвоїти значення 1, елементам, що знаходяться вище головної діагоналі - значення 0, елементів, що знаходяться нижче головної діагоналі - значення 2. тобто отримати такий масив (приклад для n == 3):

1 0 0

2 1 0

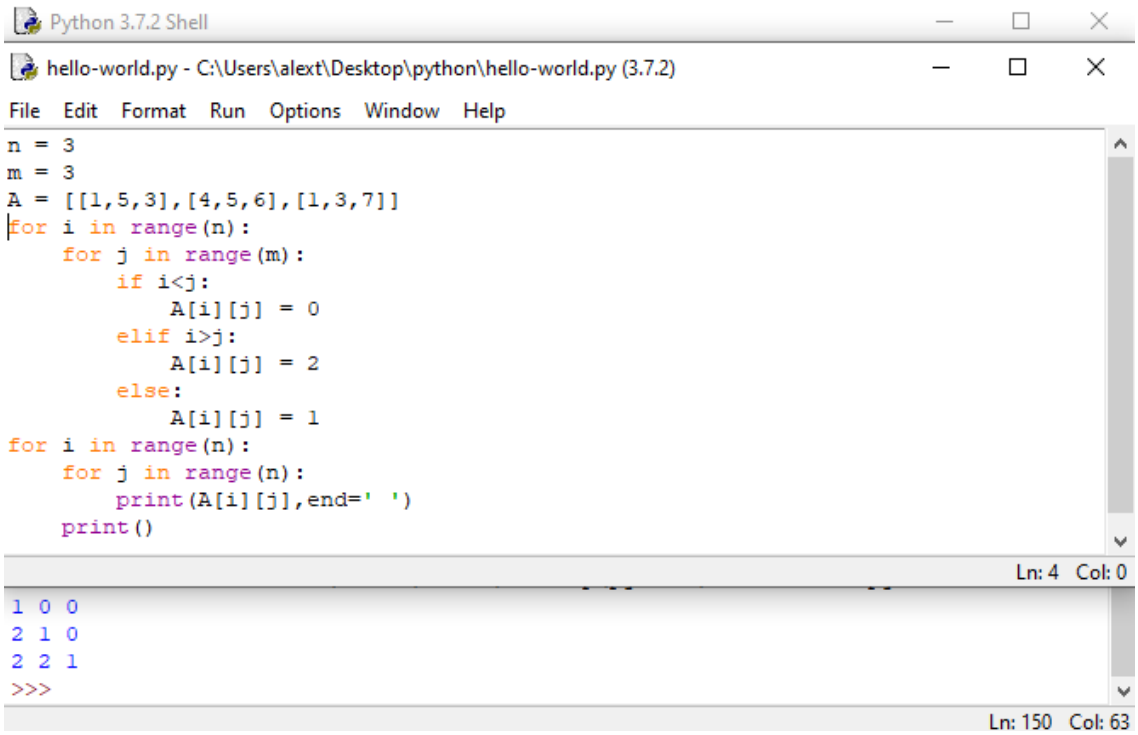
2 2 1

Перший спосіб

Елементи, які лежать вище головної діагоналі - це елементи A [i] [j], для яких i <j, а для елементів нижче головної діагоналі i> j. Таким чином, ми можемо порівнювати значення i та j . Визначати значення A [i] [j]. Отримуємо наступний алгоритм:

```
    for i in range(n):
  for j in range(n):
    if i < j:
      A[i][j] = 0
    elif i > j:
      A[i][j] = 2
    else:
      A[i][j] = 1
```

Наприклад:



Другий спосіб

Попередній алгоритм виконує одну або дві інструкції if для обробки кожного елемента. Ускладнимо алгоритм та обійдемось без умовних інструкцій.

1. Спочатку заповнимо головну діагональ, для чого нам знадобиться один цикл:

for i in range(n):

A[i][i] = 1

2. Заповнимо значенням 0 всі елементи вище головної діагоналі, для чого нам знадобиться в кожній з рядків з номером **i** привласнити значення елементів A [i] [j] для j = i + 1, ..., n-1. Тут нам знадобляться вкладені цикли:

for i in range(n):

for j in range(i + 1, n):

A[i][j] = 0

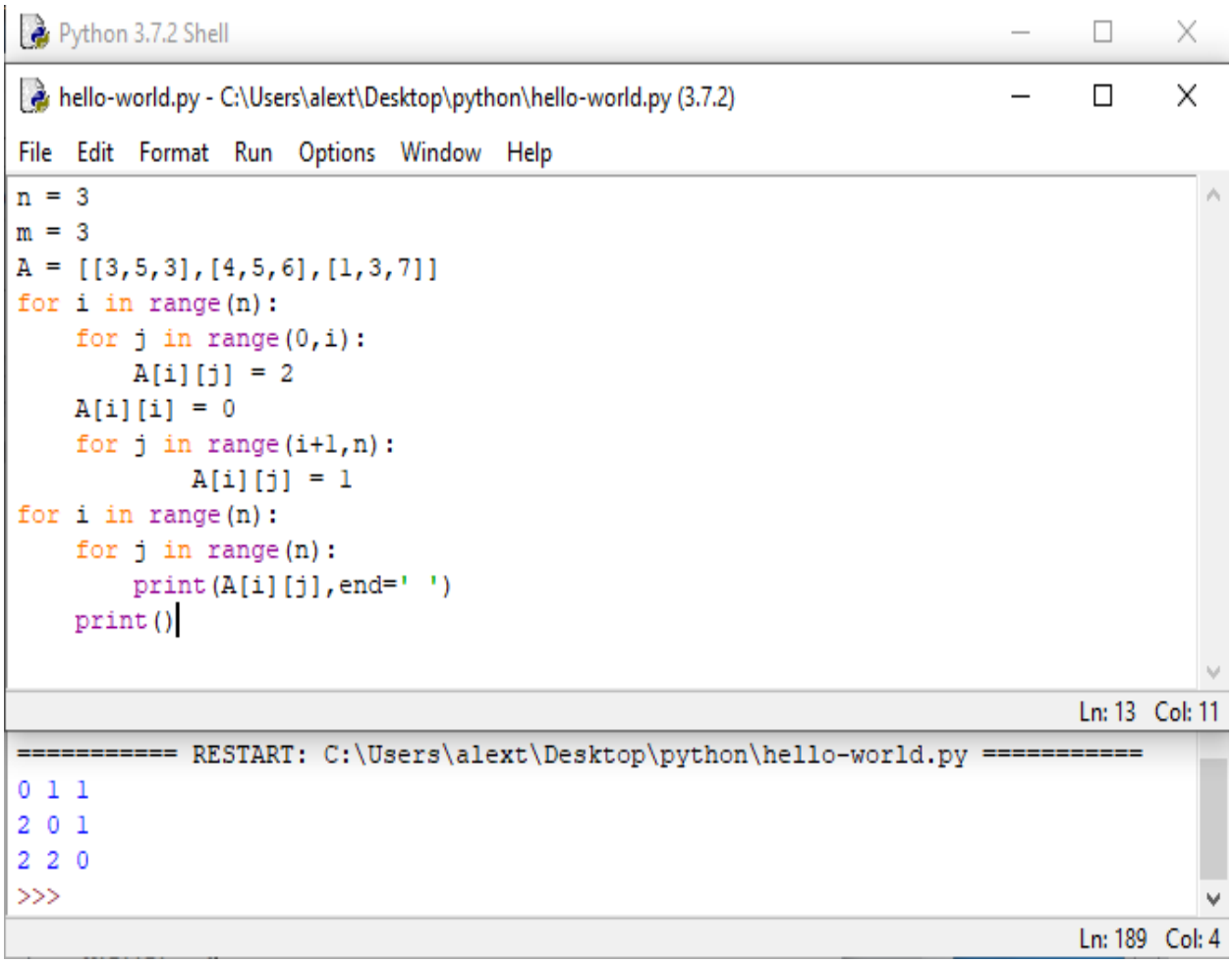
3. Аналогічно присвоюємо значення 2 елементам A [i] [j] для j = 0, ..., i-1:

```
for i in range(n):
    for j in range(0, i):
        A[i][j] = 2
```

4. Можна також зовнішні цикли об'єднати в один і отримати ще одне, більш компактне рішення:

```
for i in range(n):
    for j in range(0, i):
        A[i][j] = 2
    A[i][i] = 1
    for j in range(i + 1, n):
        A[i][j] = 0
```

Наприклад:



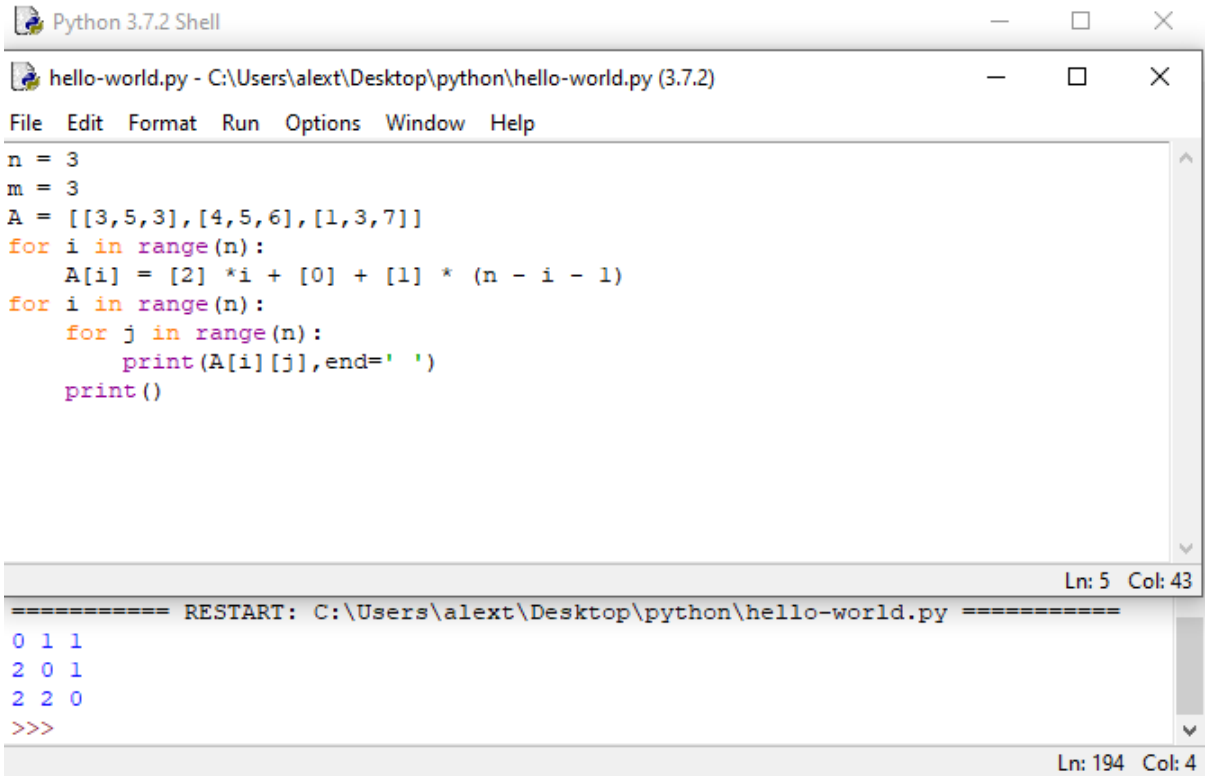
Третій спосіб

Використовується операція повторення списків для побудови чергового рядка списку. і-й рядок списку складається з **i** чисел 2, потім йде одне число 1, потім йде n-i-1 число 0:

for i in range(n):

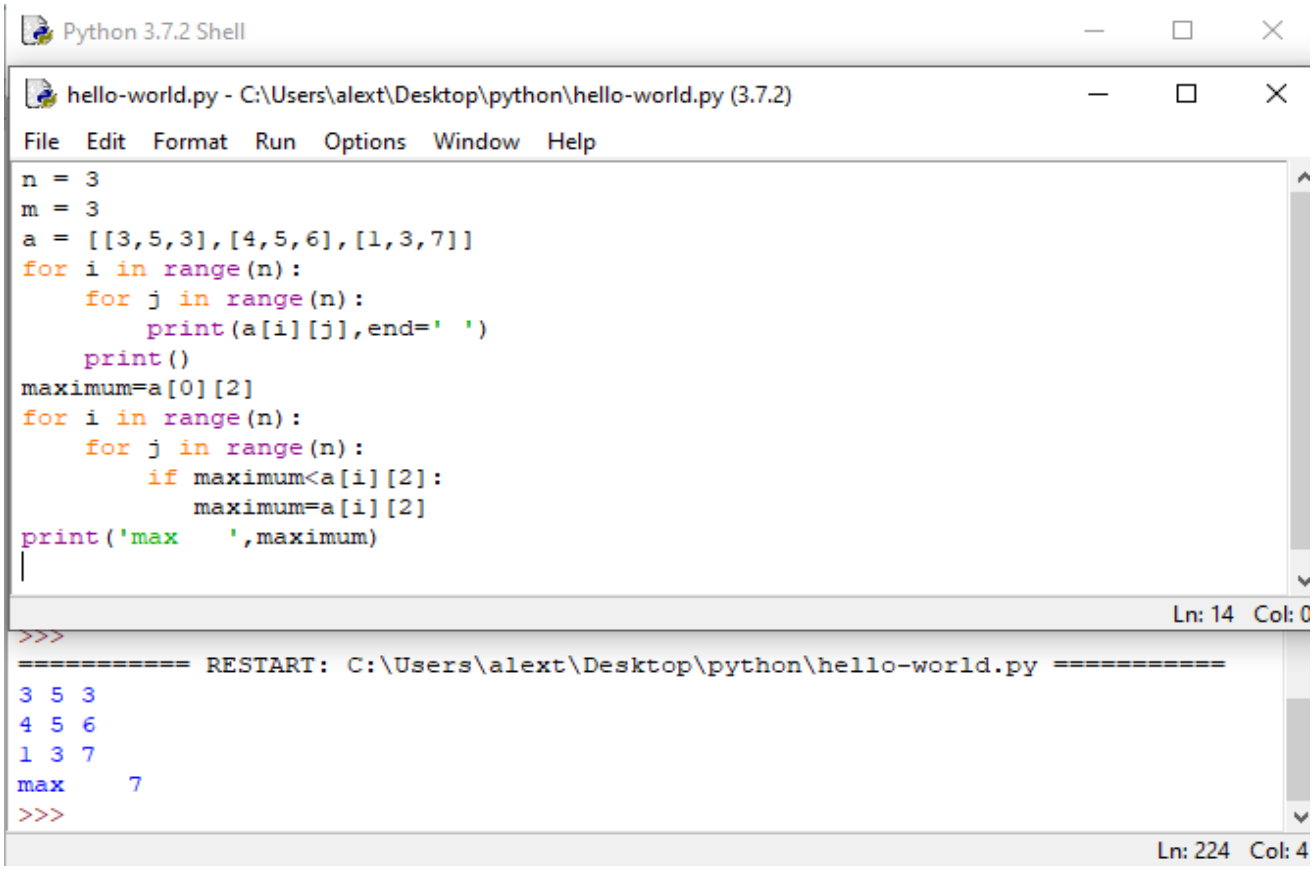
A[i] = [2] * i + [1] + [0] * (n - i - 1)

Наприклад:



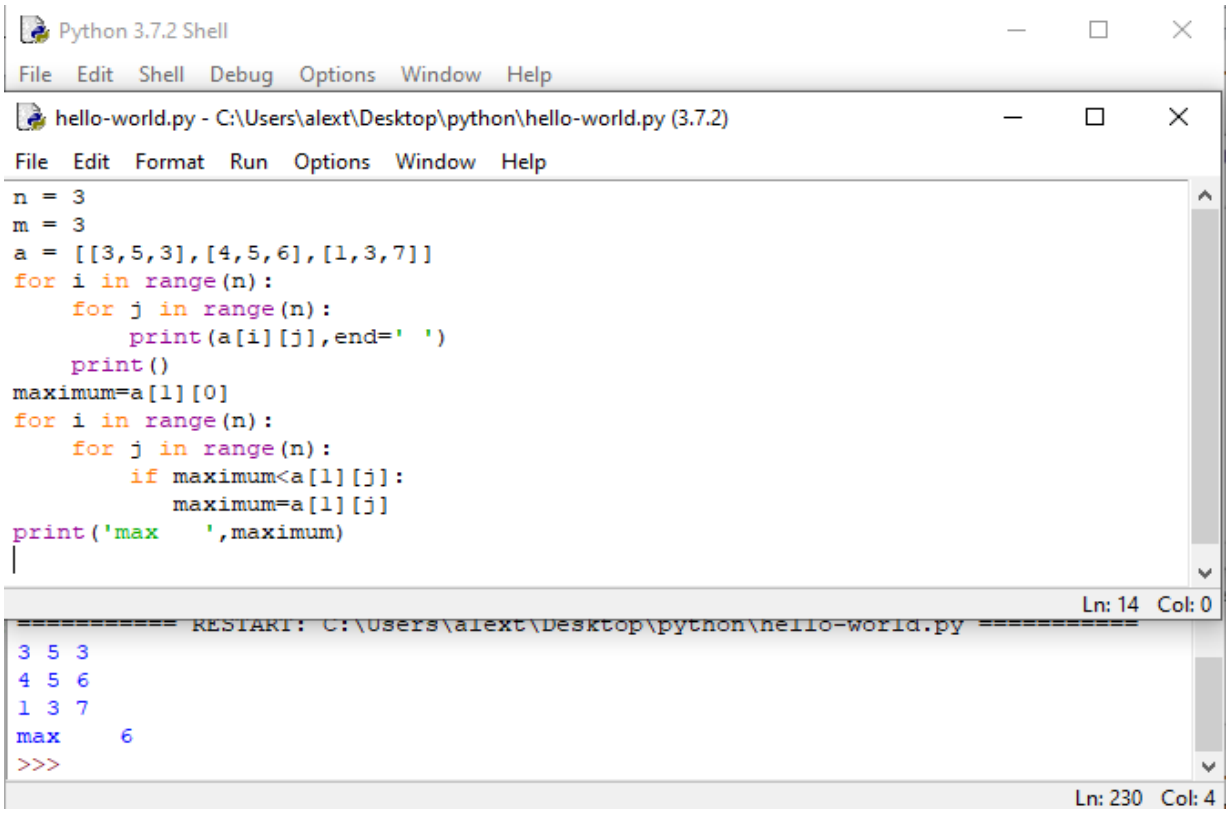
Наприклад:

Знайти максимальне значення між елементами третього стовпчика.



Наприклад:

Знайти максимальне значення між елементами другої строки.



Практичне завдання

1. Дано список A = [[1,7,№],[8,№,29],[№,2,4]]. Де № - остання цифра у списку групи. Виведіть матрицю на екран. Ввести три пробілу між елементами матриці.
2. Присвоїти всім елементам матриці A значення 9. Вивести на екран.
3. Присвоїти елементам, що знаходяться на головній діагоналі, що проходить з лівого верхнього кута в правий нижній (тобто тих елементів A [i] [j], для яких i == j) значення 3, елементам, що знаходяться вище головної діагоналі - значення 4, елементів, що знаходяться нижче головної діагоналі - значення 5. Програму написати трьома способами.
4. Дано список A = [[1,7,№],[8,№,29],[№,2,4]]. Де № - остання цифра у списку групи. Введіть список з екрану. Знайти мінімальне значення між елементами другого стовпчика. Вивести отримані значення.
5. Дано список A = [[1,7,№],[8,№,29],[№,2,4]]. Де № - остання цифра у списку групи. Введіть список з екрану. Знайти мінімальне значення між елементами третьої строки. Вивести отримані значення.
6. Дано - двовимірний масив розміром m n (будь який). Сформувати новий масив замінивши позитивні елементи одиницями, а негативні нулями. Вивести обидва масиви.

Практична робота №8

Тема. Розширені можливості функцій. Рекурсивні функції.

Мета роботи. Освоїти розширені можливості функцій. Рекурсивні функції.

Зміст.

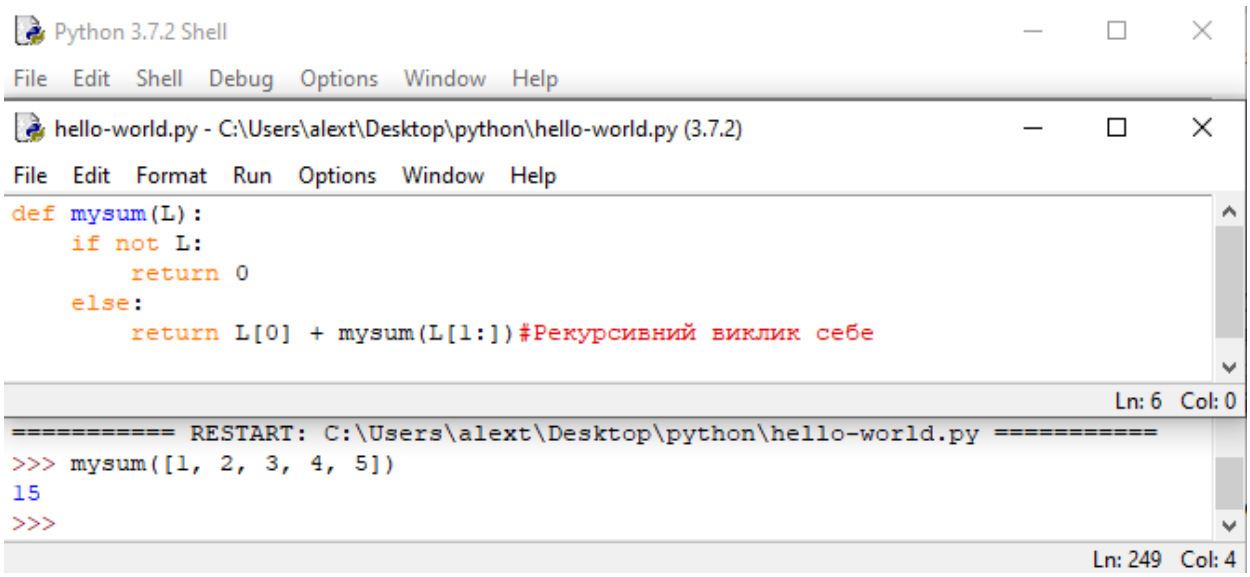
1. Вивчення відомостей про розширені можливості функцій. Рекурсивні функції.
2. Виконання роботи.
3. Отримання результату.

Ключові положення.

Рекурсія дозволяє програмами обходити структури, які мають довільні і непередбачувані форми та глибини, наприклад, при плануванні маршрутів в подорож, аналізі мови і проходження по посиланнях в веб-мережі. Рекурсія навіть є альтернативою нескладним циклам і ітераціям, хоча не обов'язково більш простою або ефективною.

Підсумовування за допомогою рекурсії

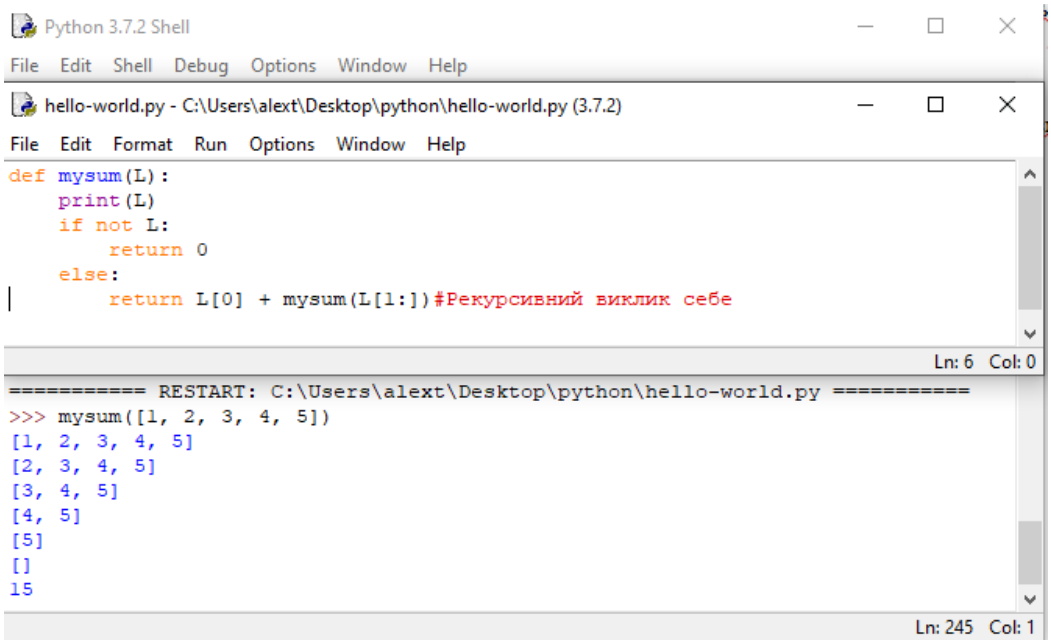
Щоб отримати суму списку (або іншій послідовності) чисел, ми можемо або використовувати вбудовану функцію sum, або написати власну більш спеціалізовану версію. Ось як може виглядати спеціальна функція підсумовування, в якій застосовується рекурсія:



На кожному рівні функція mysum рекурсивно викликає саму себе, щоб обчислити суму залишку списку, яка пізніше додається до елементу в голові списку.

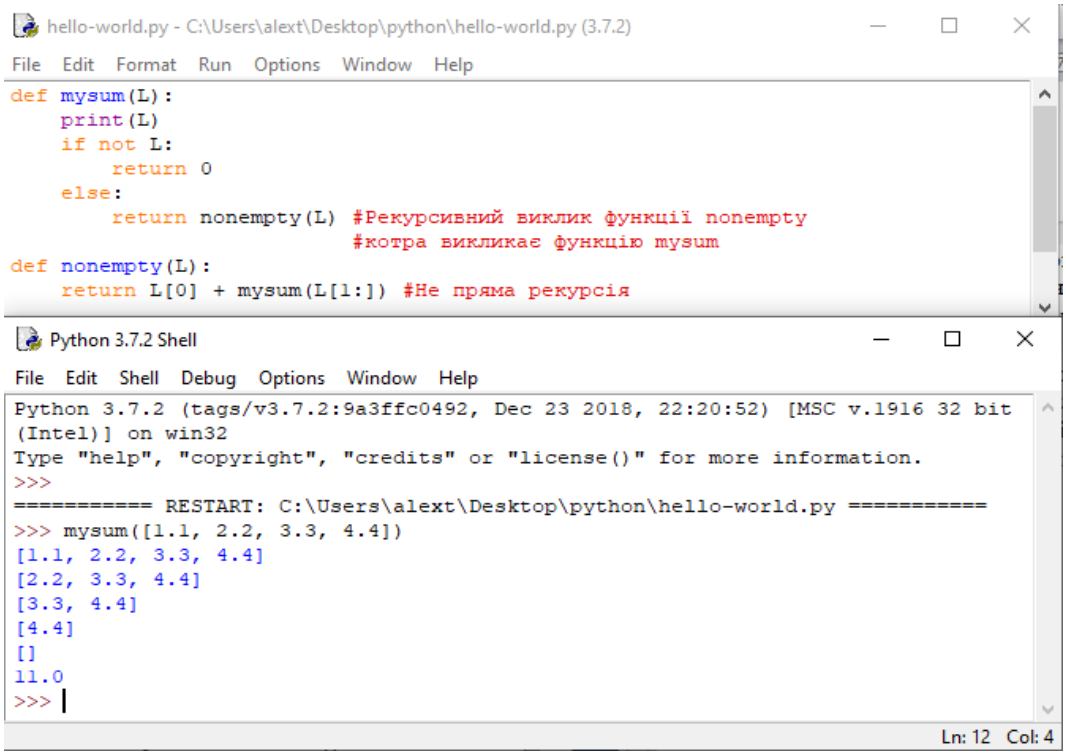
Коли список стає порожнім, рекурсивний цикл закінчується і повертається нуль. У разі використання рекурсії такого роду кожен відкритий рівень виклику функції має власну копію локальної області видимості функції в стек викликів часу виконання - тут це означає, що змінна L на кожному рівні різна.

Додамо в функцію вивід L на екран і запустимо програму знову, щоб відстежити поточний список на кожному рівні виклику.



Як легко помітити, підсумовуючий список на кожному рівні рекурсії стає все менше, поки остаточно не спорожніє - кінець рекурсивного циклу. Сума обчислюється при розкручуванні рекурсивних викликів по поверненню.

Рекурсія може бути прямою, як було показано в прикладах чи непрямою, як в наступному прикладі (функція, що викликає іншу функцію, яка знову викликає першу функцію). Сукупний ефект виявляється таким же, хоча на кожному рівні є два виклики функцій замість одного:

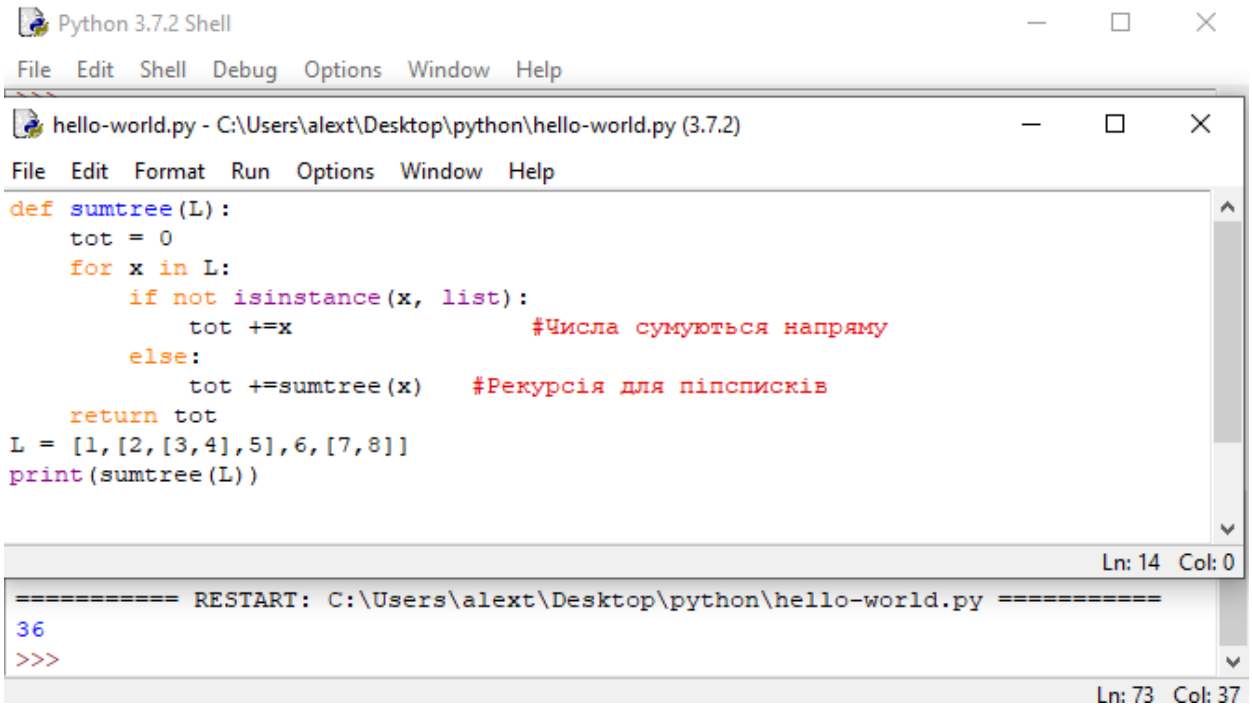


Обробка довільних структур

Рекурсія (або еквівалентні явні алгоритми, засновані на стеці) може вимагатися для обходу структур довільної форми. Як простий приклад рекурсії в такому контексті візьмемо задачу обчислення суми всіх чисел в структурі з вкладеними підсписками такого вигляду:

[1, [2, [3, 4], 5], 6, [7, 8]]

Прості оператори циклів тут не підходять, оскільки це не лінійна ітерація. Вкладених операторів циклу теж не буде достатньо, тому що підсписки можуть бути вкладеними на довільну глибину і в довільній формі. Нема ніякого способу дізнатися, скільки вкладених циклів необхідно написати для обробки всіх випадків. Натомість наступний код пристосовується до такого універсального вкладенню за рахунок застосування рекурсії для відвідування підсписків:



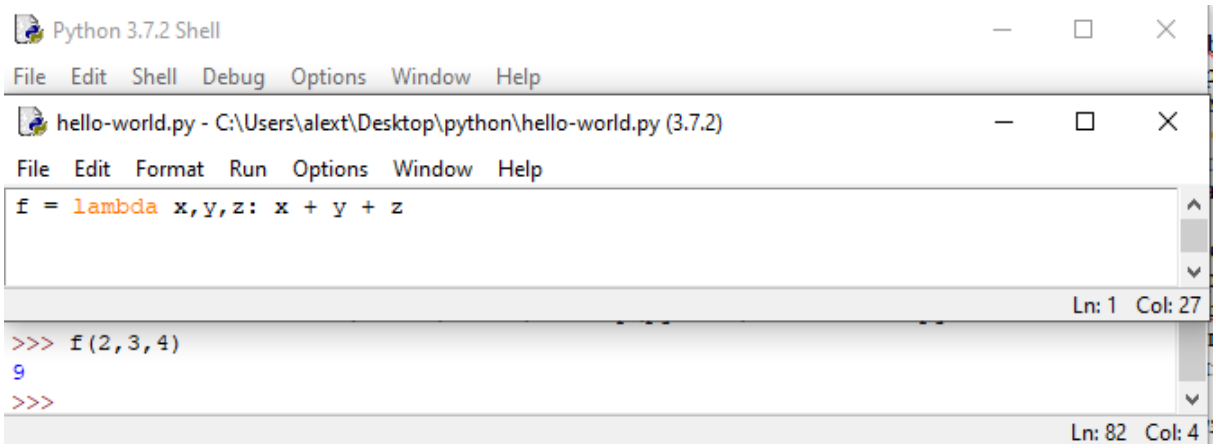
Анонімні функції: вираження lambda

Крім оператора def в Python також пропонується форма вираження, яка генерує об'єкти функцій. Через схожість з інструментом в мові Lisp вона називається lambda. Подібно def такий вислів створює функцію, яка буде викликатися пізніше, але повертає сам об'єкт функції, не привласнюючи його імені. З цієї причини вираження lambda іноді називають анонімними (тобто безіменними) функціями.

На практиці вони часто застосовуються для того, щоб вбудувати визначення функції в рядок або відкласти виконання порції коду.

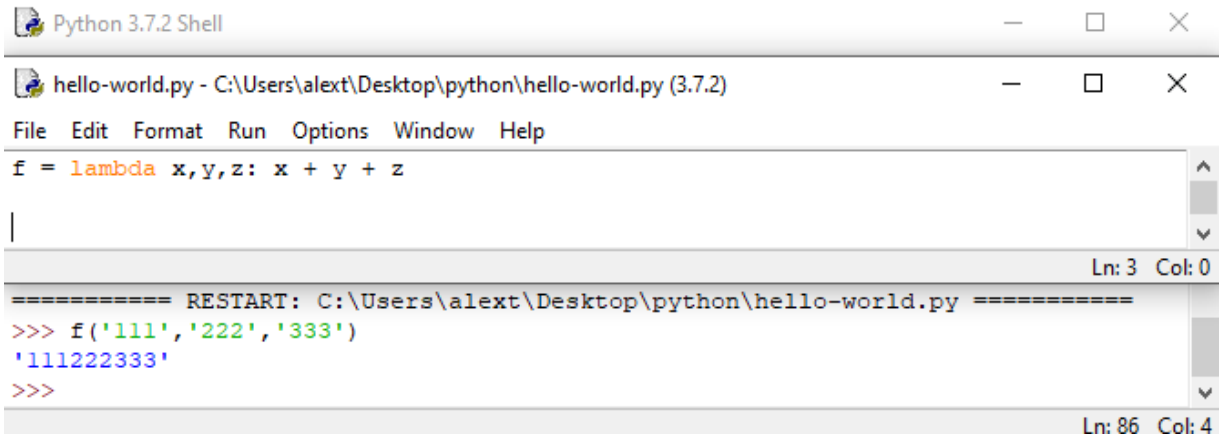
Загальна форма lambda виглядає як ключове слово lambda, за яким слідують один або більше аргументів (дуже схоже на список аргументів, укладений в круглі дужки в заголовку def) і далі вираз після двокрапки:

lambda аргумент 1, аргумент2. . . аргумент№: вираз, що використовує аргументи



Тут імені f присвоюється об'єкт функції, створений виразом lambda; так працює оператор def, але він виробляє присвоювання автоматично.

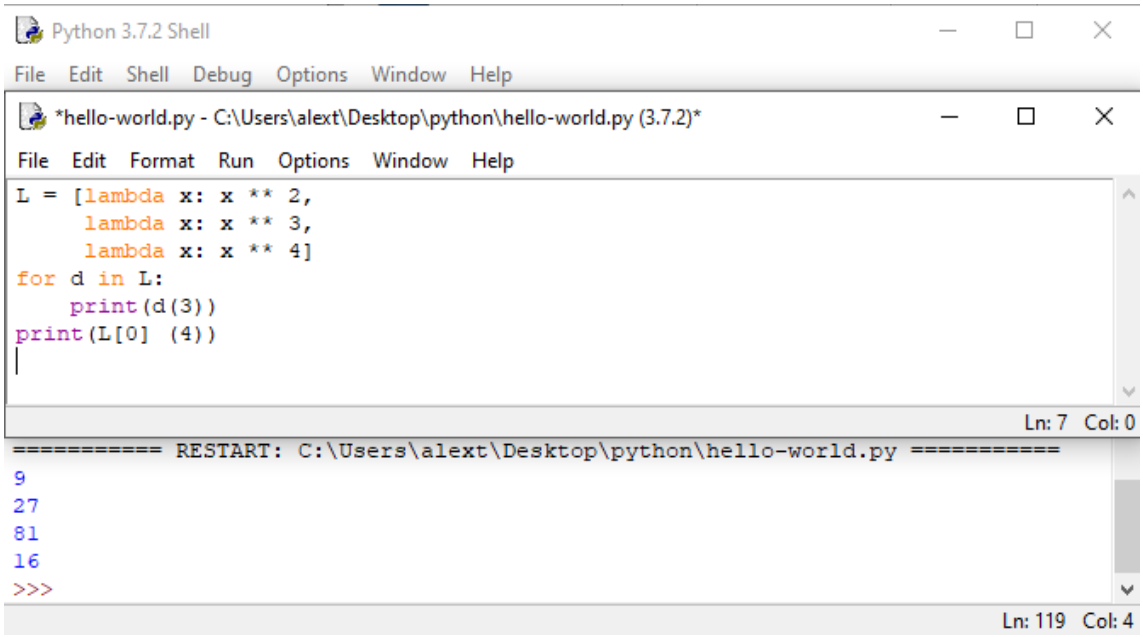
Як і в разі def, для аргументів lambda можна вказувати стандартні значення:



Функції lambda, на відміну від звичайної, не потрібна інструкція return, а в іншому, поводиться точно так же.

Вираз lambda корисний як свого роду коротке умовне позначення функції, яке дозволяє вбудовувати визначення функції всередину коду, де воно застосовується. Вираз lambda зовсім необов'язково використовувати завжди. Замість нього використовується оператор def, якщо функція вимагає мощі повних операторів, яку lambda не здатна легко надати. Але в сценаріях, де потрібно всього лише вбудовувати невеликі порції виконуваного коду в місцях їх застосування, вираження lambda виявляться більш простими кодовими конструкціями.

Вирази lambda також широко використовуються при реалізації таблиць переходів, які представляють собою списки або словники дій, що підлягають виконанню за запитом.

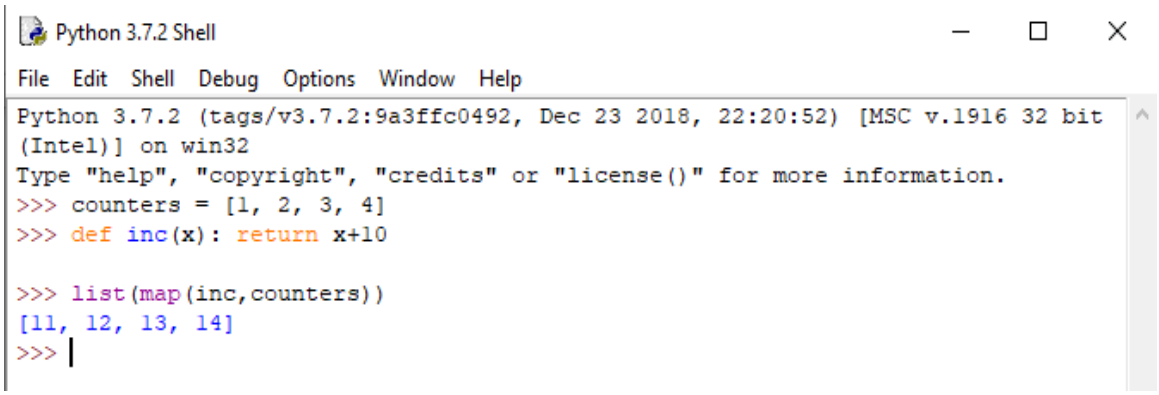


Інструменти функціонального програмування

Вбудована функція **map**

Одним з найбільш частих дій, які виконуються в програмах зі списками і іншими послідовностями, є застосування якоїсь операції до кожного елементу і накопичення результатів - вибір стовпців в таблицях бази даних, збільшення значень в полях із заробітною платою співробітників компанії, розбір вкладень в повідомленнях електронної пошти і т.д.

Вбудована функція **map** застосовує передану функцію до кожного елементу в ітеріруемому об'єкті і повертає список, що містить всі результати викликів функції.



def inc(x): return (x + 10) – Функція яка буде виконуватись.

list(map(inc,counters))- Накопичення результатів.

У списку **map** - викликає inc на кожному елементі списку і збирає все повернені значення в новий список. Для виведення всіх результатів використовується виклик list.

Через те, що таке використання куля еквівалентно циклам for, додавши трохи коду, ви можете отримати універсальну утиліту відображення:

```
>>> def inc(x): return x + 10

>>> def mymap(func, seq):
        res = []
        for x in seq: res.append(func(x))
        return res

>>> list(map(inc,[1, 2, 3]))
[11, 12, 13]
>>>
```

Функцію **map** можна використовувати більш розвиненими способами,

ніж показано тут. Скажімо, отримавши в якості аргументів кілька послідовностей, **map** передає витягнуті з послідовностей елементи як індивідуальні аргументи функції **pow**:

```
>>> pow(3, 4)
81
>>> list(map(pow, [1, 2, 3], [2, 3, 4]))
[1, 8, 81]
>>>
```

Вибір елементів з ітеріруемих об'єктів: **filter**

Функція map - початковий і щодо прямолінійний представник

інструментального набору для функціонального програмування в Python. Її близькі родичі, **filter** та **reduce**, вибирають елементи з ітеріруемих

об'єктів на основі перевірочної функції і застосовують функції до пар елементів відповідно.

Через повернення ітеріруемого об'єкта функція **filter** (подібно range) вимагає виклику **list** при відображенні всіх її результатів . Наприклад filter вибирає з послідовності елементи більше нуля:

```
>>> list(range(-5, 5))
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]
>>> list(filter((lambda x: x > 0), range(-5, 5)))
[1, 2, 3, 4]
>>>
```

Як і **map**, функція **filter** приблизно еквівалентна циклу for, але вона є вбудованої, лаконічною і часто більш швидкою:

```
>>> res = []
>>> for x in range(-5, 5):
        if x > 0:
                res.append(x)


>>> res
[1, 2, 3, 4]
>>>
```

Також подібно **map** функцію **filter** можна емалювати за допомогою синтаксису спискового включення з часто більш простими результатами і за допомогою схожого генераторного виразу, коли бажано відкладати випуск результатів.

```
>>> [x for x in range(-5, 5) if x > 0]
[1, 2, 3, 4]
>>>
```

Комбінування елементів з ітеріруемих об'єктів: **reduce**

Виклик функції reduce, яка знаходиться в модулі functools, виглядає складніше. Вона приймає ітеріруемий об'єкт, що підлягає обробці, але сама не є ітеріруемим об'єктом, а повертає одиночний результат. Нижче показані два виклики reduce, які обчислюють суму і добуток елементів у списку:

```
>>> from functools import reduce
>>> reduce((lambda x, y: x + y), [1, 2, 3, 4])
10
>>> reduce((lambda x, y: x * y), [1, 2, 3, 4])
24
>>>
```

На кожному кроці reduce передає поточну суму або добуток разом з черговим елементом зі списку зазначеної функції lambda. За замовчуванням початкове значення відповідає першому елементу послідовності.

Практичне завдання

1. За допомогою вбудованої функції **map** збільшити у 5 разів усі елементи списку A=[22, 33, 44, 55, 66]. Вивести новий список на екран.
2. За допомогою вбудованої функції **map** ввести масив[5, 6, 7, 8, 9] у ступінь [3, 4, 5, 8, №]. Де № - остання цифра номеру студента у списку групи. Перший елемент першого масиву возводиться у ступень першого елемента другого масиву, другий елемент першого масиву возводиться у ступень другого елемента другого масиву….
3. За допомогою lambda створити список з чотирьох функцій: перша збільшує число на №, друга зводить у ступінь 3, третя зменшує на одиницю, наступна зводить у ступінь 5. Де №- остання цифра номеру студента у списку групи. Використати число №+2 для перевірки функції.
4. За допомогою filter вибрати з генерованої послідовності на інтервалі [-15, 15] елементи менше №. Де №- остання цифра номеру студента у списку групи.
5. За допомогою функції reduce обчислити суму і добуток елементів у списку [12, 34, 45, 66, №]. Де №- остання цифра номеру студента у списку групи.
6. Написати програму розрахунку №+10 числа послідовності Фібоначчі. Де №- остання цифра номеру студента у списку групи.
7. Написати програму розрахунку суми №+10 перших чисел послідовності Фібоначчі. Де №- остання цифра номеру студента у списку групи.
8. Написати програму розрахунку факторіалу числа №+10. Де №- остання цифра номеру студента у списку групи.

9. За допомогою lambda створити список з чотирьох функцій: перша збільшує число у №, друга зводить у ступінь 6, третя зменшує на два, наступна зводить у ступінь 2. Де №- остання цифра номеру студента у списку групи. Використати число №+3 для перевірки функції.

Практична робота №9

Тема. Включення і генератори.

Мета роботи. Освоїти спискові включення і інструменти функціонального програмування.

Зміст.

1. Вивчення відомостей про спискові включення. Генератори.
2. Виконання роботи.
3. Отримання результату.

Ключові положення.

Інструменти map і filter - головні члени раннього інструментального набору для функціонального програмування на Python. Вони відображають операції на ітеруємі об'єкти і накопичують результати. Через те, що ця задача настільки поширена при написанні з'явився новий вираз - спискові включення, які володіють навіть більшою гнучкістю, ніж інші інструменти.

Формальний синтаксис включень.

Насправді спискові включення навіть більш універсальні. Їх найпростіша форма передбачає зазначення виразу, який накопичується і одиночної конструкції for:

[*Вираз for мета in ітеруємий_об'єкт*]

Незважаючи на необов'язковість всіх інших частин, вони дозволяють висловлювати розвиненіші ітерації - в списковому включенні допускається записувати будь-яку кількість вкладених циклів for, кожний з яких може мати необов'язкову асоційовану перевірку if, що діє як фільтр. Загальна структура спискових включень виглядає наступним чином:

[Вираз for мета1 in ітеруємий_об'єкт1 if умова1
for мета2 in ітеруємий_об'єкт2 if умова 2. . .
for мета № in ітеруємий_об'єкт№ if умова №]

Такий самий синтаксис успадкований включеннями множин і словників, а також генераторними виразами, які з'явилися пізніше, хоча вони використовують інші символи (фігурні дужки або часто необов'язкові круглі дужки), а включення словника починається з двох виразів, розділених двокрапкою (для ключа і значення).

Наприклад:

```
>>> [x + y for x in 'spam' for y in 'SPAM']
['sS', 'sP', 'sA', 'sM', 'pS', 'pP', 'pA', 'pM', 'aS',
'aP', 'aA', 'aM', 'mS', 'mP', 'mA', 'mM']
>>>
```

Наприклад:

```
>>> res =[x + y for x in [0, 1, 2] for y in [100, 200, 300]]
>>> res
[100, 200, 300, 101, 201, 301, 102, 202, 302]
>>>
```

Кожна конструкція for може мати асоційований фільтр if незалежно від того, наскільки глибоко вкладені цикли.

Наприклад:

```
>>> [x + y for x in 'spam' if x in'sm' for y in 'SPAM' if y in ('P', 'A')]
['sP', 'sA', 'mP', 'mA']
```

Наприклад:

```
>>> [x + y +z for x in 'spam' if x in'sm'
              for y in 'SPAM' if y in ('P', 'A')
              for z in '246' if z > '2']
['sP4', 'sP6', 'sA4', 'sA6', 'mP4', 'mP6', 'mA4', 'mA6']
>>>
```

Вираз комбінує парні числа від 0 до 4 з непарними числами від 0 до 4. Конструкції if фільтрують елементи на кожній ітерації.

Наприклад:

```
>>> [(x, y) for x in range(5) if x % 2 == 0
            for y in range(5) if y % 2 == 1]
[(0, 1), (0, 3), (2, 1), (2, 3), (4, 1), (4, 3)]
>>>
```

Спискові включення і матриці

Наприклад, в наступному коді визначаються дві матриці 3 x 3 у вигляді списків, що складаються з вкладених списків:

```
>>> M = [[1, 2, 3],
         [4, 5, 6],
         [7, 8, 9]]
>>> N = [[2, 2, 2],
         [3, 3, 3],
         [4, 4, 4]]
>>> M[1]          #Строка 2
[4, 5, 6]
>>> M[1][2]       #Строка 2, елемент 3
6
>>>
```

Спискові включення є потужним інструментом для обробки таких структур як матриці, тому що вони автоматично переглядають рядки і стовпці.

```
>>> M = [[1, 2, 3],
         [4, 5, 6],
         [7, 8, 9]]
>>> N = [[2, 2, 2],
         [3, 3, 3],
         [4, 4, 4]]
>>> [row[1] for row in M] # Стовбчик 2
[2, 5, 8]
>>> [M[row][1] for row in (0, 1, 2)]  #Використання зсуву
[2, 5, 8]
>>> [M[row][0] for row in (0, 1, 2)]  #Використання зсуву
[1, 4, 7]
>>>
```

За заданими позиціями ми також можемо виконувати завдання на зразок вилучення діагоналі. У першому з наведених далі виразів із застосуванням функції range генерується список зсувів і проводиться індексація з однаковими

номерами рядків і стовпців, що вибирає M [0] [0], потім M [1] [1] і т.д. У другому рядку індекс стовпця врівноважується для вилучення M [0] [2], M [1] [1] і т.д. Ми припускаємо, що матриця має рівну кількість рядків і стовпців.

Наприклад:

```
>>> M = [[1, 2, 3],
         [4, 5, 6],
         [7, 8, 9]]
>>> [M[i][i] for i in range(len(M))]
[1, 5, 9]
>>>
```

Витягування зворотної діагоналі:

```
>>> M = [[1, 2, 3],
         [4, 5, 6],
         [7, 8, 9]]
>>> [M[i][len(M)-1-i] for i in range(len(M))]
[3, 5, 7]
>>>
```

Приклад зміни кожного члена матриці на 10:

```
>>> M = [[1, 2, 3],
         [4, 5, 6],
         [7, 8, 9]]
>>> [[col + 10 for col in row] for row in M]
[[11, 12, 13], [14, 15, 16], [17, 18, 19]]
>>>
```

Приклад перемноження матриць:

```
>>> N = [[2, 2, 2],
         [3, 3, 3],
         [4, 4, 4]]
>>> M = [[1, 2, 3],
         [4, 5, 6],
         [7, 8, 9]]
>>> [[M[row][col]*N[row][col] for col in range(3)] for row in range(3)]
[[2, 4, 6], [12, 15, 18], [28, 32, 36]]
>>>
```

Генераторні функції і вирази

Генераторні функції записуються як нормальні оператори def, але в них застосовуються оператори yield, щоб повертати по одному результату за раз, призупиняти виконання зі збереженням стану і відновлювати його між видачами.

В великих програмах генератори можуть бути краще в плані пам'яті і продуктивності. Вони дозволяють функціям уникнути виконання всієї роботи заздалегідь, що особливо корисно, коли результуючі списки великі або отримання кожного значення вимагає тривалих обчислень. Генератори розподіляють час, необхідний для виробництва серії значень, за всіма итерациям циклу. Крім того, для ускладнених сценаріїв застосування генераторів здатне запропонувати більш просту альтернативу ручному збереженню стану між ітераціями. В об'єктах класів - генератори забезпечують автоматичне збереження і відновлення змінних, доступних в області видимості функцій.

Генераторні функції yield або return

Генераторні функції тісно пов'язані з поняттям протоколу ітерації в Python. Об'єкти ітераторів визначають метод, який або повертає черговий елемент в ітерації, або ініціює спеціальне виключення Stopiteration для закінчення ітерації. Итератор ітеруємого об'єкта спочатку витягується за допомогою вбудованої функції iter, хоча цей крок нічого не робить для об'єктів, які самі є ітераторами.

Цикли for в Python і всі інші ітераційні контексти використовують протокол ітерації для проходу по генератору послідовностей або значень, якщо протокол підтримується (якщо ні, тоді ітерація натомість робить багаторазову індексацію послідовностей). Будь-який об'єкт, що підтримує такий інтерфейс, працює з усіма ітераційним інструментами.

Для підтримки протоколу ітерації функції, що містять оператор yield, компілюються в особливий спосіб як генератори - вони не будуть нормальними функціями, а будуються з метою повернення об'єкта з очікуваними методами з протоколу ітерації. При наступному виклику вони повертають об'єкт генератора, який підтримує інтерфейс ітерації з автоматично створеним методом по імені, призначеним для запуску або відновлення виконання.

Генераторні функції можуть також мати оператор return, який поряд з переміщенням за кінець блоку def просто припиняє генерацію значень - формально за рахунок ініціювання виключення Stopiteration після всіх звичайних дій по виходу з функції. З точки зору викликаючого коду, метод генератора відновлює виконання функції до тих пір, поки або не повернеться наступний результат yield, або до виникнення виключення StopIteration.

Сукупний ефект в тому, що генераторні функції, які записані як оператори def, що містять оператори yield, автоматично робляться доступними для протоколу ітерації і тому можуть застосовуватися в будь-якому ітераційному контексті, щоб виробляти результати з плином часу і за запитом.

У наступному коді визначається генераторная функція, яку можна застосовувати для генерації квадратів серії чисел з плином часу:

```
>>> def gensquares(N):
        for i in range(N):
                yield i ** 2 #Пізніше тут возобновить виконання

>>> for i in gensquares(5):  # Відновлення виконання функції
        print(i, end=' : ')   # Вивід останнього виданого значення

0 : 1 : 4 : 9 : 16 :
>>>
```

Функція gensquares видає значення і тому повертає управління коду який викликається на кожній ітерації циклу; при поновленні її виконання відновлюється попередній стан, включаючи останні значення змінних i і N, а управління знову підхоплюється безпосередньо після оператора yield. Скажімо, коли gensquares використовується в тілі циклу for, перша ітерація починає функцію і отримує перший результат; потім на кожній ітерації циклу управління повертається функції після оператора yield.

Щоб закінчити генерацію значень, функції або застосовують оператор return без значення, або дозволяють потоку управління дійти до кінця тіла функції.

Генераторні вирази:

ітеруємі об'єкти які зустрічаються з включеннями.

Генераторні вирази схожі на спискові включення, але вони не будують результуючий список, а повертають об'єкти, які виробляють результати за запитом. Синтаксично генераторні вирази схожі на нормальні спискові включення і підтримують весь їх синтаксис, в тому числі фільтри if і вкладення циклів, але вони розміщаються в круглі дужки, а не в квадратні (подібно кортежам круглі дужки часто необов'язкові).

У точності як генераторні функції, генераторні вирази забезпечують оптимізацію витрат пам'яті - вони не вимагають створення відразу всього результуючого списку, що відбувається в разі спискового включення в квадратних дужках. Також подібно генераторним функцій вони поділяють роботу з випуску результатів на невеликі тимчасові інтервали - результати видаються поступово замість того, щоб викликати код та очікувати створення повного набору в єдиному виклику.

З іншого боку, на практиці генераторні вирази можуть виконуватися трохи повільніше спискових включень, а тому їх найкраще застосовувати для дуже великих результуючих наборів або в додатках, які не можуть чекати генерації повних результатів.

Наприклад:

Генераторний вираз будує інтеруємий об’єкт.

```
>>> (x**2 for x in range(4))
<generator object <genexpr> at 0x03549C70>
```

```
>>> list(x**2 for x in range(4))
[0, 1, 4, 9]
>>>
```

Для примусового формування всіх результатів відразу використовуємо вбудовану функцію list.

Практичне завдання

1. Написати програму, яка комбінує непарні числа від 0 до 7 з парними числами від 0 до 7 за допомогою списків включень.
2. Для матриці A [[11, №, 44, 55],[1, 2, №, 4],[11, 12, 13, 14],[21, 31, 41, №]]. Вивести на екран: головну діагональ, зворотну діагональ, другий стовпчик матриці. Збільшити кожен член матриці на №. Помножити матрицю А на матрицю М де М = [[1, 2, 3,6], [4, 5, 6,0], [7, 8, 9,7], [71, 8, 93,7]]. Де №- остання цифра номеру студента у списку групи.
3. За допомогою генераторної функції та **yield** написати код для генерації кубів серії чисел з плином часу. Вивести № сгенерованих чисел. Де №- остання цифра номеру студента у списку групи.

Практична робота №10

Тема. Об'єктно-орієнтоване програмування на Python 3. Конструктор класса – метод __init__().

Мета роботи. Об'єктно-орієнтоване програмування на Python 3.

Зміст.

1. Вивчення відомостей про об'єктно-орієнтоване програмування на Python 3. Конструктор класса – метод __init__().
2. Виконання роботи.
3. Отримання результату.

Ключові положення.

Цикли, розгалуження і функції — все це елементи процедурного програмування. Його можливостей цілком достатньо для написання невеликих,

простих програм. Однак великі проекти часто реалізують, використовуючи парадигму об'єктно-орієнтованого програмування (ООП).

Припустимо, команда програмістів займається розробкою гри. Програму-гру можна представити як систему, яка складається з цифрових героїв і середовища їх існування, яке включає багато предметів. Кожен воїн, зброя, дерево, будинок — це цифровий об'єкт, в якому "упаковано" його властивості і дії, за допомогою яких він може змінювати свої властивості і властивості інших об'єктів.

Кожен програміст може розробляти свою групу об'єктів. Розробникам достатньо домовитись між собою тільки про те, як об'єкти будуть взаємодіяти між собою.

Ключову різницю між програмою, написаною в структурному стилі, і об'єктно-орієнтованою програмою можна висловити так: у першому випадку на перший план виходить логіка, розуміння послідовності виконання дій для досягнення поставленої цілі. У другому — важливіше представити програму як систему об'єктів, які взаємодіють.

Основними поняттями в ООП є клас, об'єкт, успадкування, інкапсуляція і поліморфізм.

Поняття –клас. Тип int — це клас цілих чисел. Числа 5, 100134, -10 і т. д. — це конкретні об'єкти цього класу. В Python об'єкти також прийнято називати екземплярами.

Поняття — успадкування. Нехай є клас столів, який описує загальні властивості усіх столів. Однак можна розділити усі столи на письмові, обіді і журнальні і для кожної групи створити свій клас, який буде спадкоємцем загального класу, але також вносити ряд своїх особливостей. Таким чином, загальний клас буде батьківським, а класи груп — дочірніми.

Дочірні класи успадковують особливості батьківських, однак доповнюють або у деякій мірі модифікують їх характеристики. Коли ми створюємо конкретний екземпляр стола, то повинні обрати, до якого класу столів він буде належати. Якщо він належить класу журнальних столів, то

отримає усі характеристики загального класу столів і класу журнальних столів. Але не особливості письмових і обідніх.

Поняття—Інкапсуляція. Приховування даних, тобто неможливість напряму отримати доступ до внутрішньої структури об'єкта, так як це небезпечно. Інший смисл інкапсуляції — об'єднання властивостей і поведінки в одне ціле, тобто в клас.

Поняття — Поліморфізм. Об'єкти різних класів, з різною внутрішньою реалізацією можуть мати однакові інтерфейси. Наприклад, для чисел є операція додавання, яка позначається знаком +. Однак ми можемо визначити клас, об'єкти котрого також будуть підтримувати операцію, яка позначається цим знаком. Але це зовсім не означає, що об'єкти повинні бути числами, і буде отримуватись якась сума. Операція + для об'єктів нашого класу може означати щось інше. Але інтерфейс, в даному випадку це знак +, у чисел і нашего класу буде однаковим. Поліморфність же проявляється у внутрішній реалізації і результаті операції.

Створення класів і об'єктів.¶

В мові програмування Python класи створюють за допомогою команди **class**, після якої вказують ім'я класу, потім ставиться двокрапка, далі з нового рядка і з відступом реалізується тіло класу:

```
class Ім’я класу:
    pass
```

Якщо клас є дочірнім, то батьківські класи перераховуються у круглих дужках після імені класу:

```
class MyClass(ParentClass):
    pass
```

Об'єкт створюється шляхом виклику класу за його іменем. При цьому після імені класу обов'язково ставляться дужки. Оскільки у програмному коді важливо не згубити посилання на щойно створений об'єкт, то зазвичай його пов'язують зі змінною. Отже створення двох об'єктів найчастіше виглядає так:

```
>>> class A:
...     pass
...
>>> a = A()
>>> b = A()
```

Клас як модуль

В Python клас можна представити подібно модулю. Так само як у модулі у ньому можуть бути свої змінні зі значеннями і функції. Так само як у модулі у класу є власний простір імен, доступ до якого можливий через ім'я клас:

```
>>> class B:
...     n = 5
...     def adder(v):
...             return v + B.n
...
>>> B.n
5
>>> B.adder(4)
9
>>>
```

Однак у випадку класів використовується дещо інша термінологія. Імена, визначені в класі, називаються атрибутами цього класу. В вищенаведеному прикладі імена n і adder — це атрибути класу B. Атрибути-змінні часто називають полями і деколи властивостями. Атрибути-функції називаються методами. Кількість полів і методів у класі може бути довільною.

Клас як створювач об'єктів¶

```
>>> 1 = B()
```

```
>>> 1.n
5
>>> 1.adder(4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: add() takes 1 positional argument but 2 were given
>>>
```

Інтерпретатор повідомляє нам, що `adder()` приймає тільки один аргумент, а було передано два. Звідки ж взявся другий аргумент, і хто він такий, якщо у дужках було вказано тільки одне число 4?

Клас створює об'єкти, які у певному сенсі є його спадкоємцями. Це означає, що якщо у об'єкта немає власного поля n, то інтерпретатор шукає його на один рівень вище, тобто у класі. Таким чином, якщо ми присвоюємо об'єкту поле с таким самим ім'ям як у класі, то воно перекриває, або перевизначає, поле клас:

```
>>> 1.n = 10
>>> 1.n
10
>>> B.n
5
>>>
```

Тут змінні `1.n` і `B.n` — це різні змінні. Перша знаходиться у просторі імен об'єкта 1, друга — у просторі класу `B`. Якщо б ми не додали поле `n` до об'єкта `a`, то інтерпретатор піднявся б вище по дереву наслідування і прийшов би у клас, де би і знайшов це поле. Щодо методів, то вони також наслідуються об'єктами клас. У даному випадку у об'єкта `a` немає свого власного метода `add`, отже, він шукається в клас `Adder`. Однак від клас `Adder` може бути породжено багато об'єктів. Методи ж найчастіше призначаються для обробки об'єктів. Таким

чином, коли викликається метод, у нього треба передати конкретний об'єкт, який він буде обробляти. Зрозуміло, що екземпляр, що передається — це об'єкт, до якого застосовується метод. Вираз `1.adder(100)` виконується інтерпретатором наступним чином:

1. Шукаю атрибут `adder()` у об'єкта `1`. Не знайшов.
2. Тоді йду шукати у клас `B`, так як він створив об'єкт `1`.
3. Тут знайшов метод. Передаю йому об'єкт, до якого цей метод треба застосувати, і аргумент, що вказано у дужках.

Іншими словами, вираз

1.adder(100)

перетвоюється у вираз

B.adder(1, 100)

Таким чином, інтерпретатор спробував передати у метод adder() класу B два аргумента — об'єкт 1 і число 100. Але ми запрограмували метод adder() так, що він приймає тільки один параметр. В Python визначення методів не передбачають прийняття об'єкта як зрозуміле за замовчуванням. Об'єкт що приймається треба вказувати явно. За згодою у Python для посилання на об'єкт використовується ім'я `self`. Ось так повинен виглядати метод adder(), якщо ми плануємо викликати його через об'єкти:

```
>>> class B:
...         n = 5
...         def adder(self, v):
...                 return v + self.n
...
>>>
```

Змінна self зв'язується з об'єктом, до якого було застосовано даний метод, і через цю змінну ми отримуємо доступ до атрибутів об'єкта. Коли цей же метод застосовується до іншого об'єкта, то self зв'яжеться вже з саме цим іншим об'єктом, і через цю змінну будуть вилучатись тільки його поля.

Давайте протестуємо оновлений метод:

```
>>> l = B()
>>> m = B()
>>> l.n = 10
>>> l.adder(3)
13
>>> m.adder(4)
9
```

Тут від класу B створюється два об'єкта – l та m. Для об'єкта l заводиться власне поле n. Об'єкт m, не має такого поля, отже успадковує n від класу B. Переконаємось у цьому:

```
>>> m.n is B.n
True
>>> l.n is B.n
False
```

У методі adder() вираз self.n – це звернення до поля n, переданого об'єкта, і не важливо, на якому рівні наслідування його буде знайдено.

Зміна полів об'єкта¶

Прийнято привласнювати поля, а також отримувати їх значення, шляхом виклику методів:

```
>>> class User:
...     def setName(self, n):
...         self.name = n
```

```
...     def getName(self):
...         try:
...             return self.name
...         except:
...             print("No name")
...
>>> first = User()
>>> second = User()
>>> first.setName("Bob")
>>> first.getName()
'Bob'
>>> second.getName()
No name
```

Методи називають сетерами (set – встановить) та гетерами (get – отримати).

Конструктор класу – метод __init__().

В ООП конструктором класу називають метод, який автоматично викликається при створенні об'єктів. Його також можна назвати конструктором об'єктів класу. Ім'я такого метода зазвичай регламентується синтаксисом конкретної мови програмування. В Python роль конструктора виконує метод `__init__()`. Об'єкт створюється в момент виклику класу по імені, і в цей момент викликається метод **init**(), якщо його визначено в класі.

Необхідність конструкторів пов'язана з тим, що часто об'єкти повинні мати власні властивості одразу. Припустимо маємо клас Person, об'єкти котрого обов'язково повинні мати ім'я та прізвище. Якщо клас буде описано наступним способом:

```
class Person:
    def setName(self, n, s):
        self.name = n
        self.surname = s
```

то створення об'єкта можливе без полів. Для встановлення імені метод set_name() необхідно викликати окремо:

```
>>> from test import Person
>>> p1 = Person()
>>> p1.setName("Bill", "Ross")
>>> p1.name, p1.surname
('Bill', 'Ross')
```

Наявність конструктора не дозволить створити об'єкт без полів:

```
class Person:
    def __init__(self, n, s):
        self.name = n
        self.surname = s

p1 = Person("Sam", "Baker")
print(p1.name, p1.surname)
```

Тут при виклику класу в круглих дужках передаються значення, котрі будуть присвоєні параметрам метода **init**(). Перший параметр – self – посилання на сам щойно створений об'єкт.

Буває, що необхідно допустити створення об'єкта, якщо деякі дані в конструктор не передаються. У такому випадку параметрам конструктора класу задаємо значення за замовчуванням:

```
class Rectangle:
    def __init__(self, w = 0.5, h = 1):
        self.width = w
        self.height = h
    def square(self):
        return self.width * self.height

rec1 = Rectangle(5, 2)
rec2 = Rectangle()
```

```
rec3 = Rectangle(3)
rec4 = Rectangle(h = 4)
print(rec1.square())
print(rec2.square())
print(rec3.square())
print(rec4.square())
```

Деструктор класа¶

Окрім конструктора об'єктів в ООП є зворотній йому метод – деструктор. Він викликається, коли об'єкт знищується.

В Python об'єкт знищується, коли зникають усі пов'язані з ним змінні або їм присвоюється інше значення, в результаті чого зв'язок з старим об'єктом втрачається. Видалити змінну також можна за допомогою `del`.

В Python функцію деструктора виконує метод `__del__()`. Але в Python деструктор використовується рідко, інтерпретатор і без нього добре впорається зі "сміттям".

Практичне завдання

1. Є клас Person, конструктор якого приймає три параметри (не враховуючи self) - ім'я, прізвище і оцінку студента. Оцінка має значення задане за замовчуванням, рівне одиниці.
2. У класу Person є метод, який повертає рядок, що включає в себе всю інформацію про студента.
3. Клас Person містить деструктор, який виводить на екран фразу "Ви отримали стипендію ..." (замість три крапки повинні виводитися ім'я та прізвище об'єкта).
4. В основній гілці програми створіть три об'єкти класу Person. Подивіться інформацію про студента і об’явить, що він отримав стипендію.

5. В кінці програми додайте функцію input (), щоб скрипт не завершився сам, поки не буде натиснуто Enter. Інакше ви відразу побачите як видаляються всі об'єкти при завершенні роботи програми.

Практична робота №11

Тема. Проектування ігри «Шибениця» за допомогою блок-схем.

Мета роботи. Освоїти ASCII-графіку. Освоїти проектування гри за допомогою блок-схем. Списки, оператор in, методи, строкові методи split(), lower(), upper(), startswith() і endswith(), інструкції elif.

Зміст.

4. Вивчення відомостей про ASCII-графіку.
5. Освоїти проектування гри за допомогою блок-схем.
6. Отримання результату.

Ключові положення.

Шибениця - це гра для двох, в якій один гравець загадує слово (це буде комп’ютер) і малює на сторінці окремі порожні клітини для кожної літери. А другий гравець намагається вгадати букви, які можуть бути в даному слові, а потім і все слово. Якщо другий гравець правильно вгадує букву, перший гравець вписує її в відповідну порожню клітину. А якщо помиляється, перший гравець малює одну з частин тіла повішеного чоловічка. Щоб перемогти, другий гравець повинен вгадати всі букви в слові до того, як повішений чоловічок буде повністю намальований.

Графіка для гри «Шибениця» - це символи клавіатури, надруковані на екрані. Такий вид графіки називається ASCII-графіка. Нижче показано шибеницю, яка намальована за допомогою ASCII-графіки:

```
 +---+   +---+   +---+   +---+   +---+   +---+   +---+
     |   0   |   0   |   0   |   0   |   0   |   0   |
     |       |   |   |  /|   |  /|\  |  /|\  |  /|\  |
     |       |       |       |       |  /    |  / \  |
    ===     ===     ===     ===     ===     ===     ===
```

Блок-схема являє собою діаграму, яка відображатиме серію кроків у вигляді полів (блоків), пов'язаних один з одним стрілками. Кожне поле - це крок, а стрілки показують можливі кроки. На рис.1 показана повна блок-схема гри «Шибениця».

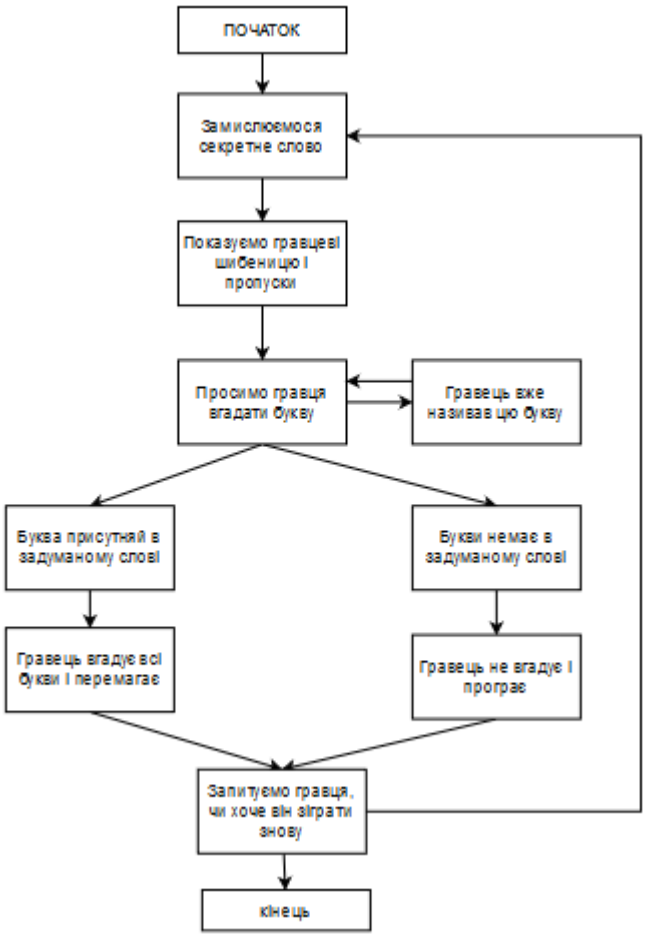


Рис.1

Підключимось до бібліотеки випадкових чисел за допомогою команди import random. Рядки з 2 по 37 - це одна довга інструкція присвоювання для змінної HANGMAN_PICS.

```
import random
HANGMAN_PICS= ['''
+---+
   |
   |
   |
  ===''','''
+---+
 0  |
    |
    |
  ===''','''
+---+
 0  |
 | |
    |
  ===''','''
+---+
 0  |
/|  |
    |
  ===''','''
+---+
 0  |
/|\ |
    |
```

```
  ===''','''
+---+
 0  |
 /|\ |
 /   |
  ===''','''
+---+
 0  |
 /|\ |
 / \ |
  ===''']
```

```
words = 'аист акула бабуин барсук бобр бык верблюд'.split()
```

У цьому рядку коду гри використовується метод split (). Код виглядає довгим, але, насправді, це просто рядок слів, розділених пробілами, з викликом методу split () в кінці. Метод split () створює список, в якому кожне слово з рядка стає окремим елементом списку words.

```
def getRandomWord(wordList):
    # Ця функція повертає випадковий рядок з переданого списку
    wordIndex = random.randint(0, len(wordList) - 1)
    return wordList[wordIndex]
```

У рядку визначається функція getRandomWord (). Значення елементів списку передаються аргументу wordlist в якості параметрів. Ця функція повертає одиничне секретне слово з списку words. wordIndex = random.randint (0, len (wordList) - 1) З списку в якості значення змінної wordIndex зберігається елемент з випадковим індексом. Випадковий вибір елемента проводиться функцією randint () з двома аргументами. Перший аргумент – 0 а другий - значення виразу len (wordList) - 1, що визначає останній можливий індекс. Змінна wordIndex зберігає випадковий індекс зі списку, переданого за допомогою параметра wordList. return wordList[wordIndex] - повертає зі списку

wordList значення елемента з відповідним індексом, який зберігається як ціле число в wordIndex.

Відображення ігрового поля для гравця

Далі необхідна функція промальовування ігрового поля гри «Шибениця». На ньому має відображатися кількість введених гравцем букв, вгаданих як вірно, так і помилково.

missedLetters - Рядок букв, які гравець назвав, але їх немає в загаданому слові; **correctLetters** - Рядок букв, які гравець вгадав в загаданому слові; **secretWord** - Рядок з загаданим словом, яке гравець намагається відгадати.

```
def displayBoard(missedLetters, correctLetters, secretWord):
    print(HANGMAN_PICS[len(missedLetters)])
    print()
    print('Ошибочные буквы:', end=' ')
    for letter in missedLetters:
        print(letter, end= ' ')
    print()
    blanks = '_' * len(secretWord)
    for i in range(len(secretWord)): # замінює пропуски відгаданими буквами
        if secretWord[i] in correctLetters:
            blanks = blanks[:i] + secretWord[i] + blanks[i+1:]

    for letter in blanks: # Показує секретне слово з пробілами між буквами
print(letter, end=' ')
    print()
```

Спочатку функція print () викликає відображення ігрового поля. Глобальна змінна HANGMAN_PICS містить список строкових змінних для промальовування всіх можливих етапів гри.

Код HANGMAN_PICS [0] відображає порожню «шибеницю», код HANGMAN_PICS [1] показує голову.

Число букв, яке зберігається в змінної missedLetters, відображає кількістю неправильних припущень, зроблених гравцем. Для визначення цього числа викликається функція len (missedLetters). Тобто якщо значення змінної missedLetters одно, наприклад, 'лелека', то код len ('лелека') поверне 6. Код HANGMAN_PICS [6] відобразить повішеного, відповідного 6 промахів.

У циклі for відбувається перебір всіх сімволів змінної missedLetters і виведення їх на екран. Вираз end = " заміщає символ нового рядка, який є в кінці кожного рядка, - одиничним пропуском. Наприклад, якщо значення missedLetters одно 'аизх', то такий цикл for виведе на екран а и з х.

Інша частина коду функції displayBoard () виводить на екран літери і формує рядок - секретне слово, в якому ще не угадані букви заміщені пробілами. Це досягається за допомогою функції range () і зрізу списку.

Вивід секретного слова з пробілами.

Строкова змінна blanks, з підкресленнями по числу букв секретного слова. Спочатку потрібно створити рядок з символів нижнього підкреслення для всіх букв секретного слова. Потім замінити прогалини цими символами. Наприклад, для секретного слова 'ворон' порожній рядок-підкреслення виглядає так: '_ _ _ _ _' (п'ять підкреслень).

У циклі for виробляє послідовний перебір всіх букв секретного слова і заміщає підкреслення буквами, що містяться в змінної correctLetters.

```
def getGuess(alreadyGuessed):
    # Повертає букву, введену гравцем.
    # Ця функція перевіряє, що гравець ввів тільки одну букву і нічого більше
    while True:
        print('Введить букву.')
```

```
guess = input()
guess = guess.lower()
if len(guess) !=1:
    print('Введить букву.')
elif guess in alreadyGuessed:
    print('Ви вже називали цю букву. Назвить іншу.')
elif guess not in 'абвгдеежзийклмнопрстуфхцчшщъьэюяы':
    print(Будь ласка, введіть ЛІТЕРУ.')
else:
    return guess
```

Отримання припущень гравця при виконанні функції getGuess () гравець може ввести передбачувану букву. Ця функція повертає букву, запропоновану гравцем, у вигляді рядка. Далі функція getGuess () перевіряє допустимість введеного символу, перш ніж передати його в функцію.

Рядок букв, запропонованих гравцем, передається в якості аргументу в параметр alreadyGuessed функції. Потім функція getGuess () просить гравця ввести одну букву. Цю букву функція getGuess () поверне як своє значення.

Перевірка допустимості припущення гравця

Змінна guess містить букви, запропоновані гравцем. Програма повинна упевнитися, що вводяться символи допустимі, це повинна бути тільки одна буква, при цьому не повинні застосовуватися раніше. Якщо ця умова не виконується, цикл повертається до початку і знову запитує букву.

if len(guess) !=1: перевіряється, що введено не більше одного символу,

elif guess in alreadyGuessed: перевіряє, що запропонована буква не міститься в змінної alreadyGuessed, elif guess not in 'абвгдеежзийклмнопрстуфхцчшщъьэюяы': перевіряє, що введений символ стандартного російського алфавіту. Якщо хоча б одна умова не виконується, гравцеві пропонується ввести іншу букву. Якщо виконані всі умови, програма

переходить до виконання коду блоку і повертає значення запропонованої букви в рядку return guess.

Пропозиція гравцеві зіграти зановоФункція playAgain () містить лише виклик функції print () і инструк-цію return.

```
def playAgain():
    # Ця функція повертає значення True, якщо гравець хоче зіграти заново;
    #в іншому випадку повертає False
    print('Хочете зіграти ще? (так чи ні)')
    return input().lower().startswith('т')
```

Функція playAgain () дозволяє гравцеві відповісти «Так» або «Ні» на пред-ложение ще одного раунду. Гравець повинен ввести Так, так, Т або що-небудь ще, що починається з букви т, і це значення буде означати «Так». Якщо гравець ввів ТАК, функція input () повертає значення 'ТАК'. Вираз 'ТАК'.lower () повертає рядок в нижньому регістрі (маленькими літерами). Тобто значення 'ТАК' перетворюється в 'так'. Но, крім цього, проводиться ще й виклик методу startswith (' т '). Ця функція повертає True, якщо викликає рядок починається з указаного параметра, або False, якщо це не так. Наприклад, вираз 'так'.startwith (' т ') повертає значення True.

Ігровий цикл

Основна частина коду програми «Шибениця» відображає на екрані назву гри, містить кілька змінних і запускає цикл while. У цьому розділі розглянемо послідовне виконання решти програмного коду

```
print(‘Ш И Б Е Н И Ц Я ')
missedLetters = ''
correctLetters = ''
secretWord = getRandomWord(words)
```

gameIsDone = False

Код print(‘Ш И Б Е Н И Ц Я ') - викликає функцію print (), яка виводить на екран заголовок гри в момент її запуску. Потім строкової змінної missedLetters присвоюється порожнє значення, змінної correctLetters теж присвоюється порожнє значення, так як гравець не запропонував ще ніяких букв.В рядку secretWord = getRandomWord(words)- викликається функція getRandomWord (words), яка вибирає випадковим чином секретне слово з списка. Код gameIsDone = False - привласнює змінної gameIsDone значення False. Код присвоїть цієї змінної значення True, коли надійде сигнал завершення гри, і запропонує гравцеві зіграти заново.

Виклик функції displayBoard () Частина програми складається з циклу while. Умова циклу завжди істинно, а це значить, що він буде виконуватися нескінченно довго, до тих пір, поки не буде ініційовано виконання інструкції break.

```
while True:
    displayBoard(missedLetters, correctLetters, secretWord)
    # Дозволяє гравцеві ввести букву
    guess = getGuess(missedLetters + correctLetters)
    if guess in secretWord:
        correctLetters = correctLetters + guess
        foundAllLetters = True
        for i in range(len(secretWord)):
            if secretWord[i] not in correctLetters:
                foundAllLetters = False
                break
        if foundAllLetters:
            print('Так! Секретне слово - "' + secretWord + '"! Ви вгадали!')
            gameIsDone = True
    else:
```

```
        missedLetters = missedLetters + guess
        # Перевіряє, чи перевищив гравець ліміт спроб і програв
        if len(missedLetters) == len(HANGMAN_PICS)-1:
            displayBoard(missedLetters, correctLetters, secretWord)
            print('Ви вичерпали усі спроби!\nНе вгадано букв: ' + str(len(missedLetters)) + ' та вгадано букв: ' + str(len(correctLetters)) + '. Было загадано слово "' + secretWord + '".')
            gameIsDone = True
    if gameIsDone:
        if playAgain():
            missedLetters= ''
            correctLetters = ''
            gameIsDone = False
            secretWord = getRandomWord(words)
        else:
            break
```

Код викликає функцію displayBoard (), передаючи їй значення трьох змінних missedLetters, correctLetters, secretWord. Залежно від того, скільки букв гравець вгадав правильно і скільки разів помилився, ця функція виводить на екран відповідне зображення «повішеного».

Введення гравцем букви, що вгадується

Далі guess = getGuess(missedLetters + correctLetters) - викликається функція getGuess (), щоб гравець міг ввести букву, що вгадується. У функцію передається параметр alreadyGuessed для визначення – чи вводив гравець таку букву раніше або ще ні. guess = getGuess(missedLetters + correctLetters)- конкатенуються рядкові змінні missedLetters і correctLetters, а потім передає результат як аргумент параметру alreadyGuessed.

Перевірка наявності букви в секретному слові

Якщо запропонована буква є в секретному слові, вона додається в кінець строкової змінної correctLetters. Та додає нове значення в змінну correctLetters.

```
if guess in secretWord:
    correctLetters = correctLetters + guess
```

Цикл починається в припущенні, що всі букви секретного слова вгадані. Але в рядку foundAllLetters = False, у процесі виконання циклу, значення змінної foundAllLetters змінюється на False, як тільки виявлена перша буква з змінної secretWord, що не міститься в змінної correctLettrs. Якщо всі букви секретного слова виявлені, гравцеві повідомляється про його перемогу, а змінна gameIsDone приймає значення True.

Обробка помилкових припущень

Неправильно вгадані букви додаються в строкову змінну missedLetters в рядку missedLetters = missedLetters + guess. Це відбувається так само як в рядку correctLetters = correctLetters + guess з буквами, вгаданими вірно.

Перевірка - чи не програв гравець. Кожен раз, коли гравець вводить неправильну букву, вона додається в змінну missedLetters. Таким чином, довжина значення змінної missedLetters (або в коді - len (missedLetters)) стає дорівнює числу помилкових припущень.

Змінна HANGMAN_PICS містить сім рядків з ASCII-символами рисунка. Таким чином, якщо довжина рядка missedLetters дорівнює len (HANGMAN_PICS) - 1 (тобто 6), гравець вичерпав ліміт припущень. Якщо малюнок «повішеного» завершено - гравець програв.

Завершення або перезавантаження ігри

В не залежності від перемоги або програшу гра повинна запитати гравця - чи хоче він зіграти знову. Функція playAgain () обробляє отриману відповідь («Так» або «Ні») в рядку if playAgain(). Якщо гравець хоче почати гру заново,

значення змінних missedLettersi correctLetters треба обнулити, змінної gameIsDone привласнити значення False і вибрати нове секретне слово. Після того як виконання циклу while повертається до початку, ігровий інтерфейс, який відображається на екрані, перезавантажується і готовий до нової гри. Якщо гравець не ввів нічого, що починається з букви т, при запиті - чи хоче він зіграти заново, умова стає хибною і виконується блок else.

Увесь код програми представлено в додатку 1.

Практичне завдання

1. Змінити гру так, щоб один гравець вводив слово з клавіатури. Воно не відображалось на екрані, другий відгадував це слово.
2. Змінити шибеницю на відображення цифр від 9 до 0 по кількості поразок.

Практична робота №12

Тема. Робота з класами.

Мета роботи. Освоїти створення 2D гри на Python 3.

Зміст.

1. Вивчення відомостей про роботу з класами.
2. Виконання роботи.
3. Отримання результату.

Ключові положення.

Є ігрове поле - простий прямокутник з твердими межами. Коли кулька торкається стінки або стелі, вона відскакує в іншу сторону. Якщо вона впаде на підлогу - ви програли. Щоб цього не сталося, внизу уздовж підлоги літає платформа, а ви керуєте нею за допомогою стрілок. Ваше завдання -

підставляти платформу під кульку якомога довше. За кожний вдалий порятунок кульки ви отримуєте одне очко.

Щоб реалізувати таку логіку гри, потрібно передбачити такі сценарії поведінки:

1. Гра починається;
2. Кулька починає рухатися;
3. Якщо натиснуті стрілки вліво або вправо - рухаємо платформу;
4. Якщо кулька торкнулася стінок, стелі або платформи - робимо відскік;
5. Якщо кулька торкнувся платформи - збільшуємо рахунок на одиницю;
6. Якщо кулька впала на підлогу - виводимо повідомлення і закінчуємо гру.

Все це відбувається паралельно і незалежно один від одного. Тобто поки кулька літає, ми цілком можемо рухати платформу, а можемо і залишити її на місці. І коли кулька відскакує від стін, це теж не заважає іншим об'єктам рухатися і взаємодіяти між собою.

Виходить, що нам потрібно визначити три класи - платформу, саму кульку і рахунок, і визначити, як вони реагують на дії один одного. Поле нам самим визначати не потрібно - для цього є вже готова бібліотека. А потім в цих класах ми пропишемо методи - вони якраз і будуть відповідати за поведінку наших об'єктів.

Щоб у нас з'явилася графіка в грі, використовуємо бібліотеку Tkinter.
from tkinter import * - Вона входить в набір стандартних бібліотек Python і дозволяє малювати найпростіші об'єкти - лінії, прямокутники, кола і фарбувати їх в різні кольори. Підключаємо модуль часу- import time для обмеження швидкості платформи, та модуль випадкових чисел-import random.

Щоб створити вікно, де буде видно графіка, використовують клас Tk (). tk = Tk() -він просто робить вікно, але без вмісту. Щоб з'явилося вміст, створюють полотно - видиму частину вікна. Робимо заголовок за допомогою властивості об’єкту tk.title('Game'). За допомогою властивості resizable не

дозволяємо змінювати розміри вікна - tk.resizable(0, 0). Та розташуємо ігрове вікно зверху- tk.wm_attributes('-topmost', 1).

Саме на ньому ми будемо малювати нашу гру. За полотно відповідає клас Canvas (), тому нам потрібно буде створити свій об'єкт з цього класу і далі вже працювати з цим об'єктом- canvas = Canvas(tk, width=500, height=400, highlightthickness=0). canvas.pack() – у кожного елемента гри, які ми бачимо будуть свої координати. tk.update()- команда оновлює вікно.

```
from tkinter import *
import time
import random
tk = Tk()
tk.title('Game')
tk.resizable(0, 0)
tk.wm_attributes('-topmost', 1)
canvas = Canvas(tk, width=500, height=400, highlightthickness=0)
canvas.pack()
tk.update()
```

Кулька повинна вміти:

1. Задавати своє початкове положення і напрямок руху;
2. Розуміти, коли він торкнувся платформи;
3. Малювати сама себе і розуміти, коли потрібно відрендерити себе в новому положенні (наприклад, після відскоку від стіни).

Описуємо клас Ball, який відповідає за кульку.

```
class Ball:
def __init__(self, canvas, paddle, score, color):
self.canvas = canvas
self.paddle = paddle
self.score = score
```

self.id = canvas.create_oval(10,10, 25, 25, fill=color)

Колір потрібен був для того, щоб ми зафарбувати усю кульку. Нова властивість id, в якої зберігається внутрішнє назва кульки а ще командою create_oval створюється коло радіусом 15 пікселів і зафарбовується у потрібні кольори.

Початкова позиція кульки координата (245, 100), список для старту [-2, -1, 1, 2]. Та перемішуємо його, щоб кожний наступний раз кулька з'являлась у іншому місці. Задаємо перший із перемішаного вектор.

```
self.canvas.move(self.id, 245, 100)
starts = [-2, -1, 1, 2]
random.shuffle(starts)
self.x = starts[0]
self.y = -2
self.canvas_height = self.canvas.winfo_height()
self.canvas_width = self.canvas.winfo_width()
self.hit_bottom = False
```

Початковий рух кульки завжди вниз тому зменшуємо його значення по осі y- self.y = -2. При цьому кулька визначає свої координати self.canvas_height = self.canvas.winfo_height() self.canvas_width = self.canvas.winfo_width(). self.hit_bottom = False – властивість яка відповідає досягла чи ні дна кулька.

Створюємо функцію яка відповідає за торкання платформи.

def hit_paddle(self, pos): -координати платформи будемо отримувати через об'єкт – paddle.

Створюємо умови, в яких перевіряємо чи співпадають координати платформи з координатами кульки if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]: if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:

Якщо співпадають - додаємо до рахунку за допомогою команди self.score.hit()

Інакше не зіткнулись.

def hit_paddle(self, pos):

```
            paddle_pos = self.canvas.coords(self.paddle.id)
        if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
            if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
            self.score.hit()
            return True
        return False
```

Створюємо рух кульки за допомогою функції draw. За допомогою self.canvas.move(self.id, self.x, self.y)- пересуваємо кульку на задані координати . Та запам’ятовуємо нові координати кульки. pos = self.canvas.coords(self.id)

Задаємо умови руху кульки.

1.Якщо кулька падає зверху if pos[1] <= 0- задаємо падіння на наступному кроці 2- self.y = 2.

2. Якщо кулька торкнулась правим нижнім кутом дна if pos[3] >= self.canvas_height- помічаємо це в окремій змінній, та виводимо кількість очок.

3. Якщо відбулось торкання платформи-if self.hit_paddle(pos) == True. Кулька відправляється вверх - self.y = -2.

4. Якщо кулька торкнулася лівої стінки if pos[0] <= 0, то кулька повинна рухатися вправо self.x = 2.

5. Якщо кулька торкнулася правої стінки if pos[2] >= self.canvas_width, то кулька повинна рухатися вліво self.x = -2.

```
        def draw(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
            self.y = 2
        if pos[3] >= self.canvas_height:
            self.hit_bottom = True
            canvas.create_text(250, 120, text='Ви програли', font=('Courier', 30),
    fill='red')
```

```
if self.hit_paddle(pos) == True:
    self.y = -2
if pos[0] <= 0:
    self.x = 2
if pos[2] >= self.canvas_width:
    self.x = -2
```

Платформа повинна:

1. рухатися вліво або вправо в залежності від натиснутої стрілки;
2. розуміти, коли гра почалася і можна рухатися.

Намалюємо платформу, для цього створимо клас Paddle, який відповідає за платформи. Клас canvas говорить о том що платформа буде намальована, розміром 100 на 10 пікселів у вибраному кольорі. Стартові позиції [40, 60, 90, 120, 150, 180, 200] за допомогою random.shuffle(start_1) перемішуємо та обираємо першу з перемішаних позицій self.starting_point_x = start_1[0]. Переміщуємо платформу у стартову позицію та завмираємо self.canvas.move(self.id, self.starting_point_x, 300), self.x = 0.

```
class Paddle:
def __init__(self, canvas, color):
    self.canvas = canvas
    self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
    start_1 = [40, 60, 90, 120, 150, 180, 200]
    random.shuffle(start_1)
    self.starting_point_x = start_1[0]
    self.canvas.move(self.id, self.starting_point_x, 300)
    self.x = 0
```

Платформа, як і кулька узнає свою ширину self.canvas_width = self.canvas.winfo_width()

Задаємо обробник натискань:

1. Якщо натиснута стрілка вправо - виконується метод turn_right ().
2. Якщо натиснута стрілка вліво - виконується метод turn_left().

3. Якщо гра не почалася - чекаємо self.started = False.

4. Як тільки гравець натисне клавішу ENTER, гравець починається.

```
self.canvas_width = self.canvas.winfo_width()
self.canvas.bind_all('<KeyPress-Right>', self.turn_right)
self.canvas.bind_all('<KeyPress-Left>', self.turn_left)
self.started = False
self.canvas.bind_all('<KeyPress-Return>', self.start_game)
```

Описуємо рух платформи функціями

Якщо рухаємося вправо – додаємо по 2 пікселя по осі x.

Якщо рухаємося вліво – додаємо по -2 пікселя по осі x.

```
def turn_right(self, event):
self.x = 2
def turn_left(self, event):
self.x = -2
def start_game(self, event):
self.started = True
```

Метод, який відповідає за рух платформи. Ми рухаємо платформу на задану кількість пікселів self.canvas.move(self.id, self.x, 0). Отримуємо координати холста pos = self.canvas.coords(self.id). Перевіряємо межі. Якщо платформа стикнулася з лівою межею – стоп if pos[0] <= 0,self.x = 0. Якщо платформа стикнулася з правою межею – стоп elif pos[2] >= self.canvas_width, self.x = 0.

```
def draw(self):
self.canvas.move(self.id, self.x, 0)
pos = self.canvas.coords(self.id)
if pos[0] <= 0:
```

```
            self.x = 0
        elif pos[2] >= self.canvas_width:
            self.x = 0
```

Від рахунку нам потрібно щоб він правильно реагував на дотик платформи, збільшував число очок і виводив їх на екран:

Опишемо клас Score, який відображує підрахунки. На початку він дорівнюється нулю self.score = 0. Створюємо напис, який показує поточний рахунок, робимо його потрібного кольору і запам'ятовуємо внутрішнє ім'я цього напису self.id = canvas.create_text(450, 10, text=self.score, font=('Courier', 15), fill=color. Створимо функцію торкання платформи def hit(self). Збільшення рахунку на одиницю та записуємо нове значення рахунку.

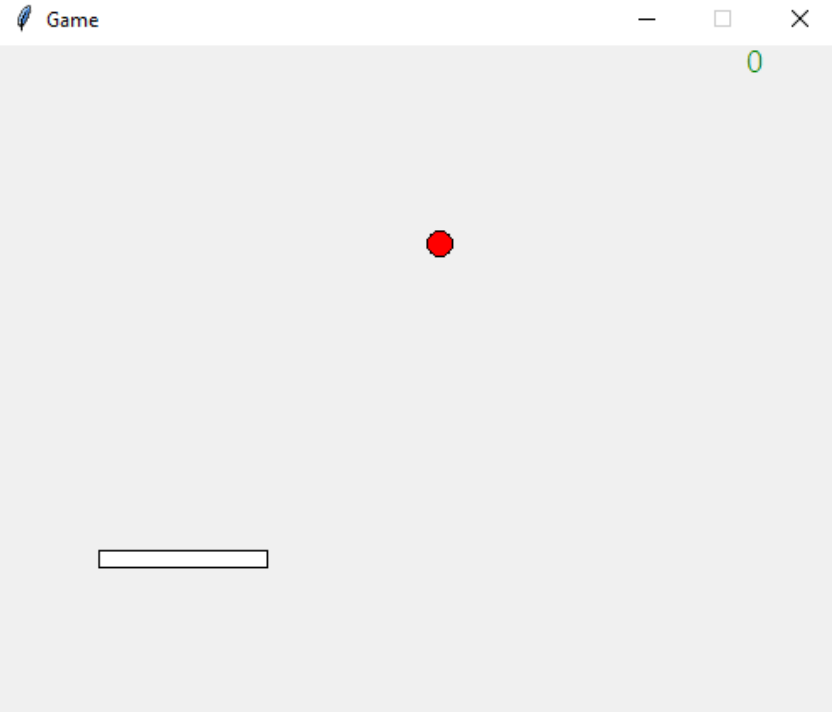
```
class Score:
def __init__(self, canvas, color):
        self.score = 0
        self.canvas = canvas
                self.id = canvas.create_text(450, 10, text=self.score,
      font=('Courier', 15), fill=color
def hit(self):
        self.score += 1
        self.canvas.itemconfig(self.id, text=self.score)
```

Зараз все є для того, щоб почати писати саму гру. Сенс гри в тому, щоб не впустити кульку. Поки цього не сталося - все рухається, але як тільки кулька впала - потрібно показати повідомлення про кінець гри і зупинити програму.

1. Зробимо об’єкт – зелений розрахунок score = Score(canvas, 'green').
2. Зробимо об’єкт – білу платформу paddle = Paddle(canvas, 'White').
3. Зробимо об’єкт - червону кулю ball = Ball(canvas, paddle, score, 'red').
4. Організуємо цикл доки куля не торкнулась дна ми можемо рухати платформу.

```
score = Score(canvas, 'green')
paddle = Paddle(canvas, 'White')
```

```
ball = Ball(canvas, paddle, score, 'red')
while not ball.hit_bottom:
if paddle.started == True:
        ball.draw()
        paddle.draw()
tk.update_idletasks()
tk.update()
time.sleep(0.01)
time.sleep(3)
```



Увесь код програми представлено у додатку 2.

Практичне завдання

1. Розмалювати елементи гри у інші кольори.
2. Змінити розмір кулі та платформи.
3. Змінити швидкість руху платформи.
4. Змінити швидкість руху кульки.
5. Додати другу кульку.

Практична робота №13

Тема. Створення 2D гри на Python 3 з використанням бібліотеки PyGame (Частина 1) .

Мета роботи. Освоїти створення 2D ігри на Python 3 з використанням бібліотеки PyGame .

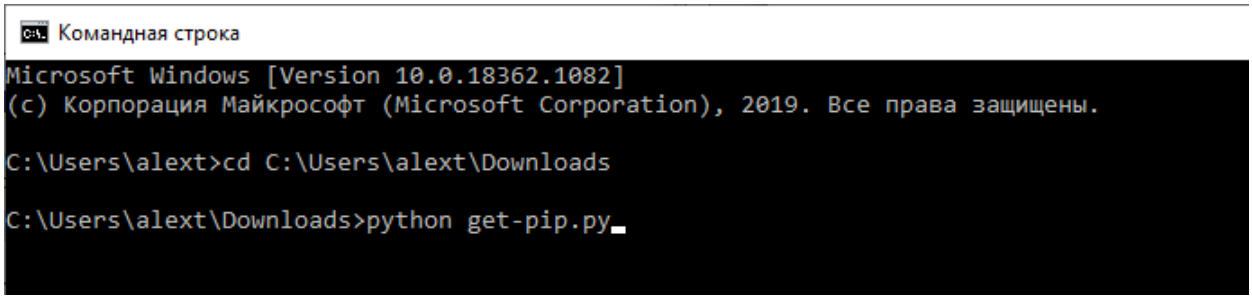
.

Зміст.

4. Вивчення відомостей про бібліотеки PyGame.
5. Виконання роботи.
6. Отримання результату.

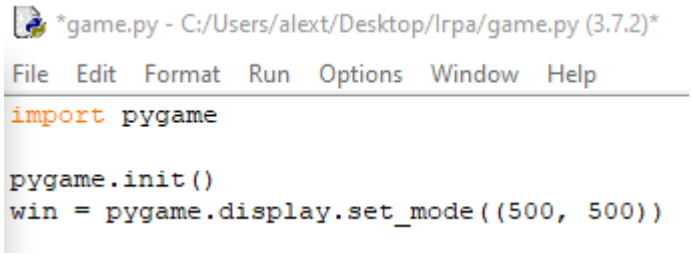
Ключові положення.

Створюємо папку Ігра, потім створимо файл game.py. Відкриваємо командну строку. Далі заходимо на сайт https://bootstrap.pypa.io/get-pip.py та завантажуємо пакетний менеджер. Далі за допомогою командної строки переходимо у місце завантаження get-pip.py та пишемо у командній строці python get-pip.py та натискаємо Enter.



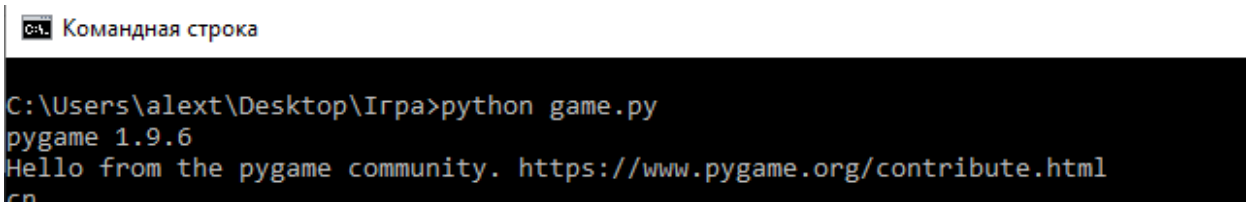
Після завантаження пакетного менеджера, переходимо до папки з нашою майбутньою грою та пишемо у командній строці наступну команду *python -m pip install -U pygame –user* та натискаємо Enter.

Після цього заходимо до game.py та пишемо наступні команди:



Команда import підключає бібліотеку pygame, що ми встановили. Команда *pygame.init()* ініціалізує бібліотеку pygame для нашого використання, потім за допомогою *win = pygame.display.set_mode((500, 500))* ми створюємо вікно розміром 500 на 500 пікселів. Основні команди бібліотеки pygame доступні у додатку 3.

Потім зберігаємо наш файл та запускаємо через командну строку.



pygame.display.set_caption("Game 1") #Додаємо заголовок к вікну.

Задаємо п’ять змінних: розташування, його розмір, швидкість руху.

x = 50 # Розташування по координаті x

y = 50 # Розташування по координаті y

width = 40 # Ширина у пікселях

height = 60 # Довжина у пікселях

speed = 5 # Швидкість

Зробимо цикл для перевірки, як поводить себе користувач - натискає кнопки чи водить курсором.

run = True # Створюємо змінну run та присвоюємо їй значення -True потім перевіряємо її

while run: # потім перевіряємо її та поки вона дорівнює -True буде нескінчений цикл

pygame.time.delay(100) # Час повторення циклу 100мс.

for event in pygame.event.get():# Створюємо цикл який можемо перебирати через змінну event

if event.type == pygame.QUIT:# Якщо натискаємо кнопку то цикл припиняється

run = False

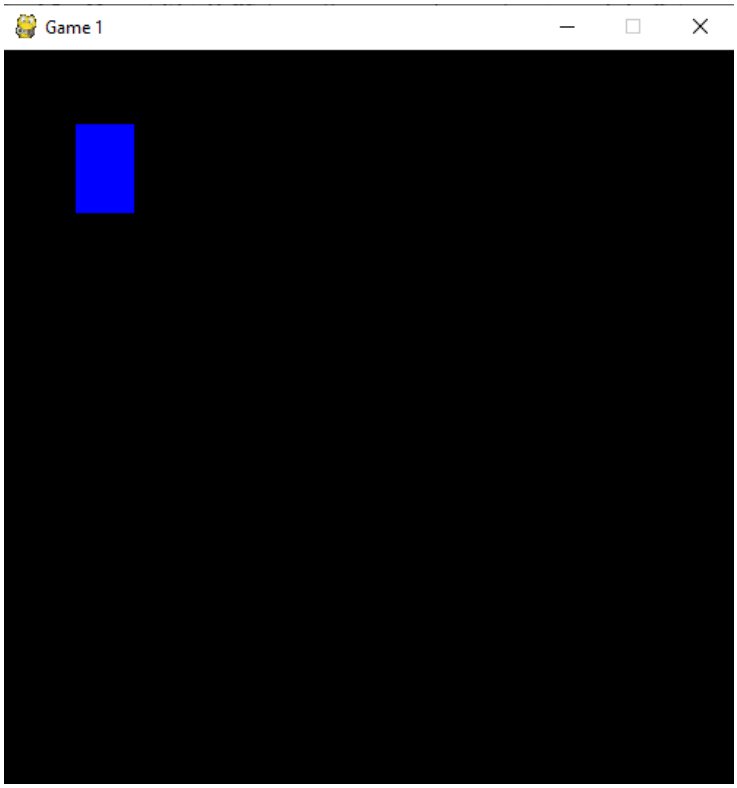
Підключаємо бібліотеку, малюємо квадрат на вікні win, вписуємо колір 0,0,255 - синій (кодування RGB) розташування, висоту та довжину.

pygame.draw.rect(win,(0,0,255),(x,y,width,height))

За допомогою pygame.display.update( ) оновлюємо вікно.

```
import pygame
pygame.init()
win = pygame.display.set_mode((500, 500))

pygame.display.set_caption("Game 1")

x = 50
y = 50
width = 40
height = 60
speed = 5

run = True
while run:
    pygame.time.delay(100)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False

    pygame.draw.rect(win, (0, 0, 225), (x, y, width, height))
    pygame.display.update()

pygame.quit()
```

Збережіть файл та запустіть програму. На екрані з’явиться ваш прямокутник синього кольору, розміром 40 на 60 пікселів.



Навчимося пересувати прямокутник за допомогою натиснення на стрілку. Додамо це після циклу.

Зробимо список кнопок, які будуть пересувати прямокутник.

keys = pygame.key.get_pressed() - клавіша стрілка вліво

x -= speed #від координати **x** при натисканні стрілки віднімається швидкість 5 пікселів при кожному натисканні.

Аналогічно допишемо для інших стрілок

if keys[pygame.K_RIGHT]: клавиша стрілка вправо

x+= speed до координати **x** при натисканні стрілки додається швидкість 5 пікселів при кожному натисканні.

if keys[pygame.K_UP]: клавиша стрілка вверх

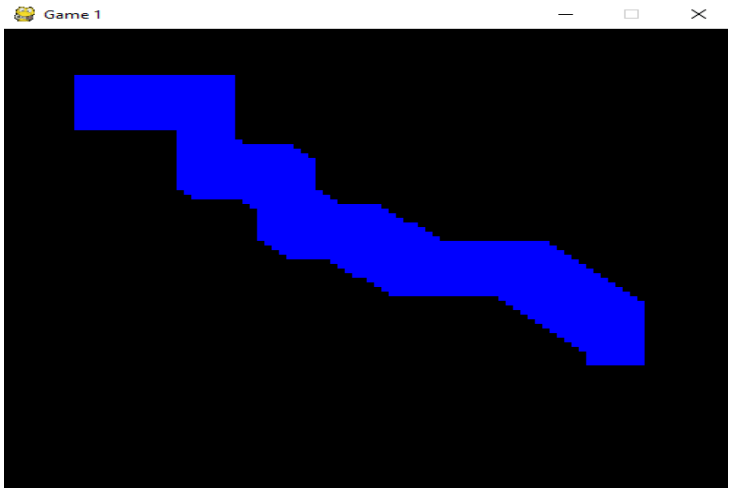
y-= speed від координати **y** при натисканні стрілки віднімається швидкість 5 пікселів при кожному натисканні.

if keys[pygame.K_DOWN]: клавиша стрілка вниз

y+= speed до координати **y** при натисканні стрілки додається швидкість 5 пікселів при кожному натисканні.

```
import pygame
pygame.init()
win = pygame.display.set_mode((500, 500))

pygame.display.set_caption("Game 1")

x = 50
y = 50
width = 40
height = 60
speed = 5

run = True
while run:
    pygame.time.delay(100)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False

    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT]:
        x-= speed
    if keys[pygame.K_RIGHT]:
        x+= speed
    if keys[pygame.K_UP] :
            y-= speed
    if keys[pygame.K_DOWN]
            y+= speed

    pygame.draw.rect(win, (0, 0, 225), (x, y, width, height))
    pygame.display.update()

pygame.quit()
```

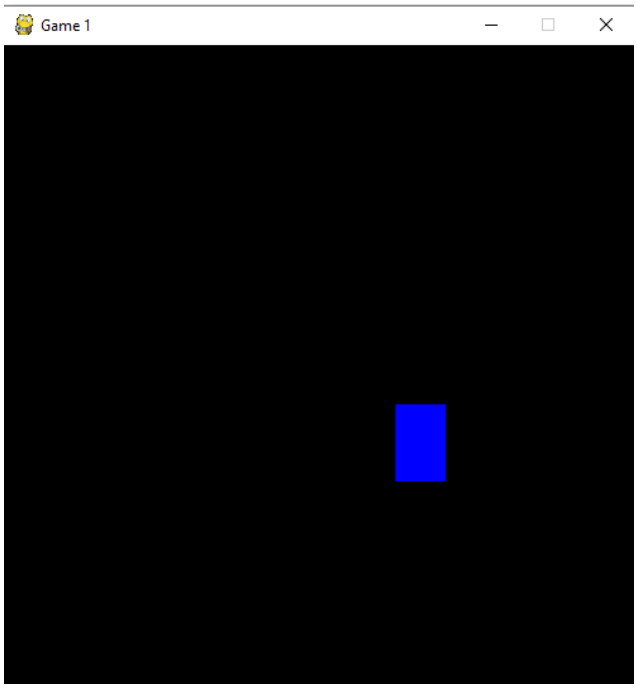
Запускаємо програму та отримаємо прямокутник, якій залишає після себе синій слід.



Для того щоб після руху прямокутника повертався колір вводимо наступну команду

win.fill((0,0,0)) де 0,0,0 - чорний колір екрану.

Збережемо та запустимо програму. Прямокутник не залишає за собою слід. Але прямокутник виходить за межі вікна.



Додаємо границі гри за допомогою команд. Точка відліку знаходиться у верхньому лівому краю вікна для вікна та прямокутника, який рухається.

Додаємо додаткову перевірку у клавіши руху. Усе вікно 500 на 500 пікселів. Прямокутник не зможе дійти 5 пікселів до лівого краю екрану.

if keys[pygame.K_LEFT] and x > 5:

Для правої границі x<500- width - 5 ширини вікна (500 пікселів).

if keys[pygame.K_RIGHT] and x < 500 - width - 5 :

```
if keys[pygame.K_LEFT] and x > 5:
    x-= speed
if keys[pygame.K_RIGHT] and x < 500- width - 5:
    x+= speed
if keys[pygame.K_UP] and y > 5 :
    y-= speed
if keys[pygame.K_DOWN] and y < 500 - height-5:
    y+= speed
```

Змінимо розташування прямокутника. Перемістимо прямокутник у нижній лівий край. Координата по x - залишається, а по *y = 500 - height - 15=425*

```
x = 50
y = 425
width = 40
height = 60
speed = 5
```

Додамо можливість прямокутнику стрибати за допомогою двух змінних. Змінна *isJump* - у неї буде False та вона вказує чи стрибає прямокутник.

Змінна *jumpCount = 10*

```
x = 50
y = 425
width = 40
height = 60
speed = 5

isJump = False
jumpCount = 10
```

Зробимо перевірку - натиснута чи ні клавіша пробіл. Якщо натиснутий пробіл тоді True

if keys[pygame.K_SPACE]:

isJump = True

```
if keys[pygame.K_LEFT] and x > 5:
    x-= speed
if keys[pygame.K_RIGHT] and x < 500- width - 5:
    x+= speed
if keys[pygame.K_UP] and y > 5 :
    y-= speed
if keys[pygame.K_DOWN] and y < 500 - height-5:
    y+= speed
if  keys[pygame.K_SPACE]:
    isJump = True
```

Додаємо швидкості грі. Замість 100 ставимо 50 у часі повторення циклу.

pygame.time.delay(50) # Час повторення циклу 50мс.

Під час стрибка – прямокутник не повинен рухатись вліво, право, або ще раз стрибати. Для цього додаємо перевірку: якщо зараз прямокутник не

стрибає, тоді ми можемо переміщувати прямокутник угору, вниз, вправо, вліво. Якщо ні - прямокутник стрибає.

If not (isJump):

if keys[pygame.K_UP] and y > 5 :

y-= speed

if keys[pygame.K_DOWN] and y < 500 - height-5:

y+= speed

if keys[pygame.K_SPACE]:

isJump = True

else:

```
if keys[pygame.K_LEFT] and x > 5:
    x-= speed
if keys[pygame.K_RIGHT] and x < 500- width - 5:
    x+= speed
if not(isJump):
    if keys[pygame.K_UP] and y > 5 :
        y-= speed
    if keys[pygame.K_DOWN] and y < 500 - height-5:
        y+= speed
    if  keys[pygame.K_SPACE]:
        isJump = True
else:
```

Після цього треба описати, як прямокутник стрибає. Використовуємо змінну *jumpCount = 10*

if jumpCount > = -10: - прямокутник стрибає поки виконується умова

if jumpCount < 0:

*y +=(jumpCount**2)/2* його координата зводиться у ступінь та ділиться на 2. Прямокутник рухається вниз

else:

*y -=(jumpCount**2)/2* - його координата зводиться у ступінь та ділиться на 2. Прямокутник рухається вверх.

jumpCount -= 1 - кожен наступний раз зменшуючись на одиницю

else: - закінчили стрибати

isJump = False

jumpCount = 10

```
if keys[pygame.K_LEFT] and x > 5:
    x-= speed
if keys[pygame.K_RIGHT] and x < 500- width - 5:
    x+= speed
if not(isJump):
    if keys[pygame.K_UP] and y > 5 :
        y-= speed
    if keys[pygame.K_DOWN] and y < 500 - height-5:
        y+= speed
    if  keys[pygame.K_SPACE]:
        isJump = True
else:
    if jumpCount >= -10:
        if jumpCount < 0:
            y +=(jumpCount**2)/2
        else:
            y -=(jumpCount**2)/2
        jumpCount  -= 1
    else:
        isJump = False
        jumpCount = 10
```

Наш прямокутник може стрибати вверх по перевернутій параболі та переміщуватись вліво, право, вверх, вниз.

Увесь код програми представлено у додатку 4.

Практичне завдання

1. Написати програму для рухомого прямокутника розміром №№ на 60 пікселів, Кольору згідно варіанту № у вікні 500+№№ на 500+№№ пікселів. Де № - остання цифра студента у списку групи.

№	Колір стрибаючого прямокутника	Швидкість руху
1	Червоний	50мс
2	Зелений	55мс
3	Синій	56мс
4	Голубий	57мс
5	Фіолетовий	58мс
6	Білий	59мс
7	Помаранчевий	49мс
8	Жовтий	48мс

9	Червоний	47мс
0	Зелений	45мс

Практична робота №14

Тема. Створення 2D гри на Python 3 з використанням бібліотеки PyGame (Частина 2 (Додаємо анімацію)**) .**

Мета роботи. Освоїти створення 2D гри на Python 3 з використанням бібліотеки PyGame.

.

Зміст.

1. Вивчення відомостей про бібліотеку PyGame.
2. Виконання роботи.
3. Отримання результату.

Ключові положення.

Додаємо анімацію. Розпакуємо архів зі спрайтами та додамо їх до папки з game.py.

1. Спершу видаляємо частину програми де прямокутник міг стрибати вверх Рис.1.
2. Створимо три змінні, в яких будемо перевіряти напрям руху гравця та на якому спрайті він находиться. (Вони у нас у папці пронумеровані) Рис.2. *left = False, right = False animCount = 0.* У початку гравець не рухається.
3. Завантажимо зображення гравця в гру. Рис.3 Задаємо список *walkright* у якому гравець повертає направо, список *walkleft* у якому гравець повертає наліво. І він стоїть прямо - змінна *playerStand.*
4. Змініть розміри ігрока 60 на 71. Це розмір картинок.

```
clock = pygame.time.Clock()

x = 50
y = 425
width = 60
height = 71
speed = 5
```

5. Завантажимо задній фон у картинці. Змінна *bg = pygame.image.load('bj.jpg')*

```
walkright = [pygame.image.load('right_1.png'),
pygame.image.load('right_2.png'),pygame.image.load('right_3.png'),
pygame.image.load('right_4.png'),pygame.image.load('right_5.png'),
pygame.image.load('right_6.png')]

walkleft = [pygame.image.load('left_1.png'),pygame.image.load('left_2.png')
pygame.image.load('left_3.png'),pygame.image.load('left_4.png'),
pygame.image.load('left_5.png'),pygame.image.load('left_6.png')]

bg = pygame.image.load('bg.jpg')
playerStand = pygame.image.load('idle.png')
```

6. Видаляємо зайві цикли Рис.4.

7. Зробимо нову функцію, яку кожен раз будемо викликати з циклу.

```
left = False
right = False
animCount = 0

def drawWindow():
    global animCount
    win.blit(bg, (0,0))
    pygame.draw.rect(win, (0,0,255), (x, y, width, height))
    pygame.display.update()
```

8. Викликаємо змінну у кінці циклу

```
        else:
            isJump = False
            jumpCount = 10

    drawWindow()

pygame.quit()                    # Закриваємо цикл
```

9. Додаємо до циклу *left = True, right = False.*

```
    keys = pygame.key.get_pressed()  #
    if keys[pygame.K_LEFT] and x > 5:
        x-= speed
        left = True
        right = False
```

10. Відповідно додаємо

```
elif keys[pygame.K_RIGHT] and x < 500- width - 5:
    x+= speed
    left = False
    right = True
```

11. Вводимо ще одну умову рис.5. Уся анімація буде прокручуватись з нуля. *animCount = 0*.

12. Зараз ми маємо не прямокутник, а гравеця, тому видаляємо наступне:

```
def drawWindow():
    global animCount
    win.blit(bg, (0,0))
    pygame.draw.rect(win, (0,0,255), (x, y, width, height))
    pygame.display.update()
```

та пишемо перевірки *if animCount +1 >= 30* - число кадрів у секунду, тоді *animCount = 0*. Всього у нас у кожному списку walkright по 6 кадров. Кожний спрайт буде проганятися по 5 кадрів.

```
def drawWindow():
    global animCount
    if animCount + 1 >= 30:
        animCount = 0

    win.blit(bg, (0,0))
    pygame.display.update()
```

13. Переносимо задній фон *win.blit(bg, (0,0))* після *global animCount*. Починаємо анімувати персонажа *if left:* тоді починається запуск гравеця, який йде наліво *win.blit(walkLeft[animCount // 5],(x,y))* - з координатами.

Теж саме прописуємо для правої сторони elif right:

win.blit(walkRight[animCount // 5],(x,y)). Кожен раз додаємо одиницю до *animCount += 1*. Якщо гравець не рухається прописуємо координати та гравеця нерухомого.

```
def drawWindow():
    global animCount

    win.blit(bg, (0,0))
    if animCount + 1 >= 30:
       animCount = 0

    if left:
        win.blit(walkleft[animCount // 5],(x,y))
        animCount += 1
    elif right:
        win.blit(walkright[animCount // 5],(x,y))
        animCount += 1
    else:
        win.blit(playerStand, (x, y))


    pygame.display.update()
```

14. Запишемо час повторення циклу. Для цього введемо нову змінну *clock = pygame.time.Clock()*

```
walkleft = [pygame.image.load('left_1.png'),pygame.image.load('left_2.png')
pygame.image.load('left_3.png'),pygame.image.load('left_4.png'),
pygame.image.load('left_5.png'),pygame.image.load('left_6.png')]

bg = pygame.image.load('bg.jpg')
playerStand = pygame.image.load('idle.png')

clock = pygame.time.Clock()
```

Потім у циклі *while run:* вписати змінну та вказати скільки фреймів повинно бути. У нас 30.

```
run = True
while run:
    clock.tick(30)
    for event in pygame.event.get():
```

15. Зберігаємо та запускаємо програму. Рис.6.

game1.py - C:/Users/alext/Desktop/lrpa/game1.py (3.7.2)

File Edit Format Run Options Window Help

```
import pygame
pygame.init()
win = pygame.display.set_mode((500, 500))

pygame.display.set_caption("Game 1")

x = 50
y = 425
width = 40
height = 60
speed = 5

isJump = False
jumpCount = 10

run = True
while run:
    pygame.time.delay(50)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT] and x > 5:
        x-= speed
    if keys[pygame.K_RIGHT] and x < 500- width - 5:
        x+= speed
    if not(isJump):
        if keys[pygame.K_UP] and y > 5 :
            y-= speed
        if keys[pygame.K_DOWN] and y < 500 - height-5:
            y+= speed
        if  keys[pygame.K_SPACE]:
            isJump = True
    else:
        if jumpCount >= -10:
            if jumpCount < 0:
                y +=(jumpCount**2)/2
            else:
```

Рис.1

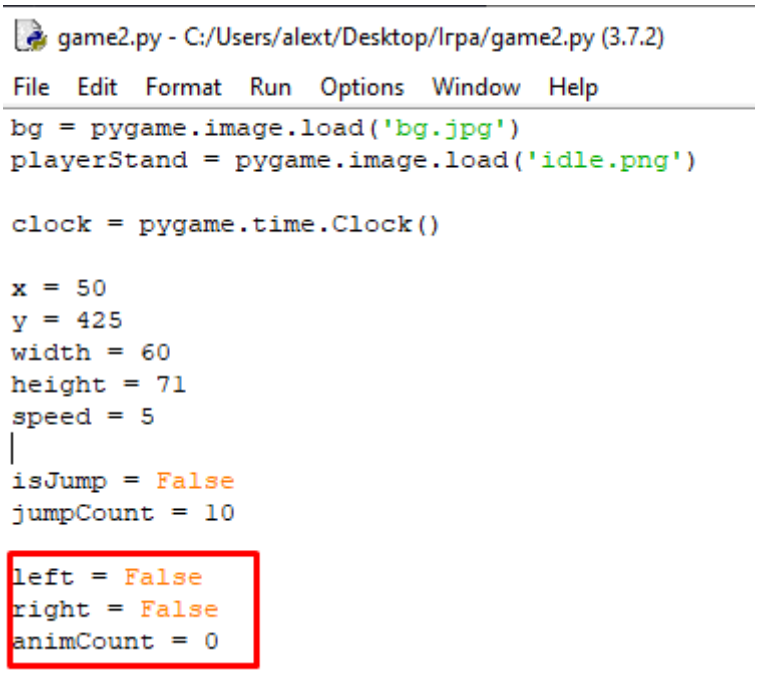

game2.py - C:/Users/alext/Desktop/lrpa/game2.py (3.7.2)
File Edit Format Run Options Window Help
bg = pygame.image.load('bg.jpg')
playerStand = pygame.image.load('idle.png')
clock = pygame.time.Clock()
x = 50
y = 425
width = 60
height = 71
speed = 5
isJump = False
jumpCount = 10
left = False
right = False
animCount = 0

Рис.2

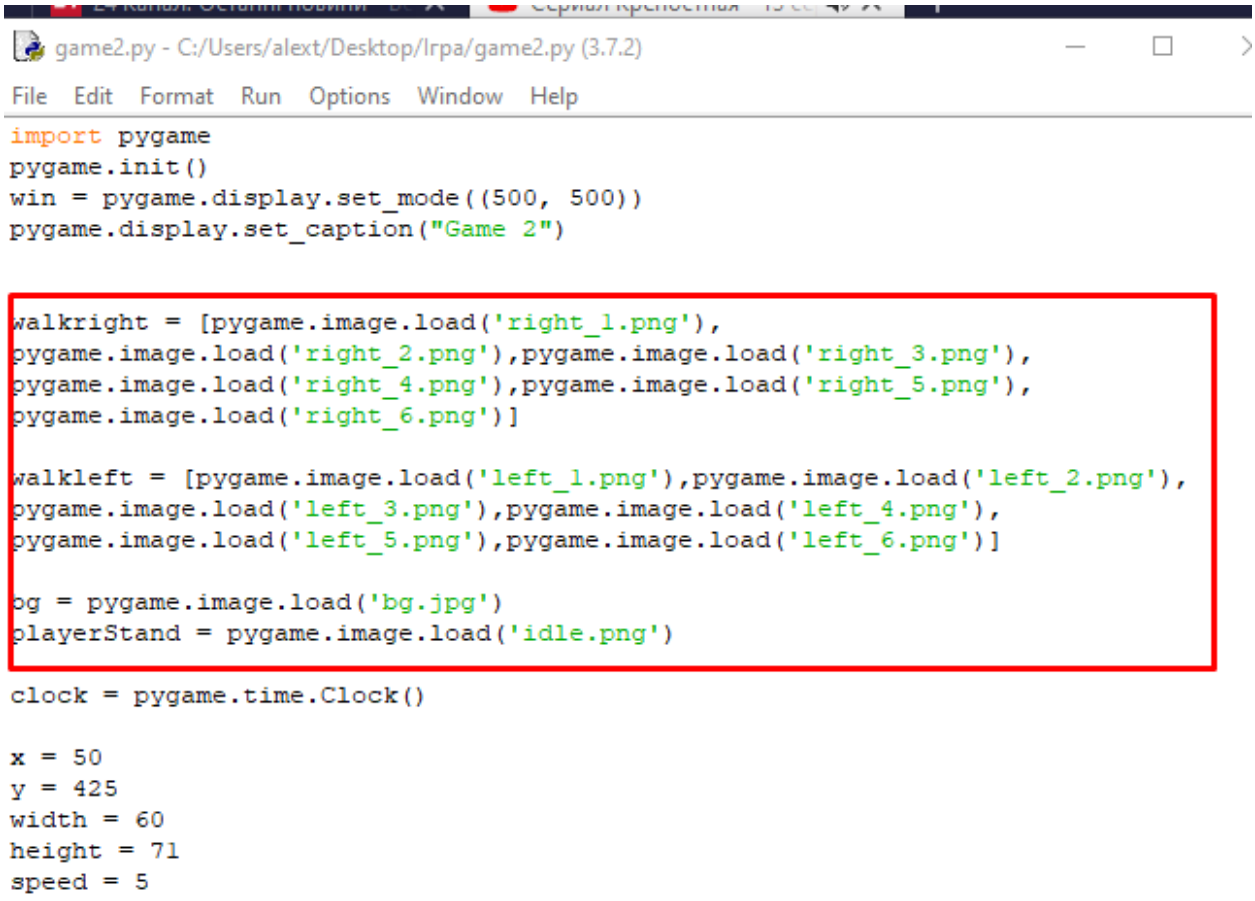

game2.py - C:/Users/alext/Desktop/lrpa/game2.py (3.7.2)
File Edit Format Run Options Window Help
import pygame
pygame.init()
win = pygame.display.set_mode((500, 500))
pygame.display.set_caption("Game 2")
walkright = [pygame.image.load('right_1.png'),
pygame.image.load('right_2.png'),pygame.image.load('right_3.png'),
pygame.image.load('right_4.png'),pygame.image.load('right_5.png'),
pygame.image.load('right_6.png')]
walkleft = [pygame.image.load('left_1.png'),pygame.image.load('left_2.png'),
pygame.image.load('left_3.png'),pygame.image.load('left_4.png'),
pygame.image.load('left_5.png'),pygame.image.load('left_6.png')]
bg = pygame.image.load('bg.jpg')
playerStand = pygame.image.load('idle.png')
clock = pygame.time.Clock()
x = 50
y = 425
width = 60
height = 71
speed = 5

Рис.3

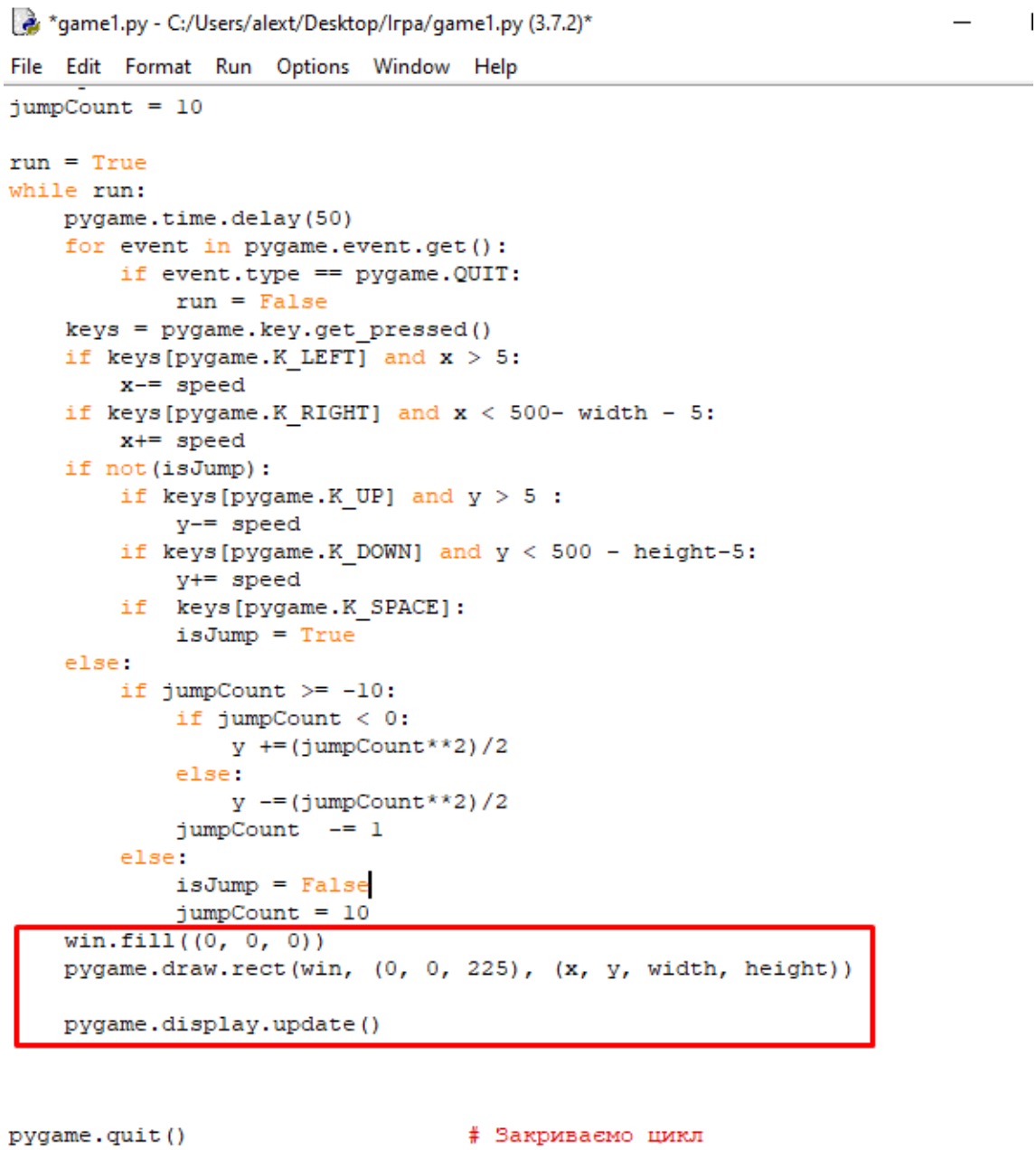


Рис.4

```
            keys = pygame.key.get_pressed()  # Робимо список усі
            if keys[pygame.K_LEFT] and x > 5:
                x-= speed
                left = True
                right = False
            elif keys[pygame.K_RIGHT] and x < 500- width - 5:
                x+= speed
                left = False
                right = True
            else:
                left = False
                right = False
                animCount = 0
            if not(isJump):
```

Рис.5

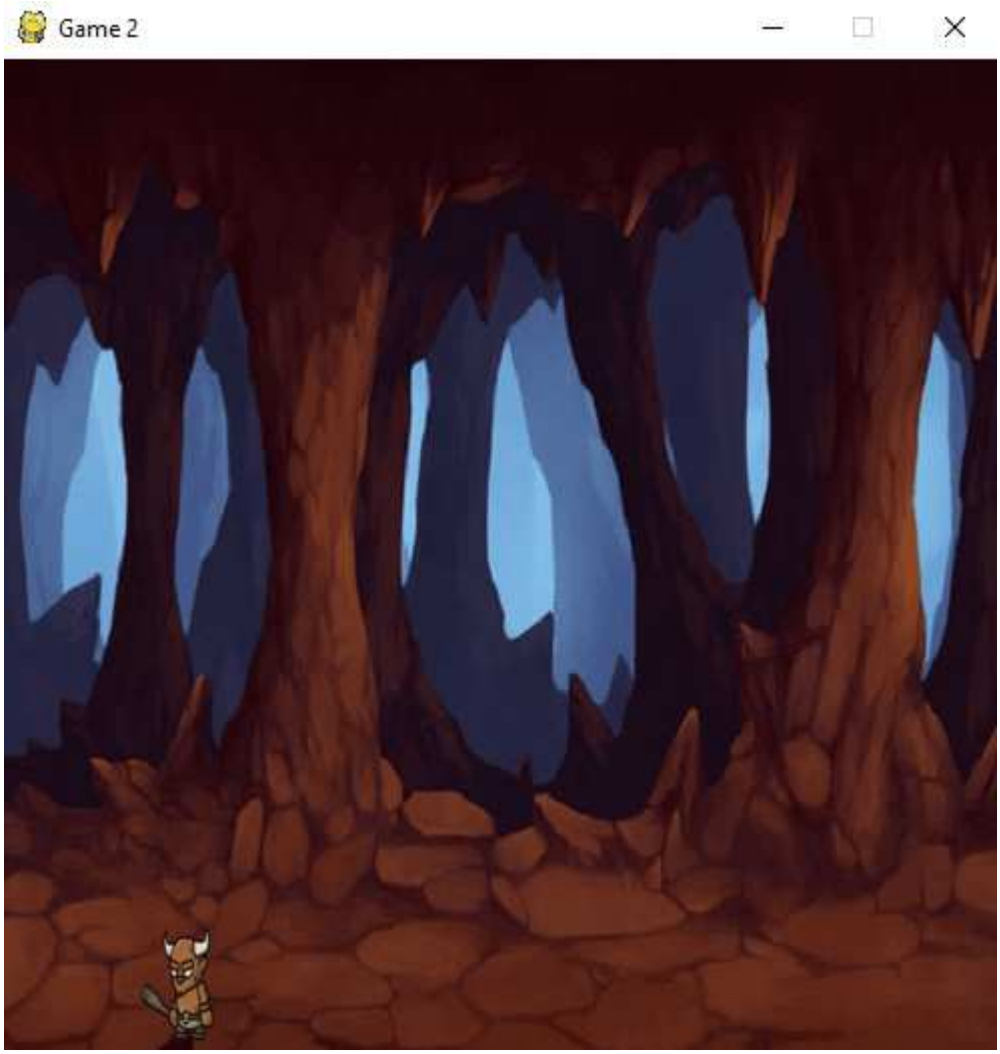


Рис.6.

Увесь код програми представлено у додатку 5.

Практичне завдання.

1. Написати програму з довільними спрайтами.
2. Змінити задній фон гри.

Практична робота №15

Тема. Створення 2D гри на Python 3 з використанням бібліотеки PyGame (Частина 3 (Додаємо анімацію - стрільбу)**) .**

Мета роботи. Освоїти створення 2D гри на Python 3 з використанням бібліотеки PyGame .

.

Зміст.

1. Вивчення відомостей про бібліотеки PyGame.
2. Виконання роботи.
3. Отримання результату.

Ключові положення.

Навчимо нашого персонажу стріляти невеликими снарядами, для чого зробимо клас, на основі якого будуть створюватись об’єкти і кожен об’єкт буде окремим ядром - снаряд (ми його намалюємо), який буде випускати наш користувач.

Створимо клас **class snaryad()** під нашими змінними. Прописуємо конструктор *def__init__(self, x, y, radius, color, facing):* - при створенні об’єкта потрібного класу, позицію об’єкта, радіус об’єкта, його колір, напрям руху снаряду. Якщо facing = 1 снаряд летить вправо, якщо -1 снаряд летить вліво. За допомогою self.x = x присвоюємо значення змінних які відносяться к класу **snaryad.** Змінна **vel** відповідає за швидкість снаряду.

```
isJump = False
jumpCount = 10

left = False
right = False
animCount = 0
lastMove =  'right'

class snaryad(object):
    def __init__(self, x, y, radius, color, facing):
        self.x = x
        self.y = y
        self.radius = radius
        self.color = color
        self.facing = facing
        self.vel = 8 * facing
```

Запишемо функцію, яка буде створювати снаряди *def draw(self, win)*. Де win- вікно, де ми створюємо снаряд. У *pygame.draw.circle(win, self.color,(self.x, self.y),self.radius)* - використовується форма, колір, позиція, радіус снаряду.

```
class snaryad(object):
    def __init__(self, x, y, radius, color, facing):
        self.x = x
        self.y = y
        self.radius = radius
        self.color = color
        self.facing = facing
        self.vel = 8 * facing
    def draw(self, win):
        pygame.draw.circle(win, self.color,(self.x, self.y),self.radius)
```

Створимо пустий список bullets = [] - список об’єктів класу **snaryad**:

```
    pygame.display.update()

run = True
bullets = []
while run:
    clock.tick(30)


    for event in pygame.event.get():
```

Створимо новий цикл, який буде відповідати за перебір усіх елементів списку bullets у кожного об’єкта якого буде: форма, колір, позиція, радіус снаряду.

```
while run:
    clock.tick(30)

    for event in pygame.event.get():

        if event.type == pygame.QUIT:
            run = False

    for bullet in bullets:
        if bullet.x < 500 and bullet.x > 0:
           bullet.x += bullet.vel
        else:
            bullets.pop(bullets.index(bullet))

    keys = pygame.key.get_pressed()
```

Починаємо створювати снаряди та випускати. Для цього введемо перевірку на натиснення клавіши **f** та будемо випускати снаряд - i*f keys[pygame.K_f]:*.

if len(bullets) < 5: # довжина списку снарядів - 5. Та *bullets.append(snaryad(round(x + width//2),round(y + height //2), 5, (255,0,0), facing))* . Де *round(x + width//2),round(y + height //2* – позиція. Число 5 – радіус. Колір червоний - 255,0,0 та змінна *facing*.

Додамо змінну *lastMove = 'right'*

```
x = 50
y = 425
width = 60
height = 71
speed = 5

isJump = False
jumpCount = 10

left = False
right = False
animCount = 0
lastMove =  'right'
```

Перевіряємо у циклу змінну *lastMove*. Якщо вона рухається вправо змінна *facing= 1* , якщо ні *facing=-1*.

```
for bullet in bullets:
    if bullet.x < 500 and bullet.x > 0:
       bullet.x += bullet.vel
    else:
        bullets.pop(bullets.index(bullet))

keys = pygame.key.get_pressed()
if keys[pygame.K_f]:
    if lastMove == 'right':
        facing = 1
    else:
        facing = -1

    if len(bullets) < 5:
        bullets.append(snaryad(round(x + width // 2), round(y + height // 2), 5, (255,0,0), facing))
```

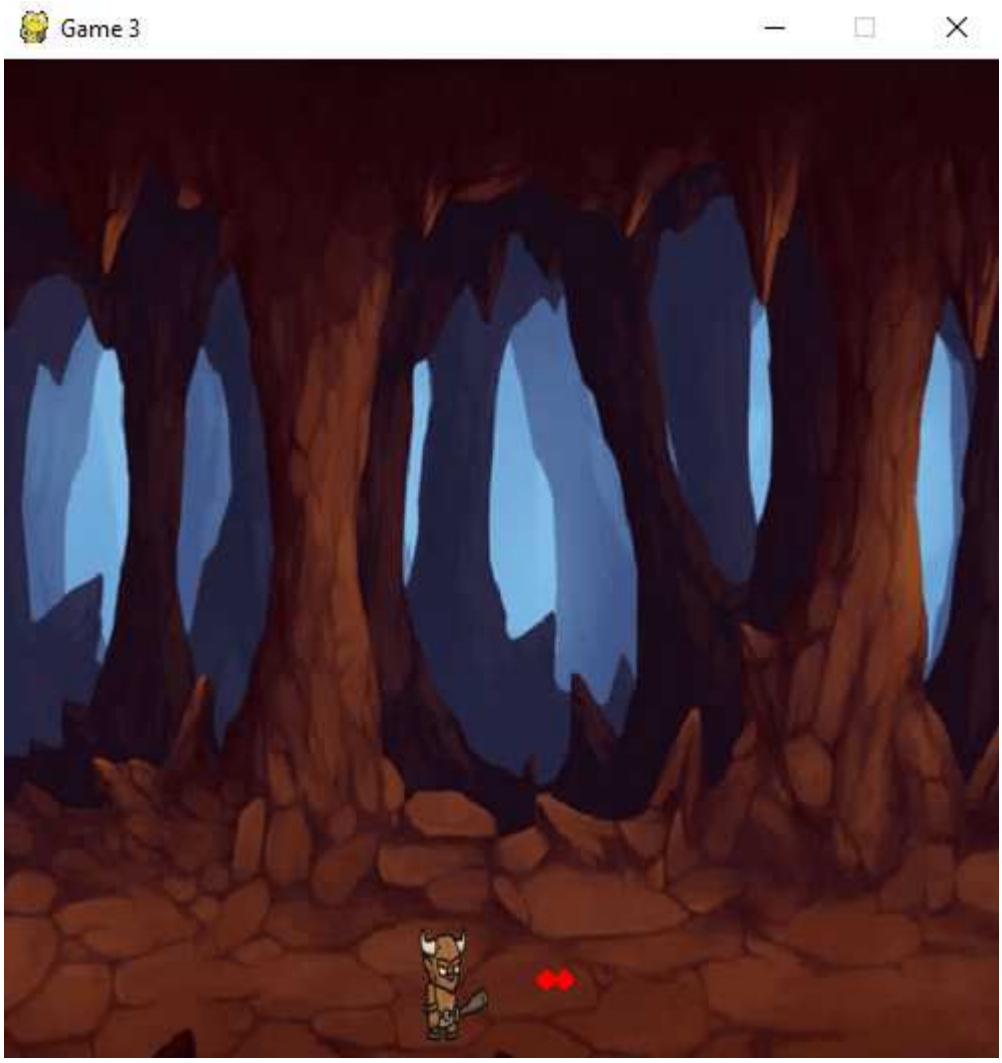
Змінюємо значення змінної **lastMove** при натисканні на стрілку. Якщо гравець не рухається, за допомогою змінної **lastMove** ми будемо знати куди гравець рухався у останній раз – вправо чи вліво. У цьому ж напряму буде рухатись наш снаряд.

```
if keys[pygame.K_LEFT] and x > 5:
    x-= speed
    left = True
    right = False
    lastMove = 'left'
elif keys[pygame.K_RIGHT] and x < 500- width - 5:
    x+= speed
    left = False
    right = True
    lastMove = 'right'
else:
    left = False
    right = False
    animCount = 0
```

Намалюємо снаряд. Для цього повертаємось до функції *drawWindow()*. Тут за допомогою цикла ми перебираємо масив bullets та застосовуємо *draw* , який вже прописано.

```
if left:
    win.blit(walkleft[animCount // 5],(x,y))
    animCount += 1
elif right:
    win.blit(walkright[animCount // 5],(x,y))
    animCount += 1
else:
    win.blit(playerStand, (x, y))
for bullet in bullets:
    bullet.draw(win)
```

Запускаємо гру та бачимо, що наш гравець вже стріляє снарядами.



Код програми представлено у додатку 6.

Практичне завдання.

1. Написати програму у якій гравець з практичної роботи 14 буде стріляти снарядами відповідного кольору та розміру. Де №- остання цифра студента у списку групи.

№	Колір снаряду	Розмір снаряду	Кількість снарядів
1	Червоний	3	5
2	Зелений	3	6
3	Синій	4	7
4	Голубий	4	8
5	Фіолетовий	5	9
6	Білий	5	10

7	Помаранчевий	6	3
8	Жовтий	4	5
9	Червоний	6	7
0	Зелений	7	9

Практична робота №16

Тема. Створення 2D гри на Python 3 з використанням бібліотеки PyGame (Частина 4 (Додаємо анімацію - ворогів, яких будемо знищувати)**).**

Мета роботи. Освоїти створення 2D гри на Python 3 з використанням бібліотеки PyGame.

Зміст.

7. Вивчення відомостей про бібліотеки PyGame.
8. Виконання роботи.
9. Отримання результату.

Ключові положення.

Створимо клас enemy. Задаємо список *walkRight* у якому enemy повертає направо, список *walkLeft* у якому enemy повертає наліво.

```
class enemy(object):
    walkRight = [pygame.image.load('R1E.png'), pygame.image.load('R2E.png'),
                 pygame.image.load('R3E.png'), pygame.image.load('R4E.png'),
                 pygame.image.load('R5E.png'), pygame.image.load('R6E.png'),
                 pygame.image.load('R7E.png'), pygame.image.load('R8E.png'),
                 pygame.image.load('R9E.png'), pygame.image.load('R10E.png'),
                 pygame.image.load('R11E.png')]
```

```
walkLeft = [pygame.image.load('L1E.png'),
pygame.image.load('L2E.png'),
        pygame.image.load('L3E.png'), pygame.image.load('L4E.png'),
        pygame.image.load('L5E.png'), pygame.image.load('L6E.png'),
        pygame.image.load('L7E.png'), pygame.image.load('L8E.png'),
        pygame.image.load('L9E.png'), pygame.image.load('L10E.png'),
        pygame.image.load('L11E.png')]
```

Прописуємо конструктор *def__init__(self, x, y, width, height, end):* - при створенні об’єкта потрібного класу - позицію об’єкта, його габарити, кінцеву координату x до якої дійде об’єкт. Змінна **vel** відповідає за швидкість.

```
def __init__(self, x, y, width, height, end):
    self.x = x
    self.y = y
    self.width = width
    self.height = height
    self.path = [x, end]
    self.walkCount = 0
    self.vel = 3
```

Створимо функцію draw яка відповідає за анімацію ворога. Де 33- число кадрів у секунду. Ворог рухається наліво та направо.

```
def draw(self, win):
    self.move()
    if self.walkCount + 1 >= 33:
        self.walkCount = 0

    if self.vel > 0:
        win.blit(self.walkRight[self.walkCount//3], (self.x,self.y))
        self.walkCount += 1
    else:
        win.blit(self.walkLeft[self.walkCount//3], (self.x,self.y))
```

```
self.walkCount += 1
```

Прописуємо функцію руху. Якщо швидкість відносно осі x >0 у даний момент часу, тоді йде наступна перевірка: якщо координата менша за суму кінцевої точки, що ми задали (400), та швидкості ворога, тоді до координати додається швидкість (3). Інакше швидкість помножується на -1, додається до координати x та анімація починається спочатку.

```
def move(self):
    if self.vel > 0:
        if self.x < self.path[1] + self.vel:
            self.x += self.vel
        else:
            self.vel = self.vel * -1
            self.x += self.vel
            self.walkCount = 0
```

Перевірка - якщо швидкість відносно осі x < 0 у даний момент часу, тоді йде наступна перевірка: якщо координата більша за різницю кінцевої точки, що ми задали (400), та швидкості ворога, тоді до координати додається швидкість (3). Інакше швидкість помножується на -1, додається до координати x та анімація починається спочатку.

```
    else:
        if self.x > self.path[0] - self.vel:
            self.x += self.vel
        else:
            self.vel = self.vel * -1
            self.x += self.vel
            self.walkCount = 0
```

Додаємо наступний код до нашої програми:

```
def drawWindow():
    global animCount

    win.blit(bg, (0,0))
    if animCount + 1 >= 30:
       animCount = 0

    if left:
        win.blit(walkleft[animCount // 5],(x,y))
        animCount += 1
    elif right:
        win.blit(walkright[animCount // 5],(x,y))
        animCount += 1
    else:
        win.blit(playerStand, (x, y))
    for bullet in bullets:
        bullet.draw(win)
    goblin.draw(win)

    pygame.display.update()

goblin = enemy(100, 436, 64, 64, 400)
run = True
bullets = []
while run:
    clock.tick(30)
```

Додаємо ситуацію - колізію перетинання коробочками в яких знаходяться наші персонажі.

Додаємо hitbox до класу enemy:

```
    def __init__(self, x, y, width, height, end):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.path = [x, end]
        self.walkCount = 0
        self.vel = 3
        self.hitbox = (self.x + 20, self.y, 28, 60)
```

До функції draw додаємо функцію для відображення hitbox для ворогів.

self.hitbox = (self.x + 15, self.y, 28, 60)- використовується для оновлення self.hitbox.

```
    def draw(self, win):
        self.move()
        if self.walkCount + 1 >= 33:
            self.walkCount = 0

        if self.vel > 0:
            win.blit(self.walkRight[self.walkCount//3], (self.x,self.y))
            self.walkCount += 1
        else:
            win.blit(self.walkLeft[self.walkCount//3], (self.x,self.y))
            self.walkCount += 1
        self.hitbox = (self.x + 15, self.y, 28, 60)
        pygame.draw.rect(win, (255,0,0), self.hitbox, 2)
```

Додаємо змінну hitbox для гравця.

```
x = 50
y = 425
width = 60
height = 71
speed = 5

isJump = False
jumpCount = 10

left = False
right = False
animCount = 0
hitbox = (x + 20, y, 28, 60)
lastMove =  'right'
```

До функції draw додаємо функцію для відображення hitbox для гравця.

hitbox = (x + 10, y, 40, 80) - використовується для оновлення hitbox.

```
def drawWindow():
    global animCount

    win.blit(bg, (0,0))
    if animCount + 1 >= 30:
       animCount = 0

    if left:
        win.blit(walkleft[animCount // 5],(x,y))
        animCount += 1
    elif right:
        win.blit(walkright[animCount // 5],(x,y))
        animCount += 1
    else:
        win.blit(playerStand, (x, y))
    for bullet in bullets:
        bullet.draw(win)
    goblin.draw(win)
    hitbox = (x + 10, y, 40, 80)
    pygame.draw.rect(win, (255,0,0), hitbox, 2)
```

Створюємо функцію hit у класі ворога та прописуємо наступні функції

```
        else:
            if self.x > self.path[0] - self.vel:
                self.x += self.vel
            else:
                self.vel = self.vel * -1
                self.x += self.vel
                self.walkCount = 0
    def hit(self):
        print('Попадание')
```

Напишемо код, що буде виконувати наступну перевірку: якщо найвища та найнижча точки снаряду знаходяться у хіт боксі ворога, тоді перевіряються

чи крайня ліва та крайня права точки знаходяться у хіт боксі. Якщо так, викликається функція hit, що ми створили, та снаряд видаляється.

```
goblin = enemy(100, 436, 64, 64, 400)
run = True
bullets = []
while run:
    clock.tick(30)

    for event in pygame.event.get():

        if event.type == pygame.QUIT:
            run = False

    for bullet in bullets:
        if bullet.y - bullet.radius < goblin.hitbox[1] + goblin.hitbox[3] and bullet.y + bullet.radius > goblin.hitbox[1]:
            if bullet.x + bullet.radius > goblin.hitbox[0] and bullet.x - bullet.radius < goblin.hitbox[1] + goblin.hitbox[3]:
                goblin.hit()
                bullets.pop(bullets.index(bullet))

        if bullet.x < 500 and bullet.x > 0:
            bullet.x += bullet.vel
        else:
            bullets.pop(bullets.index(bullet))
```

Для виправлення стрільби чергами створимо змінну shootloop = 0.

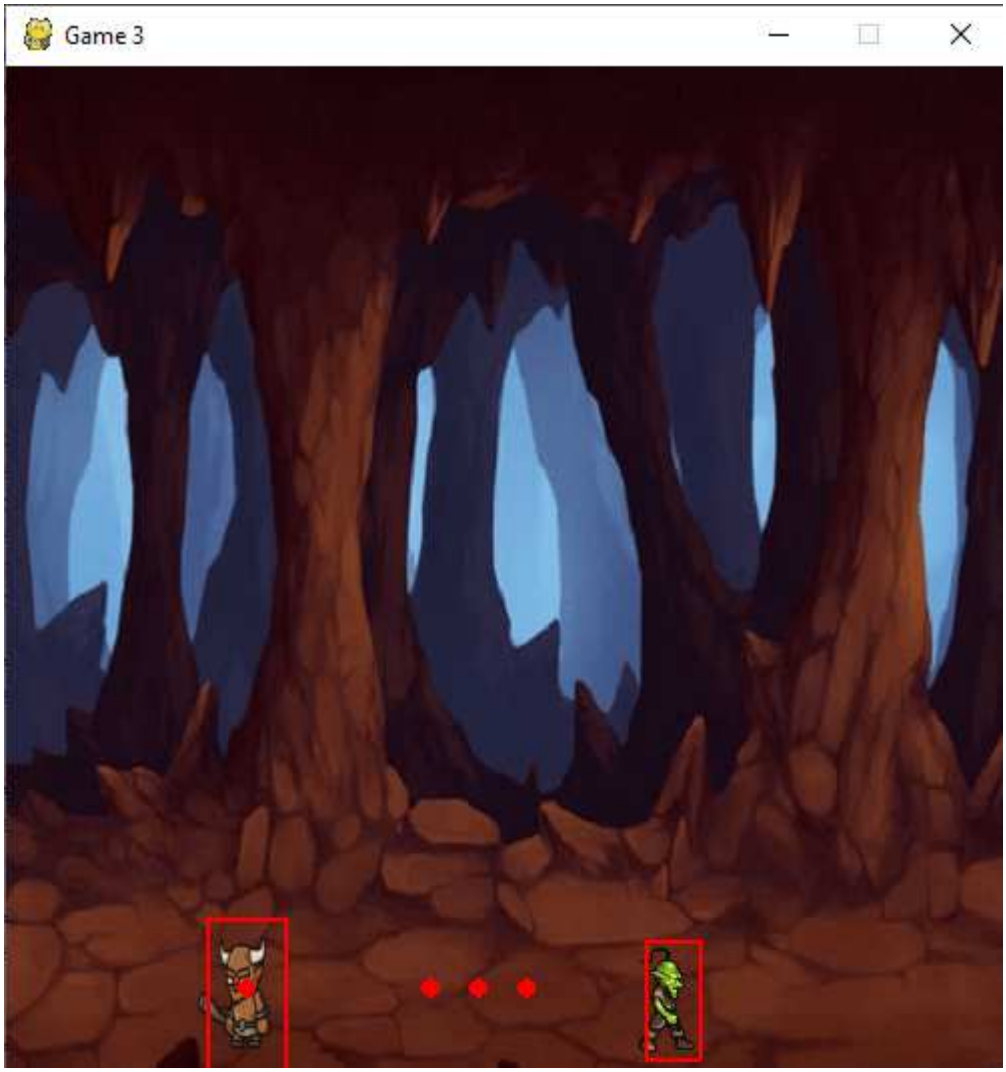
Додамо дві перевірки: якщо shootloop більша за нуль, тоді до змінна збільшується на один, та якщо shootloop більша за три, тоді їй присвоюється значення нуль.

```
goblin = enemy(100, 436, 64, 64, 400)
run = True
shootloop = 0
bullets = []
while run:
    clock.tick(30)

    if shootloop > 0:
        shootloop +=1
    if shootloop > 3:
        shootloop = 0
```

Додамо до перевірки натиснення на кнопку f, що відповідає за стрільбу, умову shootloop == 0. Також після натискання на кнопку f будемо присвоювати shootloop = 1.

```
keys = pygame.key.get_pressed()
if keys[pygame.K_f] and shootloop == 0:
    if lastMove == 'right':
        facing = 1
    else:
        facing = -1

    if len(bullets) < 5:
        bullets.append(snaryad(round(x + width // 2), round(y + height // 2)
    shootloop = 1
```

Результат програми



Увесь код програми представлено у додатку 7.

Практичне завдання.

1. Вибрати відповідні спрайти для ворога. Та написати програму, використовуючи ці спрайти. Ворог повинен рухатись на усю ширину вікна.
2. Створити hitbox для ігрока та ворога, відповідно до їх розмірів. Та налаштувати підрахування вистрілів.

Практична робота №17

Тема. Створення 2D гри на Python 3 з використанням бібліотеки PyGame (Частина 5).

Мета роботи. Освоїти створення 2D гри на Python 3 з використанням бібліотеки PyGame.

Зміст.

10. Вивчення відомостей про бібліотеки PyGame.
11. Виконання роботи.
12. Отримання результату.

Ключові положення.

Додамо змінну score = 0 на початку програми.

```
import pygame
pygame.init()
win = pygame.display.set_mode((500, 500))
pygame.display.set_caption("Game 3")

score = 0
```

При попаданні у ворога змінна score збільшується на одиницю.

```
for bullet in bullets:
    if bullet.y - bullet.radius < goblin.hitbox[1] + goblin.hitbox[3] and
        if bullet.x + bullet.radius > goblin.hitbox[0] and bullet.x - bul
            goblin.hit()
            score += 1
            bullets.pop(bullets.index(bullet))
```

Якщо треба відображати score на екрані, тоді нам треба спершу прописати шрифт у змінній font .

```
font = pygame.font.SysFont("comicsans", 30, True, True)
goblin = enemy(100, 436, 64, 64, 400)
run = True
shootloop = 0
bullets = []
```

Змінній font надаємо потрібний нам шрифт за допомогою вбудованої у pygame команди SysFont, у якій перший аргумент - тип шрифту(Comic Sans MS), другий – розмір шрифту, третій – жирний шрифт, четвертий – курсив.

Створимо змінну text та за допомогою font.render() будемо відображати текст, використовуючи шрифт, що ми задали у змінній font. У даному випадку font.render() може взяти 3 параметра, де перший – текст, що ми хочемо відобразити, другий вмикає зглажування(робить текст більш детальним), третій визначає колір.

```
    for bullet in bullets:
        bullet.draw(win)
    goblin.draw(win)
    hitbox = (x + 10, y, 40, 80)
    pygame.draw.rect(win, (255,0,0), hitbox, 2)
    text = font.render("Рахунок: " + str(score), 1, (0,0,0))
    win.blit(text, (350, 10))

    pygame.display.update()
```

Далі за допомогою команди win.blit() відображаємо text, та вписуємо координати нашого тексту.

Закоментуємо команди, що відповідають за відображення хіт боксів.

```
def draw(self, win):
    self.move()
    if self.walkCount + 1 >= 33:
        self.walkCount = 0

    if self.vel > 0:
        win.blit(self.walkRight[self.walkCount//3], (self.x,self.y))
        self.walkCount += 1
    else:
        win.blit(self.walkLeft[self.walkCount//3], (self.x,self.y))
        self.walkCount += 1
    self.hitbox = (self.x + 15, self.y, 28, 60)
    #pygame.draw.rect(win, (255,0,0), self.hitbox, 2)


else:
    win.blit(playerStand, (x, y))
for bullet in bullets:
    bullet.draw(win)
goblin.draw(win)
hitbox = (x + 10, y, 40, 80)
#pygame.draw.rect(win, (255,0,0), hitbox, 2)
text = font.render("Рахунок: " + str(score), 1, (0,0,0))
win.blit(text, (350, 10))
```

Наступна задача – створити механізм роботи полоски здоров'я. Для цього у функції draw класу enemy намалюємо прямокутник червоного кольору та прямокутник зеленого кольору, що перекриває червоний.

```
def draw(self, win):
    self.move()
    if self.walkCount + 1 >= 33:
        self.walkCount = 0

    if self.vel > 0:
        win.blit(self.walkRight[self.walkCount//3], (self.x,self.y))
        self.walkCount += 1
    else:
        win.blit(self.walkLeft[self.walkCount//3], (self.x,self.y))
        self.walkCount += 1
    self.hitbox = (self.x + 15, self.y, 28, 60)
    pygame.draw.rect(win, (255,0,0), (self.hitbox[0], self.hitbox[1] - 20, 50, 10))
    pygame.draw.rect(win, (0,255,0), (self.hitbox[0], self.hitbox[1] - 20, 50, 10))
    #pygame.draw.rect(win, (255,0,0), self.hitbox, 2)
```

Далі створимо дві змінні у фукції __init__ класу enemy. Перша буде відповідати за здоров’я ворога, друга – за видимість ворога.

```
def __init__(self, x, y, width, height, end):
    self.x = x
    self.y = y
    self.width = width
    self.height = height
    self.path = [x, end]
    self.walkCount = 0
    self.vel = 3
    self.hitbox = (self.x + 20, self.y, 28, 60)
    self.health = 10
    self.visible = True
```

Далі змінимо функцію hit() класу enemy наступним чином. Напишемо перевірку: якщо змінна health більша за нуль, тоді при попаданні від health відніметься одиниця, якщо health == 0, тоді visible надається значення False.

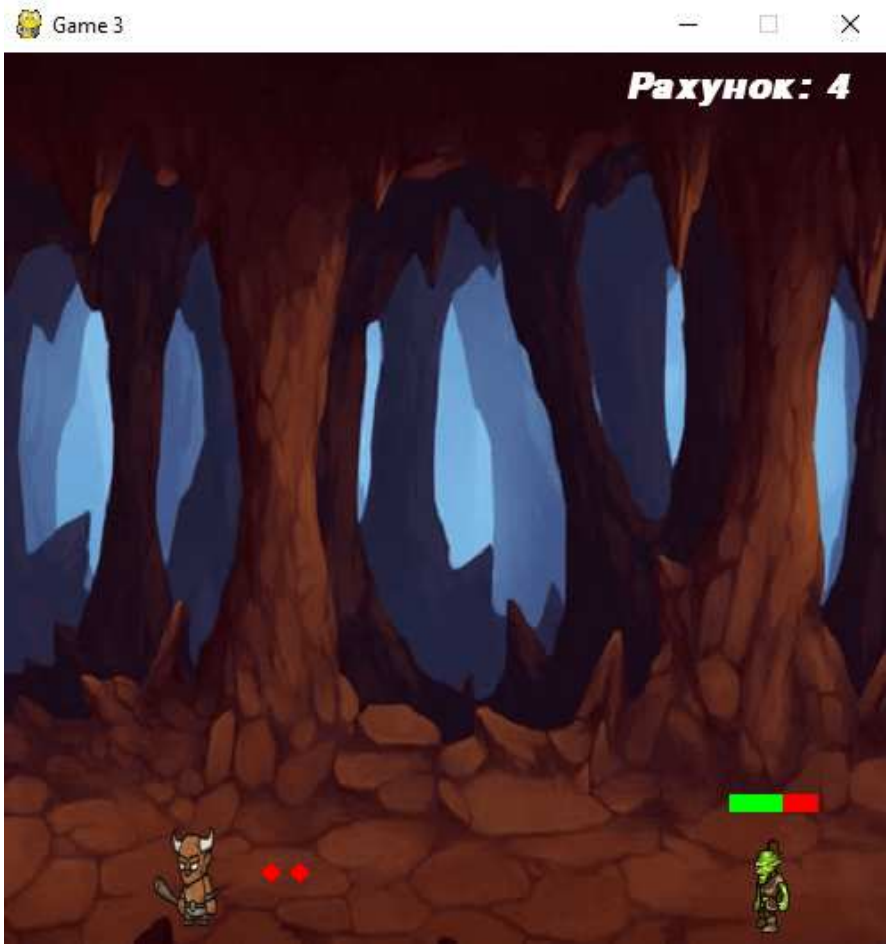
```
def hit(self):
    if self.health > 0:
        self.health -= 1
    else:
        self.visible = False
    print('Попадание')
```

Змінимо функцію draw наступним чином. Створимо перевірку: якщо self.visible = True, тоді ворог залишиться на екрані, якщо False, тоді він зникне.

```
def draw(self, win):
    if self.visible:
        self.move()
        if self.walkCount + 1 >= 33:
            self.walkCount = 0

        if self.vel > 0:
            win.blit(self.walkRight[self.walkCount//3], (self.x,self.y))
            self.walkCount += 1
        else:
            win.blit(self.walkLeft[self.walkCount//3], (self.x,self.y))
            self.walkCount += 1
        self.hitbox = (self.x + 15, self.y, 28, 60)
        pygame.draw.rect(win, (255,0,0), (self.hitbox[0], self.hitbox[1] - 20, 50, 10))
        pygame.draw.rect(win, (0,255,0), (self.hitbox[0], self.hitbox[1] - 20, 50, 10))
        #pygame.draw.rect(win, (255,0,0), self.hitbox, 2)
```

Для коректного відображення здоров'я ворога змінимо функцію draw для зеленого чотирикутника.

```
self.hitbox = (self.x + 15, self.y, 28, 60)
pygame.draw.rect(win, (255,0,0), (self.hitbox[0], self.hitbox[1] - 20, 50, 10))
pygame.draw.rect(win, (0,255,0), (self.hitbox[0], self.hitbox[1] - 20, 50 - (5 *(10 - self.health)), 10))
#pygame.draw.rect(win, (255,0,0), self.hitbox, 2)
```



Створимо три змінні, які будуть вказувати на ефекти та музику у грі. Для програвання музики на задньому фоні пишеться команда

pygame.mixer.music.play(-1), де -1 пишеться для повторення музики після її закінчення.

```
import pygame
pygame.init()
win = pygame.display.set_mode((500, 500))
pygame.display.set_caption("Game 3")

bulletSound = pygame.mixer.Sound('bullet.wav')
hitSound = pygame.mixer.Sound('hit.wav')

music = pygame.mixer.music.load('music.mp3')
pygame.mixer.music.play(-1)
```

Напишемо bulletSound.play() для програвання звуку стрільби після натискання F:

```
if keys[pygame.K_f] and shootloop == 0:
    bulletSound.play()
    if lastMove == 'right':
        facing = 1
    else:
        facing = -1
```

Аналогічно напишемо hitSound.play() для програвання звуку попадання по ворогу:

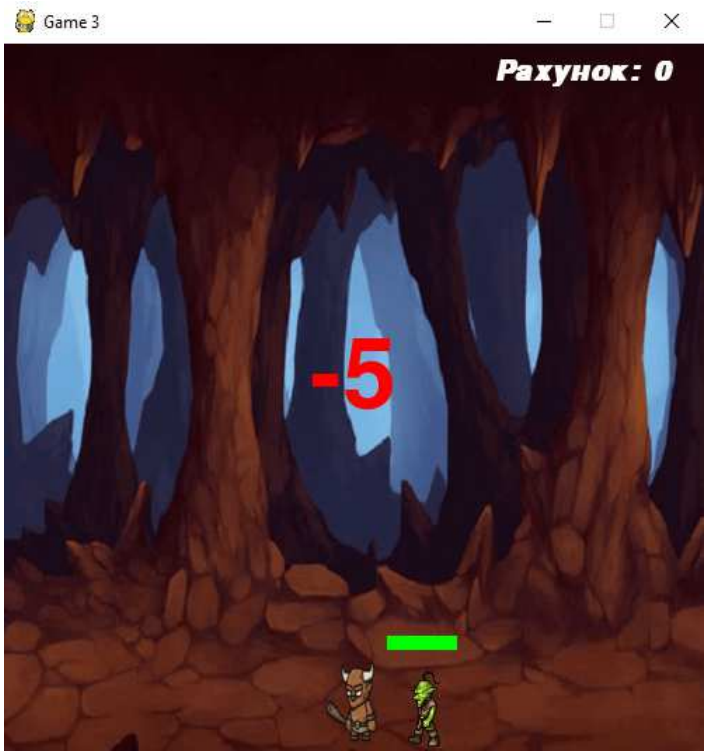
```
for bullet in bullets:
    if bullet.y - bullet.radius < goblin.hitbox[1] + goblin.h
        if bullet.x + bullet.radius > goblin.hitbox[0] and bu
            hitSound.play()
            goblin.hit()
            score += 1
            bullets.pop(bullets.index(bullet))
```

Створимо функцію hit() яка буде запускатися після стикання гравця з ворогом. Вона почне анімацію з початку, напише на екран «-5». Далі за допомогою простого таймеру зробимо так, щоб програма чекала три секунди після стикання. Якщо гравецт вирішить вийти з гри під час роботи таймеру, таймер закінчить роботу та гравець вийде.

```
def hit():
    animCount = 0
    font1 = pygame.font.SysFont('comicsans', 100)
    text1 = font1.render('-5', 1 ,(255,0,0))
    win.blit(text1, (250 - (text1.get_width()/2), 200))
    pygame.display.update()
    i = 0
    while i < 300:
        pygame.time.delay(10)
        i += 1
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                i = 301
                pygame.quit()

font = pygame.font.SysFont("comicsans", 30, True, True)
goblin = enemy(100, 436, 64, 64, 400)
run = True
shootloop = 0
```

Далі використаємо функцію, що ми створили. Зробимо перевірку: якщо хіт бокс гравця перетинає хіт бокс ворога по осям x та y, тоді виконується функція hit, що ми створили, координати гравця по осям x та y змінюються на 50 та 425 відповідно, та score зменшується на 5.

```
while run:
    clock.tick(30)
    hitbox = (x + 10, y, 40, 80)

    if hitbox[1] < goblin.hitbox[1] + goblin.hitbox[3] and hitbox[1] + hitbox[3] > goblin.hitbox[1]:
        if hitbox[0] + hitbox[2] > goblin.hitbox[0] and hitbox[0] < goblin.hitbox[0] + goblin.hitbox[2]:
            hit()
            x = 50
            y = 425
            score -= 5
```



Увесь код програми представлено у додатку 8.

Практичне завдання

1. Для вашої програми коректно змінити здоров’я ворога, кольори прямокутників.
2. Змінити кольори та розташування «Рахунку».
3. Створити клас птиць та його об’єкти, що будуть літати зверху.

Практична робота №18

Тема. Створення 2D гри на Python 3 з використанням бібліотеки PyGame (Частина 1).

Мета роботи. Освоїти створення 2D гри на Python 3 з використанням бібліотеки PyGame.

Зміст.

1. Вивчення відомостей про бібліотеку PyGame.
2. Виконання роботи.
3. Отримання результату.

Ключові положення.

Гравець буде намагається вижити серед метеоритів і інших предметів, що летять на нього.

Створення шаблону гри.

Гра буде виконана в «портретному режимі» - це значить, що висота вікна більша, ніж його ширина. Гра буде екшеном, тому важливо задати високе

значення кадрів в секунду. В такому випадку спрайт будуть рухатися максимально плавно. 60 - ідеальне значення.

```
WIDTH = 480
HEIGHT = 600
FPS = 60
```

Далі кольори та необхідно відкрити вікно гри:

```
pygame.init ()
pygame.mixer.init () # для звуку
screen = pygame.display.set_mode ((WIDTH, HEIGHT))
pygame.display.set_caption ("Моя гра")
clock = pygame.time.Clock ()
```

pygame.init () - це команда, яка запускає pygame. screen - вікно програми, яке створюється, коли ми задаємо його розмір в налаштуваннях. Далі необхідно створити clock, щоб переконатися, що гра працює із заданою частотою кадрів.

Ігровий цикл - це цикл while, контрольований змінної running. Якщо потрібно завершити гру, необхідно всього лише поміняти значення running на False. В результаті цикл завершиться. Тепер можна заповнити кожен розділ базовим кодом.

```
# Рендеринг
    screen.fill(BLACK)
    # после отрисовки всего, переворачиваем экран
    pygame.display.flip()
```

У pygame багато подій, на які він здатний реагувати. pygame.QUIT - подія, яка стартує після натискання хрестика і передає значення False змінної running, в результаті чого ігровий цикл закінчується.

```
for event in pygame.event.get():
        # проверить закрытие окна
        if event.type == pygame.QUIT:
            running = False
```

Поки що нічого помістити в розділ Update (оновлення), але потрібно переконатися, що налаштування FPS контролює швидкість гри. Це можна зробити наступним чином:

while running:

Тримаємо цикл на правильній швидкості

clock.tick (FPS)

Команда tick () просить pygame визначити, скільки займає цикл, а потім зробити паузу, щоб цикл (цілий кадр) тривав потрібен час. Якщо задати значення FPS 30, це означає, що довжина одного кадру - 1/30, тобто 0,03 секунди. Якщо цикл коду (оновлення, рендеринг та інше) займає 0,01 секунди, тоді pygame зробить паузу на 0,02 секунди. Шаблон готов.

шаблон.py - C:\Users\alext\Desktop\Irpa\шаблон.py (3.7.2)

File Edit Format Run Options Window Help

```
import pygame
import random

WIDTH =360
HEIGHT = 480
FPS = 30
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)


pygame.init()
pygame.mixer.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption('My Game')
clock = pygame.time.Clock()


running = True
while running:
    clock.tick(FPS)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
                ####

    screen.fill(BLACK)
        ####
    pygame.display.flip()

pygame.quit()
```

Прописуємо гравця.

Спрайт ігрока:

```
class Player(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((50, 40))
        self.image.fill(GREEN)
        self.rect = self.image.get_rect()
        self.rect.centerx = WIDTH / 2
        self.rect.bottom = HEIGHT - 10
        self.speedx = 0
```

Для гравця обраний розмір 50x40 пікселів. Він буде знаходиться по центру в нижній частині екрана. Також є властивість speedx, який відстежуватиме, з якою швидкістю рухається гравець по осі x.

Метод update () спрайту запускається в кожному кадрі. Він буде переміщати спрайт з конкретною швидкістю:

```
def update (self):
    self.rect.x + = self.speedx
```

Тепер потрібно показати спрайт, щоб переконатися, що він відображається на екрані:

```
all_sprites = pygame.sprite.Group ()
player = Player ()
all_sprites.add (player)
```

Не забувайте, що кожен створений спрайт повинен бути доданий до групи all_sprites, так щоб він оновлювався і промальовувався на екрані.

Наступний код встановлює швидкість speedx на значенні 0 для кожного кадру, а потім перевіряє, чи не натиснули Ви кнопку. pygame.key.get_pressed () повертає словник з усіма клавішами клавіатури і значеннями True або False, які вказують на те, чи натиснута якась із них. Якщо одна з кнопок натискається, швидкість змінюється відповідно.

```
def update(self):
    self.speedx = 0
    keystate = pygame.key.get_pressed()
    if keystate[pygame.K_LEFT]:
        self.speedx = -8
    if keystate[pygame.K_RIGHT]:
        self.speedx = 8
    self.rect.x += self.speedx
```

Тепер якщо rect спробує рухатися за межі екрану, він зупиниться.

```
if self.rect.right > WIDTH:
        self.rect.right = WIDTH
    if self.rect.left < 0:
        self.rect.left = 0
```

В першу чергу, необхідно відредагувати налаштування, додавши деякі значення в гру:

```
#ШАБЛОН
import pygame
import random
WIDTH =480
HEIGHT = 600
FPS = 60
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
YELLOW = (255, 255, 0)
```

Далі необхідно відкрити вікно гри:

```
pygame.init ()
pygame.mixer.init () # для звуку
```

```
screen = pygame.display.set_mode ((WIDTH, HEIGHT))
pygame.display.set_caption ("Моя гра")
clock = pygame.time.Clock ()
```

pygame.init () - це команда, яка запускає pygame. screen - вікно програми, яке створюється, коли ми задаємо його розмір в налаштуваннях. Далі необхідно створити clock, щоб переконатися, що гра працює із заданою частотою кадрів.

Перше, що необхідно додати - спрайт, який уособлює собою гравця. У підсумку це буде космічний корабель, але на початкових етапах можна просто ігнорувати графіку і використовувати прямокутники замість спрайтів. Важливо домогтися того, щоб вони відображалися на екрані і рухалися так, як потрібно. Замінити їх потім артами не складе труднощів.

```
class Player(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((50, 40))
        self.image.fill(GREEN)
        self.rect = self.image.get_rect()
        self.rect.centerx = WIDTH / 2
        self.rect.bottom = HEIGHT - 10
        self.speedx = 0
```

class повідомляє Python, що визначається новий об'єкт, який буде спрайтом гравця. Його тип **pygame.sprite.Sprite**. Це означає, що він буде заснований на заздалегідь визначеному в Pygame класі **Sprite**.

Перше, що потрібно в визначенні **class** - спеціальна функція **__init __ (),** що включає код, який буде запущений при створенні нового об'єкта цього типу. Також у кожного спрайту в Pygame має бути два властивості: **image** і **rect**.

Перший рядок, **Pygame.sprite.Sprite .__ init __ (self)** потрібно в Pygame - вона запускає ініціалізатор вбудованих класів **Sprite.** Далі необхідно визначити властивість **image**. Зараз просто створимо квадрат розміром 50x50 і заповнимо його зеленим (**GREEN**) кольором.

Далі необхідно визначити **rect** спрайту. Це скорочене від **rectangle** (прямокутник). Прямокутники повсюдно використовуються в Pygame для відстеження координат об'єктів. Команда **get_rect ()** оцінює зображення **image** і вираховує прямокутник, здатний оточити його. Команду **rect** можна використовувати для розміщення спрайту в будь-якому місці.

Для гравця обраний розмір 50x40 пікселів. Він буде знаходиться по центру в нижній частині екрана. Також є властивість speedx, який відстежуватиме, з якою швидкістю рухається гравець по осі x (з боку в бік).

Метод update () спрайту запускається в кожному кадрі. Він буде переміщати спрайт з конкретною швидкістю:

```
def update(self):
```

Керувати в цій грі потрібно буде за допомогою клавіатури, тому гравець повинен буде рухатися при натисканні кнопок Ліворуч або Праворуч

```
def update(self):
    self.speedx = 0
    keystate = pygame.key.get_pressed()
    if keystate[pygame.K_LEFT]:
        self.speedx = -8
    if keystate[pygame.K_RIGHT]:
        self.speedx = 8
    self.rect.x += self.speedx
```

Нарешті, потрібно зробити так, щоб спрайт не зникав з екрана. Для цього потрібно додати цей код до update () гравця:

```
if self.rect.right > WIDTH:
        self.rect.right = WIDTH
    if self.rect.left < 0:
        self.rect.left = 0
```

Тепер якщо rect спробує рухатися за межі екрану, він зупиниться.

На завершення потрібно показати спрайт, щоб переконатися, що він відображається на екрані:

```
all_sprites = pygame.sprite.Group()
player = Player()
all_sprites.add(player)
```

Почнемо з визначення властивостей спрайту ворога:

Об'єкт зверху за межами екрану (y <0), а значення x має бути в межах двох сторін.

```
class Mob(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((30, 40))
        self.image.fill(RED)
        self.rect = self.image.get_rect()
        self.rect.x = random.randrange(WIDTH - self.rect.width)
        self.rect.y = random.randrange(-100, -40)
        self.speedy = random.randrange(1, 8)
        self.speedx = random.randrange(-3, 3)
```

Для функції оновлення потрібно задати руху спрайту з певною швидкістю, але що з ним буде, коли він добереться до нижньої частини екрану?

Його можна видалити і створити новий або ж можна перенести цей же спрайт в випадкове місце у верхній частині екрану.

```
def update(self):
        self.rect.x += self.speedx
        self.rect.y += self.speedy
        if self.rect.top > HEIGHT + 10 or self.rect.left<-25 or self.rect.right>WIGHT+20:
            self.rect.x = random.randrange(WIDTH - self.rect.width)
            self.rect.y = random.randrange(-100, -40)
            self.speedy = random.randrange(1, 8)
```

Ворогів буде багато, тому потрібно створити групу mobs для них усіх. Це також спростить роботу з ними в подальшому. Після цього потрібно викликати певну кількість мобів і додати їх у групи.

```
player = Player()
all_sprites.add(player)
for i in range(8):
    m = Mob()
    all_sprites.add(m)
    mobs.add(m)
```

Потрібно додати рух по координаті x: Також потрібно змінити інструкцію if, яка створює нових мобів в той момент, коли ті пропадають з екрану. Моб, який рухається по діагоналі, пропаде з екрану задовго до того як добереться до нижньої частини, тому потрібно перезавантажити його швидше.

Додати

```
all_sprites = pygame.sprite.Group()
mobs = pygame.sprite.Group()
player = Player()
all_sprites.add(player)
for i in range(8):
```

Виявлення зіткнень на увазі необхідність розпізнати, коли один об'єкт в грі торкається іншого. Відповідь на зіткнення - це те, що трапиться в момент зіткнення. У кожного спрайту в Pygame є атрибут rect, що визначає його координати і розмір. Об'єкт rect в Pygame представлений в форматі [x, y, width, height], де x і y є верхній лівий кут прямокутника. Інша назва для цього прямокутника - обмежує рамка, тому що вона є кордоном об'єкта.

Виявлення зіткнень називається AABB (axis-aligned bounding box або «паралельний осях обмежує паралелепіпед»). Така назва пояснюється тим, що прямокутник вирівнюється відповідно до осями екрану, що не нахиляються. Виявлення зіткнень AABB таке популярне, тому що працює швидко - комп'ютер блискавично порівнює координати прямокутників, що особливо зручно, коли на екрані багато об'єктів. Щоб виявити зіткнення, у Pygame є вбудована функція spritecollide () для виконання цього.

В розділ «update» ігрового циклу необхідно додати наступну команду:

```
# Обновление
all_sprites.update()
# Проверка, не ударил ли моб игрока
hits = pygame.sprite.spritecollide(player, mobs, False)
if hits:
    running = False
```

Команда **spritecollide ()** приймає 3 аргументу: назва спрайту, який потрібно перевіряти, назва групи для порівняння і значення **True** або **False** для параметра **dokill**. Останній параметр дозволяє вказати, чи повинен об'єкт віддалятися при зіткненні. Якщо потрібно було, наприклад, перевірити, підібрав чи гравець монетку, необхідно вказати значення **True** так, щоб монетка пропала.

Результат команди **spritecollide ()** - це список спрайтів, які зіткнулися з гравцем (він може бути не один). Дамо його змінної **hits**. Якщо список **hits** буде непустою, значення інструкції **if** виявиться **True**. В результаті значення **running** зміниться на **False**, і гра закінчиться.

Додамо ще один спрайт – кулі.

```
class Bullet(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((10, 20))
        self.image.fill(YELLOW)
        self.rect = self.image.get_rect()
        self.rect.bottom = y
        self.rect.centerx = x
        self.speedy = -10

    def update(self):
        self.rect.y += self.speedy
```

```
        # убить, если он заходит за верхнюю часть экрана
        if self.rect.bottom < 0:
            self.kill()
```

У метод __init __ () спрайту кулі потрібно передати значення x і y, щоб вказати спрайту, де з'являтися. Оскільки спрайт гравця може рухатися, то місце появи буде відповідати місцю розташування гравця. Значення speedy буде негативним, щоб він спрайт рухався нагору.

Нарешті, потрібно перевірити виявився чи спрайт за межами екрану. Якщо так - його можна видаляти.

Наступний код перевіряє подія KEYDOWN, і якщо таке спостерігається, перевіряє чи натиснута кнопка K_SPACE. Якщо так - запускається метод гравця shoot ().

```
for event in pygame.event.get():
    # проверка для закрытия окна
    if event.type == pygame.QUIT:
        running = False
    elif event.type == pygame.KEYDOWN:
        if event.key == pygame.K_SPACE:
            player.shoot()
```

Необхідно додати групу для куль.

```
bullets = pygame.sprite.Group()
```

Тепер можна створювати наступний метод в класі Player:

Все що робить метод shoot () - створює кулю, використовуючи в якості місця появи верхню центральну частину гравця. Після цього потрібно переконатися, що куля додана в all_sptires (щоб вона намалювалася і оновилася) і в bullets, яка буде використовуватися для зіткнень.

```
def shoot(self):
    bullet = Bullet(self.rect.centerx, self.rect.top)
    all_sprites.add(bullet)
    bullets.add(bullet)
```

Перевірити, чи зачепила куля мобу. Відмінність в тому, що є кілька куль (в групі bullets) і кілька мобів (в групі mobs), тому не можна використовувати spritecollide () як в минулий раз, тому що в цій функції порівнюється тільки один спрайт з групою. Замість цього потрібно використовувати groupcollide ():

```
# Обновление
all_sprites.update()


# Проверка, не ударил ли моб игрока
hits = pygame.sprite.groupcollide(mobs, bullets, True, True)
for hit in hits:
    m = Mob()
    all_sprites.add(m)
    mobs.add(m)
```

Функція groupcollide () схожа на spritecollide () за винятком того, що потрібно вказувати дві групи для порівняння, а повертати функція буде список зачеплених мобів. Також є два значення dokill для кожної з груп.

Якщо просто видаляти мобів, то з'явиться проблема: вони закінчаться. Тому потрібно просто проходити циклом по hits і для кожного знищеного мобу створювати один новий.

Увесь код програми представлено у додатку 9.

Практичне завдання.

2. Написати програму для рухомого прямокутника розміром №№ на 60 пікселів, Кольору згідно варіанту № у вікні 500+№№ на 500+№№ пікселів. Де № - остання цифра студента у списку групи.

№	Колір стрибаючого прямокутника	Швидкість руху
1	Червоний	50мс
2	Зелений	55мс
3	Синій	56мс
4	Голубий	57мс
5	Фіолетовий	58мс
6	Білий	59мс

7	Помаранчевий	49мс
8	Жовтий	48мс
9	Червоний	47мс
0	Зелений	45мс

Практична робота №19

Тема. Створення 2D гри на Python 3 з використанням бібліотеки PyGame (Частина 2).

Мета роботи. Освоїти створення 2D гри на Python 3 з використанням бібліотеки PyGame.

Зміст.

13. Вивчення відомостей про бібліотеку PyGame.
14. Виконання роботи.
15. Отримання результату.

Ключові положення.

Щоб бути впевненими в тому, що код буде працювати в будь-якій операційній системі, потрібно використовувати функцію os.path, яка визначає правильний шлях до файлів незалежно від ОС. У верхній частині програми необхідно визначити місце розташування папки img:

```
#ШАБЛОН
import pygame
import random
from os import path

img_dir = path.join(path.dirname(__file__))
snd_dir = path.join(path.dirname(__file__), 'snd')
```

Тепер можна приступати до завантаження фонового зображення. Завантаження Ассетів необхідно виконувати до ігрового циклу і коду запуску:

Завантаження всієї ігрової графіки

background = pygame.image.load (path.join (img_dir, 'starfield.png')). convert ()

background_rect = background.get_rect ()

```
    def update(self):
        self.rect.y += self.speedy
        # убить, если он заходит за верхнюю часть экрана
        if self.rect.bottom < 0:
            self.kill()

            #Загрузка игровой графики

background = pygame.image.load(path.join(img_dir, "starfield.png")).convert()
background_rect = background.get_rect()
```

Тепер можна промальовувати фон в розділі Draw ігрового циклу до промальовування кожного з спрайтів:

рендеринг

screen.fill (BLACK)

screen.blit (background, background_rect)

all_sprites.draw (screen)

blit - це олдскульний термін з комп'ютерної графіки, який позначає промальовування пікселів одного зображення на іншому. В цьому випадку - промальовування фону на екрані. Тепер фон виглядає набагато краще:

```
        #Рендеринг
            screen.fill(BLACK)
            screen.blit(background, background_rect)
            all_sprites.draw(screen)
```

Тепер можна завантажити зображення спрайтів:

```
        Загрузка всей игровой графики
        background              =              pygame.image.load(path.join(img_dir,
'starfield.png')).convert()
        background_rect = background.get_rect()
        player_img              =              pygame.image.load(path.join(img_dir,
"playerShip1_orange.png")).convert()
```

```
        meteor_img              =              pygame.image.load(path.join(img_dir,
"meteorBrown_med1.png")).convert()
        bullet_img              =              pygame.image.load(path.join(img_dir,
"laserRed16.png")).convert()
        meteor_images = []
        meteor_list     =     ['meteorBrown_big1.png',     'meteorBrown_med1.png',
'meteorBrown_med1.png',
                        'meteorBrown_med3.png',          'meteorBrown_small1.png',
'meteorBrown_small2.png',
                        'meteorBrown_tiny1.png']
        for img in meteor_list:
            meteor_images.append(pygame.image.load(path.join(img_dir,
img)).convert())
```

Начнем с игрока — необходимо заменить зеленый прямоугольник, то есть `self.image` и не забыть убрать параметр `image.fill(GREEN)`, который больше не нужен:

```
        class Player(pygame.sprite.Sprite):
            def __init__(self):
                pygame.sprite.Sprite.__init__(self)
                self.image = player_img
                self.rect = self.image.get_rect()
```

Проте є пара проблем. По-перше, картинка трохи більше, ніж потрібно. Є два варіанти: 1) відкрити її в графічному редакторі (Photoshop, GIMP) і відредагувати; 2) поміняти розмір прямо в коді. Вибираємо другий варіант. Для цього знадобиться функція Pygame під назвою transform.scale (). Зменшимо зображення вдвічі - до розміру 50x30 пікселів.

Друга проблема - чорний прямокутник навколо корабля. Щоб це виправити, потрібно призначити прозорий колір за допомогою set_colorkey:

```
        self.image = pygame.transform.scale(player_img, (50, 38))
        self.image.set_colorkey(BLACK)
```

Якщо повторити те ж саме для класів Bullet і Mob (хоча їх розмір змінювати не потрібно), гра буде виглядати набагато краще:

Для обробки штовхання треба використовувати обмежуче коло. Воно ідеально вписується в форму астероїда. Не так добре підходить для крил корабля, але самій грі це не зашкодить.

Спираючись на можливості, описані вище, для функції зіткнення гравця з астероїдом будуть використовуватися кола. У Pygame це зробити простіше простого за допомогою атрибута self.radius для кожного з спрайтів.

Спочатку гравець. Якого розміру повинен бути коло? Доведеться поекспериментувати, щоб визначити потрібний радіус. Ось як зробити це в спрайт гравця __init () __:

```
self.rect = self.image.get_rect()
self.radius = 25
pygame.draw.circle(self.image, RED, self.rect.center, self.radius)
```

Намалюємо червоне коло над зображенням, щоб бачити, як він працює.

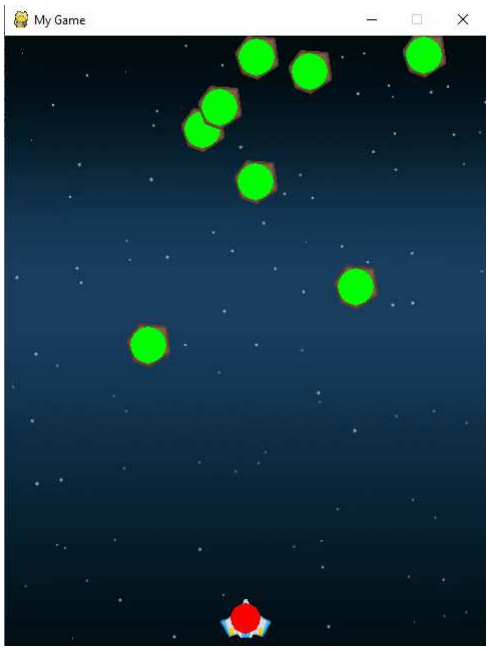
Те ж саме потрібно зробити і для астероїда:

```
class Mob(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = meteor_img
        self.image.set_colorkey(BLACK)
        self.image_orig = random.choice(meteor_images)
        self.image_orig.set_colorkey(BLACK)
        self.rect = self.image.get_rect()
        self.radius = int(self.rect.width * 0.85 / 2)
        pygame.draw.circle(self.image, GREEN, self.rect.center, self.radius)
        self.rect.x = random.randrange(WIDTH - self.rect.width)
        self.rect.y = random.randrange(-100, -40)
        self.speedy = random.randrange(1, 8)
        self.speedx = random.randrange(-3, 3)
```

Якщо з часом буде вирішено використовувати астероїди різних розмірів, то вказавши радіус як 1/2 ширини зображення, можна буде не повертатися до редагування коду.

Для астероїда добре було б, щоб його краю визирали з-під кола, тому при розрахунку потрібно зменшити ширину астероїда до 85% від оригінальної.

self.radius = int (self.rect.width * .85 / 2)



Щоб гра почала використовувати цей тип зіткнень, потрібно поміняти команду spritecollide с AABB на обмежуюче коло:

Перевірка, не вдарив чи моб гравця

```
#Оновлення
    all_sprites.update()
    #Перевірка на удар
    hits = pygame.sprite.spritecollide(player, mobs, False)
    if hits:
        running = False
```

Якщо все працює так, як хотілося, червоні кола можна прибрати. Краще їх просто закоментувати.

Всі астероїди виглядають однаково. Додати різноманітності і привабливості можна, змусивши їх обертатися. Завдяки цьому буде створюватися враження, що вони дійсно летять у космосі. Для обертання потрібно задіяти pygame.transform.rotate (). Але щоб зробити це правильно, потрібно ознайомитися з парою нюансів.

В першу чергу необхідно додати властивості спрайту Mob:

```
class Mob(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = meteor_img
        self.image.set_colorkey(BLACK)
        self.image_orig = random.choice(meteor_images)
        self.image_orig.set_colorkey(BLACK)
        self.rect = self.image.get_rect()
        self.radius = int(self.rect.width * 0.85 / 2)
        pygame.draw.circle(self.image, GREEN, self.rect.center, self.radius)
        self.rect.x = random.randrange(WIDTH - self.rect.width)
        self.rect.y = random.randrange(-150, -100)
        self.speedy = random.randrange(1, 8)
        self.speedx = random.randrange(-3, 3)
        self.rot = 0
        self.rot_speed = random.randrange(-8, 8)
        self.last_update = pygame.time.get_ticks()
```

Перше властивість rot (скорочено від «rotation» (обертання)) буде вимірювати, на скільки градусів має обертатися астероїд. Початкове значення - 0, але воно буде змінюватися з часом. rot_speed вимірює, наскільки градусів астероїд буде повертатися кожен раз - чим більше число, тим швидше буде відбуватися обертання. Підійде будь-яке значення: позитивне задасть обертання за годинниковою стрілкою, а негативне - проти.

Остання властивість необхідна для контролю швидкості анімації. Для гри не потрібно кожного разу міняти зображення спрайту в кожному кадрі. При анімації зображення спрайту потрібно лише визначити час - як часто воно буде змінюватися.

У бібліотеці є об'єкт pygame.time.Clock (), який називається clock (годинник). Він дозволяє контролювати FPS (кількість кадрів в секунду). Викликаючи pygame.time.get_ticks (), можна дізнатися, скільки мілісекунд минуло з тих пір, як годинник були запущені. Так можна буде сказати, чи пройшло достатньо часу, що в черговий раз міняти зображення спрайту.

Для цієї операції потрібно ще кілька рядків коду. Використовуємо новий метод self.rotate (), який можна додати в метод update ():

```
def update(self):
    self.rotate()
    self.rect.x += self.speedx
    self.rect.y += self.speedy
    if self.rect.top > HEIGHT + 10 or self.rect.left < -25 or self.rect.righ
        self.rect.x = random.randrange(WIDTH - self.rect.width)
        self.rect.y = random.randrange(-100, -40)
        self.speedy = random.randrange(1, 8)
```

Завдяки цьому можна буде добитися того, що в методі не буде зайвою інформацією, а обертання можна прибрати, закоментувавши всього один рядок. Ось початок методу обертання:

```
def rotate(self):
        now = pygame.time.get_ticks()
        if now - self.last_update > 50:
                self.last_update = now
                # повертання спрайтів
```

Спочатку перевіряється поточний час, потім віднімається час останнього оновлення. Якщо пройшло більше 50 мілісекунд, потрібно оновлювати зображення. Додаємо значення now в last_update і можна робити обертання. Здається, що залишилося лише застосувати його до спрайту - якось так:

self.image = pygame.transform.rotate (self.image, self.rot_speed)

Але в такому разі виникне проблема:

Це відбувається, тому що зображення являє собою сітку пікселів. При переміщенні їх в нове положення піксель не вирівнюються, і деяка інформація просто пропадає. Якщо повернути зображення один раз, то нічого страшного не станеться. Але при постійному обертанні астероїд перетворюється в кашу.

Рішення полягає в тому, щоб використовувати змінну rot для відстеження загальної ступеня обертання (додаючи rot_speed з кожним оновленням) і обертати оригінальне зображення з таким кроком. Таким чином спрайт кожен раз буде являти собою чисте зображення, яке повернеться всього один раз.

Спочатку потрібно скопіювати оригінальну картинку:

```
class Mob(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = meteor_img
        self.image.set_colorkey(BLACK)
        self.image_orig = random.choice(meteor_images)
        self.image_orig.set_colorkey(BLACK)
        self.image = self.image_orig.copy()
        self.rect = self.image.get_rect()
        self.radius = int(self.rect.width * 0.85 / 2)
        #pygame.draw.circle(self.image, GREEN, self.rect.center, self.radius)
        self.rect.x = random.randrange(WIDTH - self.rect.width)
        self.rect.y = random.randrange(-150, -100)
        self.speedy = random.randrange(1, 8)
        self.speedx = random.randrange(-3, 3)
        self.rot = 0
        self.rot_speed = random.randrange(-8, 8)
        self.last_update = pygame.time.get_ticks()
```

Потім в методі rotate потрібно оновити значення rot і застосувати обертання до вихідного зображення:

```
def rotate(self):
        now = pygame.time.get_ticks()
        if now - self.last_update > 50:
            self.last_update = now
            self.rot = (self.rot + self.rot_speed) % 360
            self.image     =     pygame.transform.rotate(self.image_orig,
self.rot)
```

Варто відзначити, що був використаний оператор залишку,%, щоб значення rot не було більше 360.

Зображення вже виглядають добре, але все ще є одна проблема: Здається, що астероїди стрибають, а не плавно обертаються.

Для цього необхідно вказати положення центру прямокутника, обчислити новий розмір і зберегти координати центру в оновлений прямокутник:

```
def rotate(self):
        now = pygame.time.get_ticks()
        if now - self.last_update > 50:
            self.last_update = now
            self.rot = (self.rot + self.rot_speed) % 360
            new_image     =     pygame.transform.rotate(self.image_orig,
self.rot)
```

```
            old_center = self.rect.center
            self.image = new_image
            self.rect = self.image.get_rect()
            self.rect.center = old_center
```

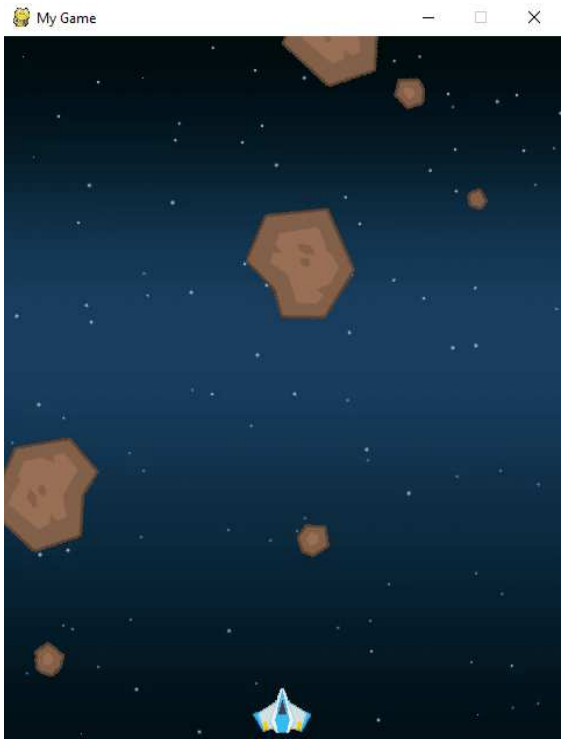
Останнє, що можна зробити, щоб астероїди виглядали цікавіше, - задавати їх розміри випадковим чином.

Спершу необхідно завантажити їх і додати в список:

```
meteor_images = []
meteor_list =['meteorBrown_big1.png','meteorBrown_med1.png',
              'meteorBrown_med1.png','meteorBrown_med3.png',
              'meteorBrown_small1.png','meteorBrown_small2.png',
              'meteorBrown_tiny1.png']
for img in meteor_list:
    meteor_images.append(pygame.image.load(path.join(img_dir,
img)).convert())
```

Далі потрібно просто кожен раз вибирати випадковий астероїд для появи в кадрі:

```
class Mob(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image_orig = random.choice(meteor_images)
        self.image_orig.set_colorkey(BLACK)
        self.image = self.image_orig.copy()
```



Ведення рахунок гравця - просте завдання. Потрібно змінна з початковим значенням 0, до якої буде додаватися +1 при кожному знищення астероїда. Оскільки астероїди різні, а у великі потрапити легше, ніж в маленькі, є сенс в тому, щоб давати більше очок за знищення тих, що поменше.

Назвемо змінну score і оголосимо її до ігрового циклу:

```
for i in range(8):
    m = Mob()
    all_sprites.add(m)
    mobs.add(m)
score = 0
```

Щоб призначати рахунок в залежності від розміру астероїда, можна використовувати властивість radius. У найбільших астероїдів зображення більше 100 пікселів, а радіус 100 * 0.85 / 2 = 43 пікселя. У той же час радіус самого маленького астероїда - всього 8 пікселів. Можна віднімати радіус з більшого числа, наприклад 50, щоб отримувати кількість очок. Так, великий буде давати всього 7 очок, а маленький - 42 очка.

```
#Оновлення
    all_sprites.update()
    #Перевірка на удар
    hits = pygame.sprite.spritecollide(player, mobs, False)
    if hits:
        running = False
    hits = pygame.sprite.groupcollide(mobs, bullets, True, True)
    for hit in hits:
        score += 50 - hit.radius
        m = Mob()
        all_sprites.add(m)
        mobs.add(m)
```

Тепер, коли є змінна з рахунком, треба малювати її на екрані, і ось це вже складніше. Робиться все в кілька етапів. Якщо ви плануєте виводити текст не один раз, тоді є сенсу створити функцію, яка буде називатися draw_text. Її параметри:

surf - поверхня, на якій текст буде написаний

text - рядок, яку потрібно відображати

x, y - положення

Також потрібно вибрати шрифт. Проблема може виникнути, якщо вибрати той шрифт, який не встановлений на комп'ютері. Вирішити це можна

за допомогою pygame.font.match_font (), яка шукає найбільш підходящий шрифт в системі.

Ось вся функція draw_text:

```
font_name = pygame.font.match_font('arial')
def draw_text(surf, text, size, x, y):
    font = pygame.font.Font(font_name, size)
    text_surface = font.render(text, True, WHITE)
    text_rect = text_surface.get_rect()
    text_rect.midtop = (x, y)
    surf.blit(text_surface, text_rect)
```

Отрісовка тексту на екрані - це, фактично, обчислення необхідної структури пікселів. У комп'ютерній графіці це називається «рендерингом». За це буде відповідати функція font.render (). У функції є невідомий параметр True, який відповідає за включення і відключення згладжування.

Тепер можна показувати рахунок на екрані. Потрібно лише додати його в розділ відтворення у верхній частині екрану по центру.

```
# Рендеринг
    screen.fill(BLACK)
    screen.blit(background, background_rect)
    all_sprites.draw(screen)
    draw_text(screen, str(score), 18, WIDTH / 2, 10)
```

Тепер можна додавати звуки в гру. По-перше, потрібно вказати, в якій папці вони знаходяться:

Далі файли потрібно завантажити. Це варто зробити в тому ж місці, де завантажувалася графіка. Спочатку звук пострілів:

```
# Загрузка мелодий игры
shoot_sound = pygame.mixer.Sound(path.join(snd_dir, 'pew.wav'))
```

Тепер, коли звук завантажений і присвоєно змінної shoot_sound, на нього можна посилатися. Важливо, щоб він відтворювався кожен раз, коли гравець стріляє, тому його потрібно додати в метод shoot () Player:

```
    def shoot(self):
        bullet = Bullet(self.rect.centerx, self.rect.top)
        all_sprites.add(bullet)
        bullets.add(bullet)
        shoot_sound.play()
```

Це все, що потрібно. Стріляти тепер буде приємніше!

```
#Оновлення
    all_sprites.update()
    #Перевірка на удар
    hits = pygame.sprite.spritecollide(player, mobs, False)
    if hits:
        running = False
    hits = pygame.sprite.groupcollide(mobs, bullets, True, True)
    for hit in hits:
        score += 50 - hit.radius
        random.choice(expl_sounds).play()
        m = Mob()
        all_sprites.add(m)
        mobs.add(m)
#Рендерінг
```

Поки що гравця знищує одне влучення астероїда. Так грати не дуже цікаво, тому додамо властивість shield, яке буде всього лише числом від 0 до 100.

```
class Player(pygame.sprite.Sprite):
    def __init__(self):
        self.speedx = 0
        self.shield = 100
```

Тепер, кожен раз, коли астероїд буде потрапляти в гравця, можна віднімати певний рівень здоров'я. Коли рівень впаде до 0, гра закінчується. Щоб було цікавіше, можна зробити, щоб великі астероїди наносили більше шкоди. Для цього варто використовувати властивість radius астероїда.

Поки що зіткнення моб-проти-гравця обробляється на найпростішому рівні:

```
  # Проверка, не ударил ли моб игрока
    hits = pygame.sprite.spritecollide(player, mobs, False, pygame.sprite.collide_circle)
    if hits:
        running = False
```

Потрібно додати кілька змін:

```
# Проверка, не ударил ли моб игрока
```

```
        hits = pygame.sprite.spritecollide(player, mobs, True,
pygame.sprite.collide_circle)
        for hit in hits:
            player.shield -= hit.radius * 2
            if player.shield <= 0:
                running = False
```

У spritecollide змінимо значення з False на True, тому що потрібно, щоб астероїд зникав після попадання. Якщо цього не зробити, то в міру просування він буде знову мати справу з гравцем в наступних кадрах. Можливо і те, що відразу кілька астероїдів вдарять по гравцеві, тому значень hit має бути кілька. Необхідно перебирати hits і віднімати певну кількість рівня здоров'я, грунтуючись на радіусі астероїда. Зрештою, коли значення впаде до 0, гра закінчиться.

Ви могли помітити, що при видаленні астероїда, що зіткнувся з гравцем, зменшується їх загальна кількість. Щоб не змінювати її, необхідно створювати нові при видаленні. Моби з'являлися за допомогою цього коду:

```
for i in range(8):
    m = Mob()
    all_sprites.add(m)
    mobs.add(m)
```

Можна було б додати деякі зміни, але це викличе повторення в коді, що не дуже добре. Замість цього краще помістити логіку створення мобів в функцію і використовувати її скрізь, де це буде потрібно:

```
def newmob():
    m = Mob()
    all_sprites.add(m)
    mobs.add(m)
```

Тепер можна використовувати цю ж функцію на початку гри, коли астероїди тільки з'являються і в тому місці, де вони пропадають після зіткнення:

```
for i in range(8):
    newmob()

# проверка, попала ли пуля в моб
hits = pygame.sprite.groupcollide(mobs, bullets, True, True)
for hit in hits:
```

```
            score += 50 - hit.radius
            random.choice(expl_sounds).play()
            newmob()

        # Проверка, не ударил ли моб игрока
        hits = pygame.sprite.spritecollide(player, mobs, True,
pygame.sprite.collide_circle)
        for hit in hits:
            player.shield -= hit.radius * 2
            newmob()
            if player.shield <= 0:
                running = False
```

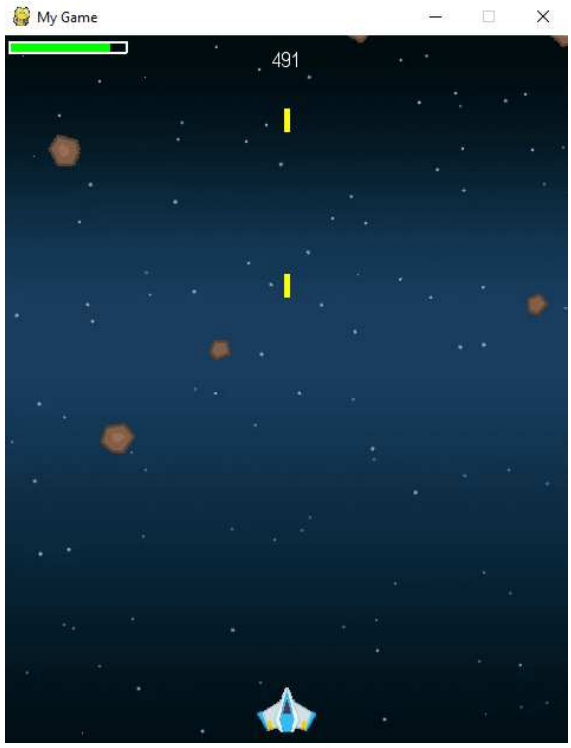
Є значення здоров'я, але воно багато в чому марно, якщо гравцеві невідомо це значення. Потрібно створити дисплей, але замість показу просто цифр краще вивести смужку, по якій буде видно, наскільки вона заповнена:

```
def draw_shield_bar(surf, x, y, pct):
    if pct < 0:
        pct = 0
    BAR_LENGTH = 100
    BAR_HEIGHT = 10
    fill = (pct / 100) * BAR_LENGTH
    outline_rect = pygame.Rect(x, y, BAR_LENGTH, BAR_HEIGHT)
    fill_rect = pygame.Rect(x, y, fill, BAR_HEIGHT)
    pygame.draw.rect(surf, GREEN, fill_rect)
    pygame.draw.rect(surf, WHITE, outline_rect, 2)
```

Ця функція буде працювати за аналогією з draw_text. Вона створить смужку з розмірами BAR_LENGHT і BAR_HEIGHT, яка буде знаходитися в (x, y) і заповнена на таке значення pct. Намалюємо два прямокутника: перший - білий контур, другий - рівень здоров'я. Додамо виклик цієї функції в розділ відтворення ігрового циклу:

```
#Рендеринг
    screen.fill(BLACK)
    screen.blit(background, background_rect)
    all_sprites.draw(screen)
    draw_text(screen, str(score), 18, WIDTH / 2, 10)
    draw_shield_bar(screen, 5, 5, player.shield)
    #Після прорисовки обертаємо екран
```

Тепер можна бачити, скільки здоров'я у гравця, і як він змінюється при попаданні астероїдів.



Увесь код програми представлено у додатку 10.

Практичне завдання

1. Для вашої програми коректно змінити ворога, швидкість його руху, швидкість обертання.
2. Змінити кольори та розташування «Рахунку».
3. Змінити героя та задній фон.
4. Створити клас об’єктів, що будуть літати зверху, яких не треба вбивати.

Практична робота №20

Розробка: студент гр.КНД-31 Шелег М.Г.

Тема. Створення RESTful додатку.

Мета роботи. Ознайомитись з архітектурою REST їх властивостями. Дослідити створення RESTful додатку.

Зміст.

16. Вивчення відомостей про архітектуру REST та властивості REST архітектури.
17. Виконання роботи.
18. Отримання результату.

Ключові положення

REST (REpresentational State Transfer) вже давно стала стандартною архітектурою для створення веб-сервісів і API для них. Що таке RESTful веб-сервіс? Архітектура REST розроблена щоб відповідати протоколу HTTP, що використовується в мережі Інтернет.

Центральне місце в концепції RESTful веб-сервісів це поняття ресурсів. Ресурси представлені URI. Клієнти відправляють запити до цих URI використовуючи методи представлені протоколом HTTP, і, можливо, змінюють стан цих ресурсів.

Методи HTTP спроектовані для впливу на ресурс стандартним способом:

Метод HTTP	Действие	Пример
GET	Отримати інформацію про ресурс	example.com/api/orders (отримати список заказів)
GET	Отримати інформацію про ресурс	example.com/api/orders/123 (отримати заказ #123)

POST	Створити новий ресурс	example.com/api/orders створити новий ресурс із даних переданих із запитом)
PUT	Обновить ресурс	example.com/api/orders/123 (оновити заказ #123 даними переданими із запитом)
DELETE	Видалити ресурс	example.com/api/orders/123 (Видалити заказ #123)

Дизайн REST не дає рекомендацій яким конкретно повинен бути формат даних переданих із запитами. Дані передані в тілі запиту можуть бути JSON blob, або за допомогою аргументів в URL.

Властивості REST архітектури

Класичні обмеження при розробці production сервісу:

- продуктивність (швидкість відповіді REST сервіса на запит)
- масштабованість як для збільшення кількості взаємодії компонентів, так і для збільшення можливого навантаження (горизонтальне - збільшення кількості серверів та вертикальне - збільшення характеристик одного серверу)

Також академічні властивості закладені Роєм Філдінгом, які непомітно використовуються на практиці, але здебільшого вже є під капотом сучасних веб-фреймворків та бібліотек:

- простота уніфікованого інтерфейсу (чітко задані типи запитів)
- надійність: стійкості до відмов на рівні системи при наявності відмов окремих компонентів, відсутності з’єднання або деяких даних
- переносимість компонентів системи шляхом переміщення програмного коду разом з даними
- прозорість зв'язків між компонентами системи для сервісних служб;

- відкритість компонентів до можливих змін для задоволення мінливих потреб (навіть при працюючому додатку);

Вимоги до REST архітектури

Існує 6 обов'язкових обмежень, закладених Філдінгом, для побудови розподілених REST-додатків. Проекти в індустрії вносять свої корективи в існуючі підходи. Тому 6 принцип є не обов'язковим, а відповідність фразі "Якщо сервіс-додаток порушує будь-яке з цих обмежувальних умов, дану систему можна вважати REST-системою" відходить на другий план.

1. Модель клієнт-сервер.
2. Відсутність стану (stateless).
3. Кешування.
4. Однорідний інтерфейс.
5. Шари абстракції.
6. Завантаження додаткового коду (необов'язковий).

Більше детально можете ознайомитися із цими пунктами https://uk.wikipedia.org/wiki/REST. Також деякі з них використовується в маніфесті 12 Factor App актуальним для розробки веб-додатків.

При проектуванні веб-сервісу або API потрібно визначити ресурси, які будуть доступні і запити, за допомогою яких ці дані будуть доступні, згідно правил REST.

Щоб написати додаток To Do List і треба спроектувати веб-сервіс для нього. Перше треба придумати кореневий URL для доступу до цього сервісу. Наприклад такий:

http://[hostname]/api/v1/todo

Тут треба включити в URL ім'я програми та версію API. Додавання імені додатки в URL це спосіб розділити між собою сервіси запущені на одному сервері. Додавання версії API в URL може допомогти, якщо захочеться зробити оновлення в майбутньому і впровадити у новій версії несумісні функції і не хочеться ламати працюючі додатки які працюють на старому API.

Наступним кроком, треба вибрати ресурси, які будуть доступні через сервіс. Оскільки це дуже простий додаток, то ресурсами можуть бути тільки завдання з нашого ToDo листа.

Для доступу до ресурсів будемо використовувати такі методи HTTP:

Метод HTTP	**URI**	**Действие**
GET	http://[hostname]/api/v1/todo/tasks	Отримати список задач
GET	http://[hostname]/api/v1/todo/tasks/{task_id}	Отримати задачу
POST	http://[hostname]/api/v1/todo/task	Створити нову задачу
PUT	http://[hostname]/api/v1/todo/task/{task_id}	Оновити існуючу задачу
DELETE	http://[hostname]/api/v1/todo/task/{task_id}	Видалити задачу

Поля:

- id: унікальний ідентифікатор задачі. Тип int.
- title: опис задачі. Тип string.
- description: детальний опис задачі. Тип string.
- done: статус виконання. Тип bool.

Введення до fastAPI

FastAPI - це відносно молодий веб-фреймворк для створення REST (та GraphQL, якщо постаратися) додатків. FastAPI надзвичайно сильно схожий на flask, однак він значно краще, бо має:

- підказки типів (type-hints)
- нативну асинхронність
- підтримку Swagger і ReDoc для швидкої генерації документації

Встановлення необхідних бібліотек:

pip install uvicorn fastapi pydantic

Hello world шаблон

main.py (Додаток 11)

```
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
async def root():

  return {"message": "Hello World"}
```

Запустимо його:

```
uvicorn main:app
```

Також uvicorn має декілька важливих параметрів, які вказуються через --{param}, якщо пишемо повне ім`я:

- reload - live reloading (після того як написали новий код, uvicorn автоматично перезапуститься)
- host - на якій адресі запускаємо, якщо будемо запускати на сервері - використовуємо 0.0.0.0 (по дефолту - 127.0.0.1 або localhost)
- port - на якому порту запускаємо (по дефолту - 800)

Створення GET/POST/PUT/DELETE, запити з fastapi

При створенні методів в fastapi ми вказуємо декоратор, який використовує об`єкт FastAPI або APIRouter та їх методи, наприклад `@app.get()`

GET метод може бути замінений на будь-який інший з методів, написаних з маленької букви (post, put, delete). В тілі цього метода (дужках) пишеться шлях за яким буде доступна ця функція, також там можуть бути параметер response_model, який використовується здебільшого для GET методів.

Response - модель, задається pydantic. BaseModel, яка парсить Python об`єкт в json та навпаки.

Також, в функціях використовуються *path* параметри, які вказають після основного URL функції, наприклад /task/{task_id}

Їх використовують для постійних елементів, наприклад id запису.

В fastapi буде ми задаємо його фігурними дужками в строковому значенні та вказуємо ім`я змінної, яке повинне співпадати із (Додаток 12):

```
@todo_router.get("/task/{task_id}", response_model=Task)
async def get_task(task_id: int):
    return db_helper.get_task(task_id)
```

Query params використовують для параметрів, які можуть використовувати за необхідністю. Їх вказують через ?, якщо їх декілька, то додається & після останнього параметра, наприклад /tasks?offset=10&limit=5

```
@todo_router.get("/tasks")
async def get_tasks(limit: int = 0, skip: int = 0):
  return {"data": db_helper.get_tasks(limit, skip)}
```

А в методі функції вказуємо query параметри в параметрах функції. Щоб в запиті query параметри були необов`язковими треба вказати їм дефолтне значення, як у прикладі вище.

Також body (тіло запиту) для передачі великої кількості параметрів, які особливо актуальні для зміни (PUT) та додавання нової інформації (POST). GET методам не рекомендується мати request body. В fastapi показуємо його використання pydantic. BaseModel, описуючи в них необхідні поля:

```
@todo_router.post("/task")
async def add_task(task: Task):
  db_helper.add_task(task)
  return {"status": "OK"}
```

Веб-додаток з такою структурою і файлами

Файл запуску main.py (Додаток 11)

```python
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from .todo_router import todo_router

# app instance creation
app = FastAPI()

# if we do not include CORS it will not work with frontend
app.add_middleware(
    CORSMiddleware,
    allow_origin_regex='https?://.*',
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# we can add many routers with different prefixes for one app with this command
app.include_router(todo_router, prefix='/api/v1/todo')
```

у

Файл models.py (Додаток 13)

```python
from pydantic import BaseModel


class Task(BaseModel):
    title: str
    description: str = ""
    done: bool = False

```

Створення абстракції для БД (Додаток 14).

```python
class DbHelper:
    def __init__(self):
        self._tasks: Dict[int, Task] = {
            1: Task(title="Build Rocket"),
            2: Task(title="Train stuff for the flight"),
            3: Task(title="Build infrastructure for launch")
        }

    def get_tasks(self, limit: int) -> List[Task]:
        task_list = list(self._tasks.values())
        if limit == 0:
            return task_list
        else:
            return task_list[:limit]

    def get_task(self, task_id: int) -> Task:
        # check whether task_id exists in the self._tasks, make it it to avoid KeyError Exception
        if task_id in self._tasks:
            return self._tasks[task_id]

    def add_task(self, task: Task):
        task_id: int = len(self._tasks) + 1
        self._tasks[task_id] = task

    def delete_task(self, task_id: int) -> bool:
        if task_id in self._tasks:
            del self._tasks[task_id]
        return task_id in self._tasks

    def update_task(self, task_id: int, task: Task) -> bool:
        if task_id in self._tasks:
            self._tasks[task_id] = task
        return task_id in self._tasks
```

При розробці веб-додатків не зберігайте ніякі дані в оперативну і постійну пам'ять серверу. При збоях та незапланованих перезавантаженнях ви можете втратити дані, якщо ви зберігаєте їх таким же як в self._tasks.

Тому в реальних проектах значно краще підключити базу даних або зберігати файли в Cloud Bucket (Amazon S3, Google Cloud Storage).

Файл todo_router.py (Додаток 12)

```python
from fastapi import APIRouter, HTTPException
from .db import DbHelper
from .models import Task

todo_router = APIRouter()
db_helper = DbHelper()


@todo_router.get("/tasks")
async def get_tasks(limit: int = 0):
    return {"data": db_helper.get_tasks(limit)}


# {task_id} is a path param for this get function
# response model is a syntactic sugar of dict formatting
@todo_router.get("/task/{task_id}", response_model=Task)
async def get_task(task_id: int):
    task = db_helper.get_task(task_id)
    if task:
        return task
    else:
        raise HTTPException(status_code=400, detail="This id doesn't exist, please, pass valid id")


@todo_router.post("/task")
async def add_task(task: Task):
    db_helper.add_task(task)
    return {"status": "OK"}


@todo_router.put("/task/{task_id}")
async def update_task(task_id: int, task: Task):
    task_exists = db_helper.update_task(task_id, task)
    if task_exists:
        return {"status": "OK"}
    else:
        raise HTTPException(status_code=400, detail="This id doesn't exist, please, pass valid id")


@todo_router.delete("/task/{task_id}")
async def delete_task(task_id: int):
    task_exists = db_helper.delete_task(task_id)
    if task_exists:
        return {"status": "OK"}
    else:
        raise HTTPException(status_code=400, detail="This id doesn't exist, please, pass valid id")
```

Як тестувати API?

Гарне питання, якщо GET запити ми можемо відправляти через Google Chrome або інший браузер, як відкриваємо звичайний сайт. Щоб отримати всі задачі запустіть додаток та введіть http://localhost:8000/api/v1/todo/tasks

А як відправляти запити на видалення, редагування та створення нових записів? Для цього ми будемо використовувати Postman.

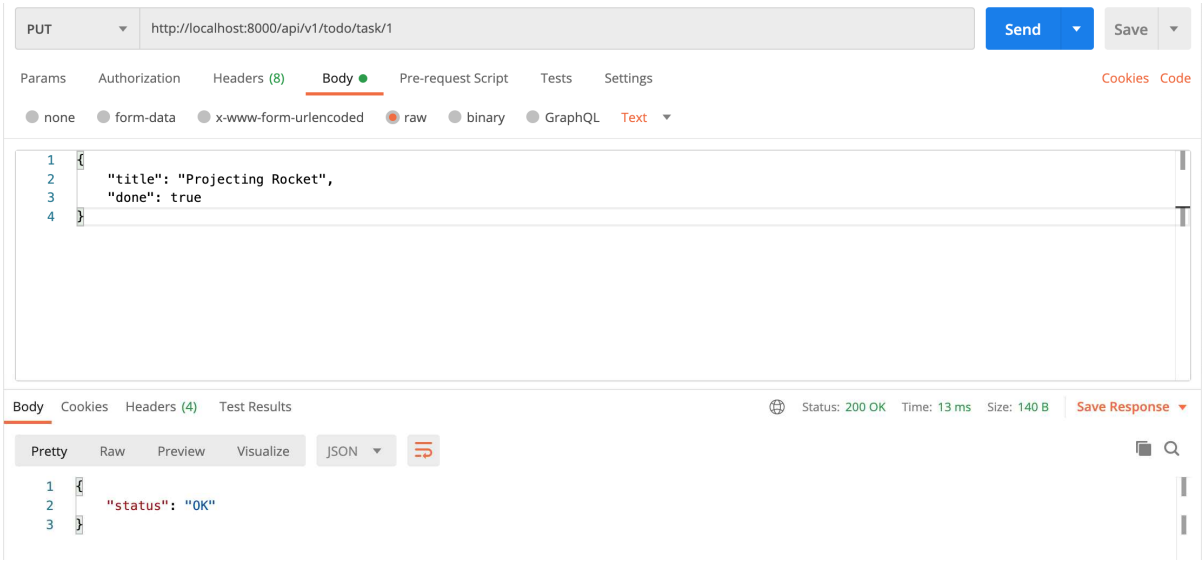
Оберемо необхідний тип запиту, наприклад PUT. Після цього введемо URL запиту:

http://localhost:8000/api/v1/todo/task/1

Відкриємо body, обираємо raw та вводимо:

```
{
   "title": "Projecting Rocket",
   "done": true
}
```

Нажимаємо на Send



Практичне завдання

1.Додати в GET /tasks такі необов'якові (запит повинен працювати і без них) query параметрі:

- *skip* (int) та реалізувати в DbHelper пропуск об'єктів, вказанної кількості
- *status* (bool) - повертати задачі із відповідним значенням *done*

2. Створити PUT метод, який буде виставляти статус done

Приймає *id* задачі як path параметр, реалізувати перевірку на наявність оновлюваного елементу, як в PUT методі з прикладу.

Доп функціонал: Захист нашого сервісу

https://fastapi.tiangolo.com/advanced/security/http-basic-auth/

https://fastapi.tiangolo.com/tutorial/security/first-steps/

Практична робота №21

Розробка: студент гр.КНД-32 Дабіжа І.О.

Тема. Початок роботи із асинхронними функціями в Python.

Мета роботи. Ознайомитись з асинхронними функціями в Python.

Зміст.

19. Вивчення відомостей про роботу з асинхронними функціями в Python.
20. Виконання роботи.
21. Отримання результату.

Ключові положення

У цій роботі ви маєте змогу дізнатися про те, що таке:

1. Синхронні програми
2. Асинхронні програми
3. Чому асинхронні програми можуть тобі сподобатись
4. Як використовувати асинхронні функції у Python

Усі приклади кода було протестовано на версії Python 3.7.2. Ви завжди можете скопіювати код до свого редактору та перевірити його.

Розуміння асинхронного програмування

Синхронна програма виконується по одному кроку. Навіть маючи умовне розгалуження, цикли та виклики функцій, ви все одно маєте думати про код з точки зору виконання одного кроку за раз. Після завершення кожного кроку програма переходить до наступного.

Ось два приклади програм, які працюють таким чином:

1. Програми пакетної обробки часто створюються як синхронні програми. Ви отримуєте певний вхід, обробляєте його та створюєте

певний результат. Кроки слідують один за одним, поки програма не досягне бажаного результату. Програмі потрібно лише звернути увагу на етапи та їх порядок.

2. Програми командного рядка - це невеликі, швидкі процеси, які працюють у терміналі. Ці сценарії використовуються для створення чогось, перетворення однієї речі в щось інше, створення звіту або, можливо, перерахування деяких даних. Це може бути виражено як ряд програмних кроків, які виконуються послідовно, поки програма не буде виконана.

Асинхронна програма поводиться по-різному. Це все одно займає один крок виконання за раз. Різниця полягає в тому, що система може не чекати завершення етапу виконання, перш ніж переходити до наступного.

Це означає, що програма перейде до наступних кроків виконання, навіть якщо попередній крок ще не закінчений і все ще працює в іншому місці. Це також означає, що програма знає, що робити, коли попередній крок закінчується.

Чому ви хочете написати програму таким чином? Решта цієї статті допоможе вам відповісти на це запитання та дасть вам інструменти, необхідні для елегантного вирішення цікавих асинхронних проблем.

Створення синхронного веб-сервера

Основною одиницею роботи веб-сервера є більш-менш те саме, що і пакетна обробка. Сервер отримає деякий вхід, обробить його і створить результат. Написаний як синхронна програма, це створить робочий веб-сервер.

Це також був би абсолютно жахливий веб-сервер.

Чому? У цьому випадку одна одиниця роботи (вхід, процес, вихід) не є єдиною метою. Справжня мета полягає в тому, щоб якомога швидше виконати сотні або навіть тисячі одиниць роботи. Це може відбуватися протягом тривалого періоду часу, і кілька робочих підрозділів можуть навіть прибути одночасно.

Чи можна вдосконалити синхронний веб-сервер? Звичайно, ви можете оптимізувати кроки виконання, щоб уся робота, яка надходить, була виконана якомога швидше. На жаль, цей підхід має обмеження. Результатом може стати веб-сервер, який не реагує досить швидко, не може впоратися з достатньою кількістю роботи, або навіть той, який відключається, коли робота накаплюється.

Примітка: Є й інші обмеження, які ви можете побачити, якщо спробували оптимізувати вищезазначений підхід. Сюди входять швидкість мережі, швидкість введення-виведення файлів, швидкість запитів до бази даних та швидкість інших підключених служб, щоб назвати декілька. Загальне у всіх - те, що всі вони є функціями вводу-виводу. Усі ці елементи на порядок повільніші, ніж швидкість обробки процесора.

У синхронній програмі, якщо на етапі виконання починається запит до бази даних, тоді центральний процесор по суті простоює, поки не буде повернутий запит до бази даних. Для пакетно-орієнтованих програм це не є пріоритетом у більшості випадків. Метою є обробка результатів цієї операції введення-виведення. Часто це може зайняти більше часу, ніж сама операція введення-виведення. Будь-які зусилля з оптимізації будуть зосереджені на роботі з обробки, а не на вводі-виводі.

Асинхронні методи програмування дозволяють вашим програмам користуватися перевагами відносно повільних процесів введення-виведення, звільняючи центральний процесор для виконання іншої роботи.

Коли ви починаєте намагатися зрозуміти асинхронне програмування, ви можете побачити багато дискусій про важливість блокування або написання неблокуючого коду.

Що таке неблокуючий код? Що ж таке блокуючий код? Чи допоможуть відповіді на ці запитання вам створити кращий веб-сервер? Якщо так, то як ви могли це зробити? Давайте дізнаємось!

Написання асинхронних програм вимагає, щоб ви по-іншому думали про програмування. Хоча цей новий спосіб мислення може бути складним для

розуміння, це також цікава вправа. Це тому, що реальний світ майже повністю асинхронний, як і те, як ви з ним взаємодієте.

Уявіть собі: ви - батько, який намагається зробити кілька речей одночасно. Ви повинні збалансувати чекову книжку, випрати білизну і стежити за дітьми. Якось ти можеш робити всі ці речі одночасно, навіть не думаючи про це! Давайте розберемо це:

- Збалансування чекової книжки - це синхронне завдання. Один крок слідує за іншим, доки його не буде зроблено. Ви виконуєте всю роботу самостійно.
- Однак, ви можете відірватися від чекової книжки, щоб випрати білизну. Ви розвантажуєте пральну машину, переносите одяг в сушильну машину і починаєте подальше завантаження пральної машини.
- Робота з пральною машиною та сушаркою є синхронним завданням, але основна частина роботи відбувається після запуску пральної машини та сушарки. Після того, як ви їх запустите, ви можете піти і повернутися до завдання чекової книжки. На даний момент завдання пральної та сушильної машин стали асинхронними. Пральна машина та сушарка працюватимуть самостійно, доки не вимкнеться таймер
- Спостереження за своїми дітьми - ще одне асинхронне завдання. Після того, як вони налаштовані та грають, вони можуть робити це самостійно здебільшого. Це змінюється, коли хтось потребує уваги, наприклад, коли хтось зголодніє або постраждає. Коли хтось із ваших дітей кричить на сполох, ви реагуєте. Діти - це довгострокове завдання з високим пріоритетом. Спостереження за ними замінює будь-які інші завдання, які ви могли виконувати, наприклад, чекову книжку чи пральню.

Ці приклади можуть допомогти проілюструвати поняття блокуючого та неблокуючого коду. Давайте подумаємо над цим з точки зору програмування.

У цьому прикладі ви схожі на центральний процесор. Поки ви пересуваєте білизну, ви (центральний процесор) зайняті та не можете робити

інші роботи, наприклад, балансування чекової книжки. Але це нормально, оскільки завдання є відносно швидким.

З іншого боку, запуск пральної та сушильної машин не заважає виконувати інші завдання. Це асинхронна функція, оскільки вам не потрібно чекати, поки вона закінчиться. Щойно воно розпочнеться, ви можете повернутися до чогось іншого. Це називається перемиканням контексту: контекст того, що ви робите, змінився, і таймер машини повідомить вас колись у майбутньому, коли завдання з пранням буде виконано.

Як людина, ти постійно працюєш. Ви, природно, жонглюєте кількома речами одночасно, часто не думаючи про це. Як розробник, фокус полягає в тому, як перевести подібну поведінку в код, який робить те саме.

Якщо ви зрозуміли, що малось на увазі у прикладі вище, то ви мали зрозуміти, як працює асинхронне програмування.

Використання функцій Python Async на практиці

Для прикладу зараз буде наведено 2 варіанти програми:

1. Синхронні (блокуючі) HTTP-виклики
2. Асинхронні (неблокуючі) HTTP-виклики

Синхронні (блокуючі) HTTP-виклики

Для перегляду коду відкрийте файл example_sync.py (Додаток 15)

Програма виконує фактичну роботу з реальним введенням-виведенням, роблячи HTTP-запити до списку URL-адрес та отримуючи вміст сторінки. Однак це робиться блокуючим (синхронним) способом.

Ось що відбувається в цій програмі:

- Рядок 2 імпортує запити, що забезпечує зручний спосіб здійснення HTTP-викликів.

- Рядок 3 імпортує код таймера з модуля кодування.
- Рядок 6 створює екземпляр таймера, який використовується для вимірювання часу, зайнятого на кожну ітерацію циклу завдань.
- Рядок 11 запускає екземпляр таймера
- Рядок 12 вводить затримку, він викликає session.get (url), який повертає вміст URL-адреси, отриманої з work_queue.
- Рядок 13 зупиняє екземпляр таймера і видає час, що минув з моменту виклику timer.start ().
- Рядки 23–32 містять список URL-адрес у черзі роботи.
- Рядок 39 створює диспетчер контексту таймера, який видасть витрачений час, який виконувався весь цикл while.

Під час запуску цієї програми ви побачите щось схоже на такий результат:

Task One getting URL: http://google.com
Task One total elapsed time: 0.3
Task Two getting URL: http://yahoo.com
Task Two total elapsed time: 0.8
Task One getting URL: http://linkedin.com
Task One total elapsed time: 0.4
Task Two getting URL: http://apple.com
Task Two total elapsed time: 0.3
Task One getting URL: http://microsoft.com
Task One total elapsed time: 0.5
Task Two getting URL: http://facebook.com
Task Two total elapsed time: 0.5
Task One getting URL: http://twitter.com
Task One total elapsed time: 0.4

Total elapsed time: 3.2

Кожне завдання отримує URL-адресу з робочої черги, отримує вміст сторінки та повідомляє, скільки часу потрібно для отримання цього вмісту.

yield дозволяє виконувати обидва ваші завдання спільно. Однак, оскільки ця програма працює синхронно, кожен виклик session.get () блокує процесор, доки сторінка не буде отримана. Зверніть увагу на загальний час, необхідний для запуску всієї програми в кінці. Це буде значущим для наступного прикладу.

Асинхронні (неблокуючі) HTTP-виклики

Для перегляду коду відкрий файл example_async.py (Додаток 16)

Ця версія програми змінює попередню, щоб використовувати функції асинхронізації Python. Він також імпортує модуль aiohttp, який є бібліотекою для асинхронного здійснення HTTP-запитів за допомогою asyncio.

Ці завдання були змінені, щоб видалити виклик yield, оскільки код для здійснення виклику HTTP GET більше не блокується. Він також виконує перемикання контексту назад до циклу подій.

Ось що відбувається в цій програмі:

- Рядок 2 імпортує бібліотеку aiohttp, яка забезпечує асинхронний спосіб здійснення викликів HTTP.
- Рядок 3 імпортує код таймера з модуля кодування.
- Рядок 5 позначає завдання () як асинхронну функцію.
- Рядок 6 створює екземпляр таймера, який використовується для вимірювання часу, зайнятого на кожну ітерацію циклу завдань.
- У рядку 7 створюється менеджер контексту сеансу aiohttp.

- Рядок 8 створює менеджер контексту відповіді aiohttp. Він також робить виклик HTTP GET до URL-адреси, взятої з work_queue.
- Рядок 11 запускає екземпляр таймера
- Рядок 12 використовує сеанс для отримання тексту, отриманого з URL-адреси асинхронно.
- Рядок 13 зупиняє екземпляр таймера і видає час, що минув з моменту виклику timer.start ().
- Рядок 39 створює диспетчер контексту таймера, який видасть витрачений час, який виконувався весь цикл while.

Під час запуску цієї програми ви побачите щось схоже на такий результат:

Task One getting URL: http://google.com
Task Two getting URL: http://yahoo.com
Task One total elapsed time: 0.3
Task One getting URL: http://linkedin.com
Task One total elapsed time: 0.3
Task One getting URL: http://apple.com
Task One total elapsed time: 0.3
Task One getting URL: http://microsoft.com
Task Two total elapsed time: 0.9
Task Two getting URL: http://facebook.com
Task Two total elapsed time: 0.4
Task Two getting URL: http://twitter.com
Task One total elapsed time: 0.5
Task Two total elapsed time: 0.3

Total elapsed time: 1.7

Погляньте на загальний час, що минув, а також на витрачений час, щоб отримати вміст кожної URL-адреси. Ви побачите, що тривалість становить приблизно половину сукупного часу всіх викликів HTTP GET. Це тому, що виклики HTTP GET виконуються асинхронно. Іншими словами, ви ефективно використовуєте ЦП, дозволяючи йому робити одночасно декілька запитів.

Оскільки процесор настільки швидкий, цей приклад може створити стільки завдань, скільки URL-адрес. У цьому випадку тривалістю програми буде час найменшого повільного пошуку URL-адреси.

Висновок

Ця робота надала вам інструменти, необхідні для того, щоб почати робити асинхронні методи програмування частиною свого репертуару. Використання асинхронних функцій Python дає вам програмний контроль над тим, коли відбуваються перемикання контексту. Це означає, що з багатьма більш жорсткими проблемами, які ви можете побачити в потоковому програмуванні, легше вирішити.

Асинхронне програмування - це потужний інструмент, але він не корисний для всіх видів програм. Наприклад, якщо ви пишете програму, яка обчислює PI до мільйонного знака після коми, тоді асинхронний код вам не допоможе. Програма такого типу пов’язана з процесором, без великого вводу-виводу. Однак якщо ви намагаєтесь реалізувати сервер або програму, яка виконує введення-виведення (наприклад, доступ до файлів або мережі), то використання асинхронних функцій Python може мати величезне значення.

Тепер, коли ви знаєте, як працюють асинхронні програми у Python ви можете вивести свої програми на новий рівень. На цьому, мені залишається лише побажати вам розвитку у програмуванні та ніколи не зупинятися на досягнутому!

* Усі залежности знаходятся у файли required.txt (Додаток 22)

Практичне завдання

1. **Детальніше ознайомитись із async await** **https://dev.to/code_enzyme/introduction-to-using-async-await-in-python-2i0n** (eng)
   https://webdevblog.ru/obzor-async-io-v-python-3-7/ (ru)

2. **Опрацювати 2 приклади коду** в файлах **task.py** та **task2.py (Додаток 17)**, з'ясувати, який приклад буде працювати повільніше і пояснити, чому.

Практична робота №22

Розробка: студент гр.КНД-32 Бабій Н.Ю.

Тема. Введення у MongoDB та Python

Мета роботи. Ознайомитись з використанням Python для взаємодії з популярною базою даних MongoDB (v 3.4.0), а також порівняння SQL і NoSQL, PyMongo (v 3.4.0) і MongoEngine (v 0.10.7) .

Зміст.

22.Вивчення відомостей про систему управління базами даних.

23. Виконання роботи.

24.Отримання результату.

Ключові положення

Python - це потужна мова програмування, що використовується спільнотою розробників для безлічі різних типів програм. Багато хто знає його як гнучку мову, що здатна впоратися практично з будь-яким завданням.

Отже, що, якщо нашому складному додатком Python потрібно база даних, настільки ж гнучка, як і сама мова? Тут на допомогу приходять NoSQL бази даних і, зокрема, MongoDB .

У цій роботі покажемо, як використовувати Python для взаємодії з популярною базою даних MongoDB (v 3.4.0), а також розглянемо порівняння SQL і NoSQL, PyMongo (v 3.4.0) і MongoEngine (v 0.10.7) .

SQL проти NoSQL

Якщо ви не знайомі з нею, MongoDB - це база даних NoSQL , яка в останні роки стала досить популярною у всій галузі. Бази даних NoSQL надають можливості пошуку і зберігання даних абсолютно інакше, ніж їх аналоги в реляційних базах даних.

Протягом десятиліть бази даних SQL були одним з небагатьох варіантів для розробників, які прагнуть створювати великі масштабовані системи. Однак постійно зростаюча потреба в можливості зберігати складні структури даних привела до народження баз даних NoSQL, які дозволяють розробнику зберігати різнорідні і безструктурні дані.

Коли справа доходить до доступних варіантів, більшість людей задають собі головне питання: «SQL або NoSQL?». І у SQL, і у NoSQL є свої сильні і слабкі сторони, і вам слід вибрати ту, яка найкраще відповідає вимогам вашого застосування. Ось кілька відмінностей між ними:

SQL

- Модель носить реляційний характер.
- Дані зберігаються в таблицях.
- Підходить для рішень, в яких всі записи однотипні і мають однакові властивості.
- Додавання нової властивості означає, що вам потрібно змінити всю схему.
- Схема бази дуже сувора.
- Підтримуються транзакції ACID.

- Добре масштабується по вертикалі.

NoSQL

- Модель нереляційна.
- Може зберігатися як JSON, ключ-значення і т. д. (В залежності від типу бази даних NoSQL).
- Не всі записи повинні бути однаковими, що робить їх дуже гнучкими.
- Додавайте нові властивості до даних, нічого не заважаючи.
- Немає вимог до схеми, яких слід дотримуватися.
- Підтримка транзакцій ACID може варіюватися в залежності від того, яка база даних NoSQL використовується.
- Цілісність може варіюватися.
- Добре масштабується по горизонталі.

Між цими двома типами баз даних є багато інших відмінностей, але згадані вище є одними з найбільш важливих відмінностей, про які слід знати.

Залежно від вашого конкретного сценарію використання бази даних SQL може бути кращим, в той час як в інших сценаріях NoSQL є більш очевидним вибором. При виборі бази даних ви повинні уважно враховувати сильні і слабкі сторони кожної бази даних.

Одна з чудових особливостей NoSQL полягає в тому, що існує безліч різних типів баз даних на вибір, і у кожної є свої варіанти використання:

• Сховище ключів і значень: DynamoDB

• Сховище документів: CouchDB , MongoDB , RethinkDB

• Орієнтована на стовпці: Cassandra

• Структури даних: Redis

Їх досить багато, але це одні з найбільш поширених типів.

В останні роки бази даних SQL і NoSQL навіть почали об'єднуватися. Наприклад, PostgreSQL тепер підтримує зберігання і запити даних JSON, як і Mongo. Завдяки цьому тепер ви можете добитися того ж з

Postgres, що і з Mongo, але ви як і раніше не отримуєте багатьох переваг Mongo (таких як горизонтальне масштабування, простий інтерфейс і т. д.). Як говориться в цій роботі:

Якщо ваші активні дані зручно розміщуються в реляційній схемі й вміст JSON є групою цих даних, тоді вам повинно бути зручно з PostgreSQL, і це набагато ефективніше уявлення JSONB і можливості індексування. Якщо ж ваша модель даних являє собою колекцію змінних документів, ви, ймовірно, захочете поглянути на базу даних, створену в загальному на основі документів JSON, таких як MongoDB або RethinkDB.

MongoDB - це програма для баз даних з відкритим вихідним кодом, орієнтована на документи, яка не залежить від платформи. MongoDB, як і деякі інші бази даних NoSQL (але не всі!), зберігає свої дані в документах, використовуючи структуру JSON. Це те, що дозволяє даними бути такими гнучкими і не вимагати схеми.

Деякі з найбільш важливих функцій:

- У вас є підтримка багатьох стандартних типів запитів, таких як зіставлення (==), порівняння (<,>) або навіть регулярний вираз.
- Ви можете зберігати практично будь-які дані – будь то структуровані, частково структуровані або навіть поліморфні.
- Щоб масштабувати і обробляти більше запитів, просто додайте більше потужності.
- Він дуже гнучкий і динамічний, що дозволяє швидко розробляти свої додатки.
- База даних на основі документів означає, що ви можете зберігати всю інформацію про вашу модель в одному документі.
- Ви можете змінити схему своєї бази даних на льоту.
- Багато функцій реляційних баз даних також доступні в MongoDB (наприклад, індексування) .

Що стосується операційної частини речей, для MongoDB є досить багато інструментів і функцій, які ви просто не можете знайти в жодній іншій системі баз даних:

- Незалежно від того, чи потрібен вам автономний сервер або повні кластери з незалежних серверів, MongoDB настільки масштабується, наскільки вам потрібно.
- MongoDB також забезпечує підтримку балансування навантаження шляхом автоматичного переміщення даних між різними шардами.
- Є підтримка автоматичного перемикання при відмові – в разі відмови вашого основного сервера новий основний буде запущений автоматично.
- Служба управління MongoDB або MMS – це дуже хороший веб-інструмент, який дає вам можливість відслідковувати ваші сервера.
- Завдяки файлам, що відображаються в пам'ять, ви значно заощадите ОЗУ, на відміну від реляційних баз даних.

Якщо ви скористаєтеся функціями індексування, велика частина цих даних буде зберігатися в пам'яті для швидкого пошуку. І навіть без індексації за конкретними ключам документа Mongo проводить кешування досить великої кількості даних, використовуючи недавно використані методи.

Хоча спочатку може здатися, що Mongo - це рішення багатьох проблем з нашою базою даних, але у нього є свої недоліки. Один з поширених недоліків Mongo - відсутність підтримки транзакцій ACID. Mongo дійсно підтримує транзакції ACID в обмеженому сенсі, але не у всіх випадках. На рівні окремого документа підтримуються транзакції ACID (саме тут і так відбувається більшість транзакцій). Однак транзакції, пов'язані з декількома документами, не підтримуються через розподілений характер Mongo.

У Mongo також відсутня підтримка власних з'єднань, які повинні виконуватися вручну (і, отже, набагато повільніше). Документи повинні бути всеосяжними, що означає, що в цілому в них не повинно бути посилань на інші документи. У реальному світі це не завжди працює, оскільки велика частина

даних, з якими ми працюємо, за своєю природою реляційна. Тому багато хто буде стверджувати, що Mongo слід використовувати в якості додаткової бази даних до бази даних SQL, але при використанні MongoDB ви виявите, що це не обов'язково так.

PyMongo

Тепер, коли ми описали, що таке MongoDB, давайте з'ясуємо, як ви насправді використовували б його з Python. Офіційний драйвер, опублікований розробниками Mongo, називається PyMongo. Це гарне місце для початку для першого запуску Python з MongoDB. Ми розглянемо тут кілька прикладів, але вам також слід ознайомитися з повною документацією , оскільки ми не зможемо охопити все.

Перше, що вам потрібно зробити, це встановити PyMongo у вашому віртуальному середовищі. Найпростіший спосіб зробити це з допомогою pip :

```
$ Pip install pymongo == 3.4.0
```

ПРИМІТКА. Для отримання більш докладного керівництва відвідайте сторінку документації по встановленню / оновленню і виконайте там дії для налаштування.

Коли ви закінчите установку, запустіть консоль Python і виконайте наступну команду:

```
>>> import pymongo
```

Якщо він запускається без будь-яких винятків в оболонці Python, ваша установка працює нормально. Якщо ні, то уважно повторіть дії ще раз. Після завершення вийдіть з оболонки.

Потім вам потрібно встановити фактичну базу даних MongoDB. Якщо у вас Mac, ми рекомендуємо використовувати Homebrew , але у вас можуть бути інші переваги. Якщо ви використовуєте Homebrew, запустіть цю команду:

```
$ Brew install mongodb
```

Після цього дотримуйтесь інструкцій тут , щоб налаштувати каталог даних для локального зберігання даних.

Якщо ви не використовуєте Mac, ви можете знайти хороші керівництва по установці на сторінці «Встановлення MongoDB» офіційних документів.

Там ви знайдете керівництва по установці MongoDB для Linux, OS X і Windows.

Після установки в новому вікні терміналу використовуйте наступну команду для запуску демона Mongo:

```
$ mongod
```

ПРИМІТКА. Залежно від методу установки вам може знадобитися запустити mongod за допомогою sudo.

Встановлення з'єднання

Щоб встановити з'єднання, ми будемо використовувати об'єкт MongoClient.

Перше, що нам потрібно зробити, щоб встановити з'єднання, - це імпортувати клас MongoClient. Ми будемо використовувати це для зв'язку з працюючим екземпляром бази даних. Для цього використовуйте наступний код (Додаток 18):

```
from pymongo import MongoClient
client = MongoClient()
```

Використовуючи наведений вище фрагмент, з'єднання буде встановлено з хостом за замовчуванням (localhost) і портом (27017). Ви також можете вказати хост і/або порт, використовуючи:

```
client = MongoClient ( 'localhost' , 27017 )
```

Або просто використовуйте формат Mongo URI :

```
client = MongoClient ( 'mongodb: // localhost: 27017' )
```

Всі ці виклики MongoClient будуть робити те ж саме; це просто залежить від того, наскільки явним ви хочете бути в своєму коді.

Доступ до баз даних

Якщо у вас є підключений екземпляр MongoClient, ви можете отримати доступ до будь-якої з баз даних на цьому сервері Mongo . Щоб вказати, яку базу даних ви дійсно хочете використовувати, ви можете отримати до неї доступ як атрибут:

```
db = client.pymongo_test
```

Або ви також можете використовувати доступ в стилі словника:

```
db = client['pymongo_test']
```

Насправді не має значення, чи створена зазначена вами база даних. Вказавши це ім'я бази даних і зберігши в ньому дані, ви автоматично створите базу даних.

Вставка документів

Зберігати дані в базі даних так само просто, як викликати всього дві строчки коду. Перший рядок вказує, яку колекцію ви будете використовувати (повідомлення в прикладі нижче). У термінології MongoDB колекція - це група документів, які зберігаються разом в базі даних. Колекції та документи схожі на таблиці і рядки SQL відповідно. Отримати колекцію так само просто, як отримати базу даних.

У другому рядку ви фактично вставляєте дані в колекцію за допомогою методу insert_ one( ).

```
posts = db.posts
post_data = {
    'title' : 'Python and MongoDB' ,
    'content' : 'PyMongo is fun, you guys' ,
    'author' : 'Scott'
}
result = posts.insert_one(post_data)
print ( 'One post: {0} '.format(result.inserted_id))
```

Ми навіть можемо вставляти багато документів за раз, що набагато швидше, ніж використання insert_one (), якщо у вас є багато документів для додавання в базу даних. Тут використовується метод insert_many (). Цей метод приймає масив даних документа:

```
post_1 = {
   'title' : 'Python and MongoDB' ,
   'content' : 'PyMongo is fun, you guys' ,
   'author' : 'Scott'
}
post_2 = {
   'title' : 'Virtual Environments' ,
   'content' : 'Use virtual environments, you guys' ,
   'author' : 'Scott'
}
post_3 = {
   'title' : 'Learning Python' ,
   'content' : 'Learn Python, it is easy' ,
   'author' : 'Bill'
}
new_result = posts.insert_many([post_1, post_2, post_3])
print ( 'Multiple posts: {0}'.format(new_result.inserted_ids))
```

При запуску ви повинні побачити щось на зразок :

```
One post: 584d947dea542a13e9ec7ae6
Multiple posts: [
   ObjectId ( '584d947dea542a13e9ec7ae7'),
   ObjectId ( '584d947dea542a13e9ec7ae8'),
   ObjectId ( '584d947dea542a13e9ec7ae9')
]
```

ПРИМІТКА. Не турбуйтеся, що ваші ObjectIds не відповідають зазначеним вище. Вони динамічно генеруються при вставці даних і складаються з епохи Unix, ідентифікатора комп'ютера й інших унікальних даних.

Отримання документів

Щоб отримати документ, ми будемо використовувати метод find_ one( ). Єдиний аргумент, який ми тут будемо використовувати (хоча він підтримує набагато більше), - це словник, який містить поля для зіставлення. У нашому прикладі нижче ми хочемо отримати повідомлення, написане Біллом:

```
bills_post = posts.find_one({ 'author' : 'Bill' })
print(bills_post)
```

Запустіть це.

```
{
   'Author': 'Bill',
   'Title': 'Learning Python',
   'Content': 'Learn Python, it is easy',
   '_Id': ObjectId ( '584c4afdea542a766d254241')
}
```

Можливо, ви помітили, що ObjectId поста задається за допомогою ключа _id, який ми пізніше можемо використовувати для його унікальної ідентифікації при необхідності. Якщо ми хочемо знайти більш одного документа, ми можемо використовувати метод find(). На цей раз давайте знайдемо всі повідомлення, написані Скоттом.

```
scotts_posts = posts.find ({ 'author' : 'Scott' })
print(scotts_posts)
```

Запустіть.

```
<Pymongo.cursor.Cursor object at 0x109852f98>
```

Основна відмінність тут в тому, що дані документа не повертаються нам безпосередньо у вигляді масиву. Замість цього ми отримуємо примірник об'єкта Cursor. Цей Cursor є повторюваний об'єкт, який містить досить багато допоміжних методів, які допомагають вам працювати з даними. Щоб отримати кожен документ, просто перебирайте результат:

```
for post in scotts_posts:
    print(post)
```

MongoEngine

Хоча PyMongo дуже простий у використанні й в цілому відмінна бібліотека, він, ймовірно, занадто низькорівневий для багатьох проектів. Іншими словами, вам доведеться написати багато власного коду для послідовного збереження, вилучення та видалення об'єктів.

Одна з бібліотек, яка забезпечує більш високу абстракцію поверх PyMongo, - це MongoEngine. MongoEngine – це засіб зіставлення об'єктних документів (ODM), яке приблизно еквівалентно об'єктно-реляційному співставленню (ORM) на основі SQL. Абстракція, що надається MongoEngine, заснована на класах, тому всі створювані вами моделі є класами.

Хоча існує досить багато бібліотек Python, які допоможуть вам працювати з MongoDB, MongoEngine - одна з кращих, оскільки вона має гарне поєднання функцій, гнучкості та підтримки спільноти, але при цьому не занадто самовпевнена.

Для установки використовуйте pip:

```
$ Pip install mongoengine == 0.10.7
```

Після установки нам потрібно направити бібліотеку для підключення до нашого запущеного екземпляру Mongo. Для цього нам потрібно буде використовувати функцію connect() і передати їй хост і порт бази даних MongoDB. В оболонці Python введіть (Додаток 19):

```
from mongoengine import *
connect( 'mongoengine_test' , host = 'localhost' , port = 27017 )
```

Тут ми вказуємо ім'я нашої бази даних і місцезнаходження. Оскільки ми все ще використовуємо хост і порт за замовчуванням, ви можете не вказувати ці параметри.

Визначення документа

Щоб налаштувати наш об'єкт документа, нам потрібно визначити, які дані ми хочемо, щоб наш об'єкт документа мав. Як і в багатьох інших ORM, ми зробимо це шляхом створення підкласу від класу Document і надання потрібних нам типів даних:

```
import datetime


class Post (Document):
    title = StringField (required = True , max_length = 200 )
    content = StringField (required = True )
    author = StringField (required = True , max_length = 50 )
    published = DateTimeField (default = datetime.datetime.now)
```

ПРИМІТКА. Однією з найбільш складних завдань з моделями баз даних є перевірка даних. Як переконатися, що зберігаються дані відповідіні до необхідного формату? Той факт, що база даних не містить схеми, не означає, що вона вільна від схеми.

У цій простій моделі ми повідомили MongoEngine, що очікуємо, що екземпляр Post матиме заголовок, контент, автора і дату публікації. Тепер базовий об'єкт Document може використовувати цю інформацію для перевірки даних, які ми йому надаємо.

Так, наприклад, якщо ми спробуємо зберегти повідомлення без заголовка, воно видасть виняток і повідомить нам про це. Ми можемо піти ще далі і додати більше обмежень, наприклад довжину рядка. Зверніть увагу, що

для деяких полів заданий параметр max_length. Це вказує Документу, як ви, мабуть, здогадалися, дозволити тільки максимальну довжину рядка, скільки символів ми вкажемо. Ми можемо встановити ще кілька таких параметрів, в тому числі:

- db_field: вкажіть інше ім'я поля.
- required : перевірка, що це поле встановлено.
- default: використовувати дане значення за замовчуванням, якщо не вказано інше значення.
- unique : перевірка, що жоден інший документ в колекції не має такого ж значення для цього поля.
- choices: перевірка, що значення поля дорівнює одному зі значень, зазначених в масиві.

Кожен тип поля має свій власний набір параметрів, тому не забудьте перевірити документацію для отримання додаткової інформації.

Збереження документів

Щоб зберегти документ в нашій базі даних, ми будемо використовувати метод save(). Якщо документ вже існує в базі даних, то всі зміни будуть внесені в існуючий документ на атомарному рівні. Однак, якщо його не існує, він буде створений.

Ось приклад створення і збереження документа:

```
post_1 = Post (
    title = 'Sample Post' ,
    content = 'Some engaging content' ,
    author = 'Scott'
)
post_1.save () # This will perform an insert
print(post_1.title)
post_1.title = 'A Better Post Title'
```

```
post_1.save() # This will perform an atomic edit on "title"
print(post_1.title)
```

Кілька зауважень з приводу виклику save():

- PyMongo виконає перевірку при виклику save(). Це означає, що він буде порівнювати дані для збереження зі схемою, оголошеної в класі. Якщо схема (або обмеження) порушується, генерується виняток і дані не зберігаються.
- Оскільки Mongo не підтримує справжні транзакції, немає способу «відкотити» виклик save(), як в базах даних SQL. Хоча ви можете наблизитися до виконання транзакцій з двоетапними комітами, вони як і раніше не підтримують відкати.

Що станеться, якщо ви пропустите назва?

```
post_2 = Post(content = 'Content goes here' , author = 'Michael' )
post_2.save()
```

Ви повинні побачити наступне виняток:

```
raise ValidationError (message, errors = errors)
mongoengine.errors.ValidationError:
ValidationError(Post: None ) (Field is required: [ 'title' ])
```

Об'єктно-орієнтовані функції

Оскільки MongoEngine є об'єктно-орієнтованим , ви також можете додавати методи в свій під-класовий документ. Розглянемо наступний приклад, в якому функція використовується для зміни набору запитів за замовчуванням (який повертає всі об'єкти колекції). Використовуючи це, ми можемо застосувати до класу фільтр за замовчуванням і отримати тільки бажані об'єкти:

```
class Post (Document):
    title = StringField ()
    published = BooleanField ()
```

```
    @queryset_manager
    def live_posts (clazz, queryset):
        return queryset.filter(published = True )
```

Посилання на інші документи

Ви також можете використовувати об'єкт ReferenceField для створення посилання з одного документа на інший. MongoEngine автоматично обробляє ліниве розйменування при доступі, що більш надійно і менш схильно до помилок, ніж необхідність не забувати робити це самостійно всюди у вашому коді. Приклад:

```
class Author (Document):
    name = StringField ()

class Post (Document):
    author = ReferenceField (Author)

Post.objects.first().author.name
```

У наведеному вище коді, використовуючи посилання на документ, ми можемо легко знайти автора першого повідомлення.

Класів полів (і параметрів) набагато більше, ніж те, що ми представили тут, тому обов'язково ознайомтеся з документацією по Полям для отримання додаткової інформації.

З усіх цих прикладів ви повинні побачити, що MongoEngine добре підходить для управління об'єктами вашої бази даних практично для будь-якого типу програм. Використання опцій в розпорядженні розробника, дозволяють неймовірно легко створити ефективну і масштабовану програму. Якщо вам потрібна додаткова допомога, пов'язана з MongoEngine , обов'язково ознайомтеся з її докладним керівництвом користувача .

Висновок

Оскільки Python є високо-рівневою, добре масштабованою сучасною мовою, йому потрібна база даних (і драйвер), які можуть відповідати його потенціалу, тому MongoDB так добре підходить. У цій роботі ми побачили, як можна використовувати сильні сторони MongoDB в своїх інтересах і створити дуже гнучку і масштабовану програму.

P . S . До роботі додається скелет проекту Python + MongoDB з повним вихідним кодом, який показує, як отримати доступ до MongoDB з Python.

Практичне завдання

Повторити та модифікувати програму.

Практична робота №23

Розробка: студент гр.КНД-32 Єгорін М.С.

Тема. Управління даними за допомогою Python, SQLite та SQLAlchemy

Мета роботи. Ознайомитись з управлінням даними за допомогою Python, SQLite та SQLAlchemy.

Зміст.

25. Вивчення відомостей про управління даними за допомогою Python, SQLite та SQLAlchemy.
26. Виконання роботи.
27. Отримання результату.

Ключові положення

Всі програми в тій або іншій формі займаються обробкою інформації. Багатьом з них необхідно зберігати ці дані і згодом отримувати. У невеликих

проектах зберігати інформацію можна в одному файлі без необхідності використання сервера баз даних. З такими завданнями цілком можна впоратися засобами Python, SQLite і SQLAlchemy.

Аналогічних результатів можна досягти, використовуючи формати CSV, JSON, XML або навіть кастомні рішення. Перевагою таких текстових сховищ є легкий для сприйняття формат даних - зазвичай структура зрозуміла з першого погляду. Нижче ми порівнюємо такий спосіб зберігання і обробки даних із застосуванням реляційних баз даних і мови SQL, а також розберемося, який метод краще підходить для вашої програми. У загальних рисах ми розглянемо такі питання:

- як використовувати плоскі файли і SQLite для зберігання даних;
- як SQL дозволяє полегшити доступ до даних;
- як застосовувати SQLAlchemy для роботи з даними в формі об'єктів Python.

Плоскі бази даних

Під плоскою базою даних (плоскою таблицею) будемо розуміти файл, який містить дані без внутрішньої ієрархії та посилань на зовнішні файли. Це пояснює ряд особливостей: в таких базах даних немає необхідності використовувати фіксовану ширину полів, а csv-файли (англ.*comma separated values*) представляють собою рядки простого тексту, в яких елементи даних розділені комами. Кожен рядок тексту представляє собою рядок даних, а кожне значення в рядку, відокремлене від решти комою, відповідає одному з полів таблиці.

/data/author_book_publisher.csv (Додаток 20)

```
first_name,last_name,title,publisher
Isaac,Asimov,Foundation,Random House
Pearl,Buck,The Good Earth,Random House
Pearl,Buck,The Good Earth,Simon & Schuster
Tom,Clancy,The Hunt For Red October,Berkley
Tom,Clancy,Patriot Games,Simon & Schuster
Stephen,King,It,Random House
Stephen,King,It,Penguin Random House
Stephen,King,Dead Zone,Random House
Stephen,King,The Shining,Penguin Random House
John,Le Carre,"Tinker, Tailor, Solider, Spy: A George Smiley Novel",Berkley
Alex,Michaelides,The Silent Patient,Simon & Schuster
Carol,Shaben,Into The Abyss,Simon & Schuster
```

У першому рядку представленого прикладу знаходиться список полів - імена стовпців даних. Кожний наступний рядок відповідає одному запису.

Переваги плоских баз даних

Невеликі плоскі бази даних легко створювати і коригувати за допомогою текстового редактора, не складає труднощів знайти невідповідність або іншу проблему. Багато додатків вміють їх імпортувати і експортувати. В Excel можна перетворити csv-файл в електронну таблицю і назад.

Ще одна перевага плоских файлів - вони автономні, даними в такій формі легко поділитися. Практично в будь-якій мові програмування є інструменти і бібліотеки для роботи з csv-файлами. Python має вбудований модуль csv і потужну сторонню бібліотеку pandas.

Недоліки плоских баз даних

Переваги плоских баз даних тьмяніють при збільшенні обсягу інформації. Файли продовжують читатися, але редагування і пошук окремих записів викликають труднощі. І не тільки для людини - файл розростається, і його обробка вимагає обчислювальних ресурсів.

Інші труднощі, пов'язані із застосуванням плоских файлів - потрібно явно створювати і підтримувати зв’язки між компонентами даних в рамках одного файлу. Щоб описати нові відносини, доводиться писати додатковий текст, створювати нові поля.

Ще одна складність - люди, з якими ми хочемо поділитися файлом даних, також повинні знати про структури і зв’язки в цьому файлі. Щоб отримати доступ до інформації, користувачі нерідко повинні розуміти не тільки структуру даних, але і супроводжуючі інструменти програмування.

Приклад роботи з плоскою базою даних

Як приклад розглянемо описаний вище файл author_book_publisher.csv, що містить список авторів, опублікованих ними книг і видавництв. Для простоти будемо вважати, що всі автори, книги і видавництва унікальні, і всі книги пишуться автором без співавторів.

Розглянемо функцію main() програми, що знаходиться в згаданому репозиторії за відносною адресою examples/example_1/main.py

examples/example_1/main.py (Додаток 21)

```
def main():
    """Основна точка входу в програму"""
    # Отримаємо ресурси для програми
    with resources.path(
        "project.data", "author_book_publisher.csv"
    ) as filepath:
        data = get_data(filepath)

    # Дізнаємося кількість книг, надрукованих кожним видавництвом
    books_by_publisher = get_books_by_publisher(data, ascending=False)
    for publisher, total_books in books_by_publisher.items():
        print(f"Publisher: {publisher}, total books: {total_books}")
    print()
```

```
# Дізнаємося кількість авторів у кожного з видавництв
authors_by_publisher = get_authors_by_publisher(data, ascending=False)
for publisher, total_authors in authors_by_publisher.items():
    print(f"Publisher: {publisher}, total authors: {total_authors}")
print()

# Виведення ієрархічних даних про авторів
output_author_hierarchy(data)

# Додаємо нову книгу до структури даних
data = add_new_book(
    data,
    author_name="Stephen King",
    book_title="The Stand",
    publisher_name="Random House",
)

# Виводимо оновлені дані в ієрархічному виді
output_author_hierarchy(data)
```

При запуску програми для оновленого csv-файлу виводиться наступний результат:

```
$ python main.py
Publisher: Simon & Schuster, total books: 4
Publisher: Random House, total books: 4
Publisher: Penguin Random House, total books: 2
Publisher: Berkley, total books: 2
```

```
Publisher: Simon & Schuster, total authors: 4
Publisher: Random House, total authors: 3
Publisher: Berkley, total authors: 2
Publisher: Penguin Random House, total authors: 1

Authors
├── Alex Michaelides
│   └── The Silent Patient
│       └── Simon & Schuster
├── Carol Shaben
│   └── Into The Abyss
│       └── Simon & Schuster
├── Isaac Asimov
│   └── Foundation
│       └── Random House
├── John Le Carre
│   └── Tinker, Tailor, Solider, Spy: A George Smiley Novel
│       └── Berkley
├── Pearl Buck
│   └── The Good Earth
│       ├── Random House
│       └── Simon & Schuster
├── Stephen King
│   ├── Dead Zone
│   │   └── Random House
│   ├── It
│   │   ├── Penguin Random House
│   │   └── Random House
│   └── The Shining
```

```
|       └── Penguin Random House
└── Tom Clancy
    ├── Patriot Games
    |   └── Simon & Schuster
    └── The Hunt For Red October
        └── Berkley
```

Для виконання основної частини роботи main() викликає інші функції, з якими можна ознайомитися в коді, який наведений вище. Сам додаток працює коректно і демонструє можливості бібліотеки pandas. Далі створимо програму з ідентичною функціональністю, використовуючи базу даних SQLite і бібліотеку SQLAlchemy для маніпуляції цими даними.

Використання SQLite для зберігання даних

У файлі author_book_publisher.csv у деяких даних є повтори. Наприклад, роман Перл Бак The Good Earth опублікували два різних видавця (Додаток 20).

```
Pearl,Buck,The Good Earth,Random House
Pearl,Buck,The Good Earth,Simon & Schuster
```

Уявіть, що було б, якщо файл містив більше пов'язаних даних, наприклад, відомості про автора, дату публікації, ISBN книги, адресу та номер телефону видавництва. Подібні відомості будуть дублюватися для кожного корінного елемента: автора, книги, видавництва. Така побудова не тільки надмірна, але і ускладнює процедури 1) зміни полів, пов'язаних з окремим об'єктом і 2) додавання нових властивостей.

Перераховані причини зумовлюють використання баз даних, які враховують зв’язки між прихованими в них структурами - реляційних баз даних. Важливою темою в цьому плані є нормалізація бази даних - приведення

структури до виду, що забезпечує мінімальну логічну надмірність. Коли структура бази даних розширюється новими типами даних, попередня нормалізація зводить до мінімуму зміни існуючої структури.

SQLite є системою керування базами даних, доступною в стандартній бібліотеці Python. Для роботи з нею не потрібен окремий сервер, формат файлу є кросплатформним і доступний в інших мовах програмування, що підтримують SQLite.

Створюємо структуру бази даних

Реляційні бази даних дозволяють зберігати структуровані дані в вигляді пов'язаних між собою таблиць. В якості основного способу взаємодії з даними використовується мова запитів SQL.

SQL - це декларативна мова, що використовується для створення, управління і пошуку даних, що містяться в базі. Декларативна мова описує, що має бути виконано, а не те, як це потрібно зробити. Ми побачимо приклади операторів SQL пізніше, коли перейдемо до створення таблиць бази даних.

Щоб скористатися перевагами SQL, нам потрібно провести нормалізацію бази даних з файлу author_book_publisher.csv (Додаток 20). Для цього ми перенесемо авторів, книги і видавництва в окремі таблиці бази даних.

Дані в цьому форматі зберігаються у вигляді двовимірних табличних структур. Дані, що містяться в полях таблиці, відносяться до заздалегідь певних типів: текст, цілі числа, числа з плаваючою комою і т.д. Це відрізняє бази даних від csv-файлів, де всі поля початково є текстовими і для розпізнавання типу повинні бути проаналізовані програмно .

Кожен запис в таблиці має первинний ключ (англ. *primary key*), визначений для присвоєння запису унікального ідентифікатора. Первинний ключ схожий на ключ в словнику Python. Двигун бази даних зазвичай сам генерує цілочисельний первинний ключ, збільшуючи значення на одиницю для кожного нового запису. Якщо дані, що зберігаються в полі, унікальні серед всіх інших даних в цьому полі, це значення також може використовуватися, як

первинний ключ. Наприклад, в таблиці, що містить дані про книги, в якості первинного ключа може використовуватися унікальний за своєю природою ISBN.

Створення таблиць бази даних

Нижче представлений приклад, як за допомогою операторів SQL можна створити три таблиці, що представляють авторів, книги і видавництва:

```
CREATE TABLE author (
    author_id INTEGER NOT NULL PRIMARY KEY,
    first_name VARCHAR,
    last_name VARCHAR
);

CREATE TABLE book (
    book_id INTEGER NOT NULL PRIMARY KEY,
    author_id INTEGER REFERENCES author,
    title VARCHAR
);

CREATE TABLE publisher (
    publisher_id INTEGER NOT NULL PRIMARY KEY,
    name VARCHAR
);
```

Зверніть увагу, що тут немає ні файлових операцій, ні змінних, ні структур для їх зберігання. Описується тільки бажаний результат: створення таблиці з певними атрибутами. Механізм бази даних визначає, як це зробити.

Уявімо, що далі ми зробили SQL-запити, щоб заповнити таблиці бази даних інформацією. Наступний оператор використовує знак *, щоб отримати і вивести всі дані з таблиці авторів:

```
SELECT * FROM author;
```

Ви можете використовувати інструмент командного рядка sqlite3 для взаємодії з файлом бази даних author_book_publisher.db:

```
sqlite3 author_book_publisher.db
```

Нижче показаний результат роботи зазначеної команди SQL і її висновок, за якою слідує команда .q для виходу з програми:

```
sqlite> SELECT * FROM author;
1|Isaac|Asimov
2|Pearl|Buck
3|Tom|Clancy
4|Stephen|King
5|John|Le Carre
6|Alex|Michaelides
7|Carol|Shaben

sqlite> .q
```

Зверніть увагу, що на відміну від csv-файлу кожен автор присутній в таблиці тільки один раз.

Маніпуляції даними за допомогою SQL

SQL надає різні способи роботи з базами даних і таблицями, додавання нових даних, оновлення та видалення вже існуючих. Приклад оператора SQL для вставки нового учасника в таблицю author:

```
INSERT INTO author
```

```
   (first_name, last_name)
VALUES ('Paul', 'Mendez');
```

Оператор INSERT вставляє строкові значення Paul і Mendez в відповідні стовпці first_name і last_name таблиці author.

Зверніть увагу, що стовпець author_id не вказано. Оскільки цей стовпець є первинним ключем, механізм бази даних сам генерує і додає значення.

Оновлення записів в таблиці бази даних - також не складний процес. Наприклад, припустимо, що Стівен Кінг хотів, щоб його знали під псевдонімом Річард Бахман:

```
UPDATE author
SET first_name = 'Richard', last_name = 'Bachman'
WHERE first_name = 'Stephen' AND last_name = 'King';
```

Оператор SQL знаходить запис за допомогою умовного оператора WHERE, а потім оновлює поля first_name і last_name свіжими значеннями. Знак рівності (=) в SQL використовується і для порівняння, і для присвоєння.

Приклад оператора SQL для видалення запису з таблиці авторів:

```
DELETE FROM author
WHERE first_name = 'Paul'
AND last_name = 'Mendez';
```

Будуємо відносини: «один до багатьох»

Дані у файлі author_book_publisher.csv відображають дані і зв'язки шляхом дублювання даних. База даних SQLite розбиває дані на три таблиці (author, book, and publisher) і встановлює між ними відносини.

Зв'язок «один до багатьох» схоже на зв'язок покупця з товарами в Інтернеті. У одного покупця може бути багато замовлень, але кожне замовлення належить одному покупцеві. У базі даних author_book_publisher.db зв'язок «один до багатьох» представлена відношенням авторів і книг. Кожен автор може написати багато книг, але кожна художня книга написана одним автором (в рамках даного прикладу).

Як реалізувати зв'язок «один до багатьох» між двома таблицями? Кожна таблиця в базі даних має поле первинного ключа, зазвичай назване за шаблоном <ім'я таблиці> _id. Крім того, таблиця book містить поле author_id, яке посилається на таблицю author (див. Вище SQL-запит для створення таблиць). Таким чином, поле author_id встановлює зв'язок «один до багатьох» між авторами та книгами, яка виглядає наступним чином.

Зв’язок між таблицями author і book

Наведена вище діаграма являє собою просту діаграму відносин сутностей (ERD), створену за допомогою програми JetBrains DataGrip. Два графічних елемента додають інформацію про зв'язки:

1. **Значки ключів** позначають первинний (жовтий колір) і зовнішній (блакитний колір) ключі.
2. **Стрілка** вказує на зв'язок між таблицями на основі зовнішнього ключа author_id.

Щоб вивести дві таблиці разом, використовуємо SQL-оператор JOIN:

```
sqlite> SELECT
   ...> a.first_name || ' ' || a.last_name AS author_name,
```

```
   ...> b.title AS book_title
   ...> FROM author a
   ...> JOIN book b ON b.author_id = a.author_id
   ...> ORDER BY a.last_name ASC;
Isaac Asimov|Foundation
Pearl Buck|The Good Earth
Tom Clancy|The Hunt For Red October
Tom Clancy|Patriot Games
Stephen King|It
Stephen King|Dead Zone
Stephen King|The Shining
John Le Carre|Tinker, Tailor, Solider, Spy: A George Smiley Novel
Alex Michaelides|The Silent Patient
Carol Shaben|Into The Abyss
```

Наведений SQL-запит збирає інформацію з author і book, використовуючи встановлені між ними зв'язок. Конкатенация строк SQL дозволяє присвоїти author_name повне ім'я автора. Дані, зібрані запитом, сортуються в порядку зростання по полю last_name. Створивши окремі таблиці для авторів і книг і встановивши між ними зв'язок, ми зменшили надмірність даних. Тепер дані про автора редагуються в одному місці, дані про книги - в іншому.

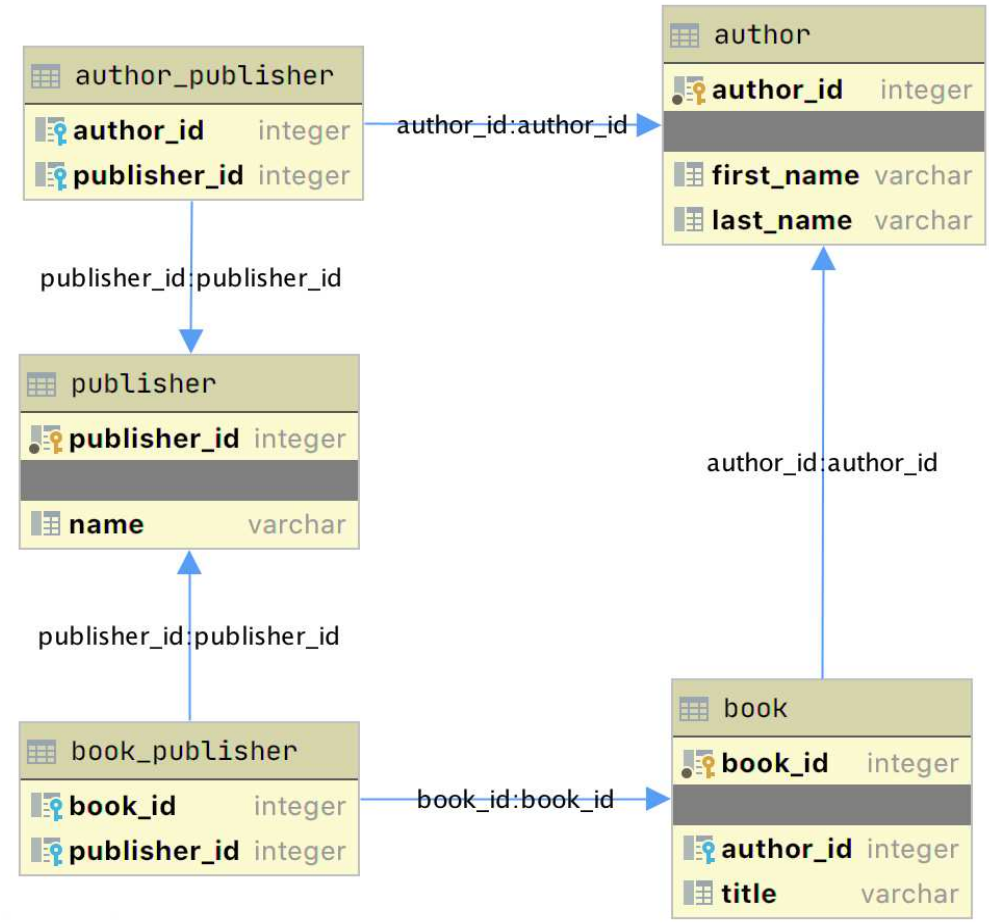
Додаємо зв'язку: «багато до багатьох»

Автор може працювати з багатьма видавцями, а видавець - з багатьма авторами. Книга може бути опублікована в кількох видавництвах, а видавець може опублікувати безліч різних книг. Тобто в базі даних author_book_publisher.db відношення «багато до багатьох» існує між авторами та видавництвами, а також між видавництвами і книгами.

Такий тип зв'язку є двостороннім і створюється за допомогою зв’язуючої таблиці. Така таблиця містить як мінімум два поля зовнішніх ключів, які є первинними ключами для кожної з двох зв'язаних таблиць:

```
CREATE TABLE author_publisher (
    author_id INTEGER REFERENCES author,
    publisher_id INTEGER REFERENCES publisher
);
```

У цьому прикладі створюється нова таблиця author_publisher, яка посилається на первинні ключі вже існуючих таблиць author і publisher. Тобто таблиця author_publisher встановлює відносини між автором і видавцем. Аналогічно створюється таблиця book_publisher.



Відносини між усіма необхідними таблицями

Оскільки зв'язок встановлюється між двома первинними ключами, немає необхідності створювати первинний ключ для самої таблиці зв'язків. Комбінація двох пов'язаних ключів створює унікальний ідентифікатор.

Об'єднати таблиці можна за допомогою оператора JOIN, але в разі зв'язку багато-до-багатьох потрібно використовувати оператор двічі:

1. Поєднуючи таблиці author і author_publisher
2. Поєднуючи таблиці author_publisher і publisher

Приклад SQL-запиту, що повертає список авторів і видавців, що публікують їх книги:

```
sqlite> SELECT
   ...> a.first_name || ' ' || a.last_name AS author_name,
   ...> p.name AS publisher_name
   ...> FROM author a
   ...> JOIN author_publisher ap ON ap.author_id = a.author_id
   ...> JOIN publisher p ON p.publisher_id = ap.publisher_id
   ...> ORDER BY a.last_name ASC;
Isaac Asimov|Random House
Pearl Buck|Random House
Pearl Buck|Simon & Schuster
Tom Clancy|Berkley
Tom Clancy|Simon & Schuster
Stephen King|Random House
Stephen King|Penguin Random House
John Le Carre|Berkley
Alex Michaelides|Simon & Schuster
Carol Shaben|Simon & Schuster
```

Наведемо приклад ще одного запиту, що відображає деякі можливості SQL:

```
sqlite> SELECT
   ...> a.first_name || ' ' || a.last_name AS author_name,
   ...> COUNT(b.title) AS total_books
   ...> FROM author a
   ...> JOIN book b ON b.author_id = a.author_id
   ...> GROUP BY author_name
   ...> ORDER BY total_books DESC, a.last_name ASC;
Stephen King|3
Tom Clancy|2
Isaac Asimov|1
Pearl Buck|1
John Le Carre|1
Alex Michaelides|1
Carol Shaben|1
```

Цей запит повертає список авторів і кількість написаних ними книг. Список сортується спочатку за кількістю книг в порядку убування, а потім по імені автора в алфавітному порядку. Тобто тут ми використовуємо SQL одночасно для агрегування і сортування результатів. Виконання обчислень в базі даних на основі вбудованих можливостей систем управління базами даних зазвичай відбувається швидше, ніж виконання тих же обчислень з необробленими наборами даних в Python.

Робота з SQLAlchemy і об'єктами Python

SQLAlchemy - це потужний набір Python-інструментів для доступу до баз даних. Коли ми програмуємо на об'єктно-орієнтованій мові, корисно мислити в категоріях об'єктів. Але між об'єктно-орієнтованою і реляційною моделями існує концептуальний розрив. Цей зазор покриває об'єктно-реляційне відображення (ORM), що надається SQLAlchemy. Між базою даних і

програмою Python з'являється модель, яка перетворює інформацію з потоку бази даних в об'єкти Python і назад. Таким чином, SQLAlchemy дозволяє мислити в термінах об'єктів, при цьому зберігаючи могутні механізми бази даних.

Модель

Для підключення SQLAlchemy до бази даних створюється модель - клас Python, успадкований від класу SQLAlchemy Base, що визначає відображення даних між об'єктами Python, що повертаються в результаті запиту, і самими таблицями бази даних.

Представлений нижче файл models.py створює моделі для представлення бази даних author_book_publisher.db:

models.py

```
# імпортуємо класи, які використовуються для визначення атрибутів моделі
from sqlalchemy import Column, Integer, String, ForeignKey, Table

# імпортуємо об'єкти для створення відносини між об'єктами
from sqlalchemy.orm import relationship, backref

# об'єкт для підключення ядра бази даних
from sqlalchemy.ext.declarative import declarative_base

# створюємо клас, від якого будуть успадковуватися моделі
Base = declarative_base()

# створюємо модель таблиці зв'язків таблиць author і publisher
author_publisher = Table(
    "author_publisher",
    Base.metadata,
```

```
    Column("author_id", Integer, ForeignKey("author.author_id")),
    Column("publisher_id", Integer, ForeignKey("publisher.publisher_id")),
)

# створюємо модель таблиці зв'язків таблиць book і publisher
book_publisher = Table(
    "book_publisher",
    Base.metadata,
    Column("book_id", Integer, ForeignKey("book.book_id")),
    Column("publisher_id", Integer, ForeignKey("publisher.publisher_id")),
)

# визначаємо моделі класів для автора, книги і видавництва
class Author(Base):
    __tablename__ = "author"
    author_id = Column(Integer, primary_key=True)
    first_name = Column(String)
    last_name = Column(String)
    books = relationship("Book", backref=backref("author"))
    publishers = relationship(
        "Publisher", secondary=author_publisher, back_populates="authors"
    )

class Book(Base):
    __tablename__ = "book"
    book_id = Column(Integer, primary_key=True)
    author_id = Column(Integer, ForeignKey("author.author_id"))
    title = Column(String)
    publishers = relationship(
```

```
        "Publisher", secondary=book_publisher, back_populates="books"
    )

class Publisher(Base):
    __tablename__ = "publisher"
    publisher_id = Column(Integer, primary_key=True)
    name = Column(String)
    authors = relationship(
        "Author", secondary=author_publisher, back_populates="publishers"
    )
    books = relationship(
        "Book", secondary=book_publisher, back_populates="publishers"
    )
```

У наведених в коді коментарях моделі зіставлені по п'яти вказаним вище таблицям бази даних author_book_publisher.db.

Клас Table служить для створення зв’язуючої таблиці, тобто для створення відносин багато-до-багатьох. Перший параметр - ім'я таблиці, певне в базі даних, другий (Base.metadata) забезпечує зв'язок між функціональністю SQLAlchemy і механізмами бази даних. Інші параметри є екземплярами класу Column, що визначає поле таблиці по імені, типу, а також відповідності зовнішньому ключу.

Клас ForeignKey визначає взаємозв'язок між двома полями Column в різних таблицях. Наприклад, наступний рядок у визначенні таблиці author_publisher повідомляє SQLAlchemy, що в таблиці author_publisher є поле з ім'ям author_id, тип цього поля - Integer, і він є зовнішнім ключем, пов'язаним з первинним ключем в таблиці author.

```
Column("author_id", Integer, ForeignKey("author.author_id"))
```

Функція relationship визначає ставлення один-ко-многим.

```
books = relationship("Book", backref=backref("author"))
```

Перший параметр функції relationship - ім'я класу Book (не плутайте з ім'ям таблиці book) - це клас, до якого належить атрибут books. Це ставлення повідомляє SQLAlchemy, що існує зв'язок між класами Author і Book. SQLAlchemy знайде зв'язок у визначенні класу Book:

```
author_id = Column(Integer, ForeignKey("author.author_id"))
```

SQLAlchemy розпізнає, що це точка з'єднання ForeignKey між двома класами. Параметр backref створює атрибут author для кожного екземпляра Book. Цей атрибут посилається на Author, з яким пов'язаний екземпляр Book. Наприклад, якщо виконати наступний код Python, то в результаті запиту буде повернутий екземпляр Book, що має атрибути, які можна використовувати для виведення інформації про книгу:

```
book = session.query(Book).filter_by(Book.title == "The
Stand").one_or_none()
print(f"Authors name: {book.author.first_name} {book.author.last_name}")
```

Наявність атрибута author в book пов'язано з визначенням backref. Зворотнє посилання дуже зручне, коли нам потрібно звернутися до батьківського об'єкту, а все, що у нас є, - це дочірній екземпляр.
Інший зв'язок у Author з класом Publisher:

```
publishers = relationship(
    "Publisher", secondary=author_publisher, back_populates="authors"
)

# для порівняння
books = relationship("Book", backref=backref("author"))
```

Як і книги, атрибут publishers позначає сукупність видавців, пов'язаних з автором. Перший параметр, "Publisher", повідомляє SQLAlchemy, що це за клас.

Другий параметр secondary повідомляє SQLAlchemy, що зв'язок з класом Publisher здійснюється через «вторинну» таблицю - таблицю author_publisher. Третій параметр back_populate повідомляє SQLAlchemy про наявність додаткової колекції в класі Publisher, званої authors.

Запити до бази даних

Стандартний запит на зразок SELECT * FROM author в SQLAlchemy можна зробити ось так:

```
results = session.query(Author).all()
```

Тут session - об'єкт SQLAlchemy, який використовується для зв'язку з SQLite в програмах Python. Тут ми повідомляємо, що хочемо виконати запит до моделі Author і повернути всі записи. На цьому етапі переваги використання SQLAlchemy замість простого SQL можуть бути неочевидні, особливо з огляду на необхідність створення моделей. Однак виявляється, що замість списку скалярних даних, тепер ми отримуємо список примірників об'єктів Author з атрибутами, відповідними заданим іменах стовпців.

Колекції книг і видавців, підтримувана SQLAlchemy, створює ієрархічний список авторів і написаних ними книг, а також видавців, які ці книги опублікували. За лаштунками SQLAlchemy перетворює виклики об'єктів і методів в оператори SQL для виконання в системі управління базами даних SQLite. І навпаки, SQLAlchemy перетворює дані, які повертаються запитами SQL, в об'єкти Python.

За допомогою SQLAlchemy ми можемо виконати запит агрегування, показаний раніше для списку авторів і кількості написаних книг, в такий спосіб:

```
author_book_totals = (
    session.query(
        Author.first_name,
        Author.last_name,
        func.count(Book.title).label("book_total")
```

```
    )
    .join(Book)
    .group_by(Author.last_name)
    .order_by(desc("book_total"))
    .all()
)
```

Запит session.query повертає ім'я і прізвище автора, а також кількість книг, написаних автором. Агрегуючий лічильник в інструкції group_by проводить підрахунок по прізвищу автора. Результати сортуються в порядку убування, посилаючись на очікувану змінної з псевдонімом book_total.

Приклад програми

Приклад програми examples/example_2/main.py (Додаток 23) містить ті ж функції, що і перший програмний приклад, але використовує SQLAlchemy виключно для взаємодії з базою даних SQLite author_book_publisher.db. Наведемо тут функцію main( ):

```
def main():
    """Main entry point of program"""
    # Підключення до бази даних через SQLAlchemy
    with resources.path(
        "project.data", "author_book_publisher.db"
    ) as sqlite_filepath:
        engine = create_engine(f"sqlite:///{sqlite_filepath}")
    Session = sessionmaker()
    Session.configure(bind=engine)
    session = Session()
```

```
# Визначаємо число книг, виданих кожним видавництвом
books_by_publisher = get_books_by_publishers(session, ascending=False)
for row in books_by_publisher:
    print(f"Publisher: {row.name}, total books: {row.total_books}")
print()

# Визначаємо число авторів у кожного видавництва
authors_by_publisher = get_authors_by_publishers(session)
for row in authors_by_publisher:
    print(f"Publisher: {row.name}, total authors: {row.total_authors}")
print()

# Ієрархічний висновок даних
authors = get_authors(session)
output_author_hierarchy(authors)

# Додаємо нову книгу
add_new_book(
    session,
    author_name="Stephen King",
    book_title="The Stand",
    publisher_name="Random House",
)

# Виведення оновлених відомостей
authors = get_authors(session)
output_author_hierarchy(authors)
```

У порівнянні з початковим кодом тепер весь код в файлі висловлює, що потрібно отримати або зробити, а не те, як це потрібно зробити. І на відміну від

другої частини нашої розповіді тепер замість використання SQL ми застосовуємо об'єкти і методи Python.

Надання доступу до даних декільком користувачам

До цього моменту ми дізналися, як для доступу до одних і тих же даних можуть використовуватися pandas, SQLite і SQLAlchemy. Одним з вирішальних факторів при виборі між використанням плоского файлу або бази даних є обсяг даних і кількість зв'язків між різними структурами даних. Ще один фактор, який слід враховувати, - з якою кількістю користувачів ми ділимося даними і питання критичності їх розсинхронізації. Проблема забезпечення однаковості даних зазвичай виникає, коли безліч користувачів взаємодіють з даними віддалено через мережу. Надання даних через серверний додаток і призначений для користувача інтерфейс вирішує проблему цілісності даних. У цьому випадку сервер - єдиний додаток, що має доступ до бази даних на рівні файлів. Останній приклад - закінчений веб-додаток та інтерфейс для зразка істотно більшої навчальної бази даних SQLite Chinook, що містить інформацію про музичних виконавців, альбоми, треки, жанри і т.п. (11 пов'язаних таблиць).

Використання Flask з SQLite і SQLAlchemy

Програма examples/example_3/chinook_server.py (Додаток 24) створює додаток Flask, з яким можна взаємодіяти за допомогою браузера. У додатку використовуються наступні технології:

Flask Blueprint - частина Flask для делегування завдань окремих модулів із заздалегідь визначеною функціональністю;

Flask SQLAlchemy - розширення Flask, що додає в веб-додатки підтримку SQLAlchemy;

Flask_Bootstrap4 - упаковує набір інтерфейсних інструментів Bootstrap, інтегруючи його з веб-додатками Flask;

Flask_WTF - розширює Flask з допомогою WTForms, надаючи вашим веб-додатків зручний спосіб створення і перевірки веб-форм;

python_dotenv - модуль Python, який використовується додатком для читання змінних середовища з файлу і збереження конфіденційної інформації за межами програмного коду.

Приклад програми досить великий, і лише деяка його частина має відношення до керівництва. З цієї причини вивчення коду залишено як вправа для читача. Проте ви можете подивитися представлений нижче скрінкаст, за яким слідує HTML-код шаблону домашньої сторінки і Python-скрипт, який динамічно надає дані.

HTML-шаблон Jinja2, який створює домашню сторінку програми:

```
{% extends "base.html" %}

{% block content %}
<div class="container-fluid">
  <div class="m-4">
    <div class="card" style="width: 18rem;">
      <div class="card-header">Create New Artist</div>
      <div class="card-body">
        <form method="POST" action="{{url_for('artists_bp.artists')}}">
          {{ form.csrf_token }}
```

```
        {{ render_field(form.name, placeholder=form.name.label.text) }}
        <button type="submit" class="btn btn-primary">Create</button>
      </form>
    </div>
  </div>
  <table class="table table-striped table-bordered table-hover table-sm">
    <caption>List of Artists</caption>
    <thead>
      <tr>
        <th>Artist Name</th>
      </tr>
    </thead>
    <tbody>
      {% for artist in artists %}
      <tr>
        <td>
          <a href="{{url_for('albums_bp.albums', artist_id=artist.artist_id)}}">
            {{ artist.name }}
          </a>
        </td>
      </tr>
      {% endfor %}
    </tbody>
  </table>
 </div>
</div>
{% endblock %}
```

Файл Python, відповідальний за відображення сторінки:

```python
from flask import Blueprint, render_template, redirect, url_for
from flask_wtf import FlaskForm
from wtforms import StringField
from wtforms.validators import InputRequired, ValidationError
from app import db
from app.models import Artist

# Налаштовуємо blueprint
artists_bp = Blueprint(
    "artists_bp", __name__, template_folder="templates", static_folder="static"
)

def does_artist_exist(form, field):
    artist = (
        db.session.query(Artist)
        .filter(Artist.name == field.data)
        .one_or_none()
    )
    if artist is not None:
        raise ValidationError("Artist already exists", field.data)

class CreateArtistForm(FlaskForm):
    name = StringField(
        label="Artist's Name", validators=[InputRequired(), does_artist_exist]
    )

@artists_bp.route("/")
@artists_bp.route("/artists", methods=["GET", "POST"])
```

```
def artists():
    form = CreateArtistForm()

    # Перевіряємо валідність форми
    if form.validate_on_submit():
        # "Створюємо" нового артиста
        artist = Artist(name=form.name.data)
        db.session.add(artist)
        db.session.commit()
        return redirect(url_for("artists_bp.artists"))

    artists = db.session.query(Artist).order_by(Artist.name).all()
    return render_template("artists.html", artists=artists, form=form,)
```

Висновок

Розумно задатися питанням: чи є SQLite правильним вибором в якості серверної частини бази даних для веб-додатка. На веб-сайті SQLite зазначено, що SQLite - хороший вибір для сайтів, які обслуговують близько 100 тис. Звернень в день. Якщо реалізували веб-сайт за допомогою SQLAlchemy, то можна перенести дані з SQLite в іншу базу даних, таку як MySQL або PostgreSQL. Описові моделі даних SQLAlchemy залишаться колишніми.

Практичне завдання

1. На прикладі описаних вище дій, створити свій Python проект з використанням SQLAlchemy та SQLite.

Практична робота №24

Розробка: студент гр.КНД-32 Кардіян В.М.

Тема. Створення telebot

Мета роботи. **Ознайомитись з використанням Python для створення telebot.**

Зміст.

28. Вивчення відомостей про створення telebot.
29. Виконання роботи.
30. Отримання результату.

Ключові положення

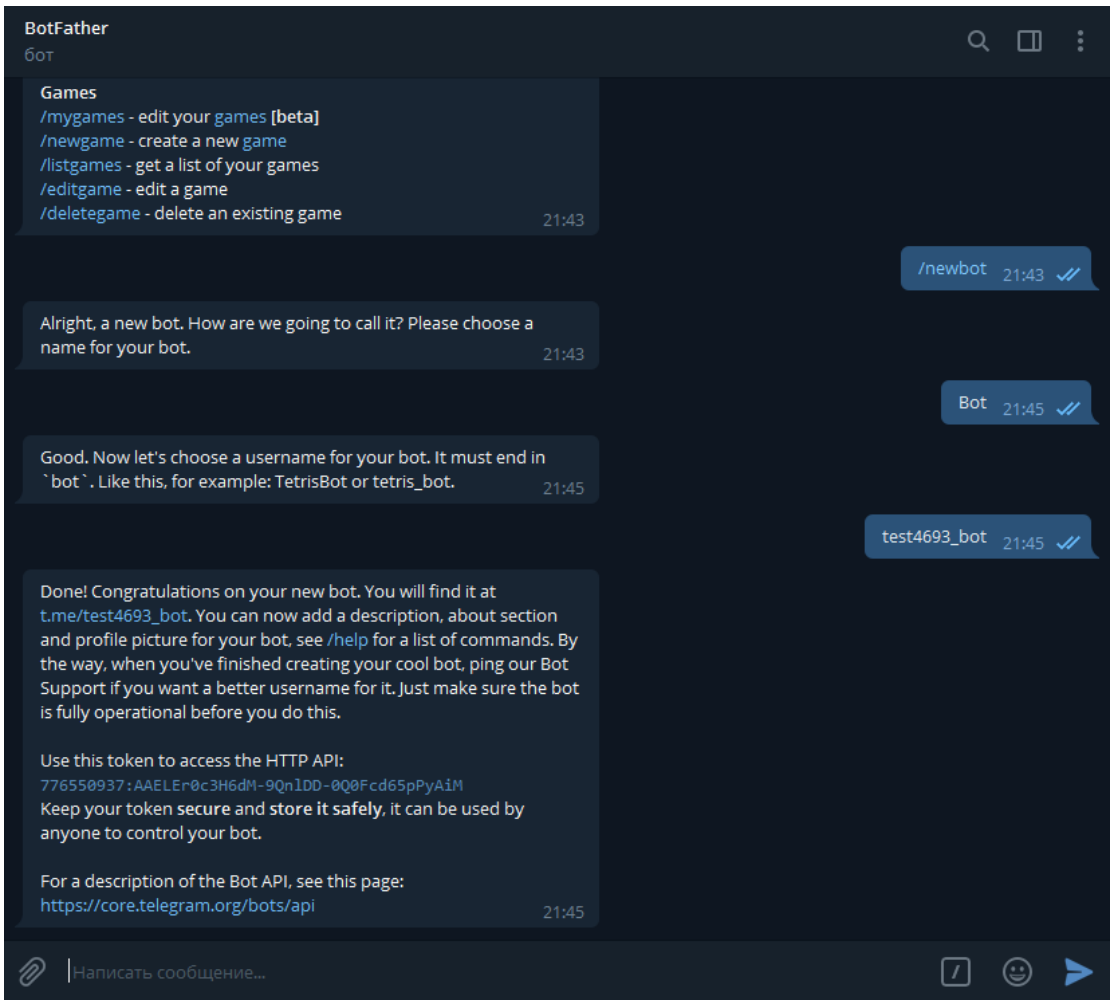
Для початку скачаємо сам python. Зробити це можна на офіційному сайті. Не забудьте поставити галочку add to PATH під час установки! Після установки python'a нам знадобиться хороший редактор коду. На допомогу приходить компанія JetBrains зі своїм безкоштовним PyCharm. Ми вже близько, залишилося завантажити бібліотеку telebot. Для цього заходимо в командний рядок і пишемо:

```
pip install pytelegrambotapi
```

Якщо все пройшло успішно, ми можемо продовжувати!

Bot Father

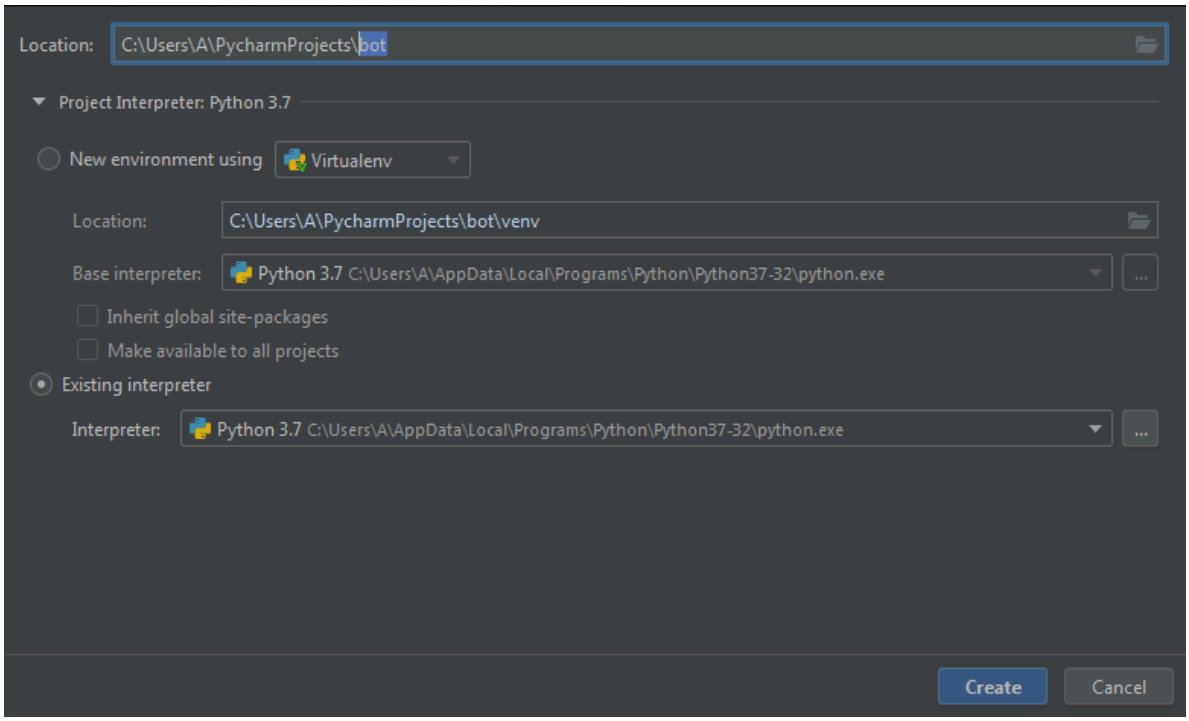
У пошуку telegram знаходимо Bot Farher'a і створюємо свого бота за допомогою команди / newbot. Потім вводимо ім'я і юзернейм. Зверніть увагу, що юзернейм повинен закінчуватися на bot!



Як ви бачите нам видали спеціальний api токен, за допомогою якого ви зможете управляти своїм ботом (в моєму випадку це: 776550937: AAELEr0c3H6dM-9QnlDD-0Q0Fcd65pPyAiM). Свій токен Ви можете запам'ятати, але я рекомендую його записати.

Код

Настав момент, якого чекали всі. Відкриваємо PyCharm і створюємо новий проект.



Тут рекомендую поставити все як у мене (назва, звичайно можна змінити). Після створення проекту, давайте створимо файл, в якому буде наш код. Клацніть правою кнопкою по папці з вашому проектом, потім New → Python File. Відмінно, почнемо писати код. Імпортуємо бібліотеку telebot, за допомогою:

```
import telebot
```

Тепер потрібно створити змінну bot. Насправді ім'я змінної може бути яким завгодно, але я звик писати bot.

```
bot = telebot.TeleBot('ваш токен')
```

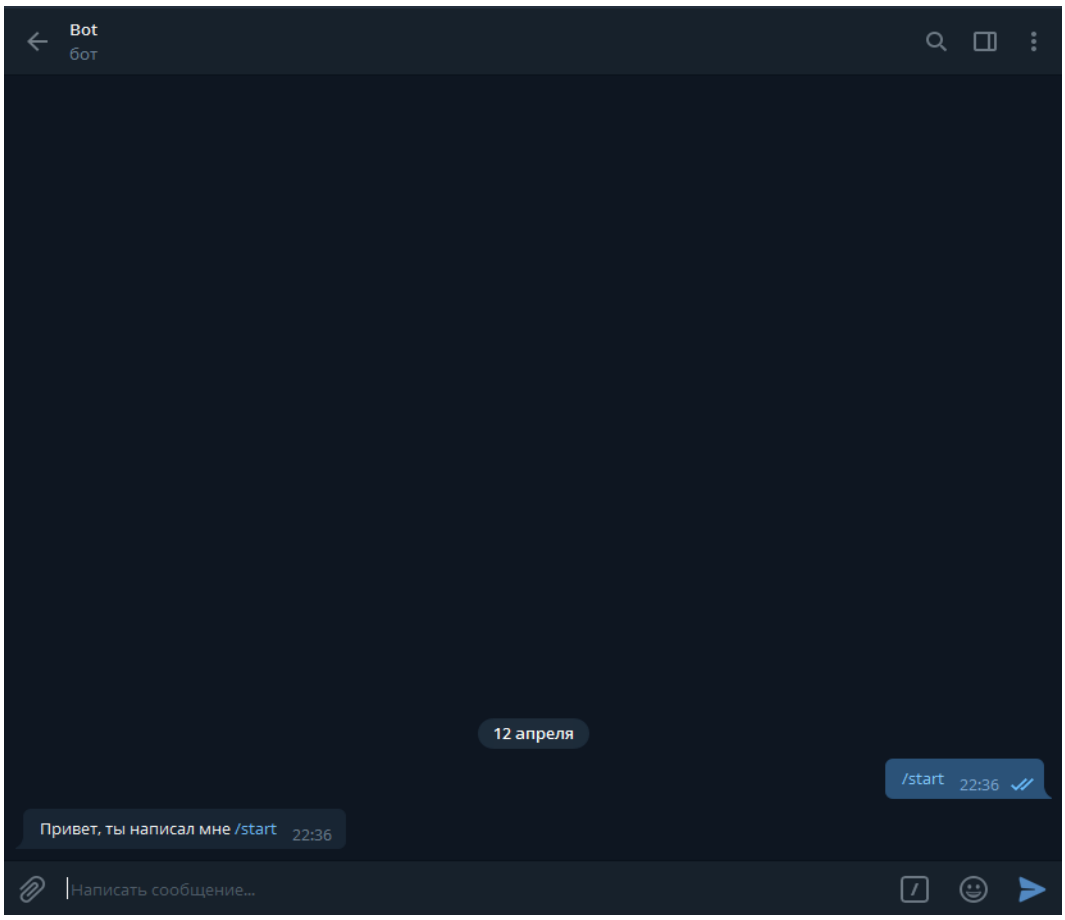
Напишемо декоратор bot.message_handler (), за допомогою якого наш бот буде реагувати на команду / start. Для цього в круглих дужках пишемо commands = ['start']. В результаті у нас повинно вийти це:

```
@bot.message_handler(commands=['start'])
```

Якщо Ви спробуєте запустити свого бота (ПКМ-> Run), то у вас нічого не вийде. По-перше в кінці коду ми повинні прописати bot.polling (). Це потрібно для того, щоб бот не вимикаючи відразу, а працював і перевіряв, чи немає на сервері нового повідомлення. А по-друге наш бот якщо вже і буде перевіряти наявність повідомлень, то все одно нічого відповісти не зможе. Після нашого декоратора створюємо функцію start_message, яка буде приймати параметр message (назва функції може бути будь-яким). Далі давайте реалізуємо відправку повідомлення від самого бота. У функції пропишемо bot.send_message (message.chat.id, 'Привіт, ти написав мені / start'). Дивіться, що у Вас повинно вийти:

```
import telebot

bot = telebot.TeleBot('776550937:AAELEr0c3H6dM-9QnlDD-0Q0Fcd65pPyAiM')

@bot.message_handler(commands=['start'])
def start_message(message):
    bot.send_message(message.chat.id, 'Привет, ты написал мне /start')

bot.polling()
```

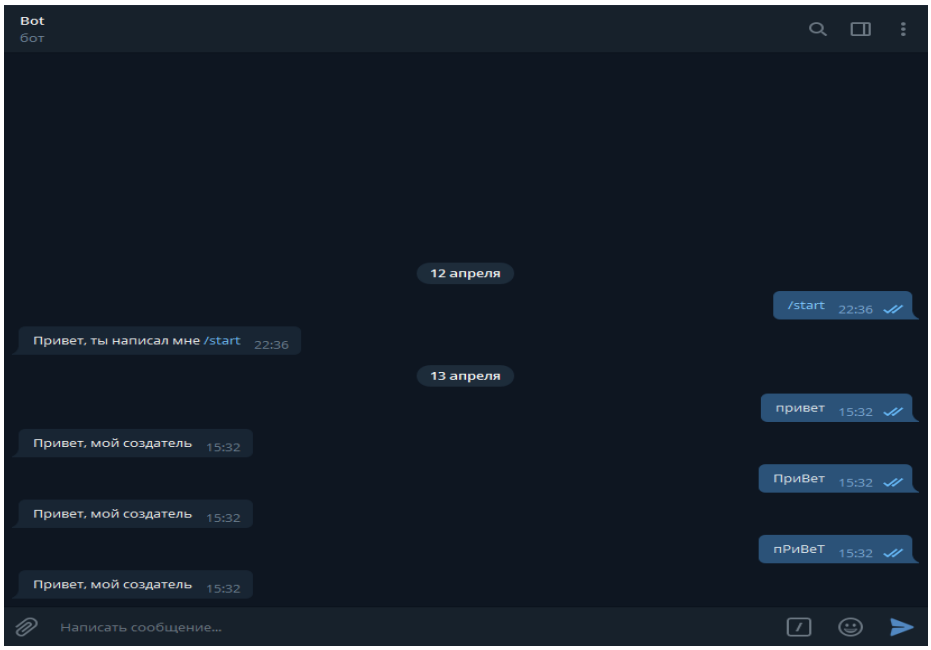
Перевіримо ...



Бот працює. Щоб він відповідав не тільки на команди, а й на повідомлення, створимо новий декоратор bot.message_handler (), а в круглі скобочки напишемо content_types = ['text']. Взагалі існує безліч видів контенту, наприклад location, photo, audio, sticker і т.д. Але нам же потрібно відповідати на текст, вірно? Тому створюємо функцію send_text, приймаючу параметр message. У функції пропишемо умову:

```
@bot.message_handler(content_types=['text'])
def send_text(message):
    if message.text == 'Привет':
        bot.send_message(message.chat.id, 'Привет, мой создатель')
    elif message.text == 'Пока':
        bot.send_message(message.chat.id, 'Прощай, создатель')
```

Якщо текст повідомлення буде дорівнює «Привіт», то бот відповідає «Привіт, мій творець», а якщо текст повідомлення буде дорівнює «Поки», то бот відповість «Прощай, творець». Тут думаю все зрозуміло. Але ви швидше за все задалися питанням, а якщо користувач пропише «привіт», ну або «Привіт», як бути в цій ситуації? Все досить просто! В умови, після message.text напишіть функцію .lower (), а в тексті усі великі літери замініть на рядкові. Тепер наш бот відповідає не тільки на «привіт», а й на «ПривеТ», і навіть «Привіт».



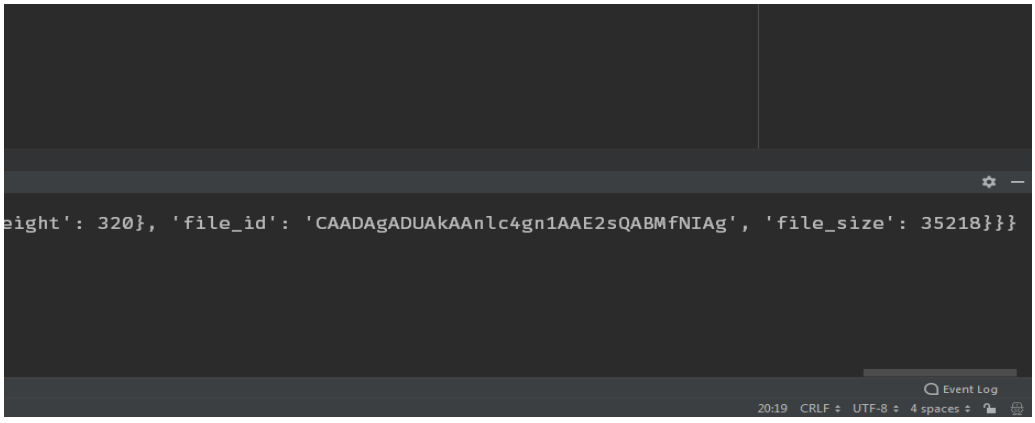
Ось що у вас повинно вийти:

```
import telebot

bot = telebot.TeleBot('776550937:AAELEr0c3H6dM-9QnlDD-0Q0Fcd65pPyAiM')

@bot.message_handler(commands=['start'])
def start_message(message):
    bot.send_message(message.chat.id, 'Привет, ты написал мне /start')

@bot.message_handler(content_types=['text'])
```

```
def send_text(message):
    if message.text.lower() == 'привет':
        bot.send_message(message.chat.id, 'Привет, мой создатель')
    elif message.text.lower() == 'пока':
        bot.send_message(message.chat.id, 'Прощай, создатель')

bot.polling()
```

Відмінно, з текстом ми розібралися, але як же відправити наприклад стікер? Все просто! У кожного стікера є свій id, відповідно знаючи id ми зможемо його відправити. Отримати id стікера можна двома способами. Перший (простий) - через спеціального бота «What's the sticker id?»

Ну і другий спосіб, для тих, хто не шукає легких шляхів. Створюємо новий декоратор bot.message_handler (), ось тільки в скобочки пишемо content_types = ['sticker']. Далі все як завжди. Створюємо функцію, приймаючу параметр message, а ось в ній пропишемо print (message). Запускаємо бота.



Дивіться, як тільки я відправив стікер, він відразу ж вивів інформацію в консоль, і в самому кінці буде наш id стікера (file_id). Давайте зробимо так, щоб коли користувач відправив боту «я тебе люблю», то бот йому відповів стікером. Створювати новий декоратор не потрібно, ми просто допишемо умова, яке було

до цього. Ось тільки замість bot.send_message () пропишемо bot.send_sticker (), а замість тексту напишемо id стікера.

Вітаю, все вийшло! Думаю як відправити аудіо, фото, і геолокацію, ви розберетеся самі. Я ж хочу показати вам, як зробити клавіатуру, яку бот покаже вам при старті. Це вже буде зробити складніше. Створюємо змінну keyboard1, в яку запишемо telebot.types.ReplyKeyboardMarkup (). Ця функція викликає клавіатуру. Далі створимо ряди, але пам'ятайте, що рядів може бути не більше 12! Для того, щоб їх створити, пишемо keyboard1.row (). В круглі скобочки запишіть все що хочете, особисто я напишу «Привіт» і «Поки». Тепер, щоб викликати клавіатуру, допишемо reply_markup = keyboard1 до функції відправки повідомлення при старті. Ось, що у вас повинно вийти:

```
keyboard1 = telebot.types.ReplyKeyboardMarkup()
keyboard1.row('Привет', 'Пока')

@bot.message_handler(commands=['start'])
def start_message(message):
    bot.send_message(message.chat.id, 'Привет, ты написал мне /start',
reply_markup=keyboard1)
```

Запускаємо бота ...

Практичне завдання

Написати програму нового бота.

Додаток 1

Програма

шаблон-копия 1бурок.py - C:/Users/alext/Desktop/Лекции Питон/шаблон-копия 1бурок....

File Edit Format Run Options Window Help

```
import random
HANGMAN_PICS= ['''
+---+
    |
    |
    |
   ===''','''
+---+
 0  |
    |
    |
   ===''','''
+---+
 0  |
 |  |
    |
   ===''','''
+---+
 0  |
/|  |
    |
   ===''','''
+---+
 0  |
/|\ |
    |
   ===''','''
+---+
 0  |
/|\ |
/   |
   ===''','''
+---+
 0  |
/|\ |
/ \ |
   ===''']
words = 'аист акула бабуин барсук бобр бык верблюд'.split()
```

```
def getRandomWord(wordList):
    wordIndex = random.randint(0, len(wordList) - 1)
    return wordList[wordIndex]

def displayBoard(missedLetters, correctLetters, secretWord):
    print(HANGMAN_PICS[len(missedLetters)])
    print()
    print('Помилкові букви:', end=' ')
    for letter in missedLetters:
        print(letter, end= ' ')
        print()
        blanks = '_' * len(secretWord)
        for i in range(len(secretWord)):
            if secretWord[i] in correctLetters:
                blanks = blanks[:i] + secretWord[i] + blanks[i+1:]
        for letter in blanks:
            print(letter, end=' ')
        print()

def getGuess(alreadyGuessed):
    while True:
        print('Введить букву.')
        guess = input()
        guess = guess.lower()
        if len(guess) !=1:
            print('Введить букву.')
        elif guess in alreadyGuessed:
            print('Ви вже називали цю букву. Назвить іншу.')
        elif guess not in 'абвгдеежзийклмнопрстуфхцчшщъьэюяы':
                print('Будь ласка, введіть ЛІТЕРУ.')
        else:
                return guess
def playAgain():
    print('Хочете зіграти ще? (так чи ні)')
    return input().lower().startswith('т')

print('Ш И Б Е Н И Ц Я')
missedLetters = ''
correctLetters = ''
secretWord = getRandomWord(words)
gameIsDone = False

while True:
    displayBoard(missedLetters, correctLetters, secretWord)
    guess = getGuess(missedLetters + correctLetters)
    if guess in secretWord:
        correctLetters = correctLetters + guess
        foundAllLetters = True
        for i in range(len(secretWord)):
            if secretWord[i] not in correctLetters:
                foundAllLetters = False
                break
            if foundAllLetters:
                print('ДА! Загадане слово - "' + secretWord + '"! Ви вгадали!')
                gameIsDone = True
```

```
    else:
            missedLetters = missedLetters + guess
            if len(missedLetters) == len(HANGMAN_PICS)-1:
                  displayBoard(missedLetters, correctLetters, secretWord)
                  print('Ви вичерпали всі спроби!\nНе вгадано букв: ' + str(len(missedLetters))
                  gameIsDone = True
    if gameIsDone:
            if playAgain():
                    missedLetters= ''
                    correctLetters = ''
                    gameIsDone = False
                    secretWord = getRandomWord(words)
            else:
                  break
```

Додаток 2

Програма

game12.py - C:/Users/alext/Desktop/Irpa/game12.py (3.7.2)

File Edit Format Run Options Window Help

```
from tkinter import *
import time
import random
tk = Tk()
tk.title('Game')
tk.resizable(0, 0)
tk.wm_attributes('-topmost', 1)
canvas = Canvas(tk, width=500, height=400, highlightthickness=0)
canvas.pack()
tk.update()
class Ball:
    def __init__(self, canvas, paddle, score, color):
        self.canvas = canvas
        self.paddle = paddle
        self.score = score
        self.id = canvas.create_oval(10,10, 25, 25, fill=color)
        self.canvas.move(self.id, 245, 100)
        starts = [-2, -1, 1, 2]
        random.shuffle(starts)
        self.x = starts[0]
        self.y = -2
        self.canvas_height = self.canvas.winfo_height()
        self.canvas_width = self.canvas.winfo_width()
        self.hit_bottom = False
    def hit_paddle(self, pos):
        paddle_pos = self.canvas.coords(self.paddle.id)
        if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
            if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
                self.score.hit()
                return True
        return False
```

```
    def draw(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
            self.y = 2
        if pos[3] >= self.canvas_height:
            self.hit_bottom = True
            canvas.create_text(250, 120, text='Ви програли', font=('Courier', 30), fill='red')
        if self.hit_paddle(pos) == True:
            self.y = -2
        if pos[0] <= 0:
            self.x = 2
        if pos[2] >= self.canvas_width:
            self.x = -2
class Paddle:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
        start_1 = [40, 60, 90, 120, 150, 180, 200]
        random.shuffle(start_1)
        self.starting_point_x = start_1[0]
        self.canvas.move(self.id, self.starting_point_x, 300)
        self.x = 0
        self.canvas_width = self.canvas.winfo_width()
        self.canvas.bind_all('<KeyPress-Right>', self.turn_right)
        self.canvas.bind_all('<KeyPress-Left>', self.turn_left)
        self.started = False
        self.canvas.bind_all('<KeyPress-Return>', self.start_game)
    def turn_right(self, event):
        self.x = 2
    def turn_left(self, event):
        self.x = -2
    def start_game(self, event):
        self.started = True
    def draw(self):


        self.canvas.move(self.id, self.x, 0)
        pos = self.canvas.coords(self.id)
        if pos[0] <= 0:
            self.x = 0
        elif pos[2] >= self.canvas_width:
            self.x = 0
class Score:
    def __init__(self, canvas, color):
        self.score = 0
        self.canvas = canvas
        self.id = canvas.create_text(450, 10, text=self.score, font=('Courier', 15), fill=color)
    def hit(self):
        self.score += 1
        self.canvas.itemconfig(self.id, text=self.score)
score = Score(canvas, 'green')
paddle = Paddle(canvas, 'White')
ball = Ball(canvas, paddle, score, 'red')
while not ball.hit_bottom:
    if paddle.started == True:
        ball.draw()
        paddle.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
time.sleep(3)
```

Додаток 3

Основні модулі пакета Pygame

Модуль	Призначення
pygame.cdrom	Доступ до CD-приводам і управління ними
pygame.cursors	Завантажує зображення курсора
pygame.display	Доступ до дисплея
pygame.draw	Малює фігури, лінії і точки
pygame.event	Управління зовнішніми подіями
pygame.font	Використовує системні шрифти
pygame.image	Завантажує і зберігає зображення
pygame.joystick	Використовує джойстики та аналогічні пристрої
pygame.key	Зчитує натискання клавіш з клавіатури
pygame.mixer	Завантажує і відтворює мелодії
pygame.mouse	Управляє мишею
pygame.movie	Відтворення відеофайлів
pygame.music	Працює з музикою і потоковим аудіо
pygame.overlay	Доступ до розширених відеозображенням
pygame	Містить функції Pygame високого рівня
pygame.rect	Управляє прямокутними областями
pygame.sndarray	Маніпулює звуковими даними
pygame.sprite	Управління рухомими зображеннями
pygame.surface	Управляє зображеннями і екраном
pygame.surfarray	Маніпулює даними пікселів зображення
pygame.time	Модуль pygame для управління часом і частотою кадрів
pygame.transform	Зміна розміру і переміщення зображень

Вікно Pygame

```
# Підключення бібліотеки PyGame
import pygame

# Ініціалізація PyGame
pygame.init()

# Вікно гри: розмір, позиція
gameScreen = pygame.display.set_mode((400, 300))

# Модуль os - позиція вікна
import os
x = 100
y = 100
os.environ['Sp_VIDEO_WINDOW_POS'] = "%d,%d" % (x,y)

# Параметры вікна
size = [500, 500]
screen = pygame.display.set_mode(size)
pygame.display.set_caption("Test drawings")
gameScreen.fill((0,0,255))
```

```
pygame.display.flip()
```

Цикл гри, вихід з гри

```
# Цикл гри
runGame = True # прапор виходу з циклу гри
while runGame:
    # Відстеження події: "закрити вікно"
    for event in pygame.event.get():
        if event.type == pygame.QUIT: runGame = False

# Вихід з гри:
pygame.quit()
```

Малювання базових елементів модуль pygame.draw

pygame.draw.rect	намалювати прямокутну форму
pygame.draw.polygon	фігуру з будь-якою кількістю сторін
pygame.draw.circle	коло навколо точки
pygame.draw.ellipse	намалювати круглу форму всередині прямокутника
pygame.draw.arc	намалювати секцію еліпса
pygame.draw.line	намалювати сегмент прямій лінії
pygame.draw.lines	для малювання декількох суміжних відрізків
pygame.draw.aaline	малювати тонку лінію
pygame.draw.aalines	намалювати пов'язану послідовність згладжених ліній

```
rect(Surface, color, Rect, wiph=0) -> Rect
polygon(Surface, color, pointlist, wiph=0) -> Rect
circle(Surface, color, pos, radius, wiph=0) -> Rect
ellipse(Surface, color, Rect, wiph=0) -> Rect
arc(Surface, color, Rect, start_angle, stop_angle, wiph=1) -> Rect
line(Surface, color, start_pos, end_pos, wiph=1) -> Rect
lines(Surface, color, closed, pointlist, wiph=1) -> Rect
aaline(Surface, color, startpos, endpos, blend=1) -> Rect
aalines(Surface, color, closed, pointlist, blend=1) -> Rect
```

Завантаження зображення

```
# Модуль pygame.image дозволяє завантажити ізображеріе з файлу і повертає об'єкт типу Surface.
pygame.image.load("путь к файлу" )

# завантажити нове зображення з файлу
load(filename) -> Surface

# Завантажити зображення (шлях до файлу для Windows)
myImage = pygame.image.load('images\\bg1.jpg')

# визначити місце розміщення
myRect = (0,0,600,400)

# вивантажити об'єкт Surface, який містить завантажене з файлу зображення (myImage), в описане місце на екрані (myRect)
```

```
screen.blit(myImage,myRect)
```

Об'єкт Rect

pygame.Rect

Pygame використовує об'єкти Rect для зберігання і маніпулювання прямокутними областями. Rect може бути створений з комбінації значень зліва, зверху, ширини і висоти. Rect також можуть бути створені з об'єктів python, які вже є Rect або мають атрибут з ім'ям «rect».

```
Rect(left, top, wiph, height) -> Rect

Rect((left, top), (wiph, height)) -> Rect

Rect(object) -> Rect
```

Методи роботи з Rect

pygame.Rect.copy	Повертає новий прямокутник, що має ту ж позицію і розмір, що і оригінал.
pygame.Rect.move	Повертає новий прямокутник, переміщуваний даними зміщенням. Аргументи x і y можуть бути будь-яким цілочисельним значенням, позитивним або негативним.
pygame.Rect.move_ip	Те ж, що і метод Rect.move (), але працює на місці.
pygame.Rect.inflate	Збільшувати або зменшувати розмір прямокутника, на місці.
pygame.Rect.inflate_ip	Збільшувати або зменшувати розмір прямокутника, на місці.
pygame.Rect.clamp	Переміщує прямокутник всередині іншого.
pygame.Rect.clamp_ip	Переміщує прямокутник всередині іншого, на місці.
pygame.Rect.clip	Обрізає прямокутник всередині іншого.
pygame.Rect.union	З'єднує два прямокутника в один.
pygame.Rect.union_ip	З'єднує два прямокутника в один, на місці.
pygame.Rect.unionall	Об'єднання багатьох прямокутників.
pygame.Rect.unionall_ip	Об'єднання багатьох прямокутників, на місці.
pygame.Rect.fit	Змінити розмір і перемістити прямокутник з огляду на співвідношення сторін.
pygame.Rect.normalize	Коригувати негативні розміри.
pygame.Rect.contains	Перевірити, чи знаходиться один прямокутник всередині іншого.
pygame.Rect.collidepoint	перевірити, чи знаходиться точка всередині прямокутника.
pygame.Rect.colliderect	Тест, чи перетинаються два прямокутника.
pygame.Rect.collidelist	Перевірити, перетинається хоч один прямокутник в

	списку.
pygame.Rect.collidelistall	Перетинаються всі прямокутники в списку.
pygame.Rect.collidedict	Перевірити, якщо один прямокутник в словнику перетинається.
pygame.Rect.collidedictall	Перетинаються всі прямокутники в словнику.

Обробка подій

Подія - це те, як Pygame повідомляє про те, що щось трапилося за межами коду програми. Події створюються, наприклад, при натисканні клавіш клавіатури, миші і розміщуються в черзі, чекаючи обробки.

Функція get в модулі pygame.event повертає остання подія, що очікує в черзі і видаляє його з черги.

Об'єкт event

Модуль pygame.event для обробки черги подій.

pygame.event.pump	Якщо ви не використовуєте інші функції подій в своїй грі, ви повинні викликати pygame.event.pump (), щоб дозволити pygame обробляти внутрішні дії.
pygame.event.get	Отримує події з черги.
pygame.event.poll	Отримати одну подію з черги.
pygame.event.wait	Чекає одиночную подію з черги.
pygame.event.peek	Перевірити, чи чекають черги події певного типу.
pygame.event.clear	Видалити всі події з черги.
pygame.event.event_name	Повертає ім'я для типу події. Рядок знаходиться в стилі WordCap.
pygame.event.set_blocked	Перевіряє, які події не дозволені в черзі.
pygame.event.set_allowed	Перевіряє, які події дозволені в черзі.
pygame.event.get_blocked	Перевірити, чи заблокований тип події з черги.
pygame.event.set_grab	Перевіряє спільне використання пристроїв введення з іншими додатками.
pygame.event.get_grab	Перевірити, чи працює програма на пристроях введення даних.
pygame.event.post	Помістити нову подію в чергу.
pygame.event.Event	Створити новий об'єкт події.
pygame.event.EventType	Об'єкт Python, що представляє подія SDL. Примірники користувальницьких подій створюються з викликом функції Event. Тип EventType не може бути безпосередньо викликаний. Примірники EventType підтримують призначення і видалення атрибутів.

Pygame відстежує всі повідомлення про події через чергу подій. Процедури в цьому модулі допомагають управляти цією чергою подій. Вхідна чергу сильно залежить від модуля відображення (display) pygame. Якщо дисплей ні инициализирован і режим відео не встановлено, чергу подій не працюватиме.

Існує безліч способів доступу до черги подій. Просто перевіряти існування подій, захоплювати їх безпосередньо з стека.

Миша

Модуль pygame.mouse для роботи з мышею

pygame.mouse.get_pressed	Отримати стан кнопок миші.
pygame.mouse.get_pos	Отримати позицію курсора миші.
pygame.mouse.get_rel	Отримати кількість рухів миші.
pygame.mouse.set_pos	Встановити позицію курсора миші.
pygame.mouse.set_visible	Приховати або показати курсор миші.
pygame.mouse.get_focused	Перевіряє, чи приймає дисплей введення миші
pygame.mouse.set_cursor	Використати картинку для курсора миші.
pygame.mouse.get_cursor	Отримати зображення для курсора миші.

Функції миші можна використовувати для отримання поточного стану пристрою миша. Ці функції також можуть змінювати курсор миші.

Коли режим відображення (display) встановлено, чергу подій почне приймати події миші. Кнопки миші генерують події pygame.MOUSEBUTTONDOWN і pygame.MOUSEBUTTONUP, коли вони натискаються і відпускаються. Ці події містять атрибут кнопки, який вказує, яка кнопка була натиснута. Колесо миші буде генерувати pygame.MOUSEBUTTONDOWN і pygame.MOUSEBUTTONUP події при прокручуванні.

Коли колесо повернене вгору, кнопка буде встановлена на 4, вниз -5. Всякий раз, коли миша переміщується, генерується подія pygame.MOUSEMOTION. Рух миші розбите на невеликі і точні події руху. У міру переміщення миші багато подій руху будуть поміщені в чергу. Події руху миші, які неправильно очищені від черги подій, є основною причиною того, що черга подій заповнюється.

Приклад. Намалювати курсор під поточною позицією миші.

```
x, y = pygame.mouse.get_pos()
x -= mouse_cursor.get_width()/2
y -= mouse_cursor.get_height()/2
screen.blit(mouse_cursor, (x, y))
```

Визначити яка кнопка була натиснута на мишці можна використовуючи значення event.button:

```
1 - left click
2 - middle click
3 - right click
4 - scroll up
5 - scroll down
```

Координати курсора при натисканні кнопки миші знаходяться в `event.pos`.

Приклад. Переміщати картинку курсором миші.

```
import pygame, sys, time
from pygame.locals import *
pygame.init()
FPS=30
fpsClock=pygame.time.Clock()
width=500
height=500
mainSurface=pygame.display.set_mode((width,height),0,32)
pygame.display.set_caption('Keyb moves')
background=pygame.image.load('images//bg1.jpg')
sprite=pygame.image.load('images//pict2.gif')

# Place image to the center of mainSurface
image_pos = ((mainSurface.get_width() - sprite.get_width())/2,
(mainSurface.get_height() - sprite.get_height())/2)
doMove = False

# game loop
while True:
    fpsClock.tick(FPS) # frame rate
    mainSurface.blit(background,(0,0))

    # get all events from the queue
    for event in pygame.event.get():
        # loop events queue
        if event.type == QUIT:
            # window close X pressed
            pygame.quit()
            sys.exit()

        if event.type == KEYDOWN and event.key == K_ESCAPE:
            # ESC key pressed
            pygame.quit()
            sys.exit()

        if event.type == pygame.MOUSEBUTTONDOWN:
            if event.button == 1:
                doMove = True
            if event.button == 3:
                image_pos = ((mainSurface.get_width() - sprite.get_width())/2,
(mainSurface.get_height() - sprite.get_height())/2)
                doMove = False

        if event.type == pygame.MOUSEBUTTONUP: doMove = False

        if event.type == MOUSEMOTION and doMove:
            image_pos = event.pos

    mainSurface.blit(sprite,image_pos)
    pygame.display.update()
```

Клавіатура

Модуль pygame.key

Цей модуль містить функції для роботи з клавіатурой.Очередь подій отримує події pygame.KEYDOWN і pygame.KEYUP при натисканні і відпусканні клавіш клавіатури.

Обидві події мають ключовий атрибут, який представляє собою цілочисельний ідентифікатор, який представляє кожну клавішу на клавіатуре.Событіе pygame.KEYDOWN має додаткові атрибути: unicode і scancode. unicode є ще однією символьний рядок, яка відповідає введеному символу. Scancode є код для конкретної платформи.

Отримати код клавіші:

```
pressed_keys = pygame.key.get_pressed()
if pressed_keys[K_SPACE]:
# Space key has been pressed
fire()
```

Існує багато клавіатурних констант, вони використовуються для подання клавіш на клавіатурі. Нижче наведено список всіх клавіатурних констант:

KeyASCII	ASCII	CommonName	KeyASCII	ASCII	CommonName
K_BACKSPACE	\b	backspace	K_DELETE		delete
K_TAB	\t	tab	K_KP0		keypad0
K_CLEAR		clear	K_KP1		keypad1
K_RETURN	\r	return	K_KP2		keypad2
K_PAUSE		pause	K_KP3		keypad3
K_ESCAPE	^[	escape	K_KP4		keypad4
K_SPACE		space	K_KP5		keypad5
K_EXCLAIM	!	exclaim	K_KP6		keypad6
K_QUOTEDBL	"	quotedbl	K_KP7		keypad7
K_HASH	#	hash	K_KP8		keypad8
K_DOLLAR	$	dollar	K_KP9		keypad9
K_AMPERSAND	&	ampersand	K_KP_PERIOD	.	keypadperiod
K_QUOTE		quote	K_KP_DIVIDE	/	keypaddivide
K_LEFTPAREN	(	leftparenthesis	K_KP_MULTIPLY	*	keypadmultiply
K_RIGHTPAREN	)	rightparenthesis	K_KP_MINUS	-	keypadminus
K_ASTERISK	*	asterisk	K_KP_PLUS	+	keypadplus
K_PLUS	+	plussign	K_KP_ENTER	\r	keypadenter
K_COMMA	,	comma	K_KP_EQUALS	=	keypadequals
K_MINUS	-	minussign	K_UP		uparrow
K_PERIOD	.	period	K_DOWN		downarrow

K_SLASH	/	forwardslash	K_RIGHT		rightarrow
K_0	0	0	K_LEFT		leftarrow
K_1	1	1	K_INSERT		insert
K_2	2	2	K_HOME		home
K_3	3	3	K_END		end
K_4	4	4	K_PAGEUP		pageup
K_5	5	5	K_PAGEDOWN		pagedown
K_6	6	6	K_F1		F1
K_7	7	7	K_F2		F2
K_8	8	8	K_F3		F3
K_9	9	9	K_F4		F4
K_COLON	:	colon	K_F5		F5
K_SEMICOLON	;	semicolon	K_F6		F6
K_LESS		less-thansign	K_F7		F7
K_EQUALS	=	equalssign	K_F8		F8
K_GREATER	>	greater-thansign	K_F9		F9
K_QUESTION	?	questionmark	K_F10		F10
K_AT	@	at	K_F11		F11
K_LEFTBRACKET	[	leftbracket	K_F12		F12
K_BACKSLASH	\	backslash	K_F13		F13
K_RIGHTBRACKET	]	rightbracket	K_F14		F14
K_CARET	^	caret	K_F15		F15
K_UNDERSCORE	_	underscore	K_NUMLOCK		numlock
K_BACKQUOTE	`	grave	K_CAPSLOCK		capslock
K_a	a	a	K_SCROLLOCK		scrollock
K_b	b	b	K_RSHIFT		rightshift
K_c	c	c	K_LSHIFT		leftshift
K_d	d	d	K_RCTRL		rightcontrol
K_e	e	e	K_LCTRL		leftcontrol
K_f	f	f	K_RALT		rightalt
K_g	g	g	K_LALT		leftalt
K_h	h	h	K_RMETA		rightmeta
K_i	i	i	K_LMETA		leftmeta
K_j	j	j	K_LSUPER		leftWindowskey
K_k	k	k	K_RSUPER		rightWindowskey
K_l	l	l	K_MODE		modeshift
K_m	m	m	K_HELP		help
K_n	n	n	K_PRINT		printscreen
K_o	o	o	K_SYSREQ		sysrq
K_p	p	p	K_BREAK		break

K_q	q	q	K_MENU		menu
K_r	r	r	K_POWER		power
K_s	s	s	K_EURO		Euro
K_t	t	t	K_DELETE		delete
K_u	u	u	K_KP0		keypad0
K_v	v	v	K_KP1		keypad1
K_w	w	w	K_KP2		keypad2
K_x	x	x	K_KP3		keypad3
K_y	y	y	K_KP4		keypad4
K_z	z	z	K_KP5		keypad5
			K_KP6		keypad6
			K_KP7		keypad7
			K_KP8		keypad8
			K_KP9		keypad9
			K_KP_PERIOD	.	keypadperiod
			K_KP_DIVIDE	/	keypaddivide
			K_KP_MULTIPLY	*	keypadmultiply
			K_KP_MINUS	-	keypadminus

Направлений рух за допомогою клавіш

Можна переміщати зображення на екрані з клавіатури, призначаючи клавіші для переміщень: вгору, вниз, вліво, вправо.

Створити картинку, наприклад:

```
sprite=pygame.image.load('images//pict2.gif')
```

Перевірити чергу подій:

```
pygame.event.get()
```

Перевірити, чи є отримане подія натисканням на клавіші зі стрілками:

```
event.type == KEYDOWN
```

Якщо - так, то получіть код натиснутою клавішою і сформувати нові координати для картинки:

```
spritex,spritey
```

Намалювати картинку в новому місці:

```
blit(sprite,(spritex,spritey))
```

```
import pygame, sys, time
```

```
from pygame.locals import *

pygame.init()

FPS=30
fpsClock=pygame.time.Clock()
width=500
height=500
mainSurface=pygame.display.set_mode((width,height),0,32)
pygame.display.set_caption('Keyb moves')

background=pygame.image.load('images//bg1.jpg')

sprite=pygame.image.load('images//pict2.gif') # Create moving image

# Place image to the center of mainSurface
spritex=(mainSurface.get_width() - sprite.get_width())/2
spritey=(mainSurface.get_height() - sprite.get_height())/2
direction=False

# --->
def move(direction, spritex, spritey):
    # This function moves recalculates new image coordinates
    if direction:
        if direction == K_UP:
            spritey-=5
        elif direction == K_DOWN:
            spritey+=5
        if direction == K_LEFT:
            spritex-=5
        elif direction == K_RIGHT:
            spritex+=5
    return spritex, spritey
# --->

# game loop
while True:
    fpsClock.tick(FPS) # define frame rate
    mainSurface.blit(background,(0,0))
    mainSurface.blit(sprite,(spritex,spritey))

    # get all events from the queue
    for event in pygame.event.get():
        # loop events queue, remember in 'direction' pressed key code
        if event.type==QUIT:
            # window close X pressed
            pygame.quit()
            sys.exit()
        if event.type==KEYDOWN and event.key == K_ESCAPE:
            # ESC key pressed
            pygame.quit()
            sys.exit()
        if event.type == KEYDOWN: direction = event.key # Other key pressed
        if event.type == KEYUP:  direction = False # Key released

    # calculate new image position
    spritex, spritey = move(direction, spritex, spritey)

    pygame.display.update()
```

Об'єкт Surface

pygame.Surface — об'єкт pygame для представлення зображень.

```
Surface((width, height), flags=0, depth=0, masks=None) -> Surface

Surface((width, height), flags=0, Surface) -> Surface
```

Накладення поверхонь, прозорість.

```
#-*-coding: utf-8-*-
import pygame
pygame.init()

# Create base surface
screen = pygame.display.set_mode((500,500))

# Create new surface
surface1 = pygame.Surface((150,150))
surface1.fill((255,0,0))
surface1.set_alpha(150)
#
surface2 = pygame.Surface((100,100))
surface2.fill((255,255,0))
surface2.set_alpha(100)

# Create image
bgImg = pygame.image.load("images//bg3.jpg")
bgImg = pygame.transform.scale(bgImg,(500,500))
#
pict1 = pygame.image.load("images//pict1.jpg")
pict1 = pygame.transform.scale(pict1,(130,130))
pict1.set_alpha(100)
#
pict2 = pygame.image.load("images//pict2.gif")
pict2 = pygame.transform.scale(pict2,(50,50))

clock = pygame.time.Clock()
running = 1
dX = dY = 1
x = y = 0

while running:
    clock.tick(50)
    event = pygame.event.poll()
    if event.type == pygame.QUIT: running = 0
    x += 8 * dX
    y += 6 * dY
    if (y<0 or y>= (screen.get_height() - pict2.get_height())) :
        dY *= -1
    if (x<0 or x>= (screen.get_width() - pict2.get_width())) :
        dX *= -1

    screen.blit(bgImg,(0,0))
    surface1.blit(pict1,(0,0))
    screen.blit(surface1,(20,50))
    screen.blit(surface2,(150,150))
    screen.blit(pict2,(x,y))

    pygame.display.flip()
```

```
pygame.quit()
```

Керування часом

Модуль pygame.time містить об'єкт Clock, який можна використовувати для відстеження часу. Щоб створити об'єкт типу: час, викликається конструктор pygame.time.Clock:
```
clock = pygame.time.Clock()
```

Коли створено об'єкт clock, можна викликати його функцію tick один раз за кадр, яка повертає час, що минув з часу попереднього виклику в мілісекундах:
```
time_passed = clock.tick ()
```

Функція tick може використовувати необов'язковий параметр для встановлення максимальної частоти кадрів. Цей параметр потрібен, якщо гра запущена на робочому комп'ютері і необхідно контролювати, щоб вона не використовувала всю його обчислювальна потужність на 100%:

Гра буде працювати зі швидкістю не більше 30 кадрів в секунду

```
time_passed = clock.tick (30)
```

Звуки

Для управління звуком використовується модуль `pygame.mixer`. Він відповідає за будь-які дії зі звуками.

Завантажуємо звуковий файл у форматі *.wav

```
sound = pygame.mixer.Sound("sound.wav")
```
(завантажуємо до ігрового циклу, тому що це дуже довга операція)

Програємо звук

```
sound.play()
```

Зіткнення (collisions)

При написанні ігор часто виникає необхідність перевіряти взаємне розташування об'єктів на екрані, відстежувати моменти їх зіткнень, пересічний.
Це завдання може бути реалізована різними способами.
Наприклад, використовуючи об'єкт Rect

```
#-*-coding:utf-8-*-
import pygame, sys, time
from pygame.locals import *

pygame.init()
FPS         = 30
fpsClock    = pygame.time.Clock()
winWidth    = 500
winHeight   = 500
mainSurface = pygame.display.set_mode((winWidth,winHeight),0,32)
pygame.display.set_caption('Collisions test')
background = pygame.image.load('images//bg1.jpg')
```

```
# Малюємо нерухомі компоненти на поверхні background
# Місця розташування кордонів
border1Rect = pygame.Rect(0,0,100,500)
border2Rect = pygame.Rect(0,150,300,50)
# кордони
border1 = pygame.draw.rect(background, (0,0,0), border1Rect, 0)
border2 = pygame.draw.rect(background, (0,0,0), border2Rect, 0)
# Записуємо їх у масив
borders = []
borders.append((border1,border1Rect))
borders.append((border2,border2Rect))
# Подвижный блок
blockWidth  = 50
blockHeight = 50
blockStep = 1
blockColor  = (255,0,0)
# Пчаткові координати рухомого блоку - по центру
blockX = (mainSurface.get_width()   - blockWidth)/2
blockY = (mainSurface.get_height()  - blockHeight)/2
blockPosition = (blockX, blockY)
direction = False

# опис функції --->
def newPosition(dirFlag, pos):
    (x,y) = pos
    # Функція перераховує координати для рухомого об'єкту
    if dirFlag:
        if dirFlag      == K_UP:
            y -= blockStep
        elif dirFlag    == K_DOWN:
            y += blockStep
        if dirFlag      == K_LEFT:
            x -= blockStep
        elif dirFlag    == K_RIGHT:
            x += blockStep
    return (x, y)
# --->

while True:   # Гра - початок
    # Частота оновлення екрану
    fpsClock.tick(FPS)
    # Малюємо нерухомі компоненти
    mainSurface.blit(background,(0,0))
    # Малюємо рухомі компоненти
    blockRect = pygame.Rect(blockPosition, (blockWidth, blockHeight))
    block = pygame.draw.rect(mainSurface, blockColor, blockRect, 1)
    # переглядаємо чергу подій
    for event in pygame.event.get():
        # loop events queue, remember in 'direction' pressed key code
        if event.type == QUIT:
            # window close X pressed
            pygame.quit()
            sys.exit()
        if event.type == KEYDOWN and event.key == K_ESCAPE:
            # ESC key pressed
            pygame.quit()
            sys.exit()
        if event.type == KEYDOWN: direction = event.key # Other key pressed
        if event.type == KEYUP:  direction = False # Key released
    # Сохраняємо старі координати рухомого блоку
    savedPosition = blockPosition
    # Розраховуємо нові координати рухомого блоку
    blockPosition = newPosition(direction, blockPosition)
```

```
    # Перевіряємо їх коректність
    for border in borders:
        testRect = pygame.Rect(blockPosition, (blockWidth, blockHeight))
        if testRect.colliderect(border[1]):
            print ("Зіткнення !")
            # Повертаємо старі координати
            blockPosition = savedPosition
        else: print ("ok")
    pygame.display.update()
# гра - кінець
```

Або використовуючи поверхні - surface

```
#-*-coding:utf-8-*-
import pygame, sys, time
from pygame.locals import *

# Блоки-обмежувачі малюються як окремі поверхні
# Між ними курсором перемещаем м'ячик

FPS = 30 # кадров в сек
# Розміри вікна гри
width  = 500
height = 500
# Заголовок вікна гри
title = "Collisions detection test"
# Повідомлення в консоль гри
info = "Нема зіткнень \n"
# Змінна - індикатор руху
direction = False
# крок руху
myStep = 2
pygame.mixer.init()
pygame.mixer.music.load("voice-prompts-reaction-reaction-1-child-3-yrs-oops-
human-voice-kid-speak-talk.mp3")

pygame.init()
fpsClock = pygame.time.Clock()
mainSurface=pygame.display.set_mode((width,height),0,32)
pygame.display.set_caption(title)
background=pygame.image.load('images//bg1.jpg')     # Фон
sprite = pygame.image.load('images//pict2.gif')     # Рухома картинка

# Початкові координати рухомої картинки
# Важливо, щоб на початку роботи вона не перекривала жоден блок
spriteX=mainSurface.get_width() - sprite.get_width()
spriteY=mainSurface.get_height() - sprite.get_height()

# Нерухомі блоки - все зберігаємо в одному списку
# Структура списку blocks:
# Кожен елемент - пара значень: (поверхня, область її розміщення)
# Blocks [0] - перший елемент списку, blocks [0] [0] - surface першого елемента,
blocks [0] [1] - Rect першого елемента
blocks = []
block1 = pygame.Surface((200,200))
block1.set_alpha(80)
block1.fill((255,0,0))
block1Rect = pygame.Rect(0,0,block1.get_width(), block1.get_height())
blocks.append((block1,block1Rect))
#
block2 = pygame.Surface((100,200))
```

```
block2.fill((0,255,0))
block2Rect = pygame.Rect(400,0,block2.get_width(), block2.get_height())
blocks.append((block2,block2Rect))
#
block3 = pygame.Surface((300,100))
block3.fill((0,0,255))
block3Rect = pygame.Rect(0,350,block3.get_width(), block3.get_height())
blocks.append((block3,block3Rect))
# закінчили з описом 3-х блоків


# *********>
def newPosition (direction, spriteX, spriteY):
    # Функція перераховує координати нової позиції рухомого об'єкта
    # Перевіряємо зіткнень з усіма блоками-кордонами
    global myStep
    if direction:
        if direction == K_UP:
            spriteY -= myStep
        elif direction == K_DOWN:
            spriteY += myStep
        elif direction == K_LEFT:
            spriteX -= myStep
        elif direction == K_RIGHT:
            spriteX += myStep
    return spriteX, spriteY
# *********>



# *********>
def collisionDetected():
    global blocks
    global spriteRectNew
    colFlag = False
    # Перевірка зіткнень з усіма блоками в масиві блоків
    for block in blocks:
        if spriteRectNew.colliderect(block[1]):
            collisionDir = direction
            colFlag = True
    return colFlag
# *********>

# Цикл гри
while True:
    fpsClock.tick(FPS) # Частота оновлення екрану

    # Обробляємо чергу подій - початок
    for event in pygame.event.get():
        # У циклі беремо з черги ще одна обставина, запам'ятовуємо в змінної
event
         # Перевіряємо тип події і виконуємо відповідні лействія
        if event.type == QUIT:
            # Тип проверяемої події НАТИСНУТЕ Х В ВІКНІ ГРИ
            pygame.quit()
            sys.exit()
        if event.type == KEYDOWN and event.key == K_ESCAPE:
            # ESC key pressed
            pygame.quit()
            sys.exit()

        # Наступний рядок отримує управління тільки тоді, коли не відпрацювали
попередні перевірки коду події
         # Тобто відбулася подія, відмінне від перерахованих вище
```

```
        if event.type == KEYDOWN:
            direction = event.key
        if event.type == KEYUP:
            direction = False # Кнопка отпущена
    # Обробляємо чергу подій - кінець
    # Поточне місце розташування рухомий картинки
    spriteRect = pygame.Rect(spriteX,spriteY,sprite.get_width(),
sprite.get_height())

    # Зберігаємо старі координати
    oldPos = (spriteX, spriteY)

    # Вичмсляем нові координати, аналізуючи натиснуті кнопки
    spriteX, spriteY = newPosition(direction, spriteX, spriteY)

    # Обчислюємо нове місце розташування картинки
    spriteRectNew = pygame.Rect(spriteX,spriteY,sprite.get_width(),
sprite.get_height())


    # Перевіряємо, не перетинає чи нове місце блоки. Якщо перетинає, то
вовращпні зображенні старі координати
    if collisionDetected():
        (spriteX, spriteY) = oldPos
        # Play OOPS!
        pygame.mixer.music.play()

    #     Фон
    mainSurface.blit(background,(0,0))
    #    Блоки
    for block in blocks:
        mainSurface.blit(block[0],(block[1].x,block[1].y))
    #    Картинка
    mainSurface.blit(sprite,(spriteRect.x,spriteRect.y))
    #    Оновлюємо екран
    pygame.display.update()
```

Екрани комп'ютерів зроблені з пікселів, кожен з яких містить 3 елементи: червоний, зелений і синій. Колір пікселя визначається тим, як горить кожен з елементів:

Красный	Зеленый	Синий	Цвет
0	0	0	Черный
255	0	0	Красный
0	255	0	Зеленый
0	0	255	Синий
0	255	255	Голубой
255	255	0	Желтый
255	0	255	Пурпурный
255	255	255	Белый

Кожен з трьох основних кольорів може мати значення від 0 (виключений) до 255 (включений на 100%), так що для кожного елемента є 256 варіантів.
Дізнатися загальна кількість відображуваних комп'ютером квітів можна, помноживши:

```
>>> 256 * 256 * 256
16,777,216
```

Додаток 4

Програма

game1.py - C:/Users/alext/Desktop/lrpa/game1.py (3.7.2)

File Edit Format Run Options Window Help

```
import pygame
pygame.init()
win = pygame.display.set_mode((500, 500))

pygame.display.set_caption("Game 1")

x = 50
y = 425
width = 40
height = 60
speed = 5

isJump = False
jumpCount = 10

run = True
while run:
    pygame.time.delay(50)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT] and x > 5:
        x-= speed
    if keys[pygame.K_RIGHT] and x < 500- width - 5:
        x+= speed
    if not(isJump):
        if keys[pygame.K_UP] and y > 5 :
            y-= speed
        if keys[pygame.K_DOWN] and y < 500 - height-5:
            y+= speed
        if  keys[pygame.K_SPACE]:
            isJump = True
```

```
            isJump = True
    else:
        if jumpCount >= -10:
            if jumpCount < 0:
                y +=(jumpCount**2)/2
            else:
                y -=(jumpCount**2)/2
            jumpCount  -= 1
        else:
            isJump = False
            jumpCount = 10
    win.fill((0, 0, 0))
    pygame.draw.rect(win, (0, 0, 225), (x, y, width, height))
    pygame.display.update()


pygame.quit()                    # Закриваємо цикл
```

Додаток 5

Програма

game2.py - C:/Users/alext/Desktop/lrpa/game2.py (3.7.2)

File Edit Format Run Options Window Help

```
import pygame
pygame.init()
win = pygame.display.set_mode((500, 500))
pygame.display.set_caption("Game 2")


walkright = [pygame.image.load('right_1.png'),
pygame.image.load('right_2.png'),pygame.image.load('right_3.png'),
pygame.image.load('right_4.png'),pygame.image.load('right_5.png'),
pygame.image.load('right_6.png')]

walkleft = [pygame.image.load('left_1.png'),pygame.image.load('left_2.png'),
pygame.image.load('left_3.png'),pygame.image.load('left_4.png'),
pygame.image.load('left_5.png'),pygame.image.load('left_6.png')]

bg = pygame.image.load('bg.jpg')
playerStand = pygame.image.load('idle.png')

clock = pygame.time.Clock()

x = 50
y = 425
width = 60
height = 71
speed = 5

isJump = False
jumpCount = 10

left = False
right = False
animCount = 0

def drawWindow():
    global animCount
```

```
    win.blit(bg, (0,0))
    if animCount + 1 >= 30:
       animCount = 0

    if left:
        win.blit(walkleft[animCount // 5],(x,y))
        animCount += 1
    elif right:
        win.blit(walkright[animCount // 5],(x,y))
        animCount += 1
    else:
        win.blit(playerStand, (x, y))

    pygame.display.update()

run = True
while run:
    clock.tick(30)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT] and x > 5:
        x-= speed
        left = True
        right = False
    elif keys[pygame.K_RIGHT] and x < 500- width - 5:
        x+= speed
        left = False
        right = True
    else:
        left = False
        right = False
        animCount = 0
    if not(isJump):
        if keys[pygame.K_UP] and y > 5 :
            y-= speed
        if keys[pygame.K_DOWN] and y < 500 - height-5:
            y+= speed
        if  keys[pygame.K_SPACE]:
            isJump = True
    else:
        if jumpCount >= -10:
            if jumpCount < 0:
                y +=(jumpCount**2)/2
            else:
                y -=(jumpCount**2)/2
            jumpCount  -= 1
        else:
            isJump = False
            jumpCount = 10

    drawWindow()
pygame.quit()                    # Закриваємо цикл
```

Додаток 6

Програма

game.py - C:\Users\alext\Desktop\lrpa\game.py (3.7.2)

File Edit Format Run Options Window Help

```
import pygame
pygame.init()
win = pygame.display.set_mode((500, 500))
pygame.display.set_caption("Game 2")


walkright = [pygame.image.load('right_1.png'),
pygame.image.load('right_2.png'),pygame.image.load('right_3.png'),
pygame.image.load('right_4.png'),pygame.image.load('right_5.png'),
pygame.image.load('right_6.png')]

walkleft = [pygame.image.load('left_1.png'),pygame.image.load('left_2.png'),
pygame.image.load('left_3.png'),pygame.image.load('left_4.png'),
pygame.image.load('left_5.png'),pygame.image.load('left_6.png')]

bg = pygame.image.load('bg.jpg')
playerStand = pygame.image.load('idle.png')

clock = pygame.time.Clock()

x = 50
y = 425
width = 60
height = 71
speed = 5

isJump = False
jumpCount = 10

left = False
right = False
animCount = 0
lastMove =  'right'

class snaryad(object):
    def __init__(self, x, y, radius, color, facing):
        self.x = x
        self.y = y
        self.radius = radius
        self.color = color
        self.facing = facing
        self.vel = 8 * facing
    def draw(self, win):
        pygame.draw.circle(win, self.color,(self.x, self.y),self.radius)
```

```
def drawWindow():
    global animCount

    win.blit(bg, (0,0))
    if animCount + 1 >= 30:
       animCount = 0

    if left:
        win.blit(walkleft[animCount // 5],(x,y))
        animCount += 1
    elif right:
        win.blit(walkright[animCount // 5],(x,y))
        animCount += 1
    else:
        win.blit(playerStand, (x, y))
    for bullet in bullets:
        bullet.draw(win)

    pygame.display.update()

run = True
bullets = []
while run:
    clock.tick(30)

    for event in pygame.event.get():

        if event.type == pygame.QUIT:
            run = False

    for bullet in bullets:
        if bullet.x < 500 and bullet.x > 0:
           bullet.x += bullet.vel
        else:
            bullets.pop(bullets.index(bullet))
'
   keys = pygame.key.get_pressed()
   if keys[pygame.K_f]:
       if lastMove == 'right':
           facing = 1
       else:
           facing = -1

       if len(bullets) < 5:
           bullets.append(snaryad(round(x + width // 2), round(y + height // 2), 5, (255,0,0), facing))

    if keys[pygame.K_LEFT] and x > 5:
        x-= speed
        left = True
        right = False
        lastMove = 'left'
    elif keys[pygame.K_RIGHT] and x < 500- width - 5:
        x+= speed
        left = False
        right = True
        lastMove = 'right'
```

```
        else:
            left = False
            right = False
            animCount = 0
        if not(isJump):
            if keys[pygame.K_UP] and y > 5 :
                y-= speed
            if keys[pygame.K_DOWN] and y < 500 - height-5:
                y+= speed
            if  keys[pygame.K_SPACE]:
                isJump = True
        else:
            if jumpCount >= -10:
                if jumpCount < 0:
                    y +=(jumpCount**2)/2
                else:
                    y -=(jumpCount**2)/2
                jumpCount  -= 1
            else:
                isJump = False
                jumpCount = 10

        drawWindow()

    pygame.quit()                    # Закриваємо цикл
```

Додаток 7

Програма

game3.py - C:\Users\alext\Desktop\lrpa\game3.py (3.7.2)

File Edit Format Run Options Window Help

```
import pygame
pygame.init()
win = pygame.display.set_mode((500, 500))
pygame.display.set_caption("Game 3")


class enemy(object):
    walkRight = [pygame.image.load('R1E.png'), pygame.image.load('R2E.png'),
                 pygame.image.load('R3E.png'), pygame.image.load('R4E.png'),
                 pygame.image.load('R5E.png'), pygame.image.load('R6E.png'),
                 pygame.image.load('R7E.png'), pygame.image.load('R8E.png'),
                 pygame.image.load('R9E.png'), pygame.image.load('R10E.png'),
                 pygame.image.load('R11E.png')]
    walkLeft = [pygame.image.load('L1E.png'), pygame.image.load('L2E.png'),
                pygame.image.load('L3E.png'), pygame.image.load('L4E.png'),
                pygame.image.load('L5E.png'), pygame.image.load('L6E.png'),
                pygame.image.load('L7E.png'), pygame.image.load('L8E.png'),
                pygame.image.load('L9E.png'), pygame.image.load('L10E.png'),
                pygame.image.load('L11E.png')]

    def __init__(self, x, y, width, height, end):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.path = [x, end]
        self.walkCount = 0
        self.vel = 3
        self.hitbox = (self.x + 20, self.y, 28, 60)

    def draw(self, win):
        self.move()
        if self.walkCount + 1 >= 33:
            self.walkCount = 0
```

```
        if self.vel > 0:
            win.blit(self.walkRight[self.walkCount//3], (self.x,self.y))
            self.walkCount += 1
        else:
            win.blit(self.walkLeft[self.walkCount//3], (self.x,self.y))
            self.walkCount += 1
        self.hitbox = (self.x + 15, self.y, 28, 60)
        pygame.draw.rect(win, (255,0,0), self.hitbox, 2)


    def move(self):
        if self.vel > 0:
            if self.x < self.path[1] + self.vel:
                self.x += self.vel
            else:
                self.vel = self.vel * -1
                self.x += self.vel
                self.walkCount = 0
        else:
            if self.x > self.path[0] - self.vel:
                self.x += self.vel
            else:
                self.vel = self.vel * -1
                self.x += self.vel
                self.walkCount = 0
    def hit(self):
        print('Попадание')

pygame.image.load('right_2.png'),pygame.image.load('right_3.png'),
pygame.image.load('right_4.png'),pygame.image.load('right_5.png'),
pygame.image.load('right_6.png')]

walkleft = [pygame.image.load('left_1.png'),pygame.image.load('left_2.png'),
pygame.image.load('left_3.png'),pygame.image.load('left_4.png'),
pygame.image.load('left_5.png'),pygame.image.load('left_6.png')]

bg = pygame.image.load('bg.jpg')
playerStand = pygame.image.load('idle.png')

clock = pygame.time.Clock()

x = 50
y = 425
width = 60
height = 71
speed = 5

isJump = False
jumpCount = 10

left = False
right = False
animCount = 0
hitbox = (x + 20, y, 28, 60)
lastMove =  'right'

class snaryad(object):
    def __init__(self, x, y, radius, color, facing):
        self.x = x
        self.y = y
        self.radius = radius
        self.color = color
        self.facing = facing
        self.vel = 8 * facing
    def draw(self, win):
        pygame.draw.circle(win, self.color,(self.x, self.y),self.radius)
```

```
def drawWindow():
    global animCount

    win.blit(bg, (0,0))
    if animCount + 1 >= 30:
       animCount = 0

    if left:
        win.blit(walkleft[animCount // 5],(x,y))
        animCount += 1
    elif right:
        win.blit(walkright[animCount // 5],(x,y))
        animCount += 1
    else:
        win.blit(playerStand, (x, y))
    for bullet in bullets:
        bullet.draw(win)
    goblin.draw(win)
    hitbox = (x + 10, y, 40, 80)
    pygame.draw.rect(win, (255,0,0), hitbox, 2)

    pygame.display.update()

goblin = enemy(100, 436, 64, 64, 400)
run = True
shootloop = 0
bullets = []
while run:
    clock.tick(30)

    if shootloop > 0:
        shootloop +=1
    if shootloop > 3:
        shootloop = 0


  for event in pygame.event.get():

      if event.type == pygame.QUIT:
          run = False

  for bullet in bullets:
      if bullet.y - bullet.radius < goblin.hitbox[1] + goblin.hitbox[3] and bullet.y + bullet.radius > goblin.hitbox[1]:
          if bullet.x + bullet.radius > goblin.hitbox[0] and bullet.x - bullet.radius < goblin.hitbox[1] + goblin.hitbox[3]:
              goblin.hit()
              bullets.pop(bullets.index(bullet))

      if bullet.x < 500 and bullet.x > 0:
         bullet.x += bullet.vel
      else:
          bullets.pop(bullets.index(bullet))

  keys = pygame.key.get_pressed()
  if keys[pygame.K_f] and shootloop == 0:
      if lastMove == 'right':
          facing = 1
      else:
          facing = -1

      if len(bullets) < 5:
          bullets.append(snaryad(round(x + width // 2), round(y + height // 2), 5, (255,0,0), facing))
      shootloop = 1
```

```
    if keys[pygame.K_LEFT] and x > 5:
        x-= speed
        left = True
        right = False
        lastMove = 'left'
    elif keys[pygame.K_RIGHT] and x < 500- width - 5:
        x+= speed
        left = False
        right = True
        lastMove = 'right'
    else:
        left = False
        right = False
        animCount = 0
    if not(isJump):
        if keys[pygame.K_UP] and y > 5 :
            y-= speed
        if keys[pygame.K_DOWN] and y < 500 - height-5:
            y+= speed
        if  keys[pygame.K_SPACE]:
            isJump = True
    else:
        if jumpCount >= -10:
            if jumpCount < 0:
                y +=(jumpCount**2)/2
            else:
                y -=(jumpCount**2)/2
            jumpCount  -= 1
        else:
            isJump = False
            jumpCount = 10

    drawWindow()

pygame.quit()                    # Закриваємо цикл
```

Додаток 8.

Програма

game4.py - C:\Users\alext\Desktop\lrpa\game4.py (3.7.2)

File Edit Format Run Options Window Help

```
import pygame
pygame.init()
win = pygame.display.set_mode((500, 500))
pygame.display.set_caption("Game 3")

bulletSound = pygame.mixer.Sound('bullet.wav')
hitSound = pygame.mixer.Sound('hit.wav')

music = pygame.mixer.music.load('music.mp3')
pygame.mixer.music.play(-1)

score = 0

class enemy(object):
    walkRight = [pygame.image.load('R1E.png'), pygame.image.load('R2E.png'),
                 pygame.image.load('R3E.png'), pygame.image.load('R4E.png'),
                 pygame.image.load('R5E.png'), pygame.image.load('R6E.png'),
                 pygame.image.load('R7E.png'), pygame.image.load('R8E.png'),
                 pygame.image.load('R9E.png'), pygame.image.load('R10E.png'),
                 pygame.image.load('R11E.png')]
    walkLeft = [pygame.image.load('L1E.png'), pygame.image.load('L2E.png'),
                pygame.image.load('L3E.png'), pygame.image.load('L4E.png'),
                pygame.image.load('L5E.png'), pygame.image.load('L6E.png'),
                pygame.image.load('L7E.png'), pygame.image.load('L8E.png'),
                pygame.image.load('L9E.png'), pygame.image.load('L10E.png'),
                pygame.image.load('L11E.png')]

    def __init__(self, x, y, width, height, end):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.path = [x, end]
        self.walkCount = 0
        self.vel = 3
        self.hitbox = (self.x + 20, self.y, 28, 60)
        self.health = 10
        self.visible = True
```

```
    def draw(self, win):
        if self.visible:
            self.move()
            if self.walkCount + 1 >= 33:
                self.walkCount = 0

            if self.vel > 0:
                win.blit(self.walkRight[self.walkCount//3], (self.x,self.y))
                self.walkCount += 1
            else:
                win.blit(self.walkLeft[self.walkCount//3], (self.x,self.y))
                self.walkCount += 1
            self.hitbox = (self.x + 15, self.y, 28, 60)
            pygame.draw.rect(win, (255,0,0), (self.hitbox[0], self.hitbox[1] - 20, 50, 10))
            pygame.draw.rect(win, (0,255,0), (self.hitbox[0], self.hitbox[1] - 20, 50 - (5 *(10 - self.health)), 10))
            #pygame.draw.rect(win, (255,0,0), self.hitbox, 2)


    def move(self):
        if self.vel > 0:
            if self.x < self.path[1] + self.vel:
                self.x += self.vel
            else:
                self.vel = self.vel * -1
                self.x += self.vel
                self.walkCount = 0
        else:
            if self.x > self.path[0] - self.vel:
                self.x += self.vel
            else:
                self.vel = self.vel * -1
                self.x += self.vel
                self.walkCount = 0
```

```
    def hit(self):
        if self.health > 0:
            self.health -= 1
        else:
            self.visible = False
        print('Попадание')
```

```
walkright = [pygame.image.load('right_1.png'),
pygame.image.load('right_2.png'),pygame.image.load('right_3.png'),
pygame.image.load('right_4.png'),pygame.image.load('right_5.png'),
pygame.image.load('right_6.png')]

walkleft = [pygame.image.load('left_1.png'),pygame.image.load('left_2.png'),
pygame.image.load('left_3.png'),pygame.image.load('left_4.png'),
pygame.image.load('left_5.png'),pygame.image.load('left_6.png')]

bg = pygame.image.load('bg.jpg')
playerStand = pygame.image.load('idle.png')

clock = pygame.time.Clock()

x = 50
y = 425
width = 60
height = 71
speed = 5

isJump = False
jumpCount = 10

left = False
right = False
animCount = 0
hitbox = (x + 20, y, 28, 60)
lastMove =  'right'
```

```
class snaryad(object):
    def __init__(self, x, y, radius, color, facing):
        self.x = x
        self.y = y
        self.radius = radius
        self.color = color
        self.facing = facing
        self.vel = 8 * facing
    def draw(self, win):
        pygame.draw.circle(win, self.color,(self.x, self.y),self.radius)


def drawWindow():
    global animCount

    win.blit(bg, (0,0))
    if animCount + 1 >= 30:
       animCount = 0

    if left:
        win.blit(walkleft[animCount // 5],(x,y))
        animCount += 1
    elif right:
        win.blit(walkright[animCount // 5],(x,y))
        animCount += 1
    else:
        win.blit(playerStand, (x, y))
    for bullet in bullets:
        bullet.draw(win)
    goblin.draw(win)

    #pygame.draw.rect(win, (255,0,0), hitbox, 2)
    text = font.render("Рахунок: " + str(score), 1, (0,0,0))
    win.blit(text, (350, 10))


def hit():
    animCount = 0
    font1 = pygame.font.SysFont('comicsans', 100)
    text1 = font1.render('-5', 1 ,(255,0,0))
    win.blit(text1, (250 - (text1.get_width()/2), 200))
    pygame.display.update()
    i = 0
    while i < 300:
        pygame.time.delay(10)
        i += 1
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                i = 301
                pygame.quit()

font = pygame.font.SysFont("comicsans", 30, True, True)
goblin = enemy(100, 436, 64, 64, 400)
run = True
shootloop = 0
bullets = []
while run:
    clock.tick(30)
    hitbox = (x + 10, y, 40, 80)

    if hitbox[1] < goblin.hitbox[1] + goblin.hitbox[3] and hitbox[1] + hitbox[3] > goblin.hitbox[1]:
        if hitbox[0] + hitbox[2] > goblin.hitbox[0] and hitbox[0] < goblin.hitbox[0] + goblin.hitbox[2]:
            hit()
            x = 50
            y = 425
            score -= 5

    if shootloop > 0:
        shootloop +=1
    if shootloop > 3:
        shootloop = 0
```

```
    for event in pygame.event.get():

        if event.type == pygame.QUIT:
            run = False

    for bullet in bullets:
        if bullet.y - bullet.radius < goblin.hitbox[1] + goblin.hitbox[3] and bullet.y + bullet.radius > goblin.hitbox[1]:
            if bullet.x + bullet.radius > goblin.hitbox[0] and bullet.x - bullet.radius < goblin.hitbox[1] + goblin.hitbox[3]:
                hitSound.play()
                goblin.hit()
                score += 1
                bullets.pop(bullets.index(bullet))

        if bullet.x < 500 and bullet.x > 0:
           bullet.x += bullet.vel
        else:
            bullets.pop(bullets.index(bullet))

    keys = pygame.key.get_pressed()
    if keys[pygame.K_f] and shootloop == 0:
        bulletSound.play()
        if lastMove == 'right':
            facing = 1
        else:
            facing = -1

        if len(bullets) < 5:
            bullets.append(snaryad(round(x + width // 2), round(y + height // 2), 5, (255,0,0), facing))
        shootloop = 1

      if keys[pygame.K_LEFT] and x > 5:
           x-= speed
           left = True
           right = False
           lastMove = 'left'
      elif keys[pygame.K_RIGHT] and x < 500- width - 5:
           x+= speed
           left = False
           right = True
           lastMove = 'right'
      else:
           left = False
           right = False
           animCount = 0
      if not(isJump):
           if keys[pygame.K_UP] and y > 5 :
                y-= speed
           if keys[pygame.K_DOWN] and y < 500 - height-5:
                y+= speed
           if  keys[pygame.K_SPACE]:
                isJump = True
      else:
           if jumpCount >= -10:
                if jumpCount < 0:
                     y +=(jumpCount**2)/2
                else:
                     y -=(jumpCount**2)/2
                jumpCount  -= 1
           else:
                isJump = False
                jumpCount = 10

      drawWindow()

pygame.quit()                    # Закриваємо цикл
```

Додаток 9.

Програма

шаблон1.py - C:\Users\alext\Desktop\Irpa\шаблон1.py (3.7.2)

File Edit Format Run Options Window Help

```
#ШАБЛОН
import pygame
import random

WIDTH =480
HEIGHT = 600
FPS = 60
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
YELLOW = (255, 255, 0)



#Створимо гру та вікно
pygame.init()
pygame.mixer.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption('My Game')
clock = pygame.time.Clock()



class Player(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((50, 40))
        self.image.fill(GREEN)
        self.rect = self.image.get_rect()
        self.rect.centerx = WIDTH / 2
        self.rect.bottom = HEIGHT - 10
        self.speedx = 0

    def update(self):
        self.speedx = 0
        keystate = pygame.key.get_pressed()
        if keystate[pygame.K_LEFT]:
            self.speedx = -8
        if keystate[pygame.K_RIGHT]:
            self.speedx = 8
        self.rect.x += self.speedx
        if self.rect.right > WIDTH:
            self.rect.right = WIDTH
        if self.rect.left < 0:
            self.rect.left = 0
    def shoot(self):
        bullet = Bullet(self.rect.centerx, self.rect.top)
        all_sprites.add(bullet)
        bullets.add(bullet)


class Mob(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((30, 40))
        self.image.fill(RED)
        self.rect = self.image.get_rect()
        self.rect.x = random.randrange(WIDTH - self.rect.width)
        self.rect.y = random.randrange(-100, -40)
        self.speedy = random.randrange(1, 8)
        self.speedx = random.randrange(-3, 3)
```

```
    def update(self):
        self.rect.x += self.speedx
        self.rect.y += self.speedy
        if self.rect.top > HEIGHT + 10 or self.rect.left < -25 or self.rect.right > WIDTH + 20:
            self.rect.x = random.randrange(WIDTH - self.rect.width)
            self.rect.y = random.randrange(-100, -40)
            self.speedy = random.randrange(1, 8)

class Bullet(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((10, 20))
        self.image.fill(YELLOW)
        self.rect = self.image.get_rect()
        self.rect.bottom = y
        self.rect.centerx = x
        self.speedy = -10

    def update(self):
        self.rect.y += self.speedy
        # убить, если он заходит за верхнюю часть экрана
        if self.rect.bottom < 0:
            self.kill()


all_sprites = pygame.sprite.Group()
mobs = pygame.sprite.Group()
bullets = pygame.sprite.Group()
player = Player()
all_sprites.add(player)
for i in range(8):
    m = Mob()
    all_sprites.add(m)
    mobs.add(m)



#Цикл гри

running = True
while running:
    #Тримаємо цикл на  вірній швидкісті
    clock.tick(FPS)
    #Ввод процесу
    for event in pygame.event.get():
        #Перевірка вікна
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                player.shoot()
#Оновлення
    all_sprites.update()
    #Перевірка на удар
    hits = pygame.sprite.spritecollide(player, mobs, False)
    if hits:
        running = False
    hits = pygame.sprite.groupcollide(mobs, bullets, True, True)
    for hit in hits:
        m = Mob()
        all_sprites.add(m)
        mobs.add(m)
#Рендерінг
    screen.fill(BLACK)
    all_sprites.draw(screen)
    #Після прорисовки обертаємо екран
    pygame.display.flip()

pygame.quit()
```

Додаток 10

Програма

шаблон4.py - C:\Users\alext\Desktop\Игра_2\шаблон4.py (3.7.2)

File Edit Format Run Options Window Help

```
#ШАБЛОН
import pygame
import random
from os import path

img_dir = path.join(path.dirname(__file__), 'img')
snd_dir = path.join(path.dirname(__file__), 'snd')

WIDTH =480
HEIGHT = 600
FPS = 60
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
YELLOW = (255, 255, 0)


#Створимо гру та вікно
pygame.init()
pygame.mixer.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption('My Game')
clock = pygame.time.Clock()



font_name = pygame.font.match_font('arial')
def draw_text(surf, text, size, x, y):
    font = pygame.font.Font(font_name, size)
    text_surface = font.render(text, True, WHITE)
    text_rect = text_surface.get_rect()
    text_rect.midtop = (x, y)
    surf.blit(text_surface, text_rect)

def newmob():
    m = Mob()
    all_sprites.add(m)
    mobs.add(m)
def draw_shield_bar(surf, x, y, pct):
    if pct < 0:
        pct = 0
    BAR_LENGTH = 100
    BAR_HEIGHT = 10
    fill = (pct / 100) * BAR_LENGTH
    outline_rect = pygame.Rect(x, y, BAR_LENGTH, BAR_HEIGHT)
    fill_rect = pygame.Rect(x, y, fill, BAR_HEIGHT)
    pygame.draw.rect(surf, GREEN, fill_rect)
    pygame.draw.rect(surf, WHITE, outline_rect, 2)
```

```
class Player(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = player_img
        self.image = pygame.transform.scale(player_img, (50, 38))
        self.image.set_colorkey(BLACK)
        self.rect = self.image.get_rect()
        self.radius = 15
        #pygame.draw.circle(self.image, RED, self.rect.center, self.radius)
        self.rect.centerx = WIDTH / 2
        self.rect.bottom = HEIGHT - 10
        self.speedx = 0
        self.shield = 100

    def update(self):
        self.speedx = 0
        keystate = pygame.key.get_pressed()
        if keystate[pygame.K_LEFT]:
            self.speedx = -8
        if keystate[pygame.K_RIGHT]:
            self.speedx = 8
        self.rect.x += self.speedx
        if self.rect.right > WIDTH:
            self.rect.right = WIDTH
        if self.rect.left < 0:
            self.rect.left = 0

    def shoot(self):
        bullet = Bullet(self.rect.centerx, self.rect.top)
        all_sprites.add(bullet)
        bullets.add(bullet)
        shoot_sound.play()

class Mob(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = meteor_img
        self.image.set_colorkey(BLACK)
        self.image_orig = random.choice(meteor_images)
        self.image_orig.set_colorkey(BLACK)
        self.image = self.image_orig.copy()
        self.rect = self.image.get_rect()
        self.radius = int(self.rect.width * 0.85 / 2)
        #pygame.draw.circle(self.image, GREEN, self.rect.center, self.radius)
        self.rect.x = random.randrange(WIDTH - self.rect.width)
        self.rect.y = random.randrange(-150, -100)
        self.speedy = random.randrange(1, 8)
        self.speedx = random.randrange(-3, 3)
        self.rot = 0
        self.rot_speed = random.randrange(-8, 8)
        self.last_update = pygame.time.get_ticks()

    def update(self):
        self.rotate()
        self.rect.x += self.speedx
        self.rect.y += self.speedy
        if self.rect.top > HEIGHT + 10 or self.rect.left < -25 or self.rect.right >
            self.rect.x = random.randrange(WIDTH - self.rect.width)
            self.rect.y = random.randrange(-100, -40)
            self.speedy = random.randrange(1, 8)
```

```
    def rotate(self):
        now = pygame.time.get_ticks()
        if now - self.last_update > 50:
            self.last_update = now
            #Обертання спрайтів
            self.rot = (self.rot + self.rot_speed) % 360
            new_image = pygame.transform.rotate(self.image_orig, self.rot)
            old_center = self.rect.center
            self.image = new_image
            self.rect = self.image.get_rect()
            self.rect.center = old_center

class Bullet(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((5, 20))
        self.image.fill(YELLOW)
        self.rect = self.image.get_rect()
        self.rect.bottom = y
        self.rect.centerx = x
        self.speedy = -10

    def update(self):
        self.rect.y += self.speedy
        # убити, якщо він заходить за верхню частину екрану
        if self.rect.bottom < 0:
            self.kill()

#Завантаження ігрової графіки
background = pygame.image.load(path.join(img_dir, "starfield.png")).convert()
background_rect = background.get_rect()
player_img = pygame.image.load(path.join(img_dir, "playerShip2_blue.png")).convert()
meteor_img = pygame.image.load(path.join(img_dir, "meteorBrown_med1.png")).convert()
bullet_img = pygame.image.load(path.join(img_dir, "laserRed01.png")).convert()
meteor_images = []
meteor_list = ['meteorBrown_big1.png', 'meteorBrown_med1.png', 'meteorBrown_med1.png',
               'meteorBrown_med3.png', 'meteorBrown_small1.png', 'meteorBrown_small2.png',
               'meteorBrown_tiny1.png']
for img in meteor_list:
    meteor_images.append(pygame.image.load(path.join(img_dir, img)).convert())

shoot_sound = pygame.mixer.Sound(path.join(snd_dir, 'pew.wav'))
expl_sounds = []

for snd in ['expl3.wav', 'expl6.wav']:
    expl_sounds.append(pygame.mixer.Sound(path.join(snd_dir, snd)))

    #Створимо групу спрайтів у грі

all_sprites = pygame.sprite.Group()
mobs = pygame.sprite.Group()
bullets = pygame.sprite.Group()
player = Player()
all_sprites.add(player)
for i in range(8):
    newmob()
score = 0
```

```
#Цикл гри

running = True
while running:
    #Тримаємо цикл на  вірній швидкісті
    clock.tick(FPS)
    #Ввод процесу
    for event in pygame.event.get():
        #Перевірка вікна
        if event.type == pygame.QUIT:
            running = False
            #перевірка події KEYDOWN
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                player.shoot()
#Оновлення
    all_sprites.update()
    #Перевірка на зіткнення
    hits = pygame.sprite.spritecollide(player, mobs, True, pygame.sprite.collide_circle)
    for hit in hits:
        player.shield -= hit.radius * 2
        newmob()
        if player.shield <= 0:
            running = False

    hits = pygame.sprite.groupcollide(mobs, bullets, True, True)
    for hit in hits:
        score += 50 - hit.radius
        random.choice(expl_sounds).play()
        newmob()

#Рендерінг
    screen.fill(BLACK)
    screen.blit(background, background_rect)
    all_sprites.draw(screen)
    draw_text(screen, str(score), 18, WIDTH / 2, 10)
    draw_shield_bar(screen, 5, 5, player.shield)
    #Після прорисовки обертаємо екран
    pygame.display.flip()

pygame.quit()
```

Додаток 11

main.py - C:\Users\alext\Desktop\MOODLE Питон\MOODLE\python_backend_tutorial\app\main.py (3.7.2)

File Edit Format Run Options Window Help

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from .todo_router import todo_router

# app instance creation
app = FastAPI()

# if we do not include CORS it will not work with frontend
app.add_middleware(
    CORSMiddleware,
    allow_origin_regex='https?://.*',
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# we can add many routers with different prefixes for one app with this command
app.include_router(todo_router, prefix='/api/v1/todo')
```

Додаток 12

todo_router.py - C:\Users\alext\Desktop\MOODLE Питон\MOODLE\python_backend_tutorial\app\todo_router.py (3.7.2)

File Edit Format Run Options Window Help

```
from fastapi import APIRouter, HTTPException
from .db import DbHelper
from .models import Task

todo_router = APIRouter()
db_helper = DbHelper()


@todo_router.get("/tasks")
async def get_tasks(limit: int = 0, skip: int = 0, status: bool = False):
    return {"data": db_helper.get_tasks(limit, skip, status)}


# {task_id} is a path param for this get function
# response model is a syntactic sugar of dict formatting
@todo_router.get("/task/{task_id}", response_model=Task)
async def get_task(task_id: int):
    task = db_helper.get_task(task_id)
    if task:
        return task
    else:
        raise HTTPException(status_code=400, detail="This id doesn't exist, please, pass valid id")
```

Додаток 13

models.py - C:\Users\alext\Desktop\MOODLE Питон\MOODLE\python_

File Edit Format Run Options Window Help

```python
from pydantic import BaseModel


class Task(BaseModel):
    title: str
    description: str = ""
    done: bool = False
```

```python
@todo_router.post("/task")
async def add_task(task: Task):
    db_helper.add_task(task)
    return {"status": "OK"}


@todo_router.put("/task/{task_id}")
async def update_task(task_id: int, task: Task):
    task_exists = db_helper.update_task(task_id, task)
    if task_exists:
        return {"status": "OK"}
    else:
        raise HTTPException(status_code=400, detail="This id doesn't exist, please, pass valid id")


@todo_router.put("/task_done/{task_id}")
async def make_task_done(task_id: int):
    task_exists = db_helper.set_done(task_id)
    if task_exists:
        return {"status": "OK"}
    else:
        raise HTTPException(status_code=400, detail="This id doesn't exist, please, pass valid id")


@todo_router.delete("/task/{task_id}")
async def delete_task(task_id: int):
    task_exists = db_helper.delete_task(task_id)
    if task_exists:
        return {"status": "OK"}
    else:
        raise HTTPException(status_code=400, detail="This id doesn't exist, please, pass valid id")
```

Додаток 14

db.py - C:\Users\alext\Desktop\MOODLE Питон\MOODLE\python_backend_tutorial\app\db.py (3.7.2)

File Edit Format Run Options Window Help

```python
from typing import Dict, List
from .models import Task


class DbHelper:
    def __init__(self):
        self._tasks: Dict[int, Task] = {
            1: Task(title="Build Rocket", done=True),
            2: Task(title="Train stuff for the flight"),
            3: Task(title="Build infrastructure for launch")
        }

    # def get_tasks(self, limit: int) -> List[Task]:
    #     task_list = list(self._tasks.values())
    #     if limit != 0:
    #         task_list = task_list[:limit]
    #     return task_list

    def get_tasks(self, limit: int, skip: int, status: bool) -> List[Task]:
        task_list = list(self._tasks.values())

        if limit != 0:
            task_list = task_list[skip:limit]

        if status:
            task_list = list(filter(lambda task: task.done, task_list))

        return task_list


    def get_task(self, task_id: int) -> Task:
        # check whether task_id exists in the self._tasks, make it it to avoid KeyError Exception
        if task_id in self._tasks:
            return self._tasks[task_id]
```

```python
    def add_task(self, task: Task):
        task_id: int = len(self._tasks) + 1
        self._tasks[task_id] = task

    def delete_task(self, task_id: int) -> bool:
        if task_id in self._tasks:
            del self._tasks[task_id]
        return task_id in self._tasks

    def update_task(self, task_id: int, task: Task) -> bool:
        if task_id in self._tasks:
            self._tasks[task_id] = task
        return task_id in self._tasks

    def set_done(self, task_id: int) -> bool:
        if task_id in self._tasks:
            self._tasks[task_id].done = True
        return task_id in self._tasks
```

Додаток 15

example_sync.py - C:\Users\alext\Desktop\MOODLE Питон\MOODLE\py\example_sync.py (3.7.2)

File Edit Format Run Options Window Help

```
import queue
import requests
from codetiming import Timer


def task(name, work_queue):
    timer = Timer(text=f"Task {name} elapsed time: {{:.1f}}")
    with requests.Session() as session:
        while not work_queue.empty():
            url = work_queue.get()
            print(f"Task {name} getting URL: {url}")
            timer.start()
            session.get(url)
            timer.stop()
            yield


def main():
    """
    This is the main entry point for the program
    """
    # Create the queue of work
    work_queue = queue.Queue()

    # Put some work in the queue
    for url in [
        "http://google.com",
        "http://yahoo.com",
        "http://linkedin.com",
        "http://apple.com",
        "http://microsoft.com",
        "http://facebook.com",
        "http://twitter.com",
    ]:
        work_queue.put(url)
            _

    tasks = [task("One", work_queue), task("Two", work_queue)]

    # Run the tasks
    done = False
    with Timer(text="\nTotal elapsed time: {:.1f}"):
        while not done:
            for t in tasks:
                try:
                    next(t)
                except StopIteration:
                    tasks.remove(t)
                if len(tasks) == 0:
                    done = True


if __name__ == "__main__":
    main()
```

Додаток 16

example_async.py - C:\Users\alext\Desktop\MOODLE Питон\MOODLE\py\example_asyn

File Edit Format Run Options Window Help

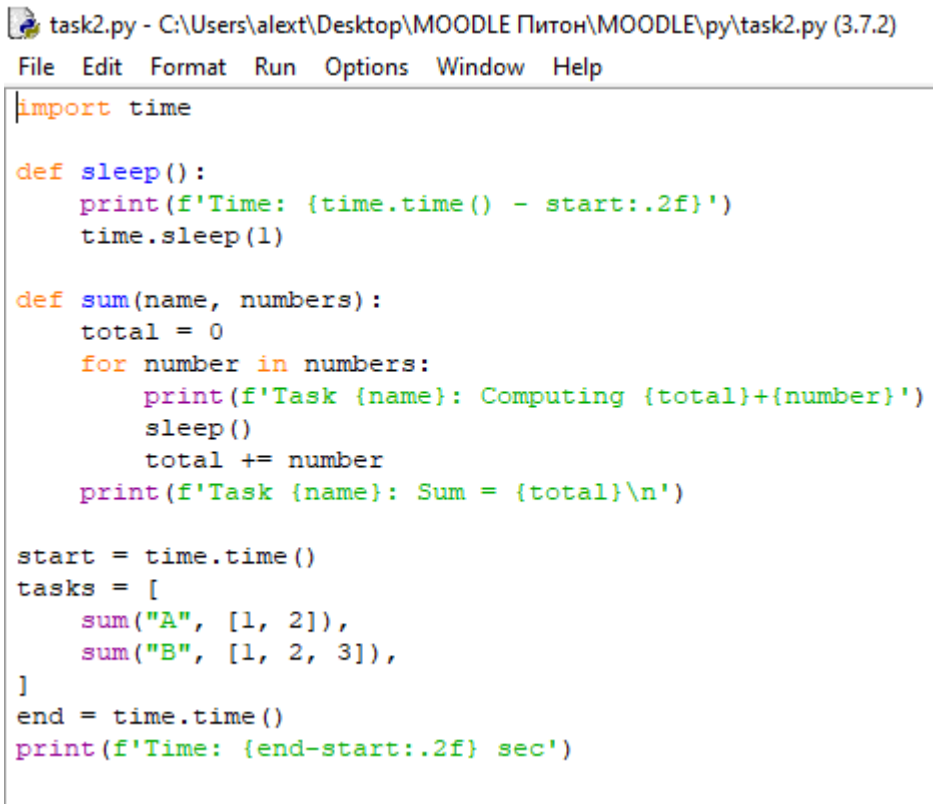
```
import asyncio
import aiohttp
from codetiming import Timer


async def task(name, work_queue):
    timer = Timer(text=f"Task {name} elapsed time: {{:.1f}}")
    async with aiohttp.ClientSession() as session:
        while not work_queue.empty():
            url = await work_queue.get()
            print(f"Task {name} getting URL: {url}")
            timer.start()
            async with session.get(url) as response:
                await response.text()
            timer.stop()


async def main():
    """
    This is the main entry point for the program
    """
    # Create the queue of work
    work_queue = asyncio.Queue()

    # Put some work in the queue
    for url in [
        "http://google.com",
        "http://yahoo.com",
        "http://linkedin.com",
        "http://apple.com",
        "http://microsoft.com",
        "http://facebook.com",
        "http://twitter.com",
    ]:
        await work_queue.put(url)


    # Run the tasks
    with Timer(text="\nTotal elapsed time: {:.1f}"):
        await asyncio.gather(
            asyncio.create_task(task("One", work_queue)),
            asyncio.create_task(task("Two", work_queue)),
        )


if __name__ == "__main__":
    asyncio.run(main())
```

Додаток 17

task.py - C:\Users\alext\Desktop\MOODLE Питон\MOODLE\py\task.py (3.7.2)

File Edit Format Run Options Window Help

```
import asyncio
import time

async def sleep():
    print(f'Time: {time.time() - start:.2f}')
    await asyncio.sleep(1)

async def sum(name, numbers):
    total = 0
    for number in numbers:
        print(f'Task {name}: Computing {total}+{number}')
        await sleep()
        total += number
    print(f'Task {name}: Sum = {total}\n')

start = time.time()

loop = asyncio.get_event_loop()
tasks = [
    loop.create_task(sum("A", [1, 2])),
    loop.create_task(sum("B", [1, 2, 3])),
]
loop.run_until_complete(asyncio.wait(tasks))
loop.close()

end = time.time()
print(f'Time: {end-start:.2f} sec')
```


task2.py - C:\Users\alext\Desktop\MOODLE Питон\MOODLE\py\task2.py (3.7.2)
File Edit Format Run Options Window Help
import time
def sleep():
print(f'Time: {time.time() - start:.2f}')
time.sleep(1)
def sum(name, numbers):
total = 0
for number in numbers:
print(f'Task {name}: Computing {total}+{number}')
sleep()
total += number
print(f'Task {name}: Sum = {total}\n')
start = time.time()
tasks = [
sum("A", [1, 2]),
sum("B", [1, 2, 3]),
]
end = time.time()
print(f'Time: {end-start:.2f} sec')

pymongo_example.py - C:\Users\alext\Desktop\MOODLE Питон\MOODLE\python-and-r

File Edit Format Run Options Window Help

```
from pymongo import MongoClient

# Establishing a Connection
client = MongoClient('localhost', 27017)

# Accessing Databases
db = client.pymongo_test

# Inserting Documents
posts = db.posts
post_data = {
    'title': 'Python and MongoDB',
    'content': 'PyMongo is fun, you guys',
    'author': 'Scott'
}
result = posts.insert_one(post_data)
print('One post: {0}'.format(result.inserted_id))

post_1 = {
    'title': 'Python and MongoDB',
    'content': 'PyMongo is fun, you guys',
    'author': 'Scott'
}
post_2 = {
    'title': 'Virtual Environments',
    'content': 'Use virtual environments, you guys',
    'author': 'Scott'
}
post_3 = {
    'title': 'Learning Python',
    'content': 'Learn Python, it is easy',
    'author': 'Bill'
}
new_result = posts.insert_many([post_1, post_2, post_3])
print('Multiple posts: {0}'.format(new_result.inserted_ids))
```

```
# Retrieving Documents
bills_post = posts.find_one({'author': 'Bill'})
print(bills_post)

scotts_posts = posts.find({'author': 'Scott'})
print(scotts_posts)
for post in scotts_posts:
    print(post)
```

Додаток 19

mongoengine_example.py - C:\Users\alext\Desktop\MOODLE Питон\MOODLE\python-and-mongo

File Edit Format Run Options Window Help

```
import datetime
from mongoengine import *

# Establishing a Connection
connect('mongoengine_test', host='localhost', port=27017)


# Defining a Document
class Post(Document):
    title = StringField(required=True, max_length=200)
    content = StringField(required=True)
    author = StringField(required=True, max_length=50)
    published = DateTimeField(default=datetime.datetime.now)

# Saving Documents
post_1 = Post(
    title='Sample Post',
    content='Some engaging content',
    author='Scott'
)
post_1.save()       # This will perform an insert
print(post_1.title)
post_1.title = 'A Better Post Title'
post_1.save()         # This will perform an atomic edit on "title"
print(post_1.title)

post_2 = Post(content='Content goes here', author='Michael')
post_2.save()
```

Додаток 20

C:\Users\alext\Desktop\MOODLE Питон\MOODLE\python-sqlite-sqlalchemy-main\project\data\author_book_publisher.

Файл Правка Поиск Вид Кодировки Синтаксисы Опции Инструменты Макросы Запуск Плагины

author_book_publisher.csv

```
first_name,last_name,title,publisher
Isaac,Asimov,Foundation,Random House
Pearl,Buck,The Good Earth,Random House
Pearl,Buck,The Good Earth,Simon & Schuster
Tom,Clancy,The Hunt For Red October,Berkley
Tom,Clancy,Patriot Games,Simon & Schuster
Stephen,King,It,Random House
Stephen,King,It,Penguin Random House
Stephen,King,Dead Zone,Random House
Stephen,King,The Shining,Penguin Random House
John,Le Carre,"Tinker, Tailor, Solider, Spy: A George Smiley Novel",Berkley
Alex,Michaelides,The Silent Patient,Simon & Schuster
Carol,Shaben,Into The Abyss,Simon & Schuster

```

Додаток 21

main.py - C:\Users\alext\Desktop\MOODLE Питон\MOODLE\python-sqlite-sqlalchemy-main\project\examples

File Edit Format Run Options Window Help

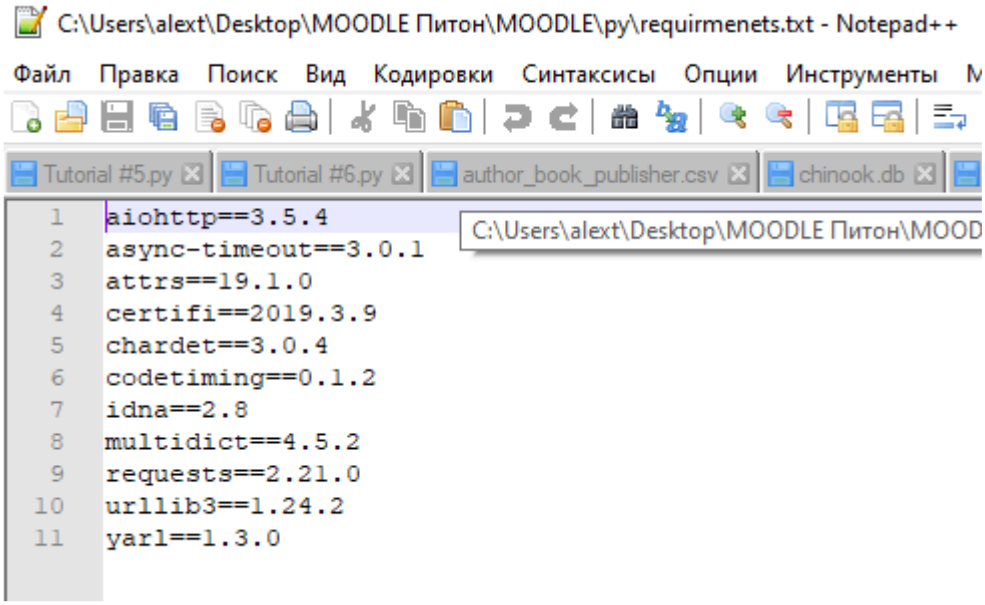
```
import pandas as pd
from treelib import Tree


def get_data(filepath):
    """Get book data from the csv file"""
    return pd.read_csv(filepath)


def get_books_by_publisher(data, ascending=True):
    """Returns the books by each publisher as a pandas series

    Args:
        data: The pandas dataframe to get the from
        ascending: The sorting direction for the returned data.
        Defaults to True.

    Returns:
        The sorted data as a pandas series
    """
    return data.groupby("publisher").size().sort_values(ascending=ascending)


def get_authors_by_publisher(data, ascending=True):
    """Returns the authors by each publisher as a pandas series

    Args:
        data: The pandas dataframe to get the data from
        ascending: The sorting direction for the returned data.
        Defaults to True.

    Returns:
        The sorted data as a pandas series
    """
    return (
        data.assign(name=data.first_name.str.cat(data.last_name, sep=" "))
        .groupby("publisher")
        .nunique()
        .loc[:, "name"]
        .sort_values(ascending=ascending)
    )
```

```
    )


def add_new_book(data, author_name, book_title, publisher_name):
    """Adds a new book to the system"""

    # Does the book exist?
    first_name, _, last_name = author_name.partition(" ")
    if any(
        (data.first_name == first_name)
        & (data.last_name == last_name)
        & (data.title == book_title)
        & (data.publisher == publisher_name)
    ):
        return data
    # Add the new book
    return data.append(
        {
            "first_name": first_name,
            "last_name": last_name,
            "title": book_title,
            "publisher": publisher_name,
        },
        ignore_index=True,
    )


def output_author_hierarchy(data):
    """Output the data as a hierarchy list of authors"""
    authors = data.assign(
        name=data.first_name.str.cat(data.last_name, sep=" ")
    )
    authors_tree = Tree()
    authors_tree.create_node("Authors", "authors")
    for author, books in authors.groupby("name"):
        authors_tree.create_node(author, author, parent="authors")
        for book, publishers in books.groupby("title")["publisher"]:
            book_id = f"{author}:{book}"
            authors_tree.create_node(book, book_id, parent=author)
```

```
                authors_tree.create_node(publisher, parent=book_id)

    # Output the hierarchical authors data
    authors_tree.show()


def main():
    """The main entry point of the program"""

    # Get the resources for the program
    with resources.path(
        "project.data", "author_book_publisher.csv"
    ) as filepath:
        data = get_data(filepath)

    # Get the number of books printed by each publisher
    books_by_publisher = get_books_by_publisher(data, ascending=False)
    for publisher, total_books in books_by_publisher.items():
        print(f"Publisher: {publisher}, total books: {total_books}")
    print()

    # Get the number of authors each publisher publishes
    authors_by_publisher = get_authors_by_publisher(data, ascending=False)
    for publisher, total_authors in authors_by_publisher.items():
        print(f"Publisher: {publisher}, total authors: {total_authors}")
    print()

    # Output hierarchical authors data
    output_author_hierarchy(data)
```

```
        # Add a new book to the data structure
        data = add_new_book(
            data,
            author_name="Stephen King",
            book_title="The Stand",
            publisher_name="Random House",
        )
        # Output the updated hierarchical authors data
        output_author_hierarchy(data)


    if __name__ == "__main__":
        main()
```

Додаток 22



Додаток 23

```python
from importlib import resources

from sqlalchemy import and_, create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.sql import asc, desc, func

from project.modules.models import Author, Book, Publisher
from treelib import Tree


def get_books_by_publishers(session, ascending=True):
    """Get a list of publishers and the number of books they've published

    Args:
        session: database session to use
        ascending: direction to sort the results

    Returns:
        List: list of publisher sorted by number of books published
    """
    if not isinstance(ascending, bool):
        raise ValueError(f"Sorting value invalid: {ascending}")

    direction = asc if ascending else desc

    return (
        session.query(
            Publisher.name, func.count(Book.title).label("total_books")
        )
        .join(Publisher.books)
        .group_by(Publisher.name)
        .order_by(direction("total_books"))
    )
```

```python
def get_authors_by_publishers(session, ascending=True):
    """Get a list of publishers and the number of authors they've published

    Args:
        session: database session to use
        ascending: direction to sort the results

    Returns:
        List: list of publisher sorted by number of authors published
    """
    if not isinstance(ascending, bool):
        raise ValueError(f"Sorting value invalid: {ascending}")

    direction = asc if ascending else desc

    return (
        session.query(
            Publisher.name,
            func.count(Author.first_name).label("total_authors"),
        )
        .join(Publisher.authors)
        .group_by(Publisher.name)
        .order_by(direction("total_authors"))
    )


def get_authors(session):
    """Get a list of author objects sorted by last name"""
    return session.query(Author).order_by(Author.last_name).all()


def add_new_book(session, author_name, book_title, publisher_name):
    """Adds a new book to the system"""

```

```
    # Get the author's first and last names
    first_name, _, last_name = author_name.partition(" ")

    # Check if the book exists
    book = (
        session.query(Book)
        .join(Author)
        .filter(Book.title == book_title)
        .filter(
            and_(
                Author.first_name == first_name, Author.last_name == last_name
            )
        )
        .filter(Book.publishers.any(Publisher.name == publisher_name))
        .one_or_none()
    )
    # Does the book by the author and publisher already exist?
    if book is not None:
        return

    # Check if the book exists for the author
    book = (
        session.query(Book)
        .join(Author)
        .filter(Book.title == book_title)
        .filter(
            and_(
                Author.first_name == first_name, Author.last_name == last_name
            )
        )
        .one_or_none()
    )
    # Create the new book if needed
```

```
    if book is None:
        book = Book(title=book_title)

    # Get the author
    author = (
        session.query(Author)
        .filter(
            and_(
                Author.first_name == first_name, Author.last_name == last_name
            )
        )
        .one_or_none()
    )
    # Do we need to create the author?
    if author is None:
        author = Author(first_name=first_name, last_name=last_name)
        session.add(author)

    # Get the publisher
    publisher = (
        session.query(Publisher)
        .filter(Publisher.name == publisher_name)
        .one_or_none()
    )
    # Do we need to create the publisher?
    if publisher is None:
        publisher = Publisher(name=publisher_name)
        session.add(publisher)

    # Initialize the book relationships
    book.author = author
    book.publishers.append(publisher)
    session.add(book)
```

```
    # Commit to the database
    session.commit()


def output_author_hierarchy(authors):
    """
    Outputs the author/book/publisher information in
    a hierarchical manner

    :param authors:         the collection of root author objects
    :return:                None
    """
    authors_tree = Tree()
    authors_tree.create_node("Authors", "authors")
    for author in authors:
        author_id = f"{author.first_name} {author.last_name}"
        authors_tree.create_node(author_id, author_id, parent="authors")
        for book in author.books:
            book_id = f"{author_id}:{book.title}"
            authors_tree.create_node(book.title, book_id, parent=author_id)
            for publisher in book.publishers:
                authors_tree.create_node(publisher.name, parent=book_id)
    # Output the hierarchical authors data
    authors_tree.show()


def main():
    """Main entry point of program"""

    # Connect to the database using SQLAlchemy
    with resources.path(
        "project.data", "author_book_publisher.db"
    ) as sqlite_filepath:
        engine = create_engine(f"sqlite:///{sqlite_filepath}")
    Session = sessionmaker()
```

```
    Session = sessionmaker()
    Session.configure(bind=engine)
    session = Session()

    # Get the number of books printed by each publisher
    books_by_publisher = get_books_by_publishers(session, ascending=False)
    for row in books_by_publisher:
        print(f"Publisher: {row.name}, total books: {row.total_books}")
    print()

    # Get the number of authors each publisher publishes
    authors_by_publisher = get_authors_by_publishers(session)
    for row in authors_by_publisher:
        print(f"Publisher: {row.name}, total authors: {row.total_authors}")
    print()

    # Output hierarchical authors data
    authors = get_authors(session)
    output_author_hierarchy(authors)

    # Add a new book
    add_new_book(
        session,
        author_name="Stephen King",
        book_title="The Stand",
        publisher_name="Random House",
    )
    # Output the updated hierarchical authors data
    authors = get_authors(session)
    output_author_hierarchy(authors)


if __name__ == "__main__":
    main()
```

Додаток 24

Tutorial #6.py | chinook.db | chinook_server.py | auth

```
from app import app


if __name__ == "__main__":
    app.run("0.0.0.0", port=5000)

```

Література

1. Mark Lutz Learning Python 2019 С-832.
2. Дмитрий Мусин Самоучитель Python М. 2019 С-149.
3. Эл Свейгарт Учим Python, делая крутые игры 2018 С-416.
4. О.Васильєв Програмування мовою Python К. 2019 С-504.
5. https://thecode.media/pygames/
6. https://www.youtube.com/watch?v=wDgZdYRQ4gU.
7. https://pythontutor.ru/lessons/2d_arrays/
8. http://labs.org.ru/python-8/
9. http://progras.ru/31-dvumernye-spiski-massivy-matricy-v-python/
10. https://habr.com/ru/post/246699/
11. https://habr.com/ru/post/478620/
12. https://developer.mozilla.org/uk/docs/Glossary/REST
13. https://uk.wikipedia.org/wiki/REST
14. https://dev.to/code_enzyme/introduction-to-using-async-await-in-python-2i0n
15. https://webdevblog.ru/obzor-async-io-v-python-3-7/