МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

МЕТОДИЧНИЙ ПОСІБНИК

СПЕЦІАЛЬНІ МОВИ ПРОГРАМУВАННЯ

ЧАСТИНА 2

Київ 2023

УДК 681.3.07

Гриф надано Навчально-науковим інститутом інформаційних технологій Державного університету телекомунікацій (протокол № 6 від 30.01.2023 року)

Рецензенти: Складанний П.М., кандидат технічних наук, доцент, завідувач кафедри інформаційної та кібернетичної безпеки імені Володимира Бурячка факультету інформаційних технологій та управління Київського університету імені Бориса Грінченка

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

СПЕЦІАЛЬНІ МОВИ ПРОГРАМУВАННЯ (Частина 2)

Методичні рекомендації підготовлено до друку для самостійної роботи студентів вищих навчальних закладів. – К.: ННІІТ ДУТ, 2023. - 202 с.

Методичні рекомендації до викладання дисципліни «Спеціальні мови програмування (Частина 2)» 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-гігантами, такими як Google і Yandex. Python практично нічим не обмежений, тому 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 фреймворку Django (Джанго) для веб-застосунків мовою Python. Переваги фреймворку Django це величезний функціонал, за допомогою якого легко організувати як систему реєстрації, так і форум на сайті, або виконати інший вид роботи. Фреймворк дозволяє зручно розбити файли на категорії: HTML-шаблони, файли моделі для роботи з базами даних та файли контролери для зв'язку моделі та HTML шаблонів між собою. У посібнику розглянуті бібліотеки NumPy та Matplotlib. Пакет NumPy використовується для аналізу даних, у машинному навчанні та наукових обчисленнях. Він суттєво полегшує обробку векторів та матриць. Вміння працювати з NumPy дає значну перевагу при налагодженні складніших сценаріїв бібліотек. Бібліотека Matplotlib у Python допомагає будувати графіки, відображати дані на графіках та візуалізовувати фігури для представлення даних.

Практична робота №1

Тема. Фреймворк Django (Джанго). Перші кроки.

Мета роботи. Ознайомитись з використанням фреймворку Django для створення сайтів.

Зміст.

1. Вивчення відомостей про фреймворк Django.
2. Виконання роботи.
3. Отримання результату.

Хід роботи

Django (Джанго) — вільний фреймворк для веб-застосунків мовою Python, що використовує шаблон проектування MVC.

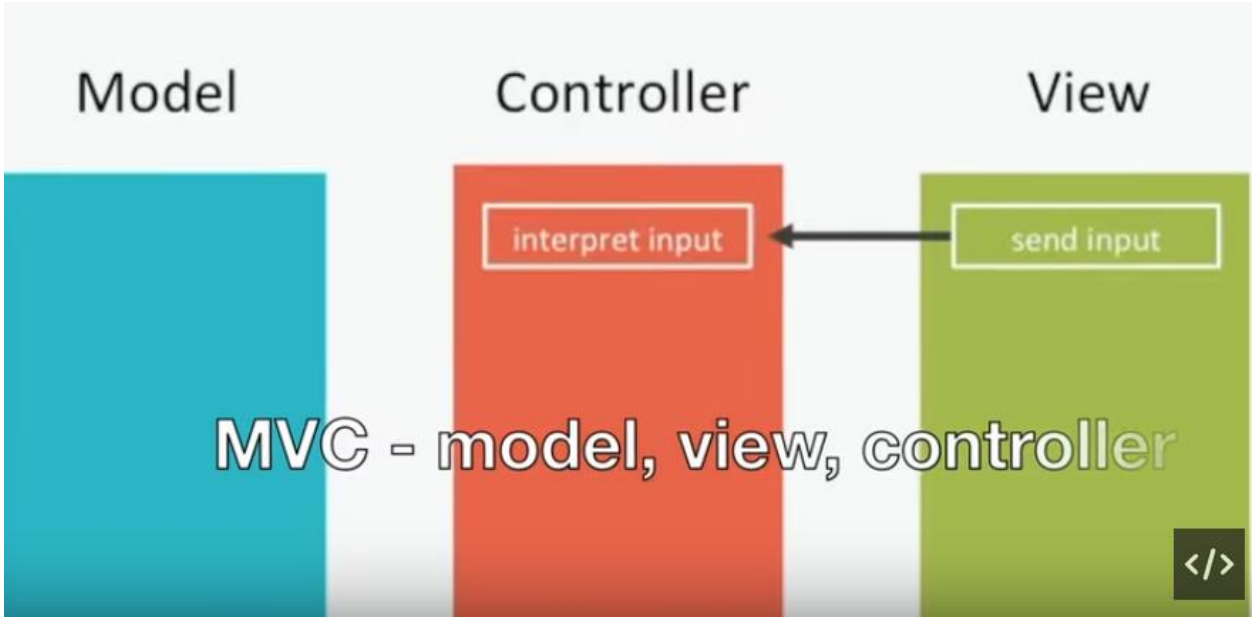


Рисунок 1.1

Проект підтримує організація Django Software Foundation. Сайт на Django будується з одного або декількох додатків, які рекомендується робити відчужуваними та такими, що підключаються. Це одна із суттєвих архітектурних відмінностей цього фреймворку від деяких інших (наприклад, Ruby on Rails). Також, на відміну від інших фреймворків, обробники URL в Django конфігуруються явно за допомогою регулярних виразів. Для роботи з базою даних Django використовує власну ORM, в якій модель даних описується класами Python, і по ній генерується схема бази даних.

Переваги – величезний функціонал, за допомогою якого легко організувати як систему реєстрації, так і форум на сайті або виконати інший вид роботи. Він дозволяє зручно розбити файли на категорії: HTML-шаблони, файли моделі для роботи з базами даних та файли контролери для зв'язку моделі та HTML шаблонів між собою. Компанія Google дуже любить цей фреймворк і на його основі написано багато проектів (Youtube і Google-пошук). Як текстовий редактор ви можете використовувати Atom, Sublime Text, Visual Studio, PyCharm. Завітайте на офіційний сайт Python і завантажте PyCharm.

Рисунок 1.2

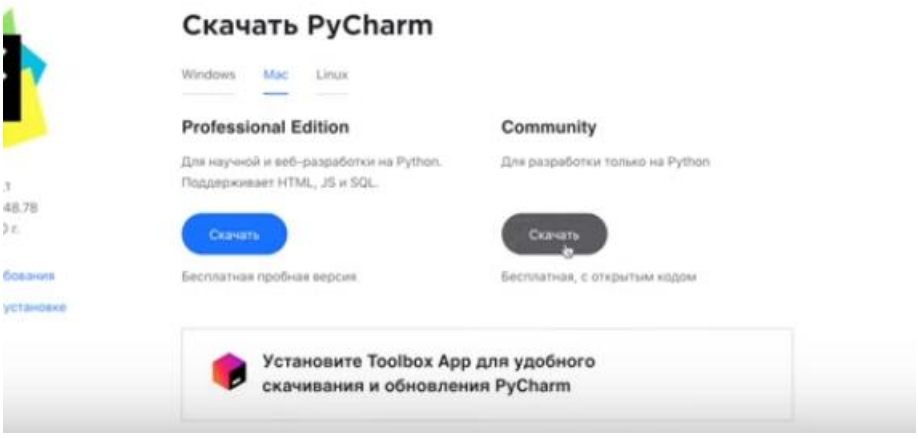


Рисунок 1.3

На робочому столі створюємо нову папку з назвою learnDjangoUA, натискаємо праву кнопку миші на новоствореній папці і натискаємо Open Folder as PyCharm Community Edition Project.

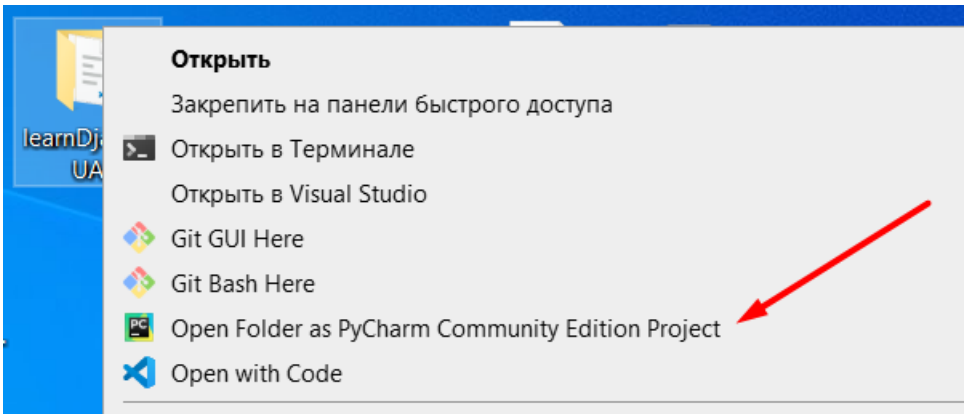


Рисунок 1.4

Встановлюємо Джанго:

1. Відкриваємо термінал.

2. За допомогою команди clear – очищаємо.

3. За допомогою пакетного менеджера pip чи pip3 встановлюємо Джанго.

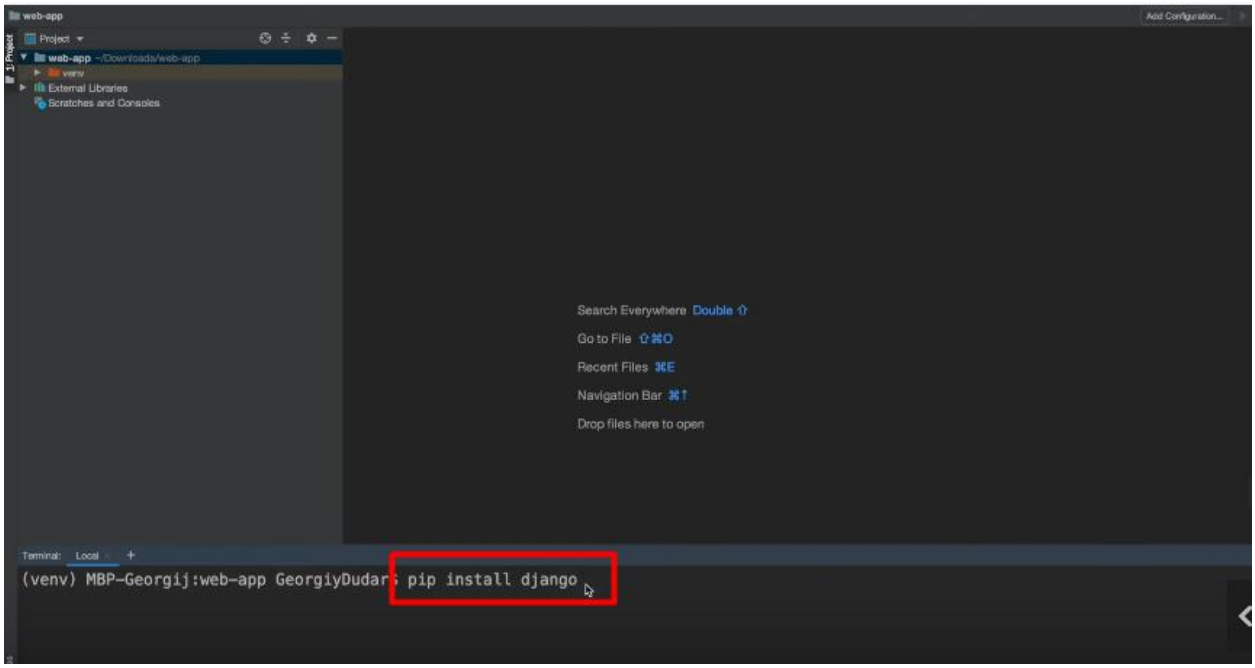


Рисунок 1.5

4. Натискаємо Enter.

5. Створюємо порожній проект з урахуванням Джанго. У терміналі вписуємо назву проекту website. Зверніть увагу, що ім'я_проекту - це ваше власне ім'я проекту, яке має бути допустимим ім'ям пакету Python.

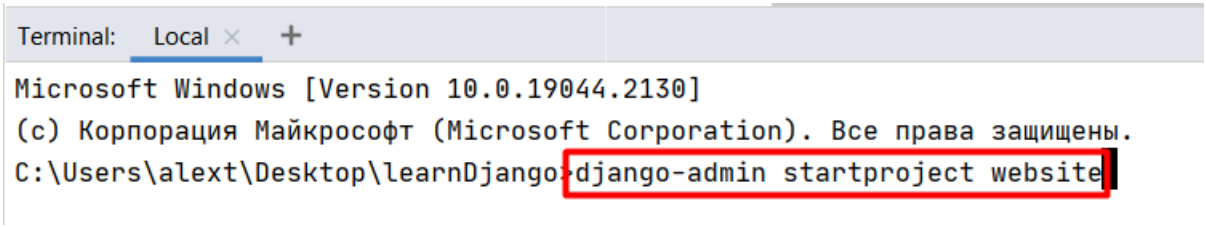


Рисунок 1.6

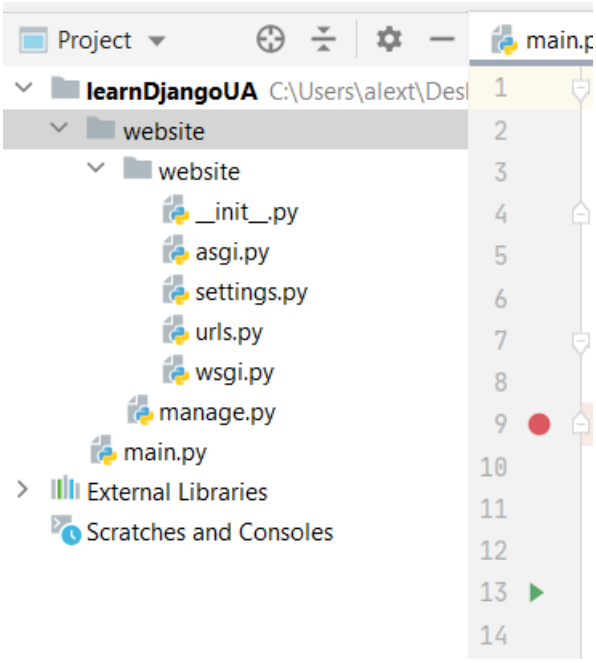


Рисунок 1.7

Manage.py автоматично створюється у кожному проекті Django. Це утиліта командного рядка Django для виконання адміністративних завдань. Він робить те саме, що й django-admin, встановлює змінну оточення DJANGO_SETTINGS_MODULE так, щоб вона вказувала на файл settings.py вашого проекту. Як правило, під час роботи над одним проектом Django простіше використовувати manage.py, ніж django-admin. Якщо вам потрібно перемикатися між кількома налаштуваннями Django, використовуйте django-admin з DJANGO_SETTINGS_MODULE або опцію командного рядка settings. Відкриваємо файл, що з'явився, **manage.py**.

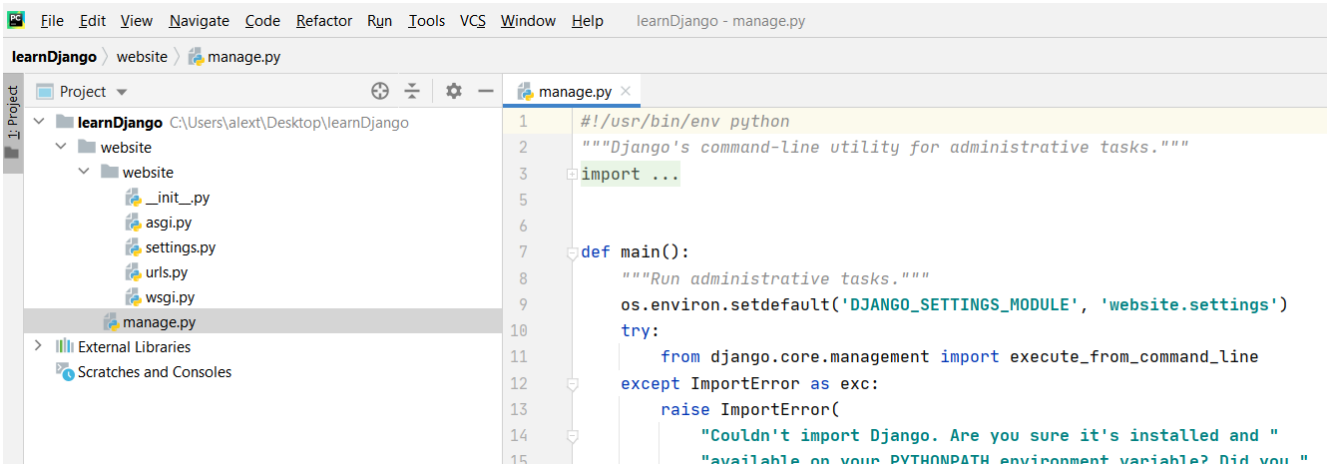


Рисунок 1.8

Змінювати файл **manage.py** не треба, він створений у каталозі проекту виключно для зручності.

__init__.py: цей файл необхідний для того, щоб Python розглядав каталог mysite як пакет (групу Python-модулів). Цей файл порожній, і додавати до нього, як правило, нічого не потрібно.

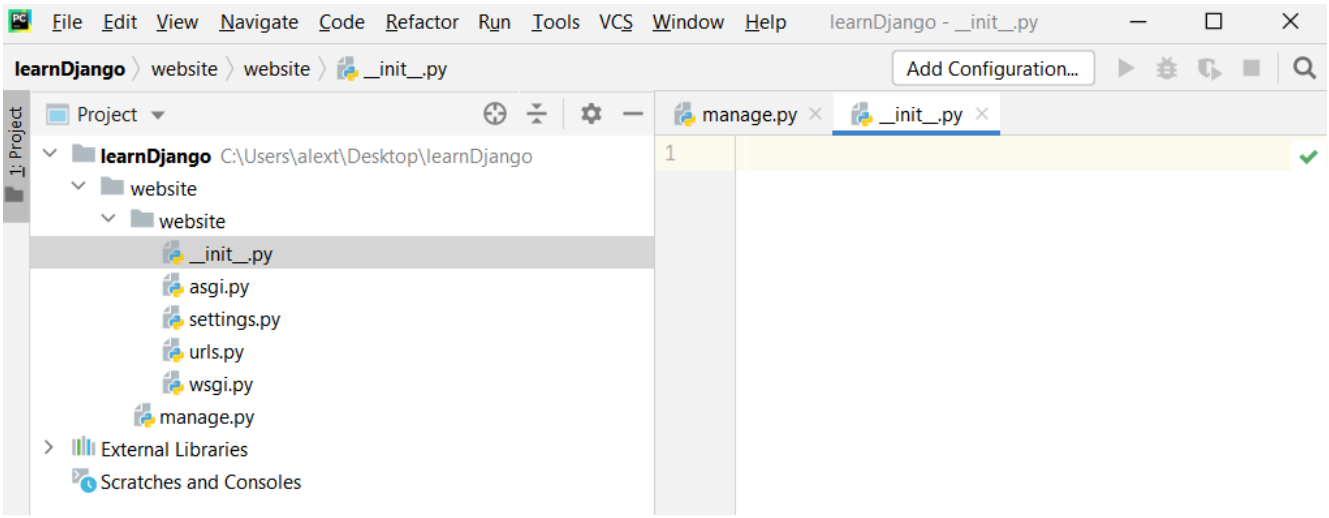


Рисунок 1.9

Поки що це порожній файл. Потім допишемо характеристики, які будуть використані в цьому проекті. Далі файли, які схожі та забезпечують коректне підключення до сервера. WSGI був написаний для синхронного коду. Новий стандарт, аналогічний WSGI для асинхронного коду – ASGI.

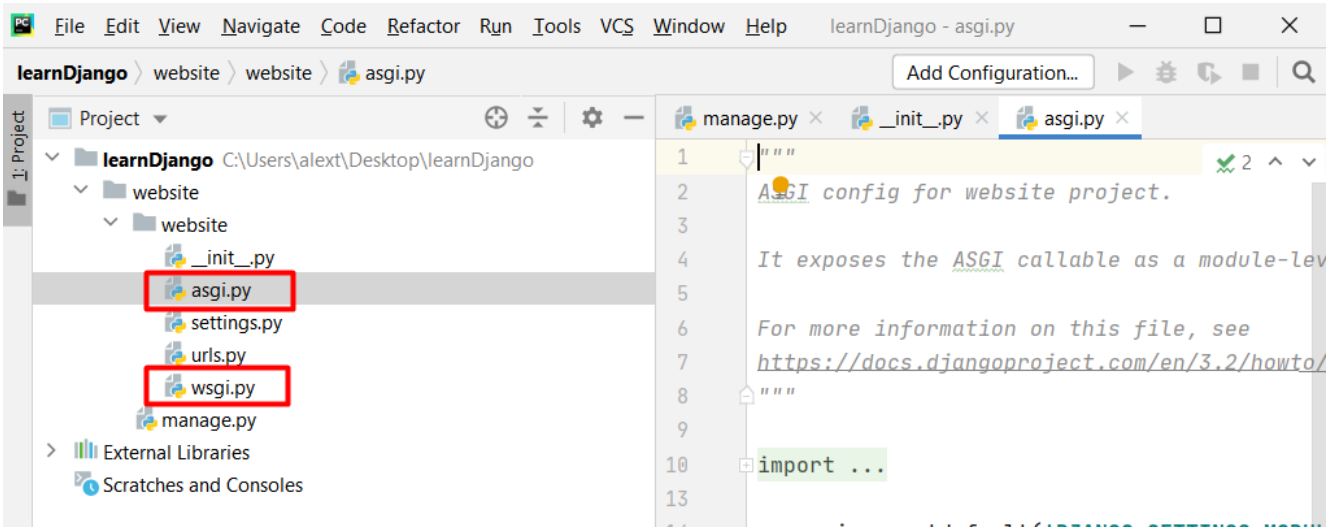


Рисунок 1.10

Файл **settings.py** описує всі глобальні налаштування для вашого проекту Джанго, змінні.

Всередині записується повний шлях до вашого проекту.

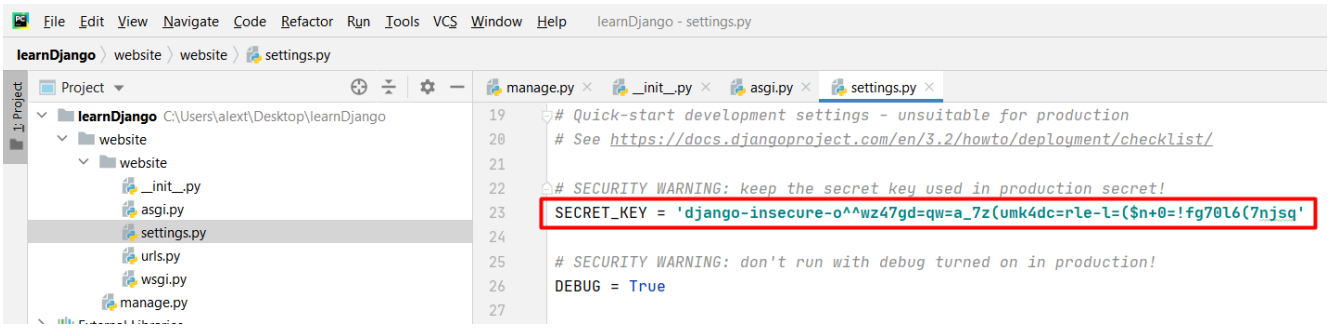


Рисунок 1.11

Секретний ключ до вашого проекту.

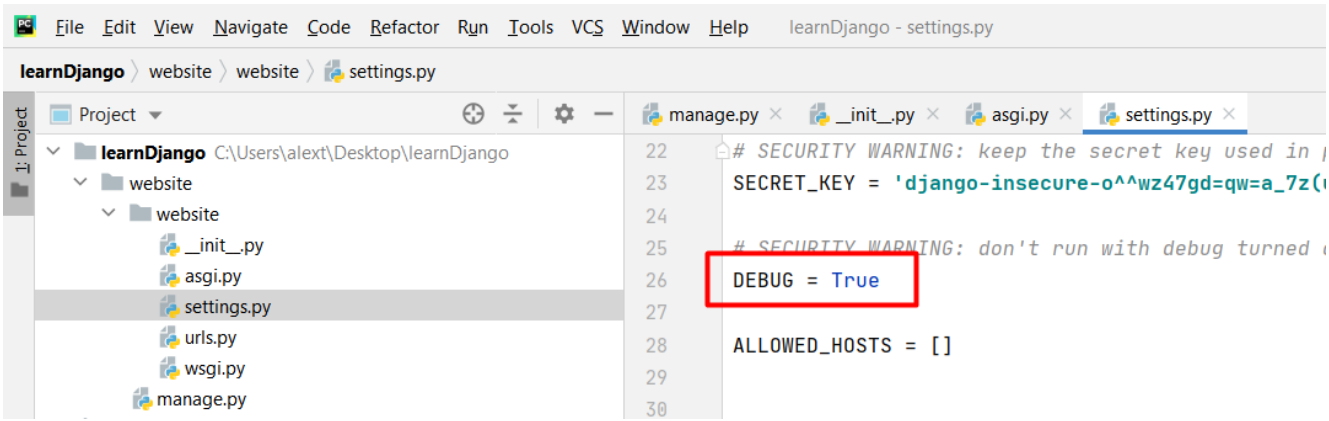


Рисунок 1.12

Змінна DEBUG - встановлена як true. Усі помилки будуть показані на сторінках веб-сайту. Коли помилок не буде, змінимо цю змінну DEBUG на false.

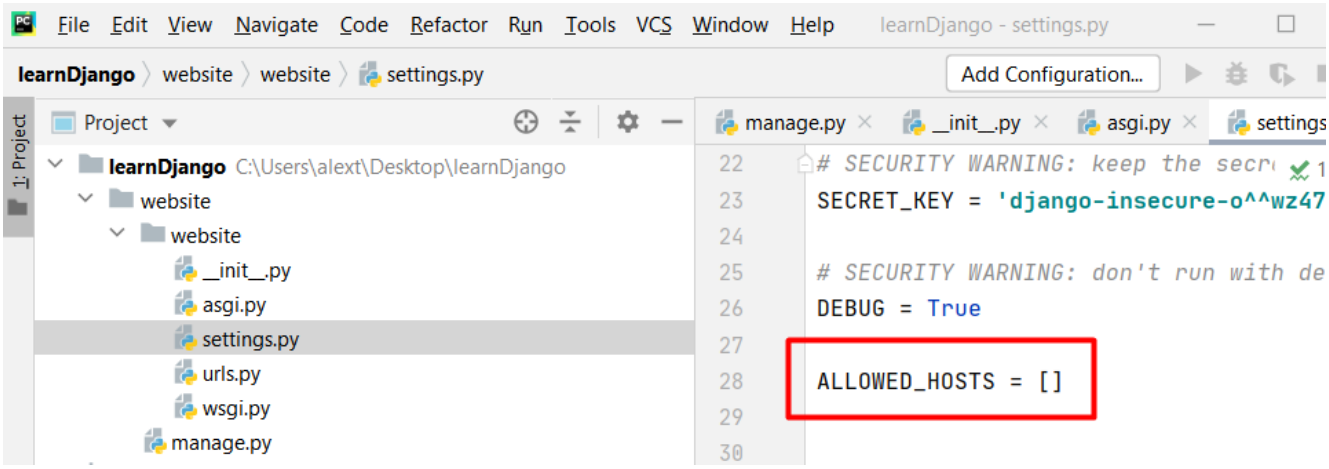


Рисунок 1.13

У файлі **settings.py** вказують ті доменні імена, на яких дозволено опубліковувати даний проект.

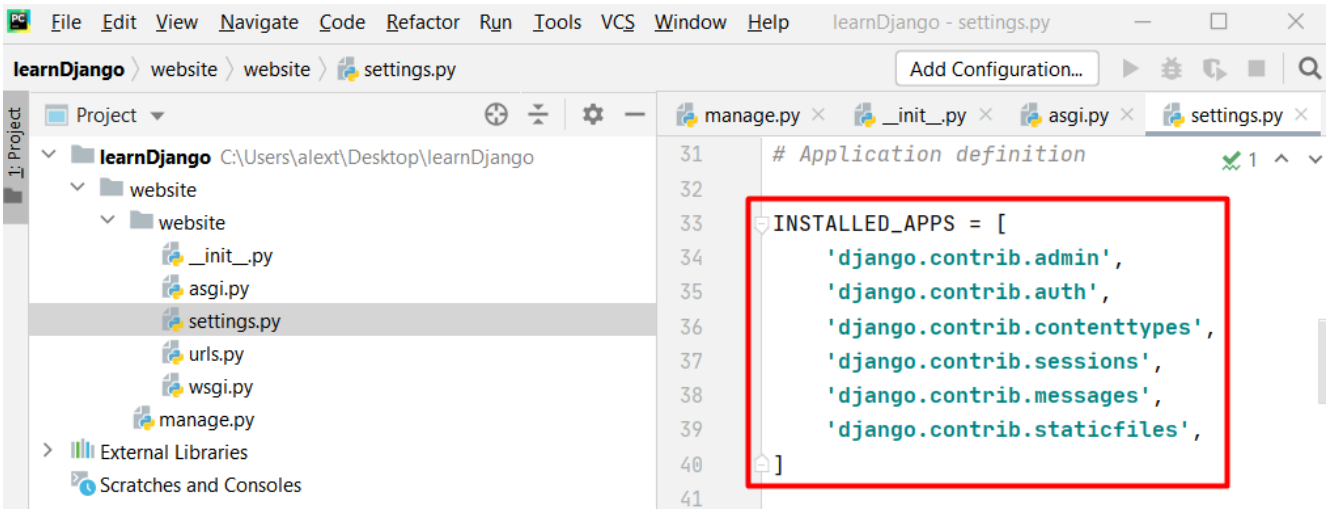


Рисунок 1.14

У файлі **settings.py** вказують ті доменні імена, на яких дозволено опубліковувати даний проект. Список зберігає набір встановлених програм, які зараз є у проекті. Перше це панель адміністратора **django.contrib.admin.** Для авторизації використовують **django.contrib.auth**. **django.contrib.contenttypes** - включає програму contenttypes, яка може відстежувати всі моделі, встановлені у вашому проекті на базі Django, надаючи високорівневий, загальний інтерфейс для роботи з вашими моделями. Додаток по роботі з сесіями **django.contrib.sessions**, по роботі з повідомленнями **django.contrib.message**s по роботі зі статичними файлами **django.contrib.staticfiles**.

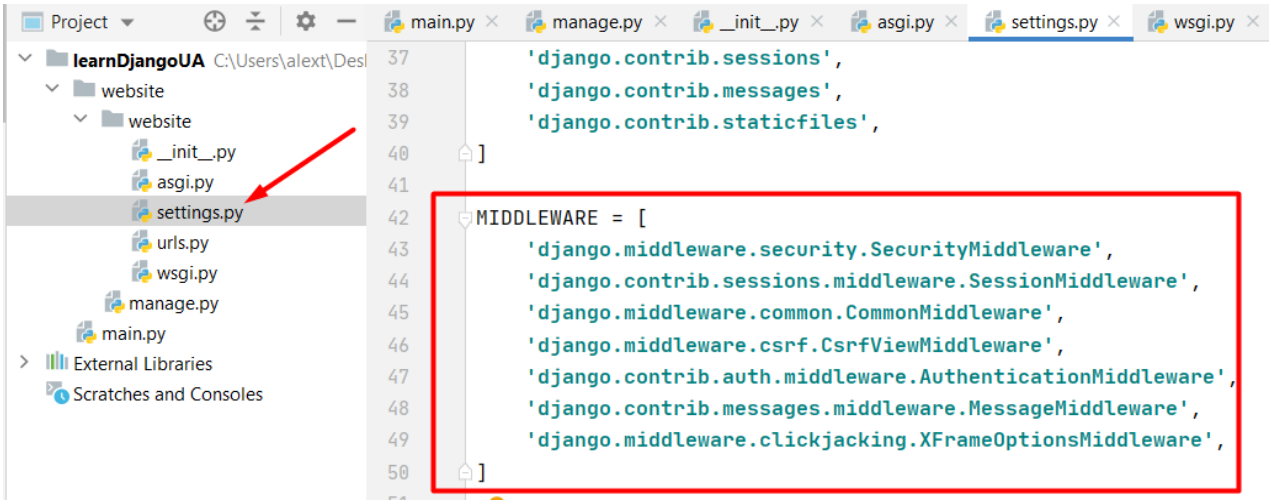


Рисунок 1.15

У списку записано різне проміжне програмне забезпечення.

Забезпечення безпеки - **django.middleware.security.SecurityMiddleware.** Цей middleware включений за замовчуванням і має бути найпершим у списку, щоб зловмисні запити блокувалися до обробки іншими middleware. Він захищає відразу від багатьох атак і кожну з цих захистів можна ввімкнути/вимкнути окремо, використовуючи змінні в settings.py. Робота із сесіями всередині проекту **django.contrib.sessions.middleware.SessionMiddleware.** Додає кілька зручностей для перфекціоністів - **django.middleware.common.CommonMiddleware.**

django.middleware.csrf.CsrfViewMiddleware. Цей Middleware захищає від міжсайтової підробки запиту. При надсиланні будь-яких небезпечних (POST, PUT, DELETE) запитів middleware перевірятиме наявність csrf токена і якщо цей токен не знайдений або не коректний запит фільтруватиметься. Підтримка авторизації **AuthenticationMiddleware**. При роботі з діями користувача може знадобитися інформувати користувачів про результати їх дій. Джанго має вбудовану платформу повідомлень, яка дозволяє відображати сповіщення для користувачів. Платформа повідомлень розташована на рівні **django.contrib.messages**.

django.middleware.clickjacking.XFrameOptionsMiddleware - при включенні цей middleware додає заголовок X-Frame-Options: SAMEORIGIN, який забороняє сучасним браузерам завантажувати сайт у (i)frame на сайтах з іншим доменом.

Також можна заборонити підключення сайту з усіх доменів за допомогою змінної X_FRAME_OPTIONS = 'DENY'

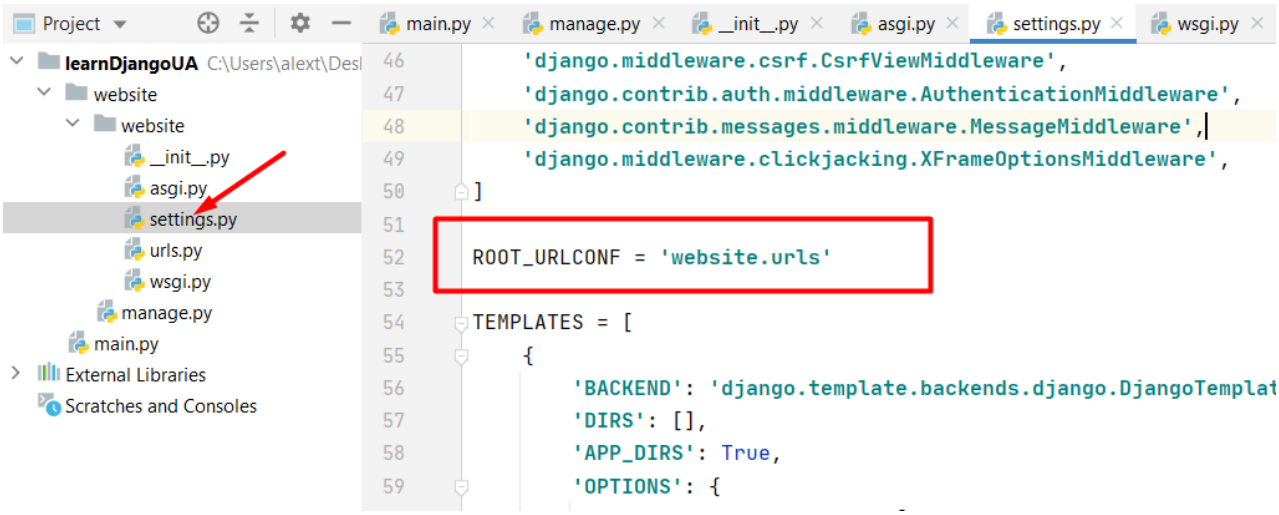


Рисунок 1.16

Вказано який urls буде використаний під час роботи проекту.

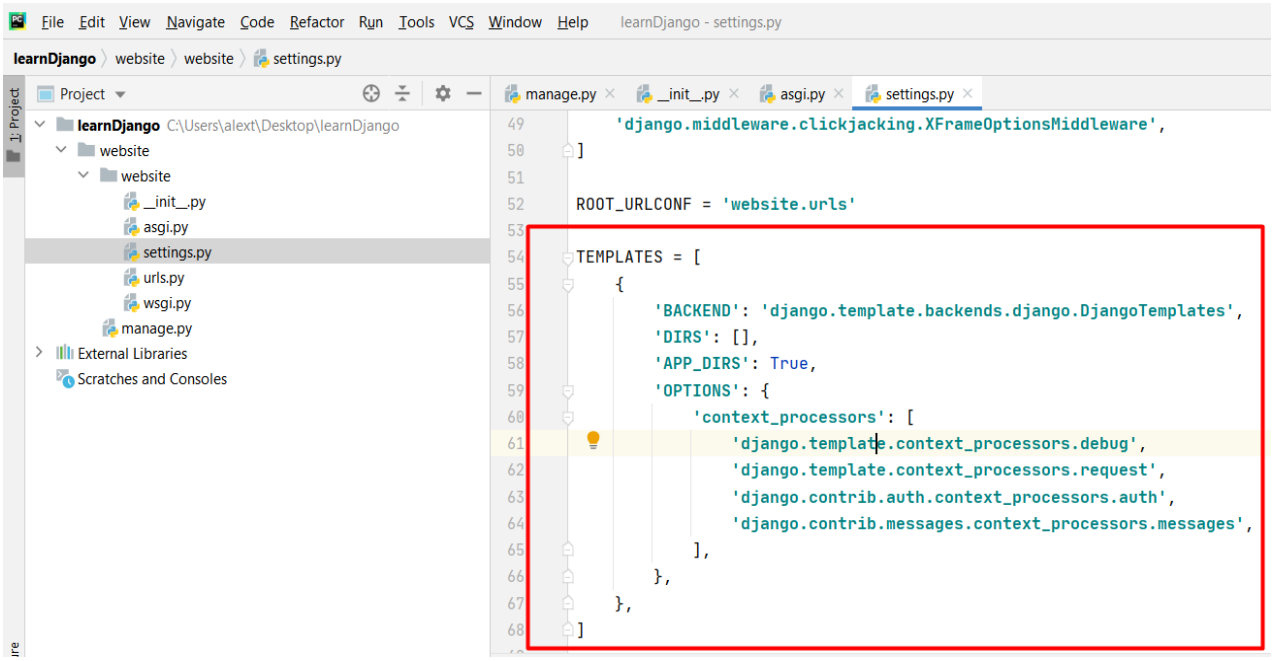


Рисунок 1.17

Шаблони, які ми будемо використовувати під час нашого проекту.

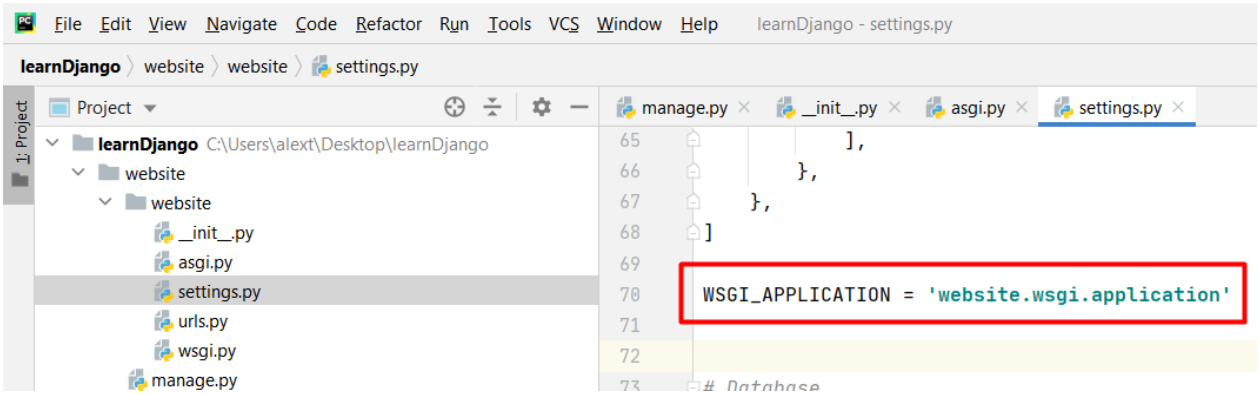


Рисунок 1.18

За допомогою змінної та цього WSGI ми зможемо вивантажити наш сайт на сервер.

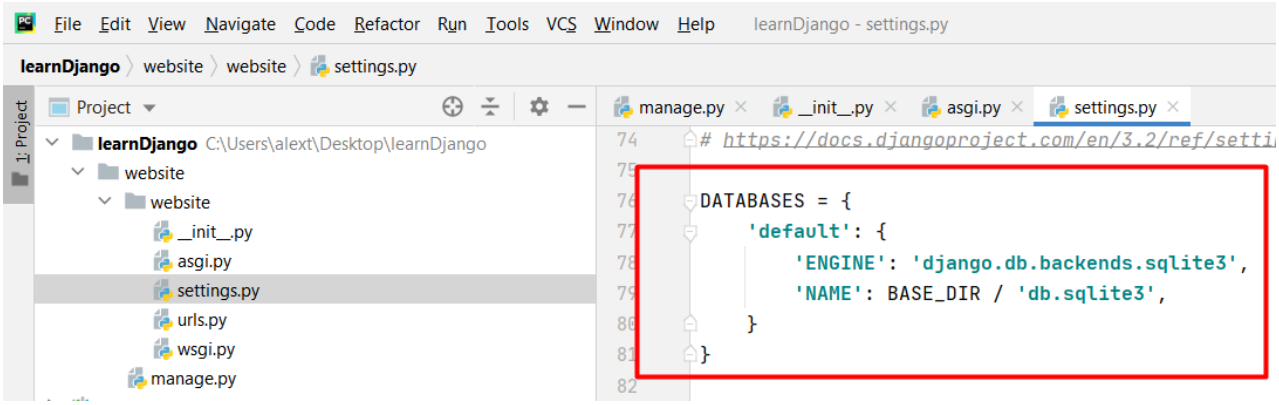


Рисунок 1.19

Вказано з якою базою даних ми працюватимемо - **db.sqlite3**.

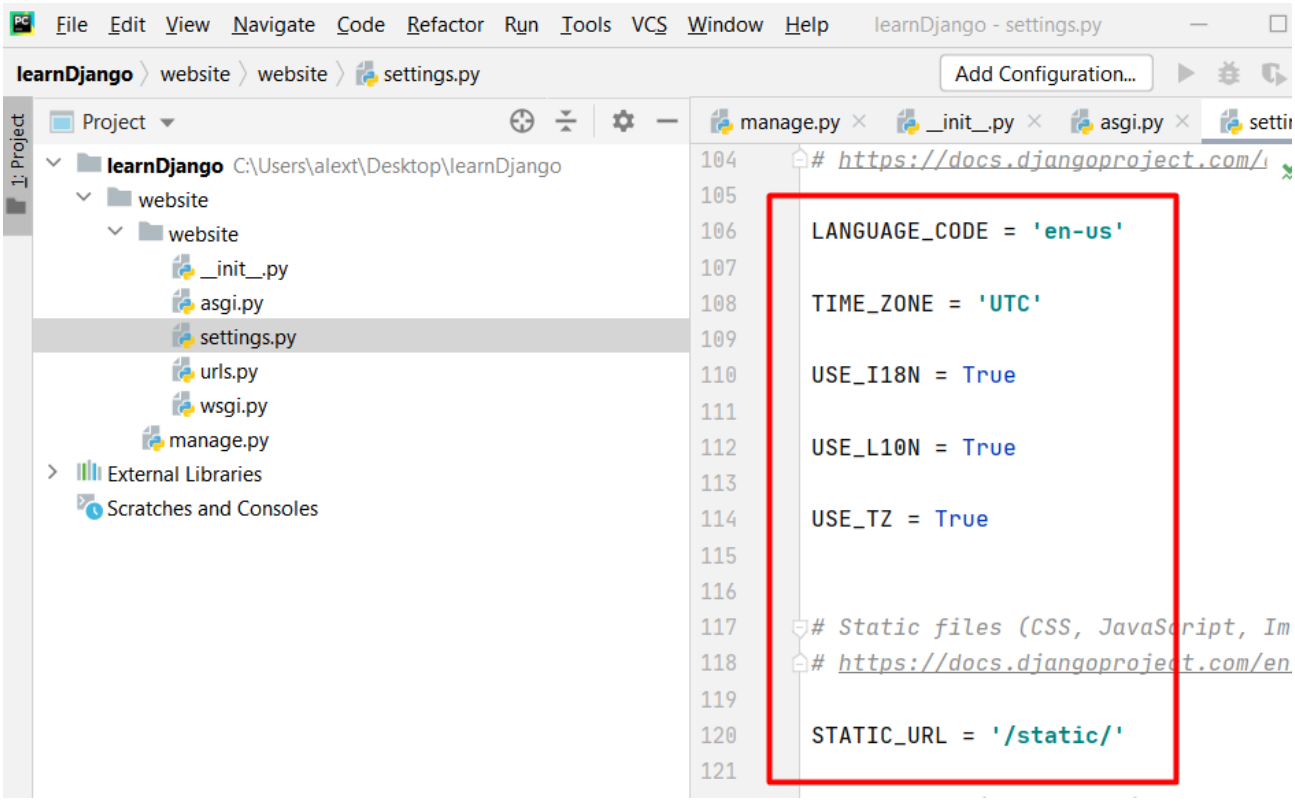


Рисунок 1.20

Ми можемо змінювати мову програми, тимчасову зону для нашої програми.

Всередині файлу **urls.py** відстежуються різні адреси urls.

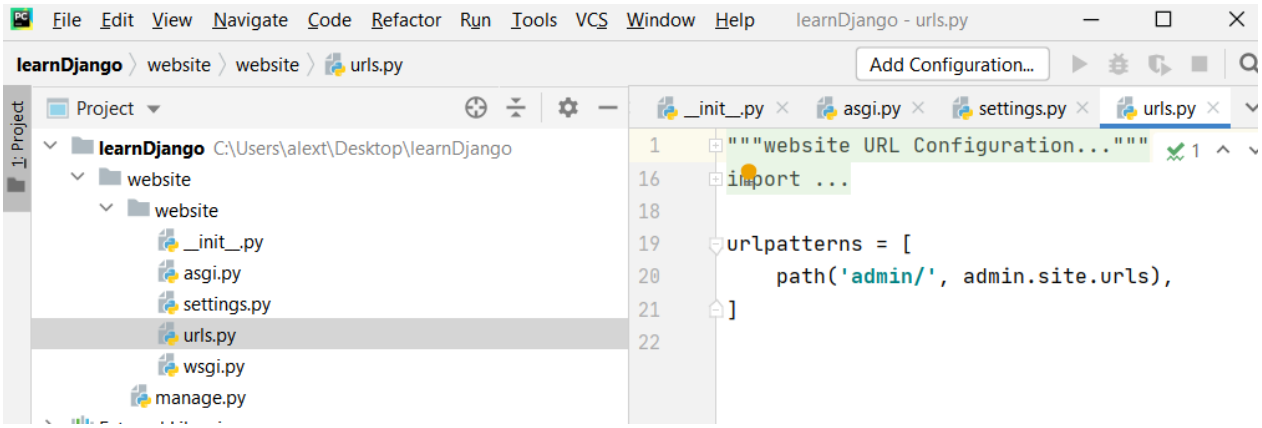


Рисунок 1.21

При переході за такою адресою urls як адмін буде відкриватися додаток теж як адмін.

Перебуваючи в папці, що містить **manage.py**, запускаємо сервер.

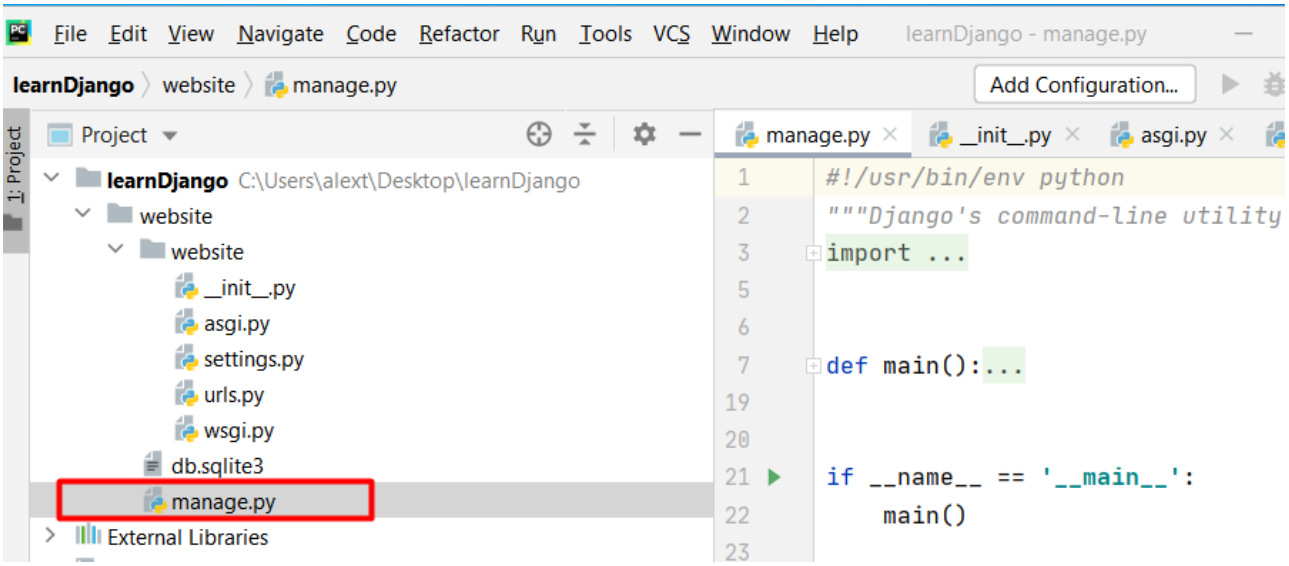


Рисунок 1.22

Запустимо локальний сервер.

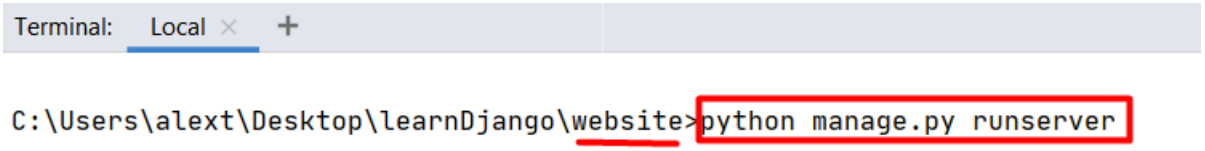


Рисунок 1.23

Через деякий час сервер запуститься.

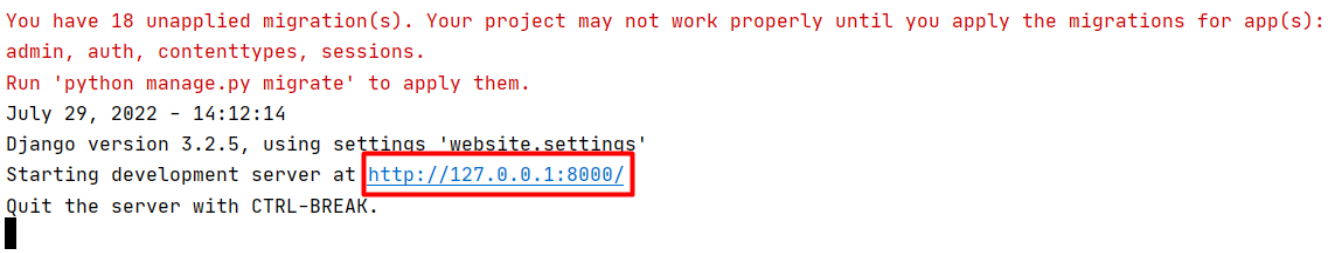


Рисунок 1.24

Натиснувши на посилання, ми перейдемо на наш сайт.

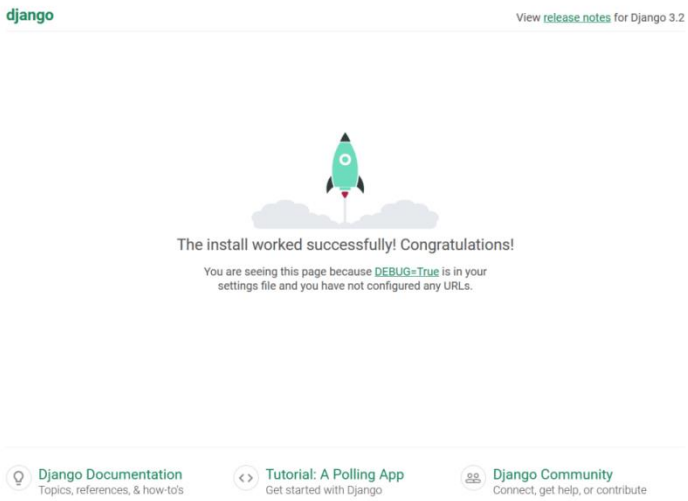


Рисунок 1.25

Створимо новий додаток. Перебуваючи в директорії, що містить **manage.py**, введемо в терміналі наступну команду:

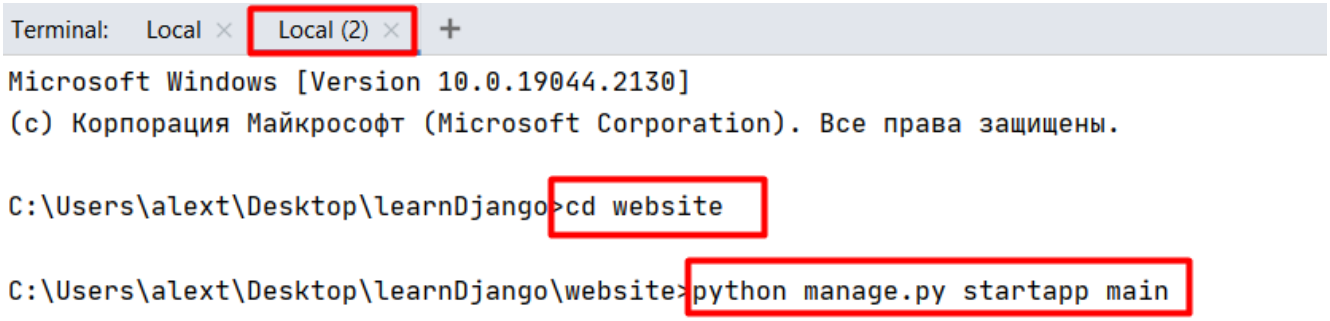


Рисунок 1.26

З'явилася нова папка main.

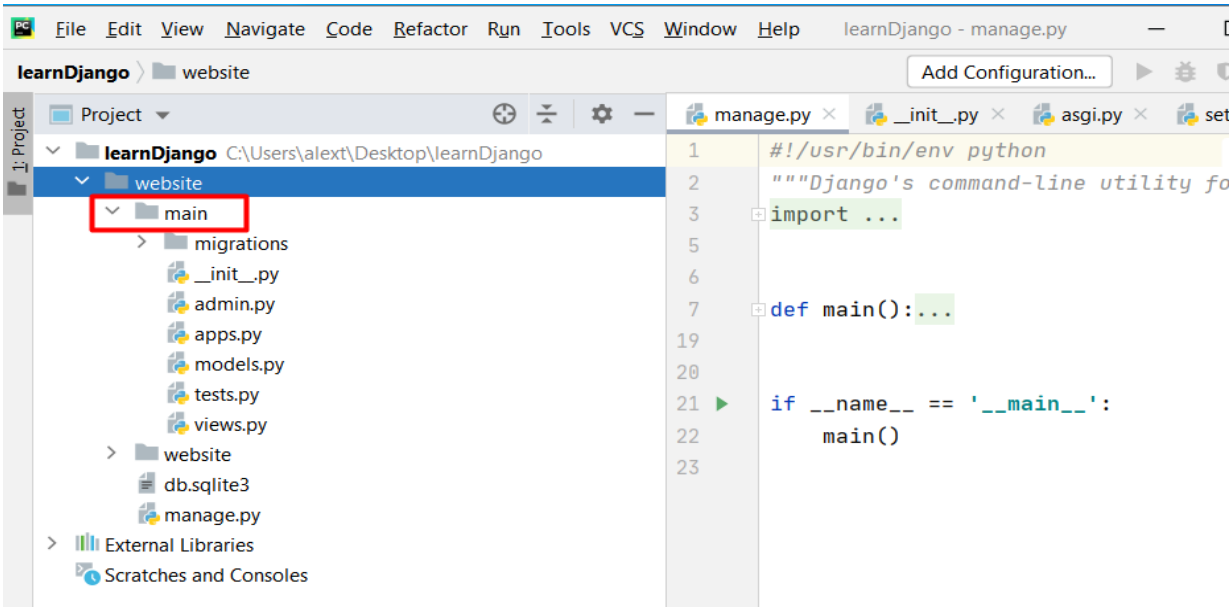


Рисунок 1.27

У додатку буде папка migration.

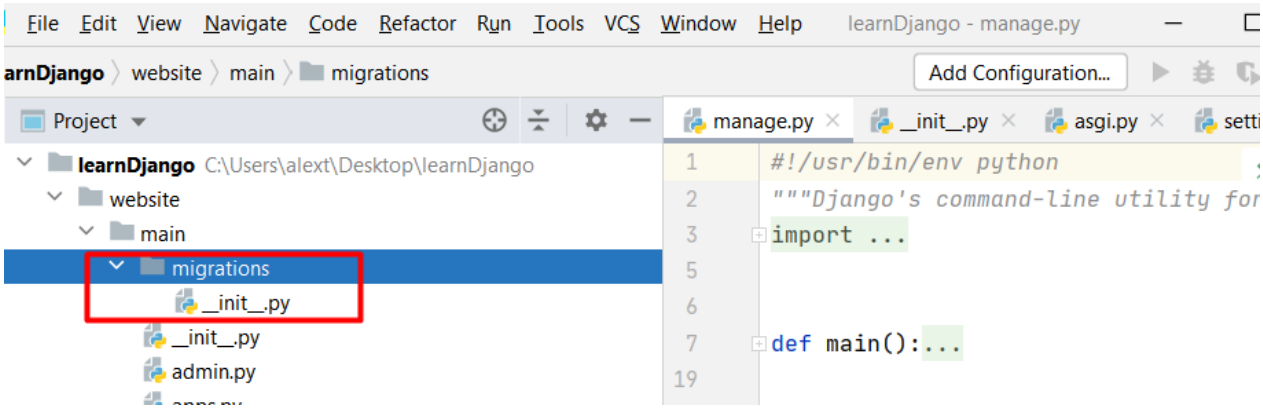


Рисунок 1.28

На основі цієї міграції ми будемо пов'язувати нашу програму з базою даних.

У файлі **models.py** можна створити клас, на основі якого буде створена таблиця в базі даних. У файлі **tests.py** можна створити тести для нашої програми. Файл **views.py** визначає методи, які будуть використовуватись при переході на іншу сторінку.

Зареєструємо новий додаток main. Заходимо у папку **website** у файл **setting.py** та вписуємо назву нової програми main.

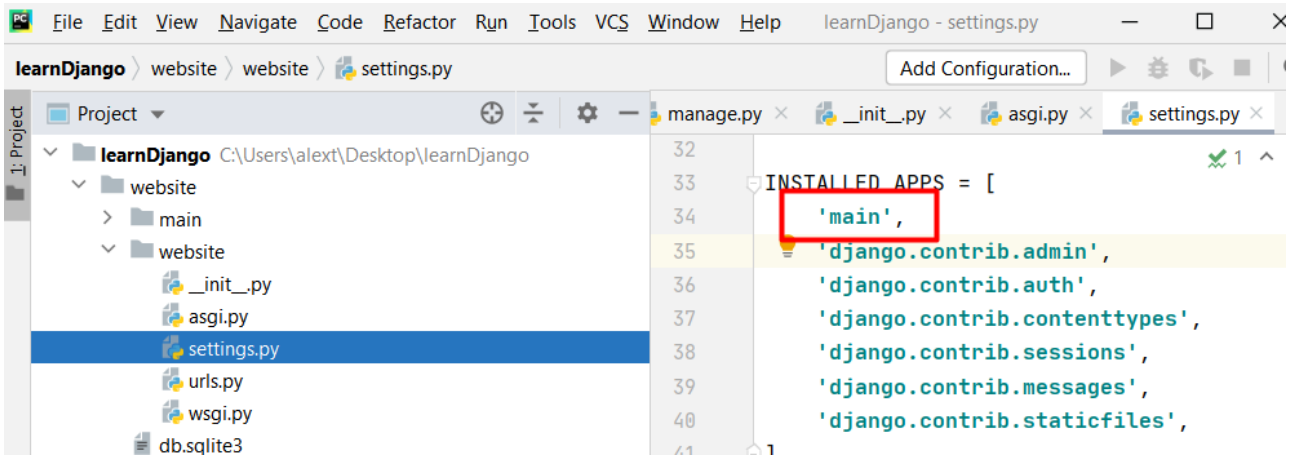


Рисунок 1.29

Навчимося відстежувати адресу url. Для цього зайдемо у папку **website** у файл **urls.py**.

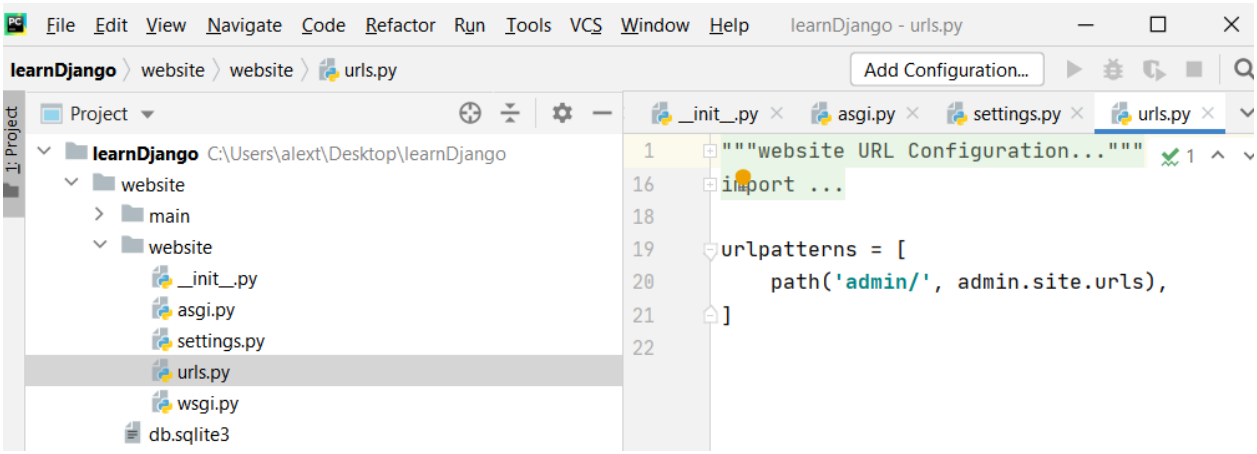


Рисунок 1.30

Тут є одна адреса – адміністратора. Додамо ще перехід на головну сторінку. Для цього пропишемо метод path. йому передамо параметри: url адресу, яку ми відстежуємо. Ми будемо відстежувати головну сторінку, лапки залишаємо порожніми.

Підключаємо додатковий метод include.

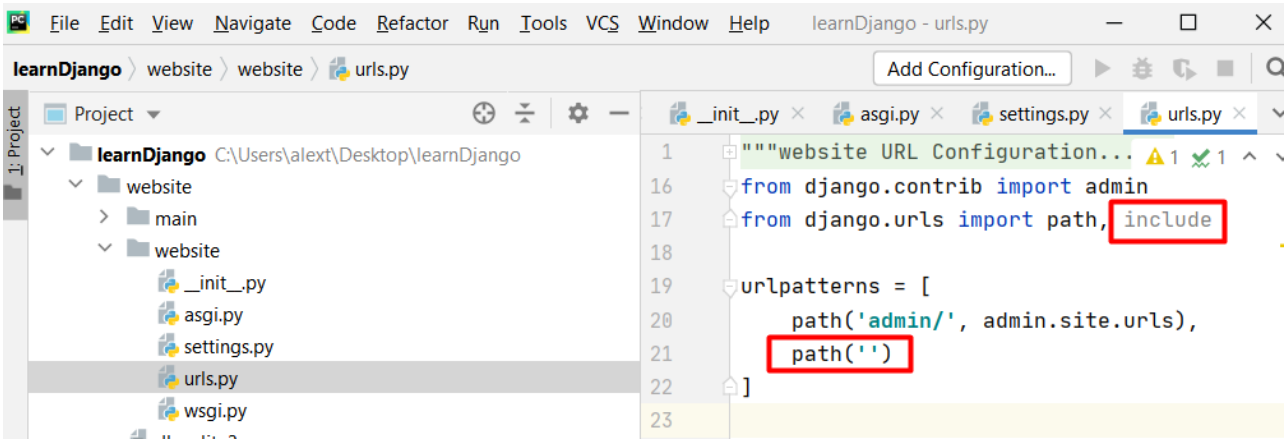


Рисунок 1.31

Потім делегуємо повноваження додатку main. Викликаємо такий же файл із програми main.

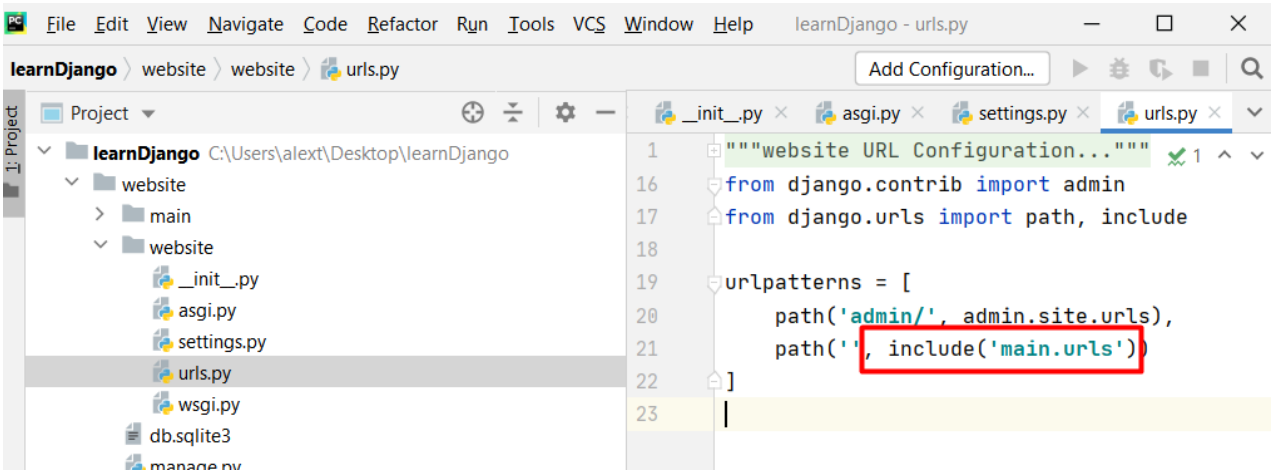


Рисунок 1.32

Наразі цього файлу в папці немає.

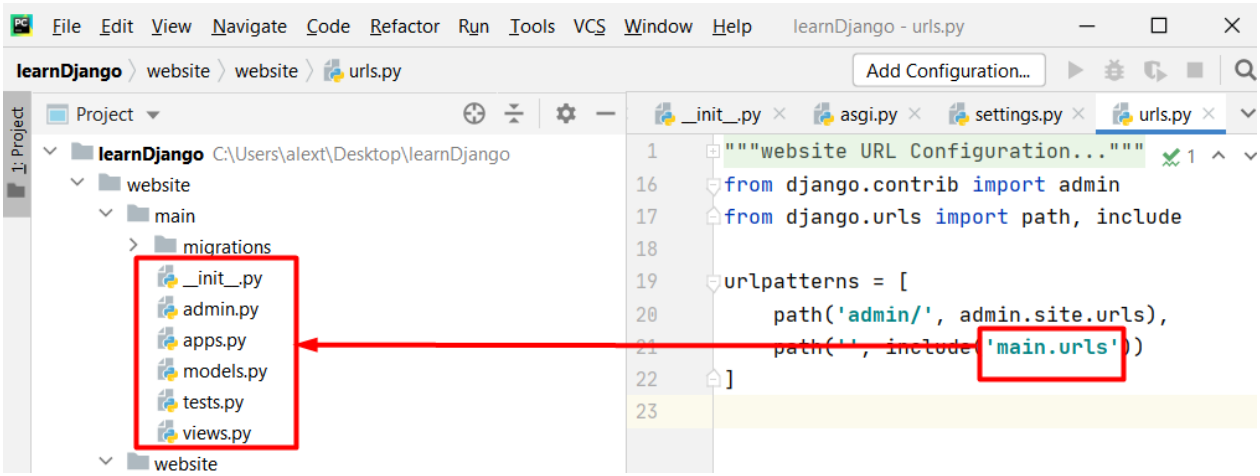


Рисунок 1.33

Створимо файл. Натиснути праву клавішу миші.

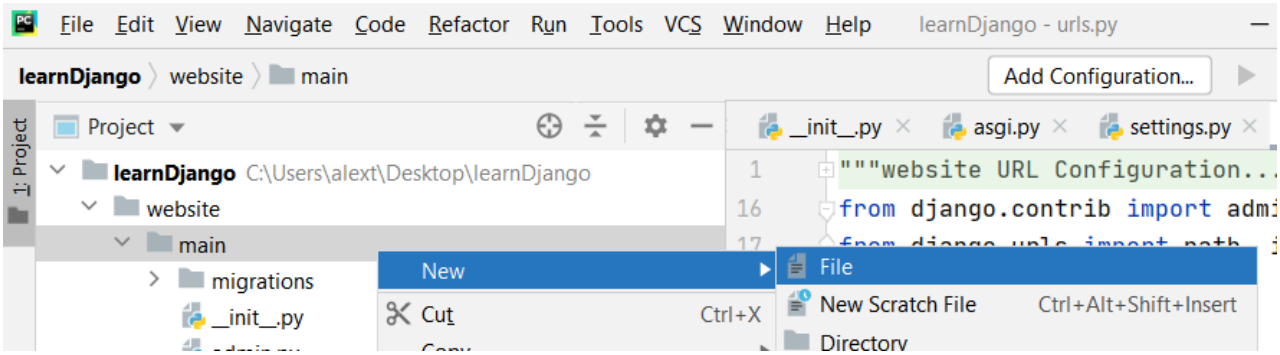


Рисунок 1.34

Вписуємо та натискаємо ENTER.

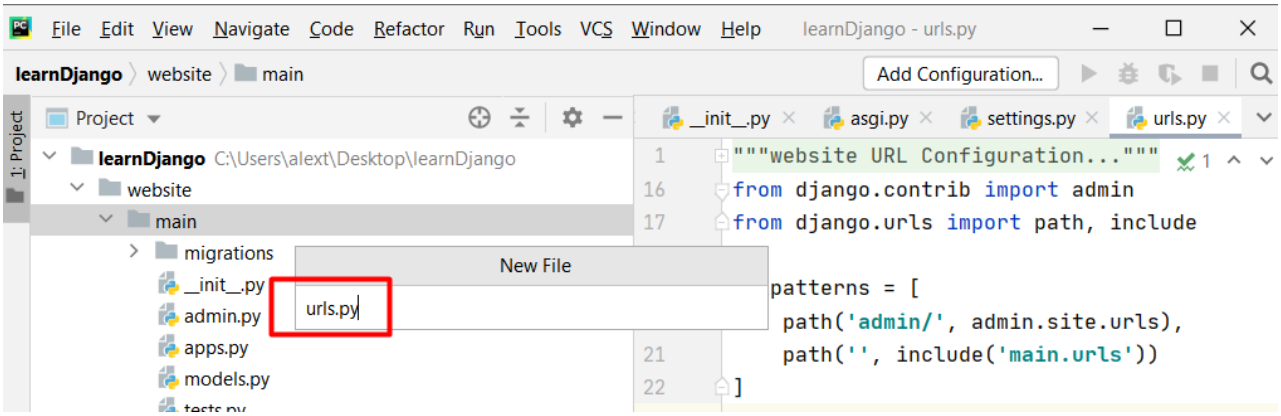


Рисунок 1.35

Файл створено у папці. Нині він порожній.

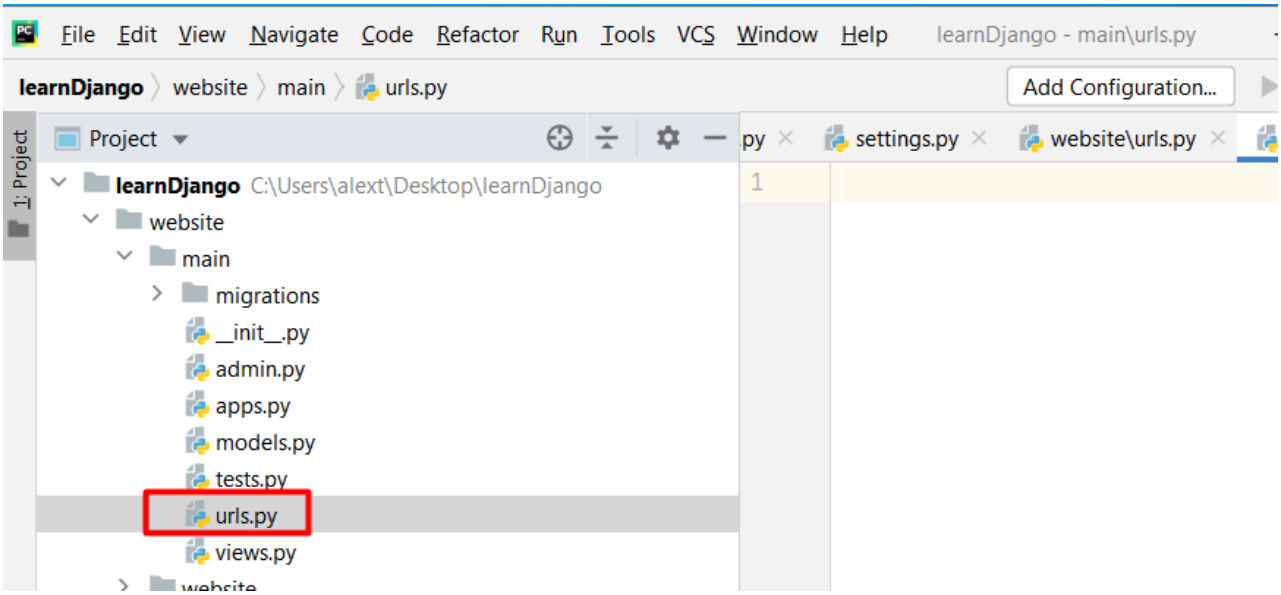


Рисунок 1.36

Перенесемо вміст **urls.py** з папки **website** до **urls.py** папки **main**.

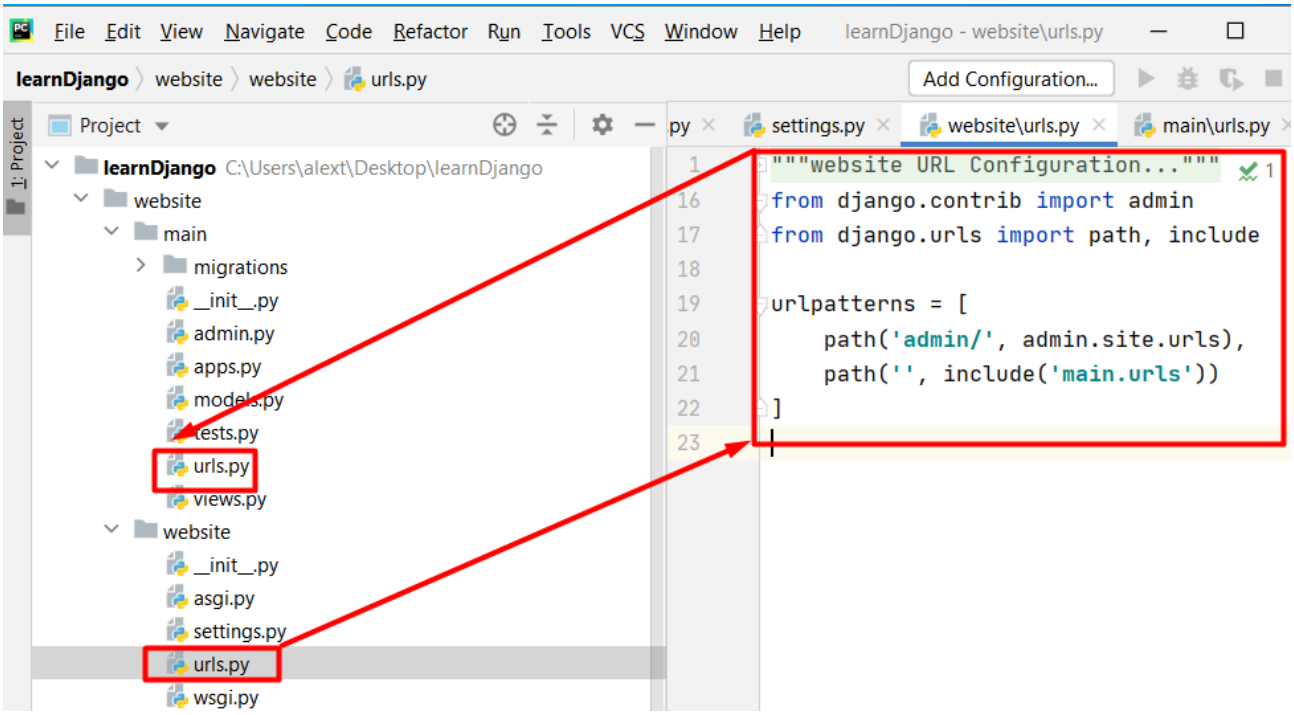


Рисунок 1.37

Отримали.

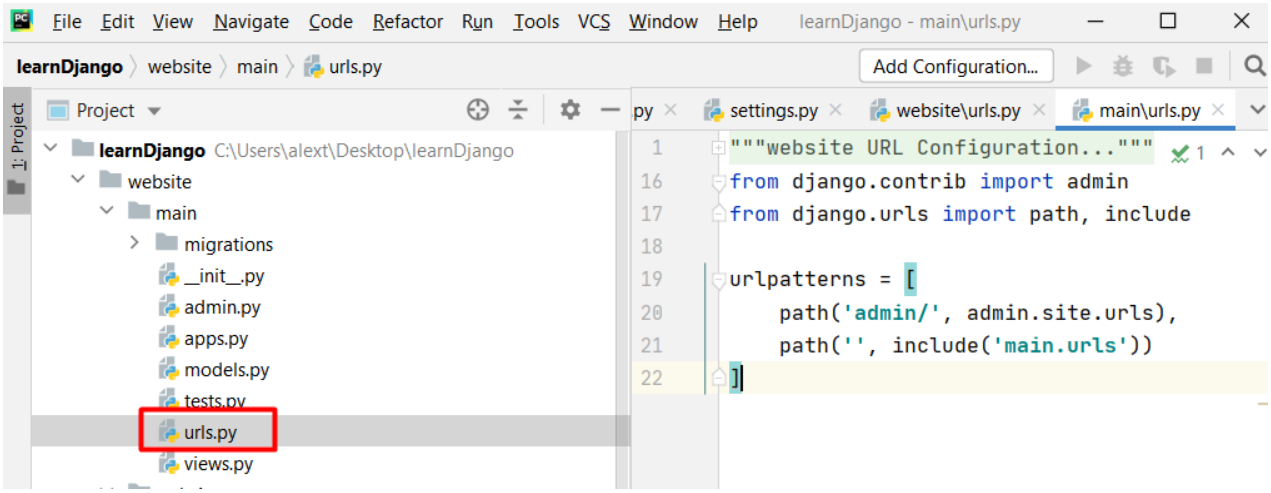


Рисунок 1.38

Прибираємо панель адміністратора та метод input. Очистимо список **urlpatterns**.

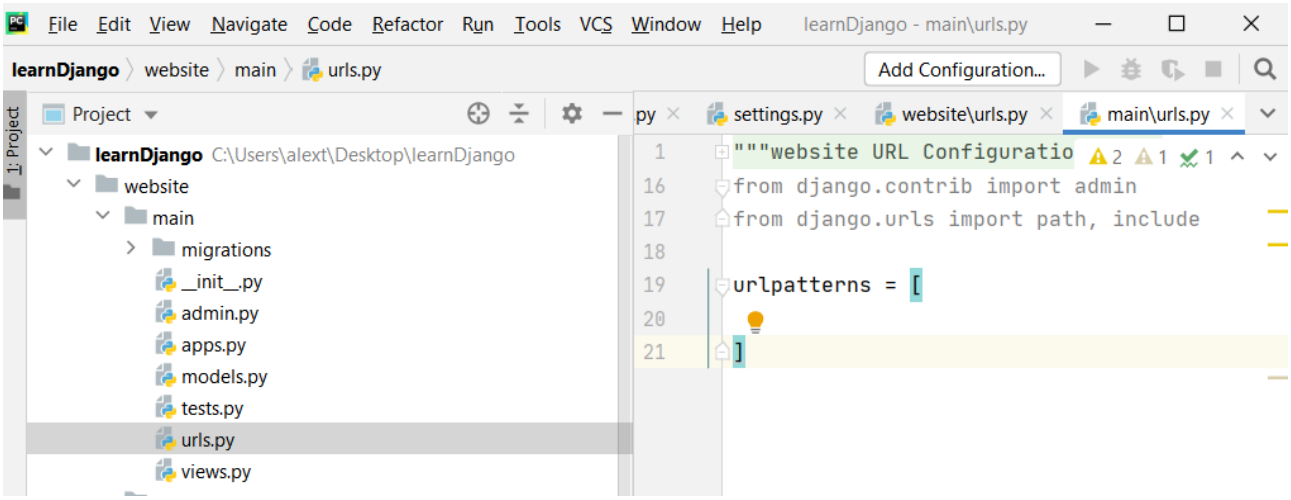


Рисунок 1.39

Логіка роботи програми:

1. Коли користувач зайде на головну сторінку, буде викликаний файл **urls.py** з папки **website**. Тому що у файлі **settings.py** він уже прописаний.

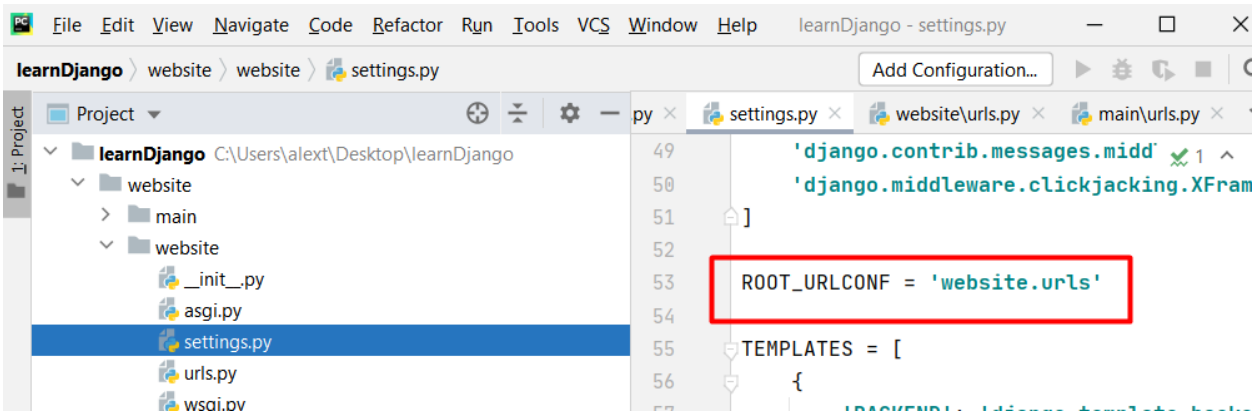


Рисунок 1.40

2. Далі викликається такий самий файл із програми **main**.

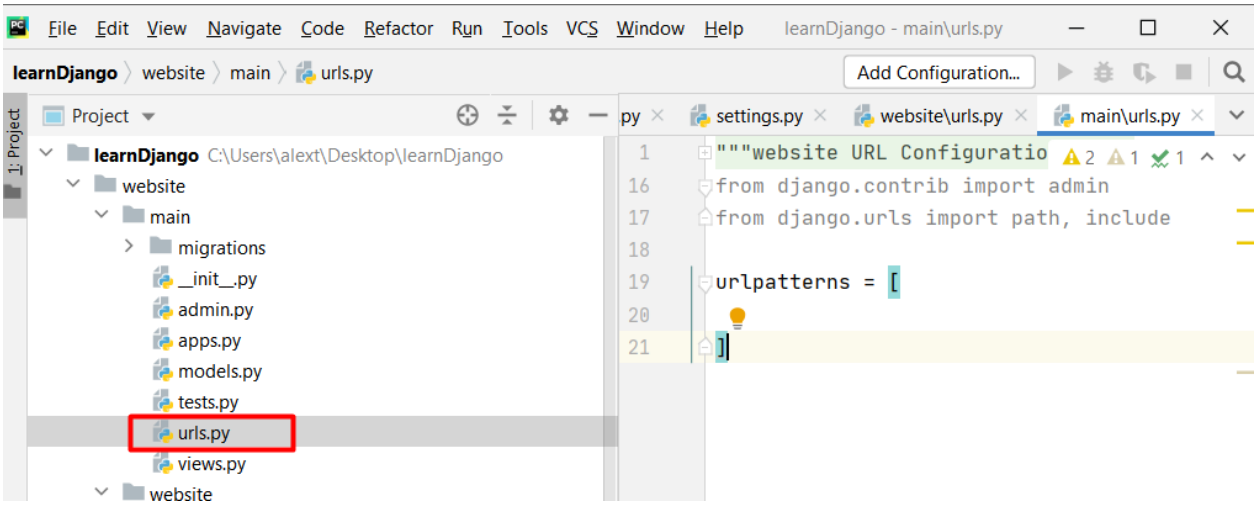


Рисунок 1.41

Всередині цієї програми треба відстежувати адреси. Треба викликати певний метод із файлу. Треба імпортувати з цієї папки де знаходиться цей файл і звернутися до того методу, який ми зараз напишемо всередині цього файлу **views.py**.

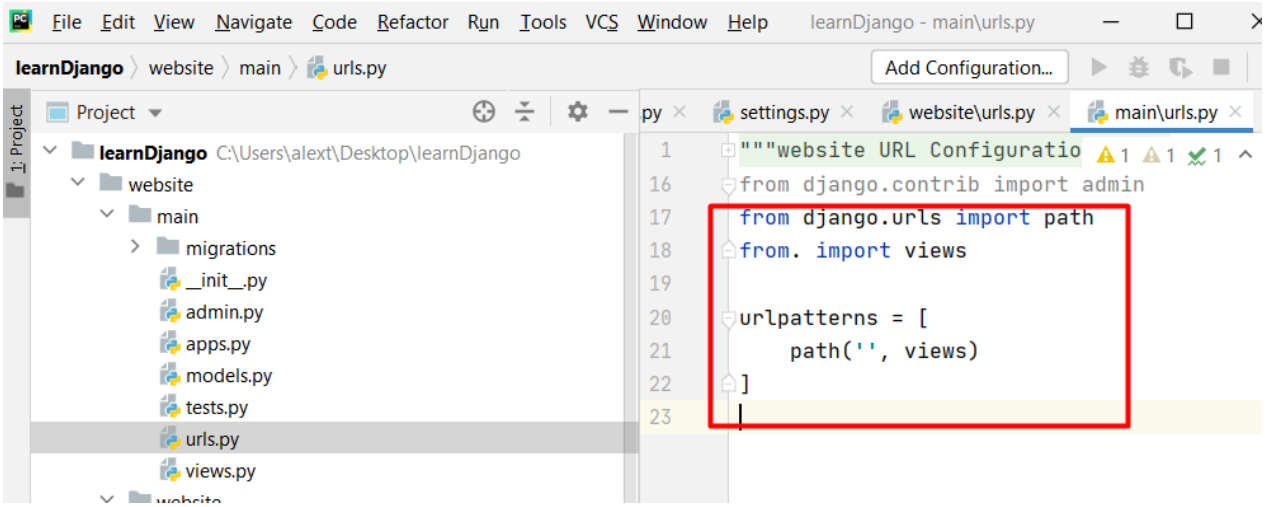


Рисунок 1.42

3. Відкриємо файл **views.py** та впишемо метод index. Поки він буде порожній.

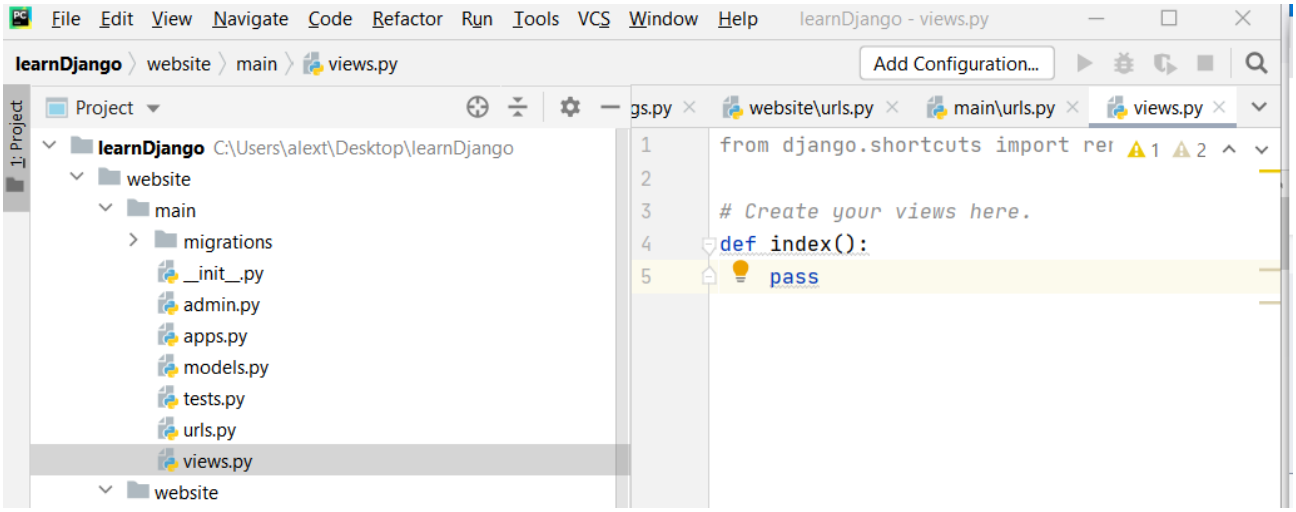


Рисунок 1.43

4. Дописуємо у **urls.py** метод index у якому будемо звертатися у файлі **views.py** .

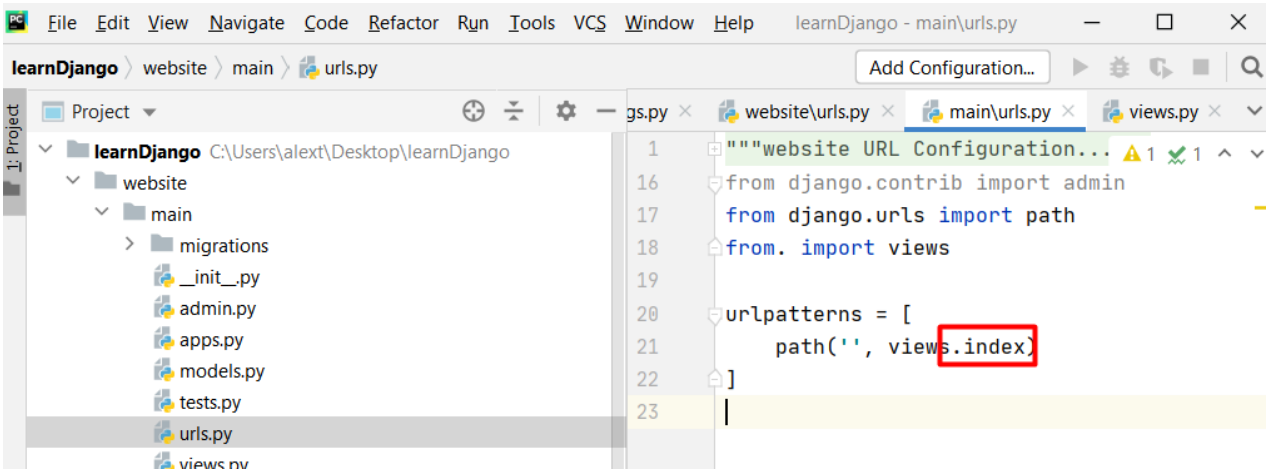


Рисунок 1.44

5. Напишемо звернення до сторінки з urls адресою about і викликатимемо метод about.

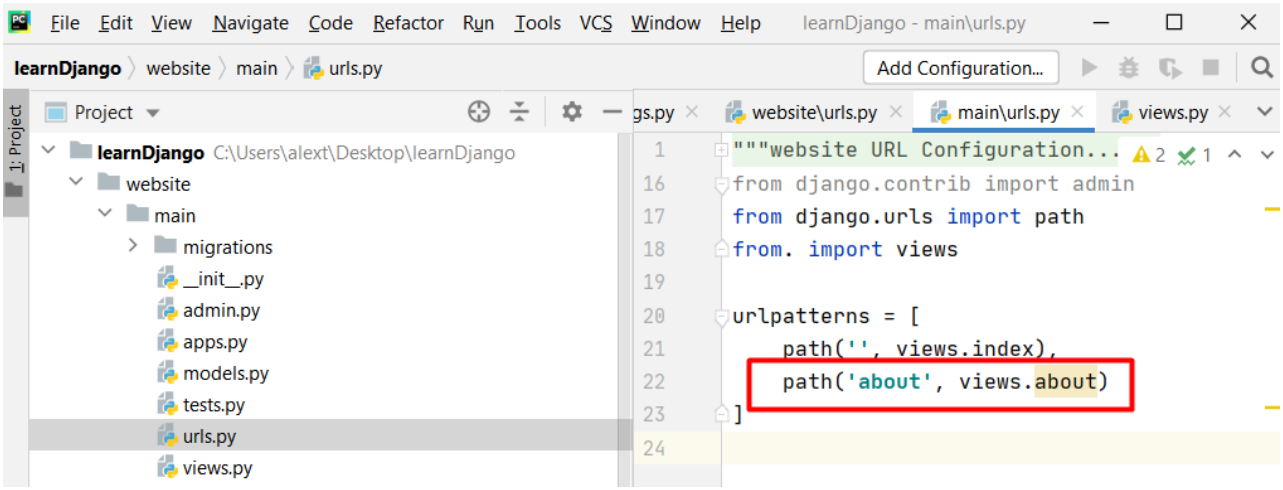


Рисунок 1.45

6. Відкриємо файл **views.py** і змінимо вміст.

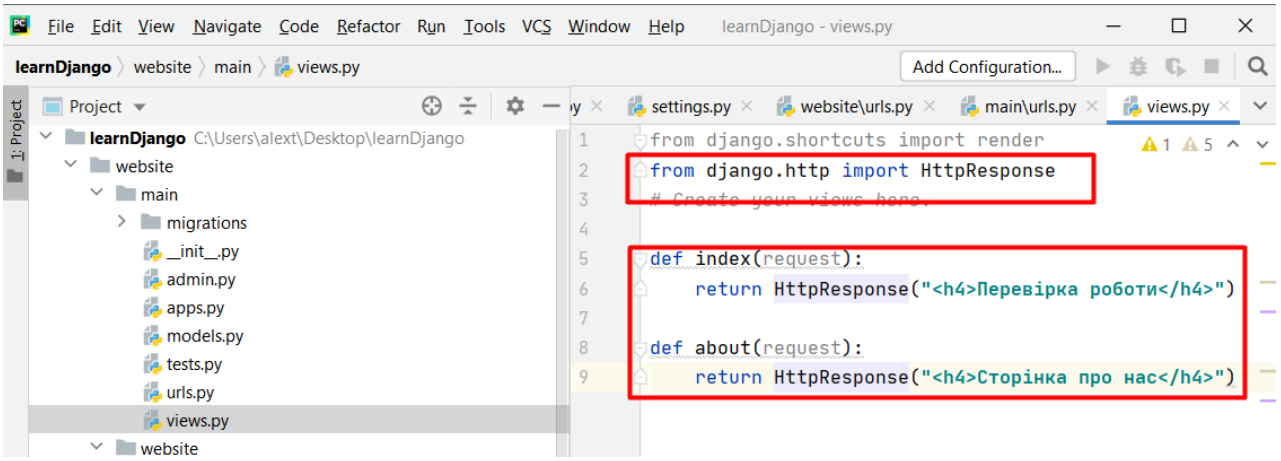


Рисунок 1.46

7. За допомогою клавіш Ctrl+S збережемо файл. Запустимо сервер і на головній сторінці отримаємо напис.

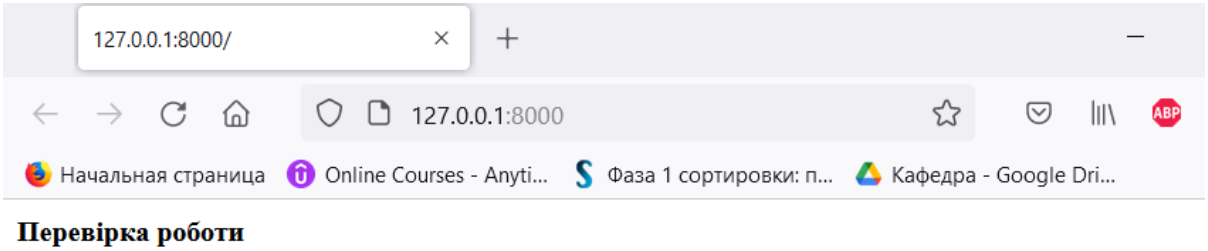


Рисунок 1.47

8. Перехід на наступну сторінку.

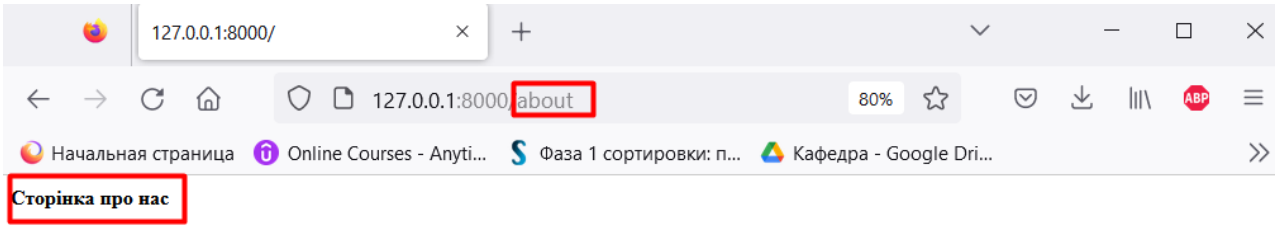


Рисунок 1.48

За допомогою класу HttpResponse не зручно виводити багато інформації. Натомість будемо використовувати імпортований метод render. Так ми можемо вивести не маленький шматочок коду, а звернутися до потрібного HTML шаблону.

У якості першого параметру прописуємо request і у якості другого параметру прописуємо назву HTML шаблону, який буде показаний користувачеві.

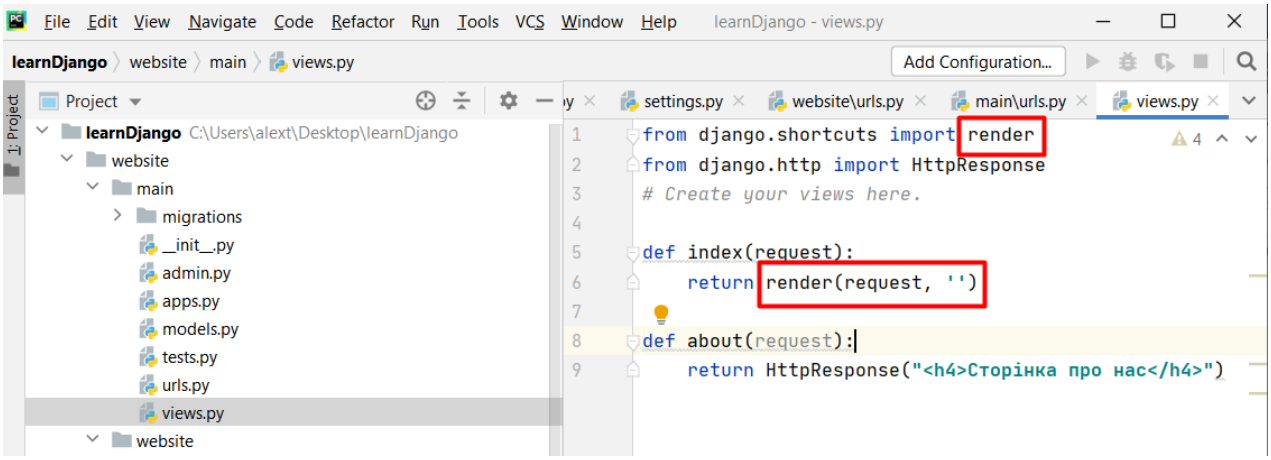


Рисунок 1.49

9. Усі HTML шаблони повинні перебувати в папці **main**. Тут ми створюємо нову папку **templates**.

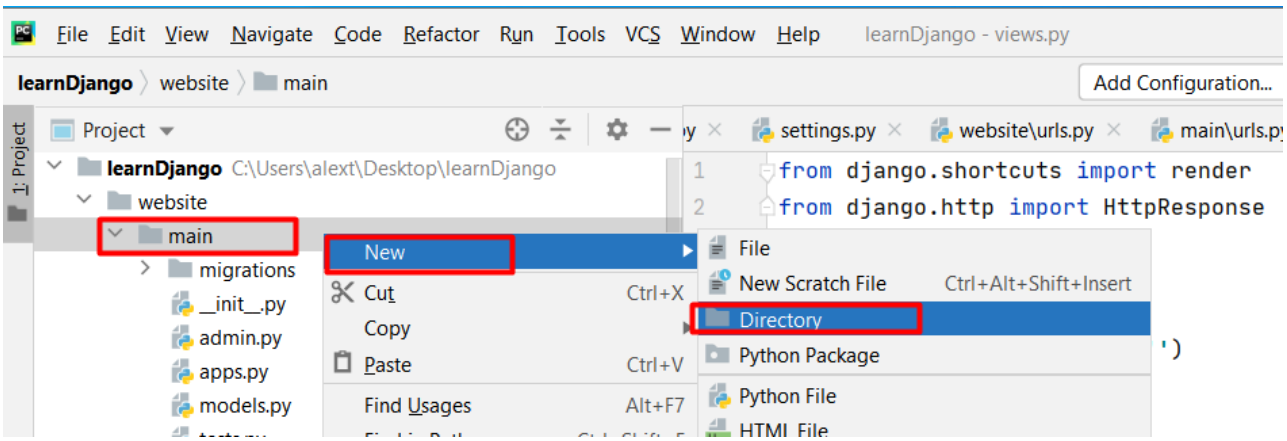


Рисунок 1.50

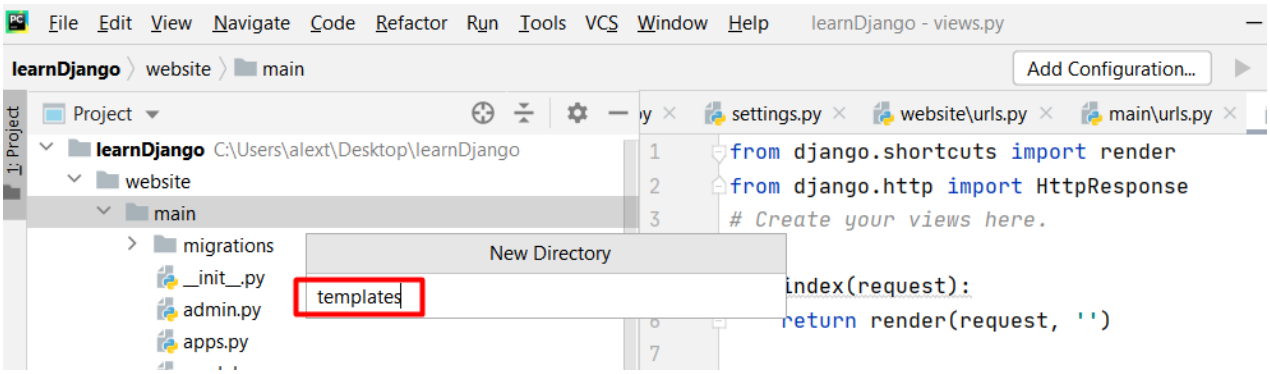


Рисунок 1.51

Вписуємо та натискаємо ENTER. З'явилася папка **templates.**

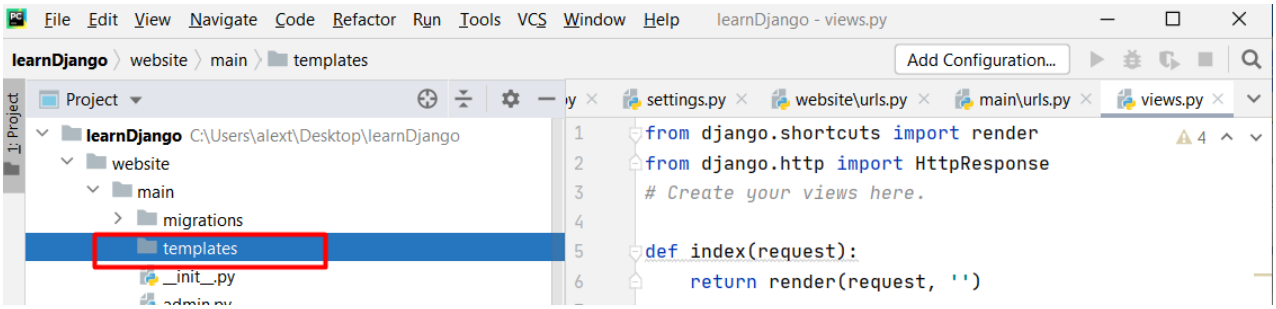


Рисунок 1.52

Всередині цієї папки, згідно з документацією, створимо ще папку, яка буде називатися як додаток main.

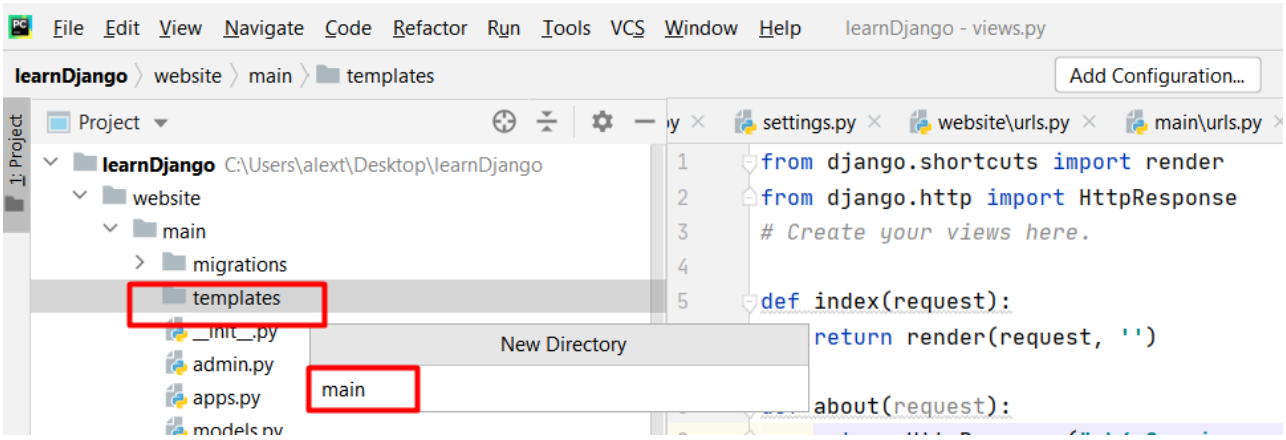


Рисунок 1.53

Так як папок templates всередині проекту може бути багато через безліч додатків, потрібно створити всередині кожного templates папку з назвою програми для коректної роботи.

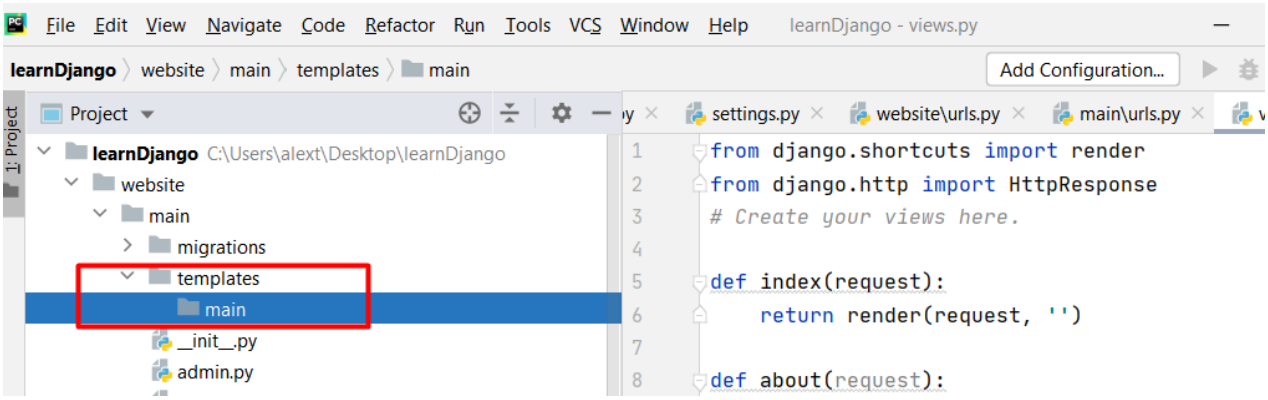


Рисунок 1.54

Тепер усередині цієї папки створимо файл, який міститиме наш шаблон.

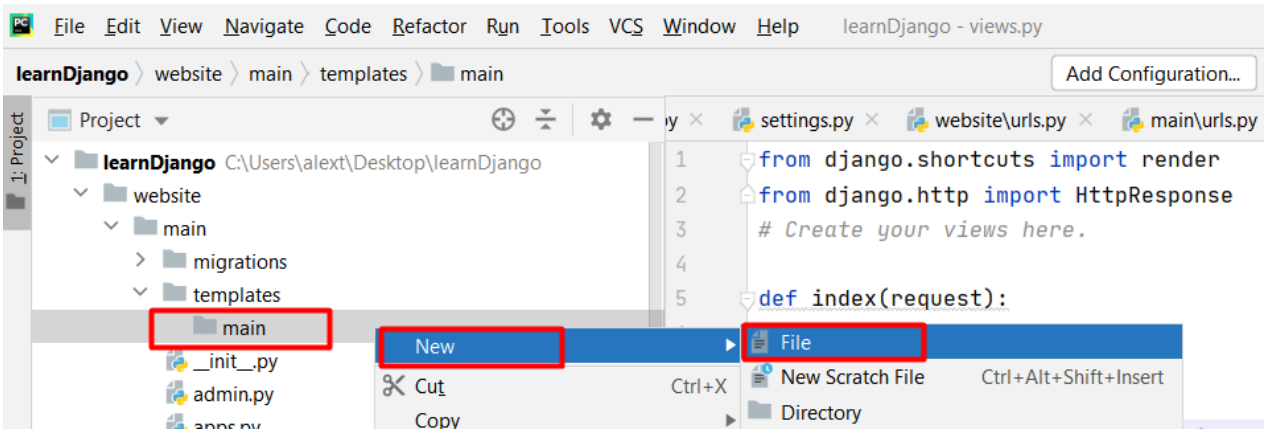


Рисунок 1.55

Файл назвемо **index.HTML.**

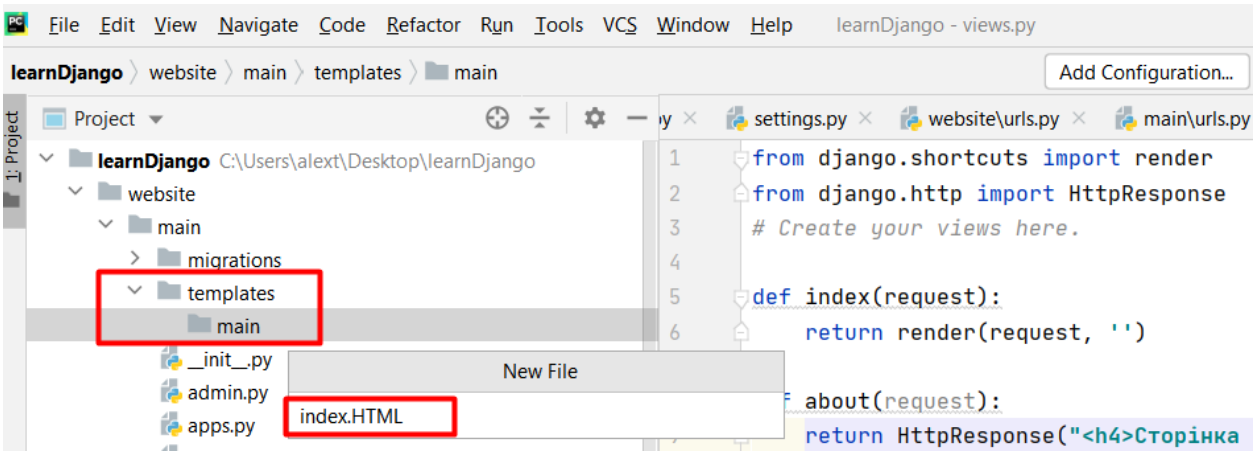


Рисунок 1.56

Нині там нічого немає. Впишемо знак оклику і натиснемо табуляцію. Весь код автоматично вписується.

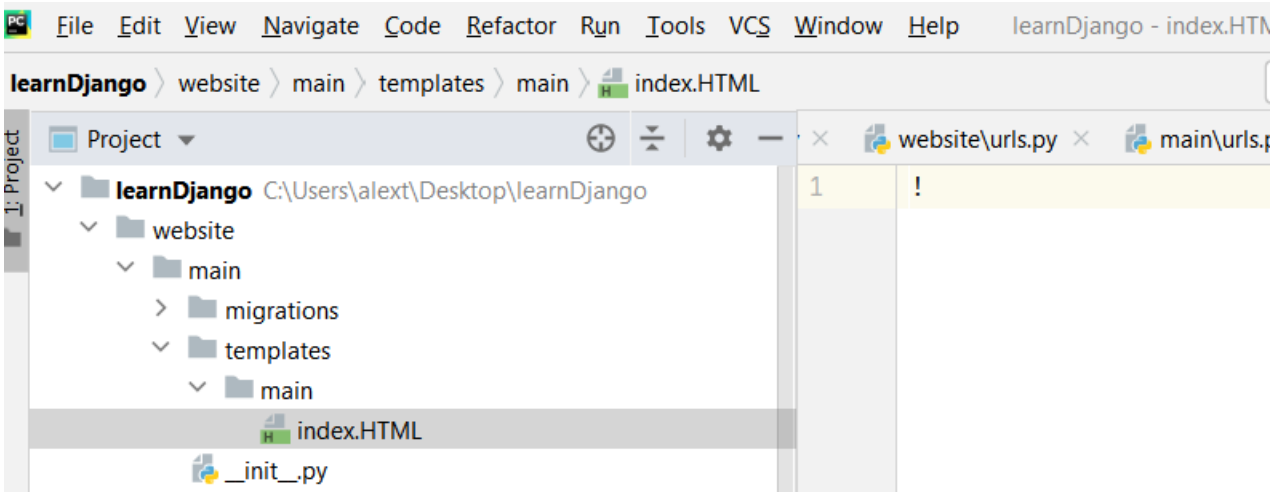


Рисунок 1.57

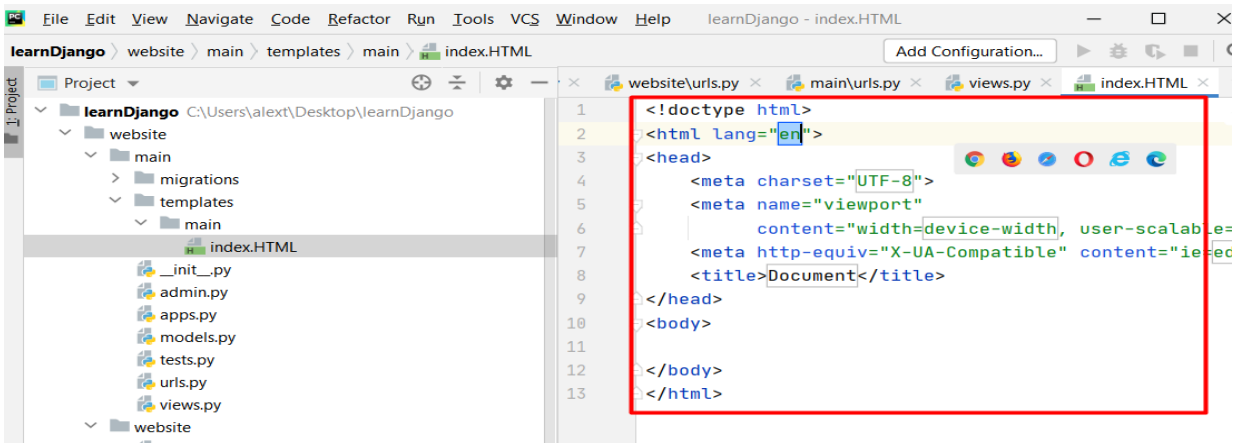


Рисунок 1.58

Впишемо свої дані.

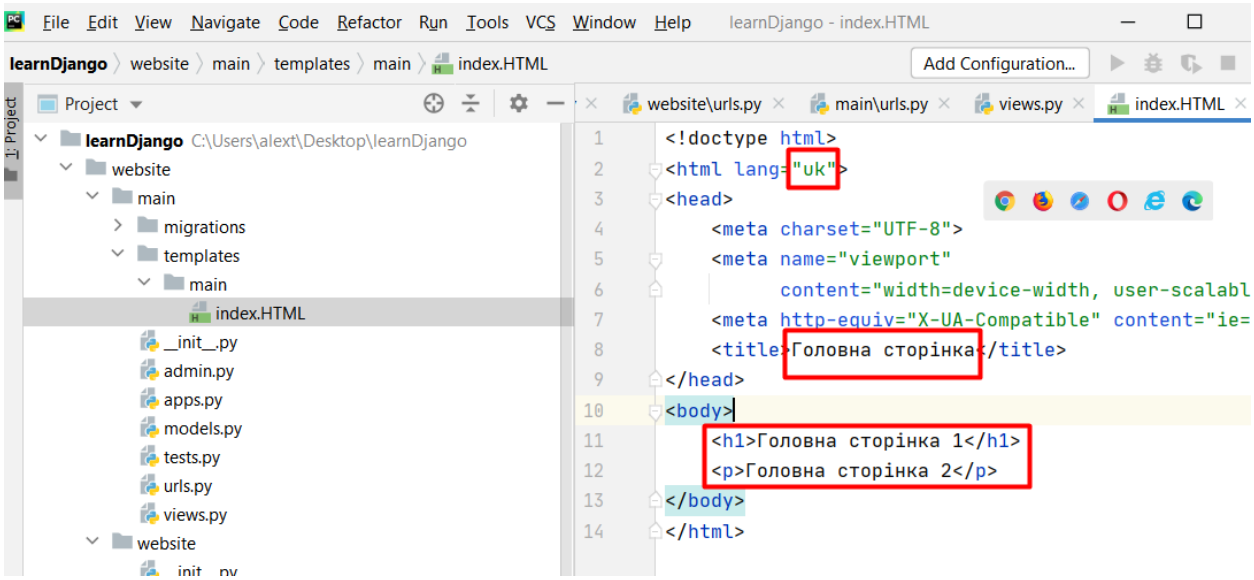


Рисунок 1.59

Повернемося у файл **views.py** та пропишемо звернення до шаблону. Звернемося до папки main та файлу **index.HTML.**

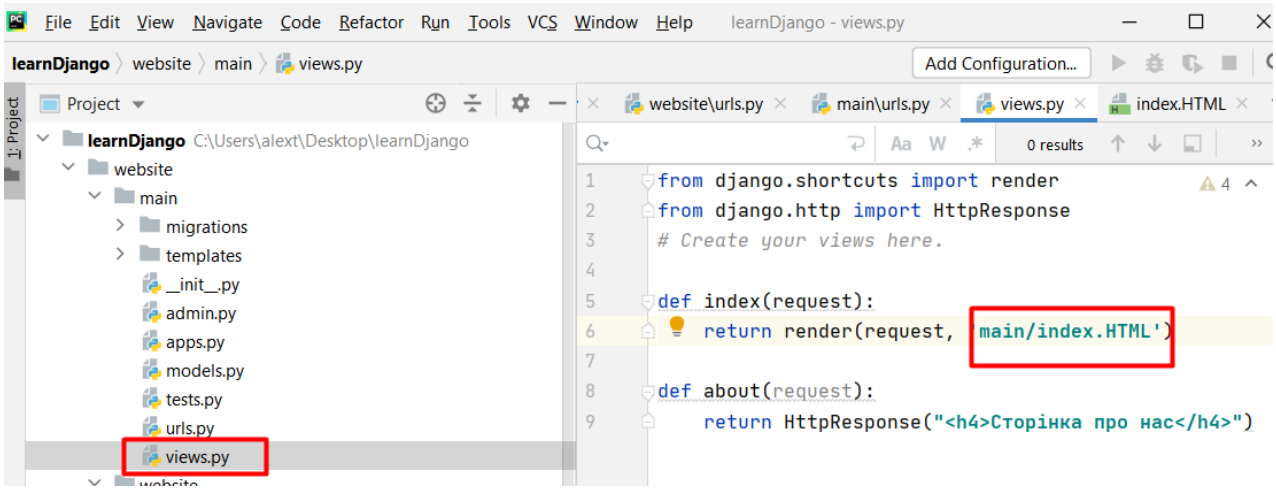


Рисунок 1.60

10. Відкриємо термінал і за допомогою команди runserver відкриємо сторінку в браузері. Видно, що відобразився вміст тегу h1 і параграф. В якості заголовку документу відобразився текст “Головна сторінка”.

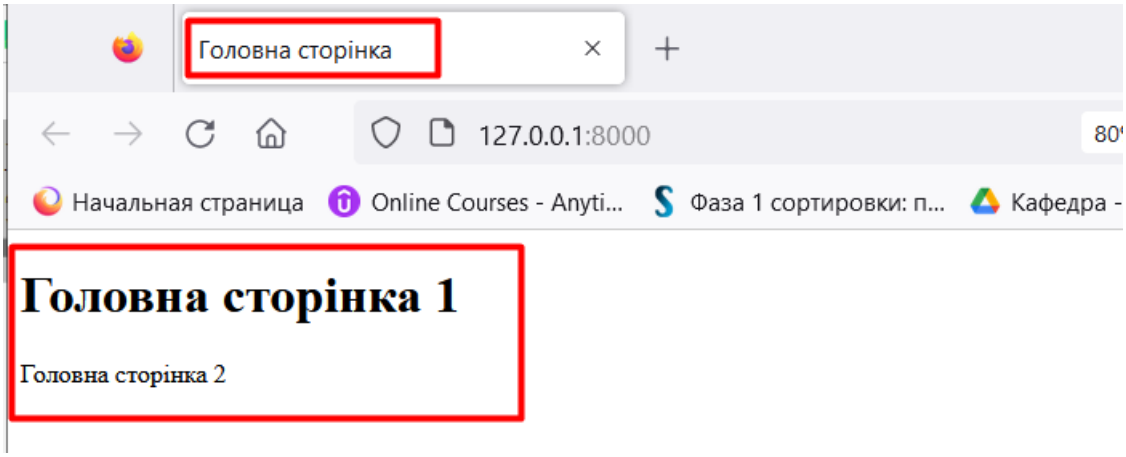


Рисунок 1.61

Додамо те саме і для сторінки about.

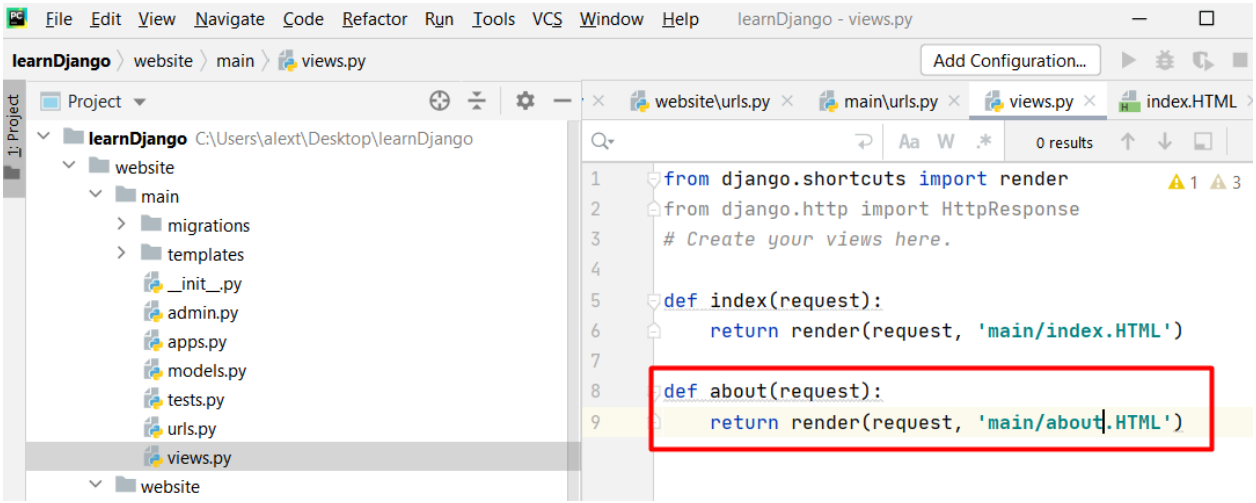


Рисунок 1.62

Такого шаблону поки що немає і ми його зараз зробимо.

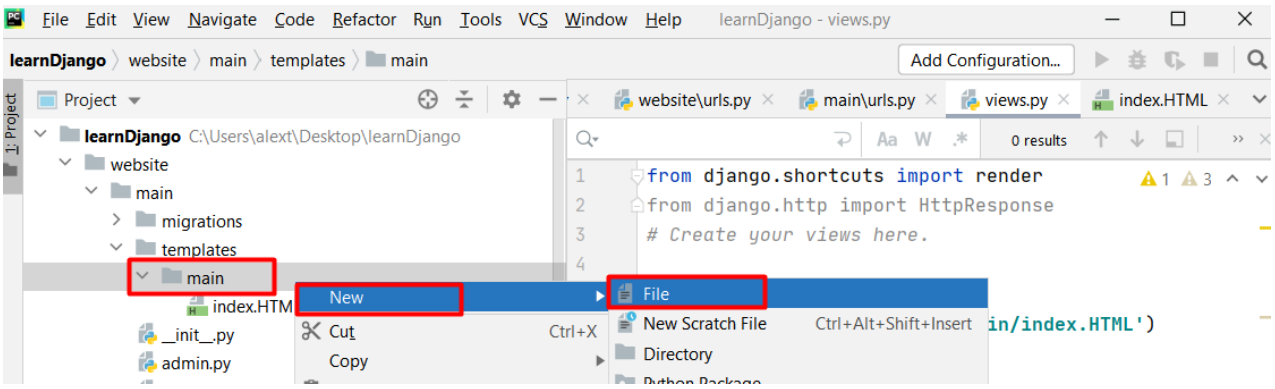


Рисунок 1.63

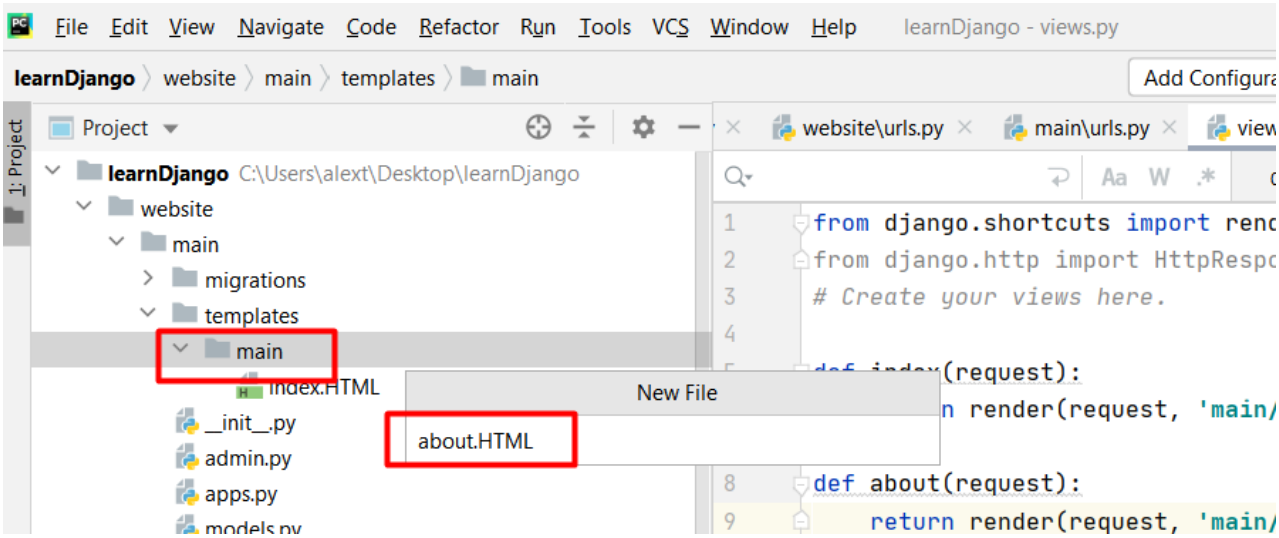


Рисунок 1.64

Впишемо туди потрібну нам інформацію. Напишемо знак оклику і натиснемо табуляцію.

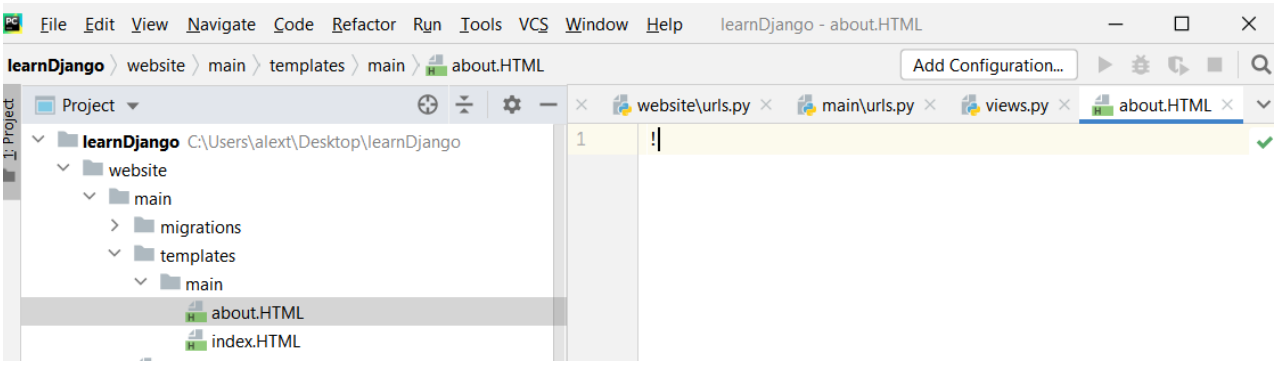


Рисунок 1.65

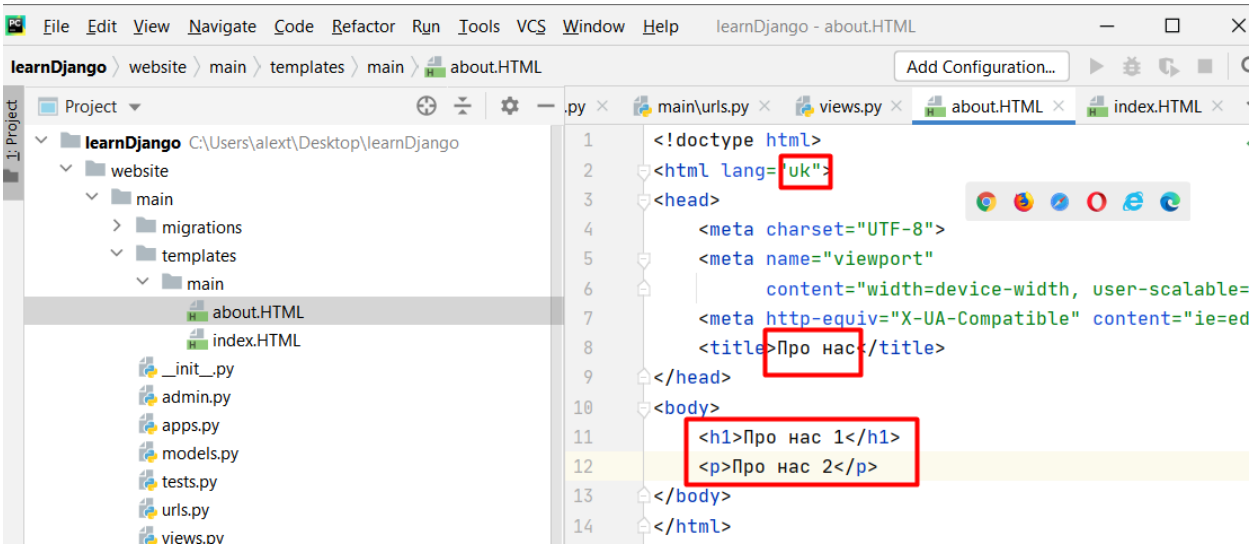


Рисунок 1.66

Отримаємо:

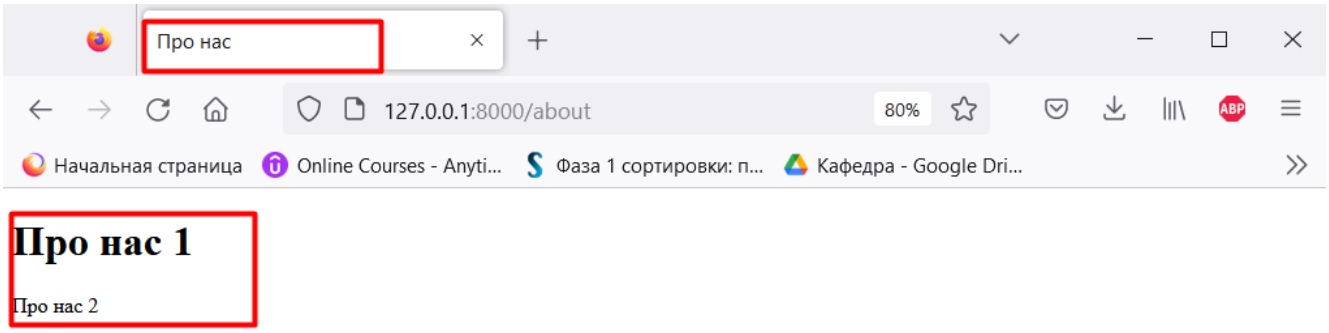


Рисунок 1.67

Тепер ми маємо два файли about і index в якому код практично однаковий.

Створимо окремий файл - загальний шаблон в якому буде знаходитися весь початковий код і там будуть ті секції, які ми видозмінюватимемо від файлу до файлу.

Створимо новий файл.

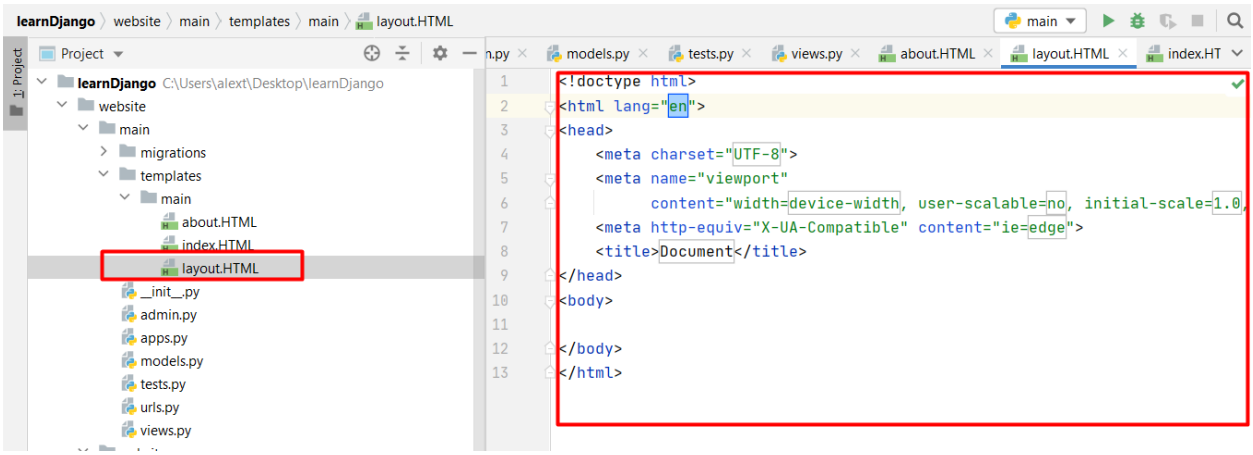


Рисунок 1.68

Перейдемо на українську і тут змінюватиметься лише **title, body**.

У будь-якому джанго-проекті ми можемо використовувати шаблонізатор Jinja. Він надає набір різних можливостей (цикли, змінні…)

Вписуємо ключове слово block і назву content або title, а потім прописуємо конструкцію - закрити блок {% endblock %}.

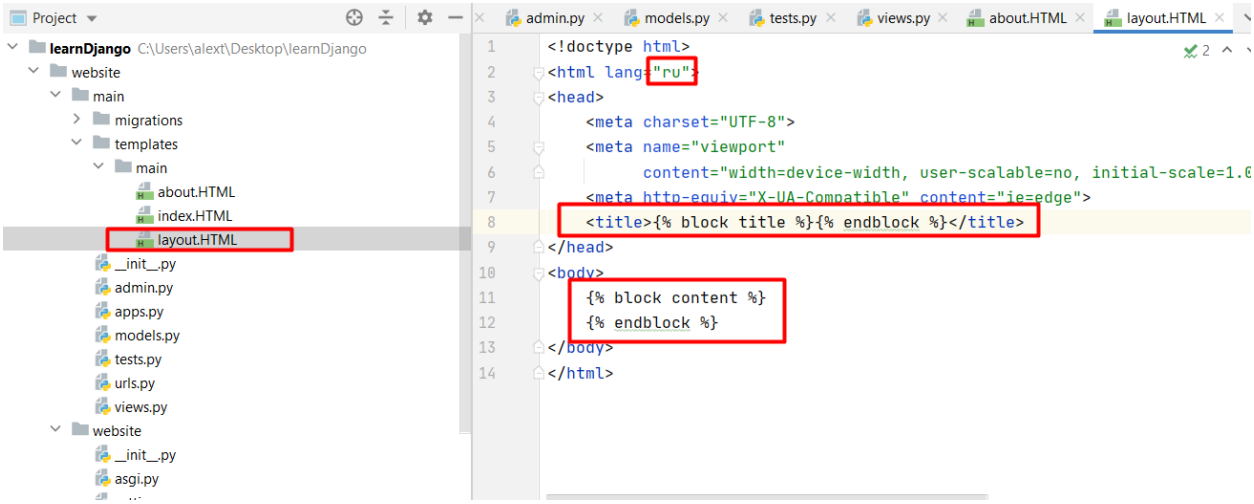


Рисунок 1.69

Увійдемо у файл index.HTML та проведемо там зміни. Очистимо все.

Тепер успадковуємо все від файлу **layout.HTML**. Відкриваємо стандартну структуру {% %} та всередину вписуємо команду extends. (Принцип такий самий як і для команди render) і вписуємо файл, від якого ми збираємося успадковувати – **layout.HTML**, який знаходиться у папці main.

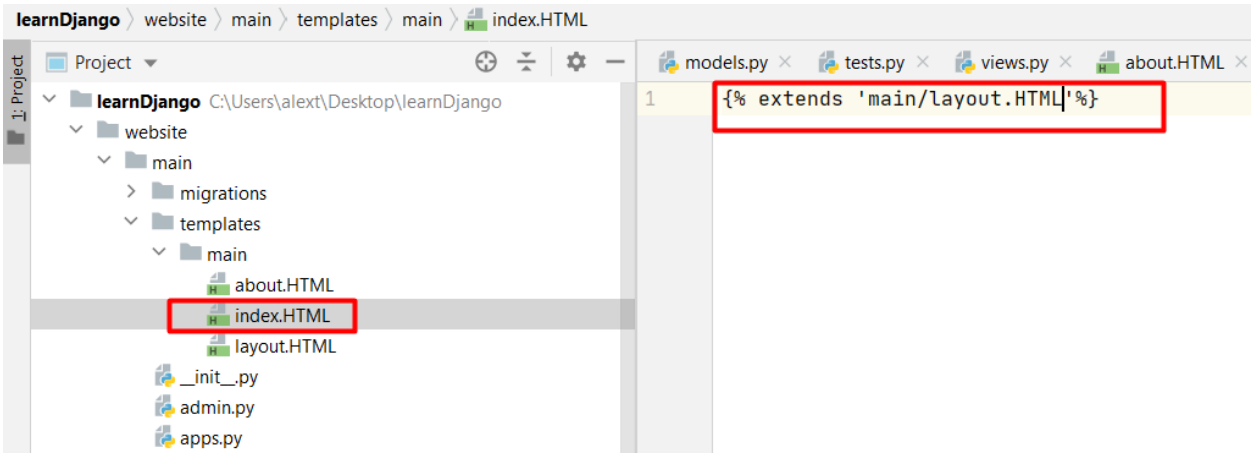


Рисунок 1.70

Тепер помістимо текст всередину блоків. Можна використовувати ключове слово block або section щоб розмістити потрібний код. І не забуваємо закрити блок.

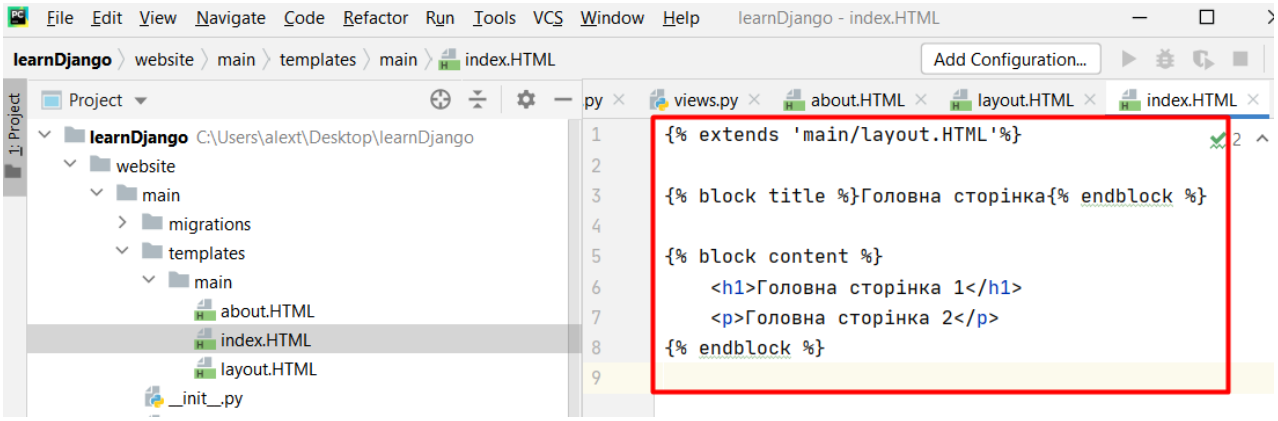


Рисунок 1.71

Теж зробимо і з файлом **about.HTML**

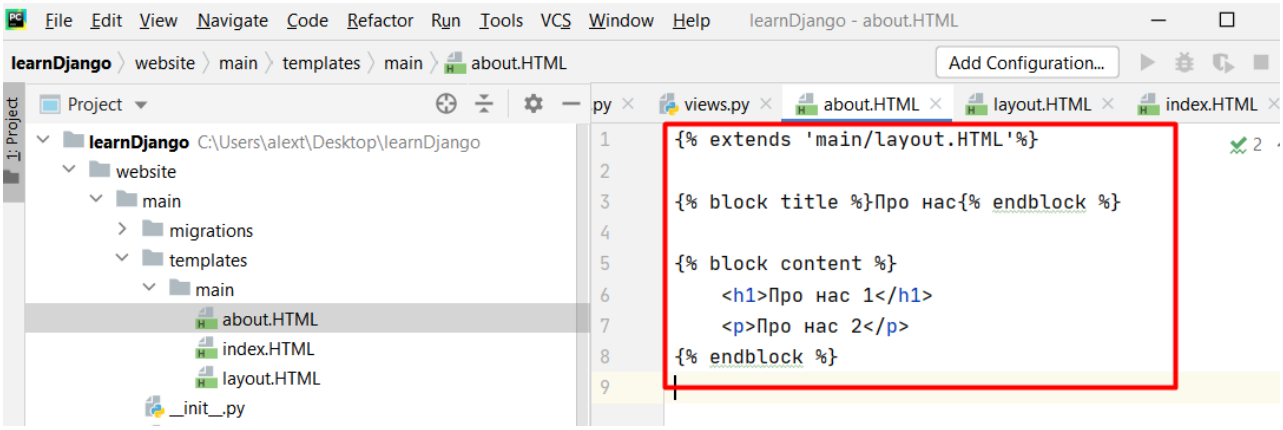


Рисунок 1.72

Перевіряємо роботу.

Спробуємо створити файл, який можна буде вбудувати за допомогою ключового слова include.

Створимо файл test із змістом тексту «Просто перевірка».

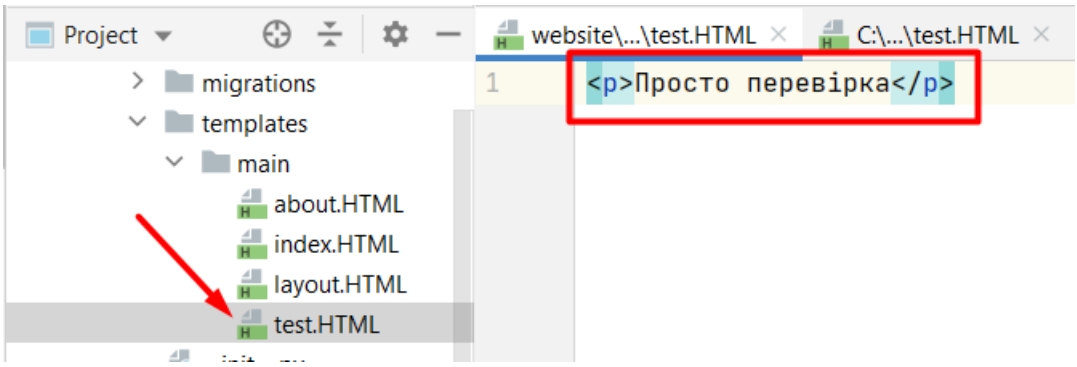


Рисунок 1.73

Тепер в файл **layout.HTML**, який містить шаблонізатор Jinja, впишемо ключове слово **include** для підключення файлів.

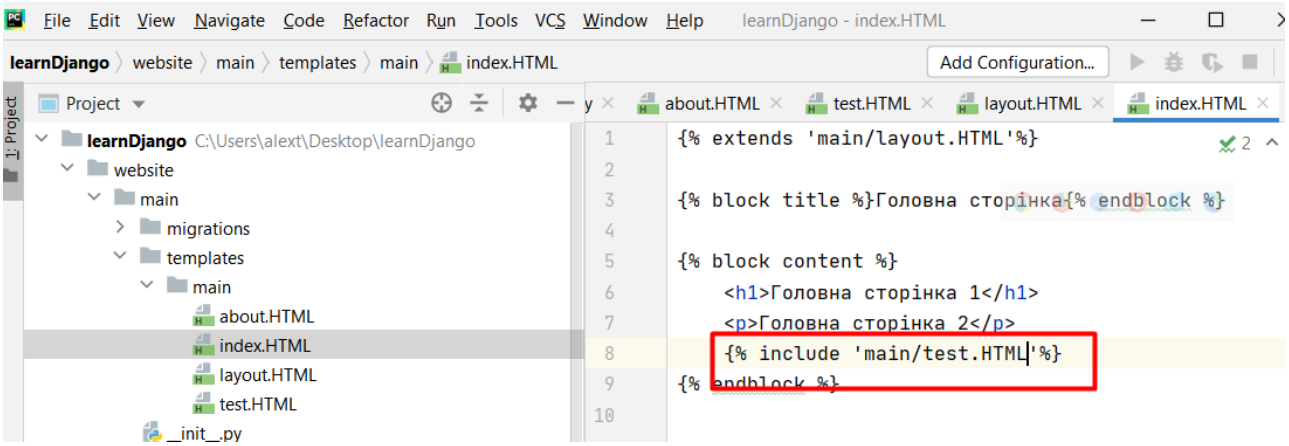


Рисунок 1.74

Відобразиться наша головна сторінка та вміст файлу **test.HTML**.

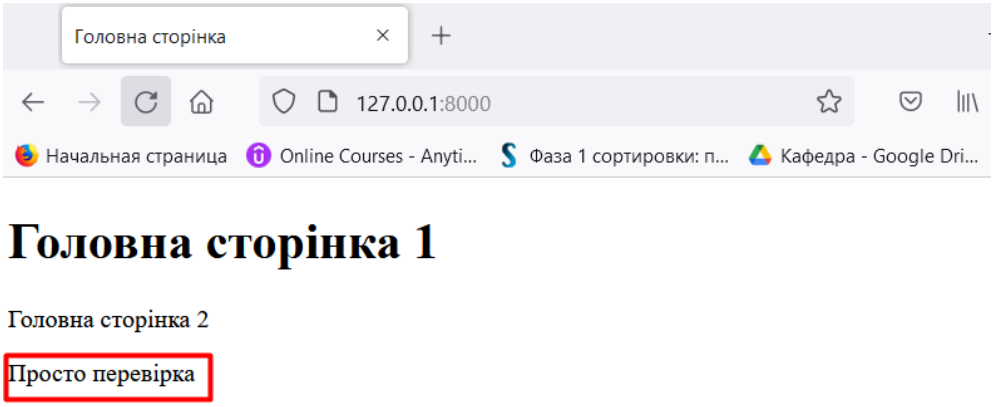


Рисунок 1.75

Це вигідно використовувати, якщо цей файл часто використовується в різних варіантах.

Додамо Bootstrap до нашого сайту. Це безкоштовний набір інструментів з відкритим кодом, призначений для створення вебсайтів та вебдодатків, який містить шаблони CSS та HTML для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу, а також додаткові розширення JavaScript. Він спрощує розробку динамічних вебсайтів і вебдодатків. Bootstrap — це клієнтський фреймворк, тобто інтерфейс для користувача, на відміну від коду серверної сторони, який знаходиться на сервері. CSS фреймворк що забезпечує нас

красивими стилями. Переходимо за посиланням на офіційний сайт **https://www.bootstrapcdn.com.**

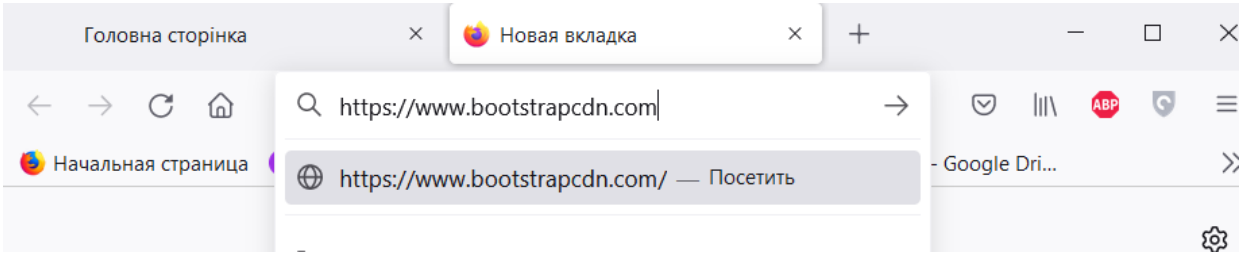


Рисунок 1.76

Копіюємо посилання:

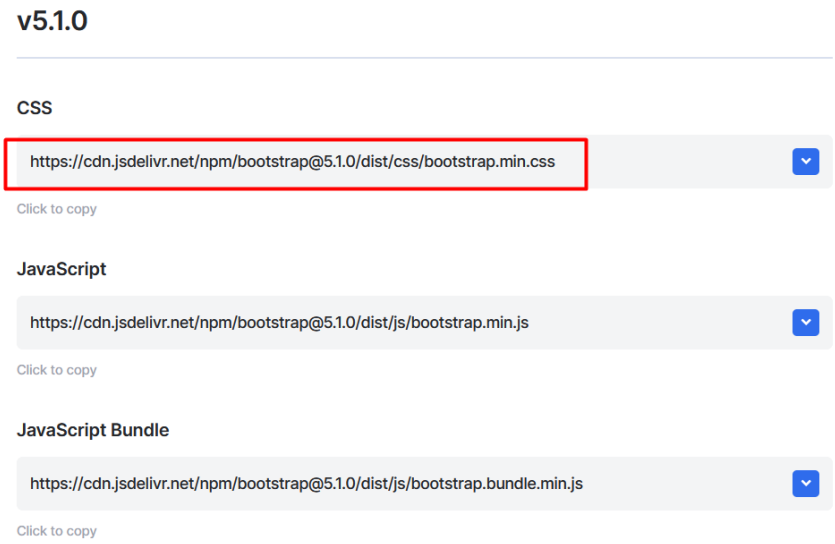


Рисунок 1.77

Вставимо посилання у файл.

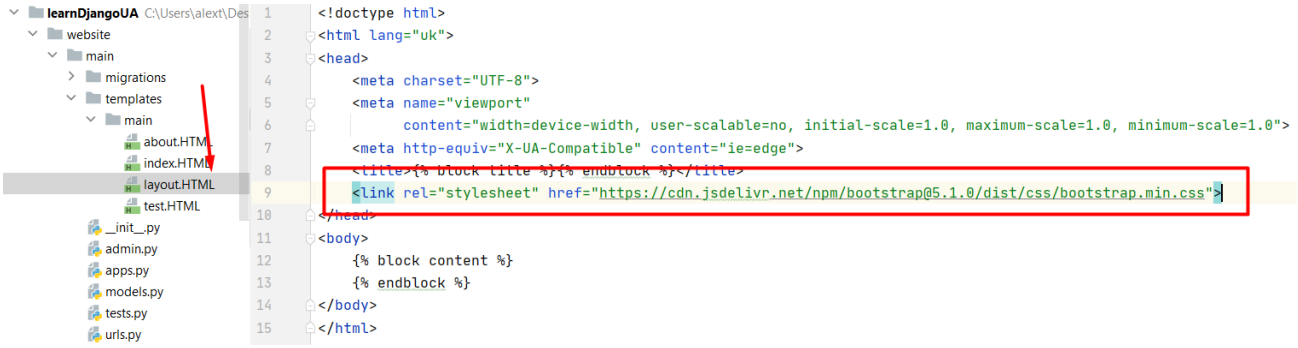


Рисунок 1.78

Заходимо на сайт і бачимо, що все працює правильно.

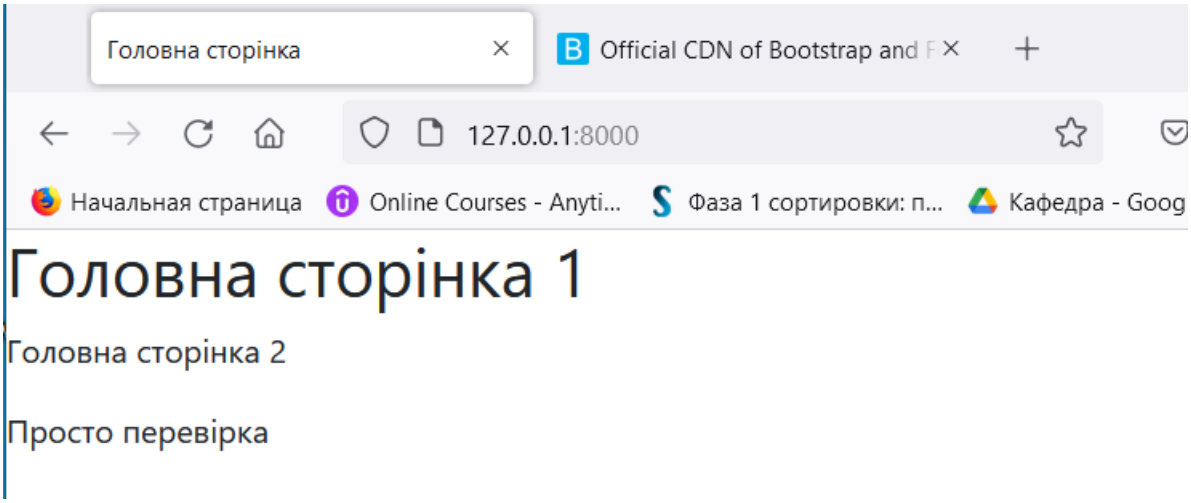


Рисунок 1.79

Створимо новий файл CSS і підключимо його до основного шаблону.

За правилами, всі шаблони повинні зберігатися в папці **templates,** а всі статичні файли потрібно зберігати папці **static**.

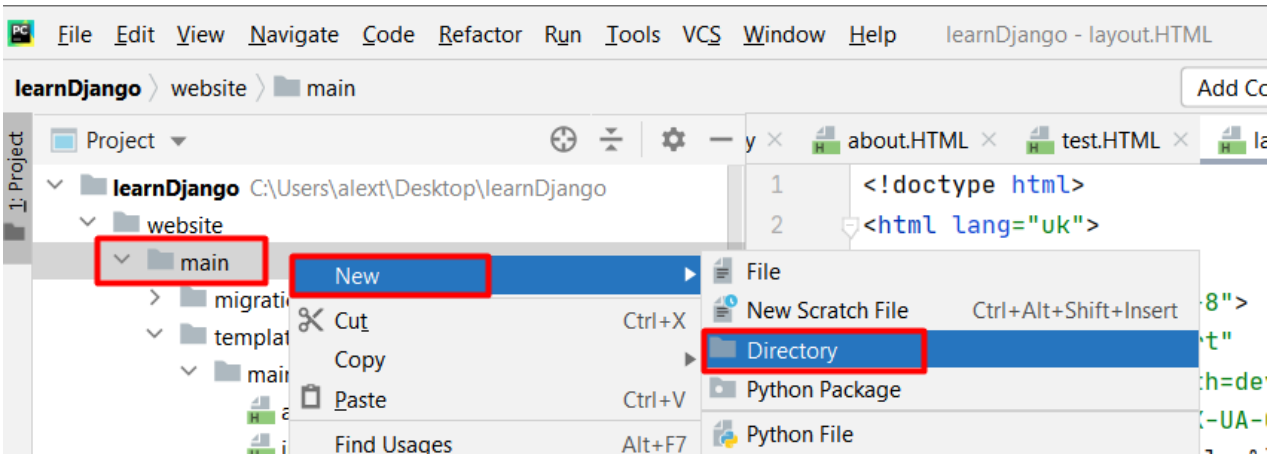


Рисунок 1.80

І всередині неї створимо папку, яка називається як додаток main.

Також створимо папки зі зберігання.

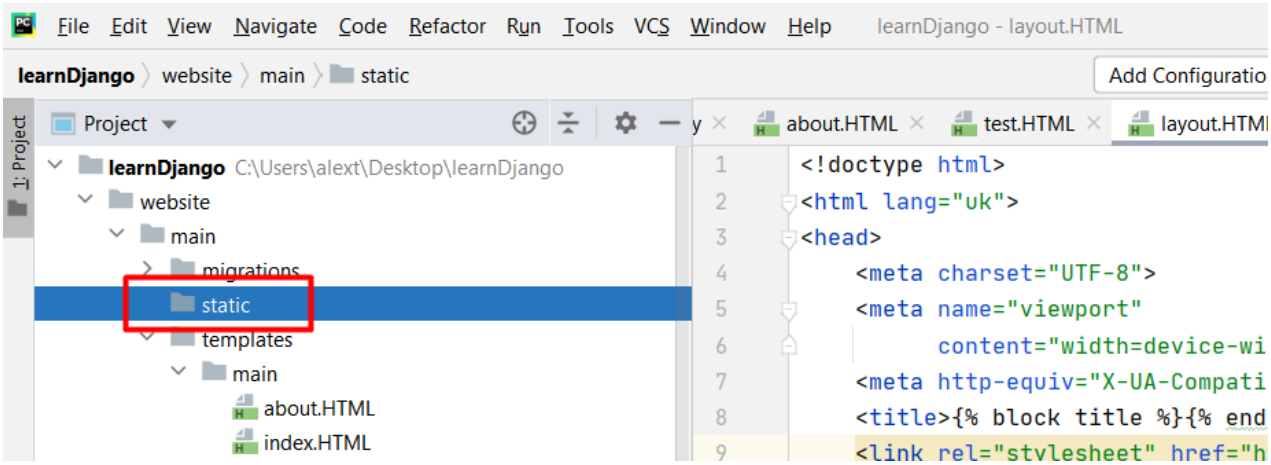


Рисунок 1.81

Створимо файл **main.css** у папці **main, css**.

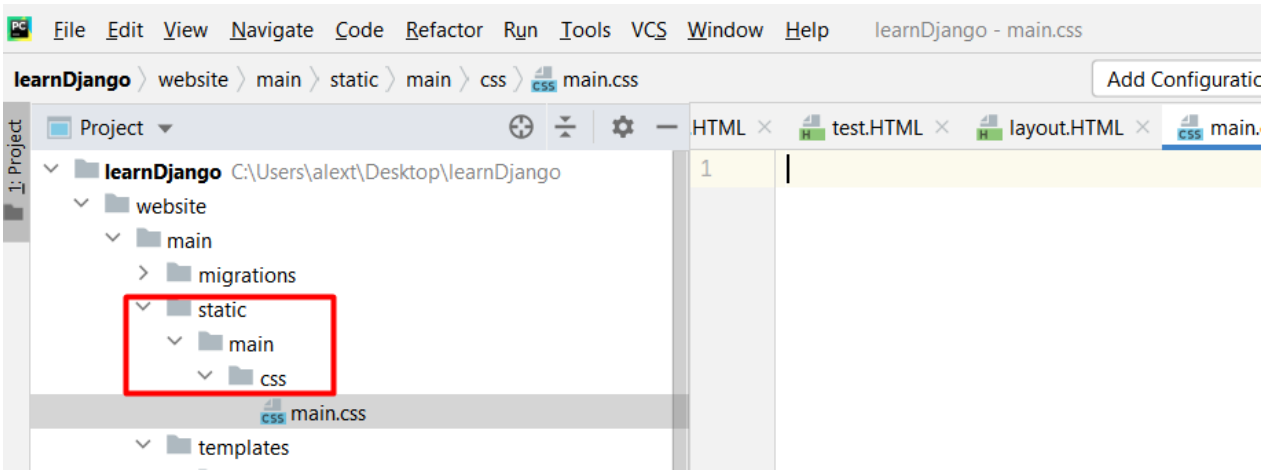


Рисунок 1.82

Впишемо у нього задній фон – червоний.

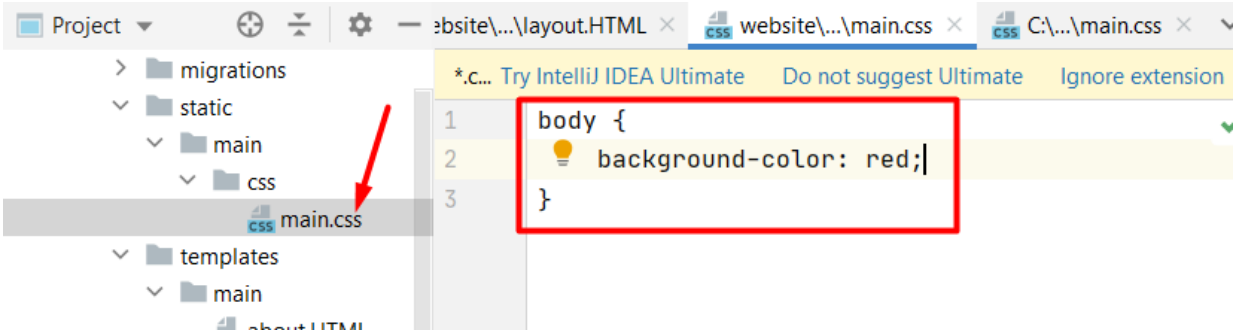


Рисунок 1.83

Підключимо файл. Для цього у файл **layout.HTML** впишемо на початку файлу підключення до папки static у шаблоні {% %}.

Рисунок 1.84

Тепер потрібно прокласти шлях до файлу **main.css**, який знаходиться в папці static. Обов'язково підключення до файлу **main.css** повинно стояти ПІСЛЯ підключення до bootstrap.

Рисунок 1.85

Підключаємося у **settings.py**.

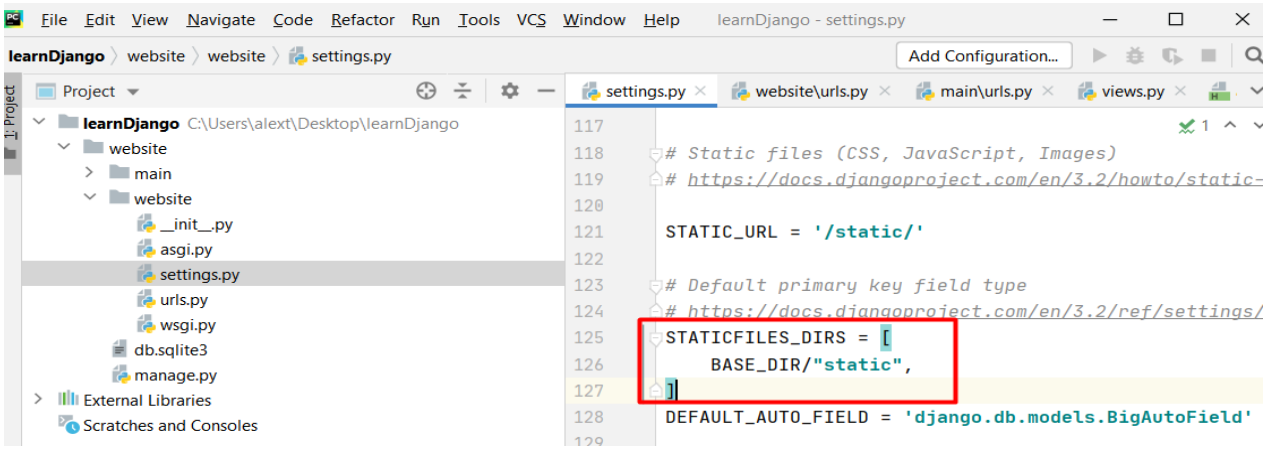


Рисунок 1.86

Додаємо підключення статичних файлів.

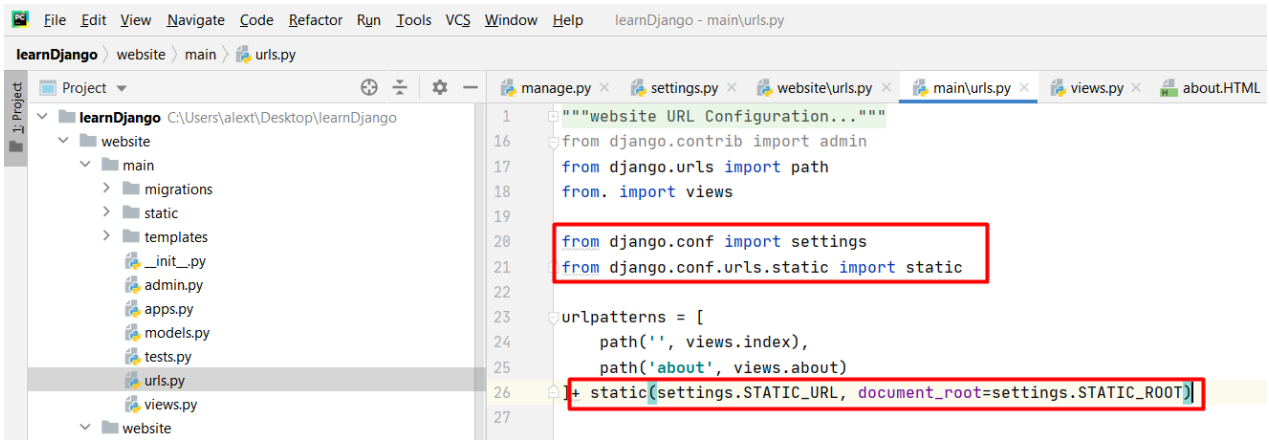


Рисунок 1.87

Перевіримо роботу програми.

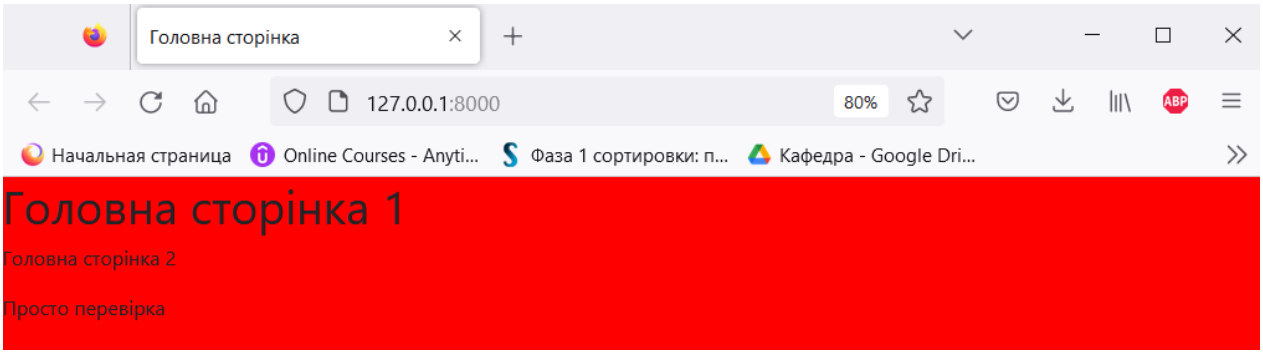


Рисунок 1.88

Якщо програма не спрацювала, натисніть правою кнопкою миші та натисніть “переглянути код”.

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0,
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Главная страница</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/css/bootstra
    <link rel="stylesheet" href="/templates/main/css/main.css">
</head>
<body>

    <h1>Главная страница 1</h1>
    <p>Главная страница 2</p>



</body>
</html>
```

Рисунок 1.89

Практичне завдання:

1. Повторити хід роботи.
2. Отримати результат.

Практична робота №2

Тема. Фреймворк Django (Джанго). Логотип. Навігація. Робота зі стилями. Частина 2.

Мета роботи. Ознайомитись зі створенням логотипу та навігації по сайту.

Зміст.

1. Оформлення сайту. Логотип. Навігація. Робота зі стилями.
2. Виконання роботи.
3. Отримання результату.

Хід роботи

Переходимо до оформлення сайту.

До папки додамо логотип

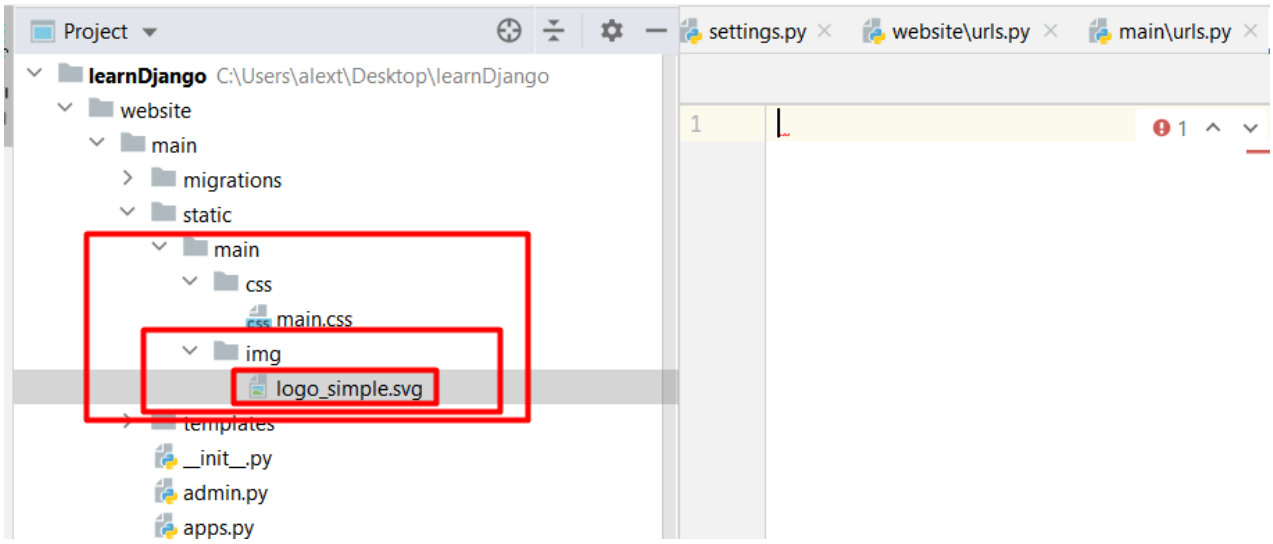


Рисунок 2.1

Прописуємо додавання логотипу.

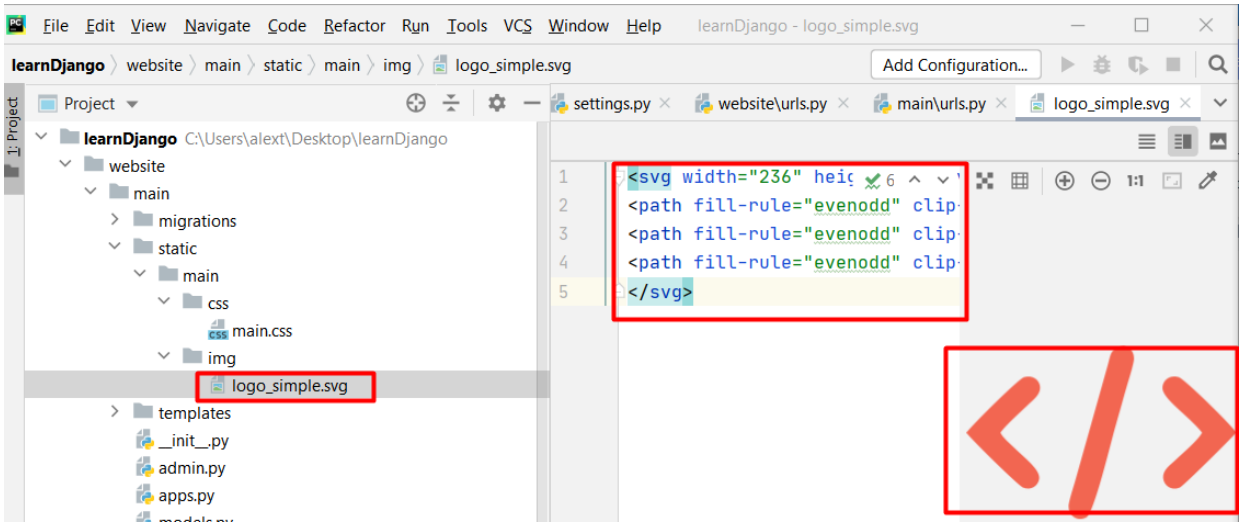


Рисунок 2.2

Код зображення логотипу у форматі svg:

```
<svg width="236" height="151" viewBox="0 0 236 151" fill="none"
xmlns="http://www.w3.org/2000/svg">
<path fill-rule="evenodd" clip-rule="evenodd" d="M0.0698242 75.5851C2.3407 79.4332
45.409 121.642 49.6375 123.281C56.8414 126.062 62.6362 121.49 63.6151
116.004C65.1031 107.852 50.8904 97.5664 41.8461 88.6901C37.8133 84.7281
32.0578 80.3093 29.787 75.547C34.5636 68.7664 48.0322 57.0708 54.7666
50.5566C58.0164 47.4328 64.7114 42.6709 64.4765 36.309C64.2416 29.8708
58.0945 25.2994 50.4989 27.3947C45.6439 28.7661 3.86767 70.0236 0.0698242
75.5851Z" fill="#F26651"/>
<path fill-rule="evenodd" clip-rule="evenodd" d="M132.37 1.65955C125.441 5.05002
126.302 7.98334 123.209 18.5738L106.177 78.6118C102.38 91.9833 98.8556
104.517 94.9013 118.345C93.57 122.955 89.537 135.183 89.3807 139.183C89.1065
147.259 96.7023 153.393 104.846 149.126C111.62 145.545 126.498 83.1833 129.904
71.793C133.898 58.536 137.069 45.7738 141.258 32.0977C142.511 27.9834 146.583
14.8786 146.583 11.2977C146.583 3.14526 139.731 -1.9595 132.37 1.65955Z"
fill="#F26651"/>
<path fill-rule="evenodd" clip-rule="evenodd" d="M206.096 75.5471C202.454 82.4427
188.673 93.6809 182.173 99.9665C178.689 103.319 171.602 107.814 172.111
114.785C172.62 121.528 179.158 126.252 186.519 123.281C189.534 122.062
232.837 79.4332 235.93 75.5089C232.289 69.376 190.708 28.8424 185.54
27.3567C178.023 25.1853 171.876 29.7948 171.484 36.3091C170.897 46.0996
194.389 60.7662 206.096 75.5471Z" fill="#F26651"/>
</svg>
```

У тегі <aside> у нас буде бокова панель, а у тезі <main> основна панель.

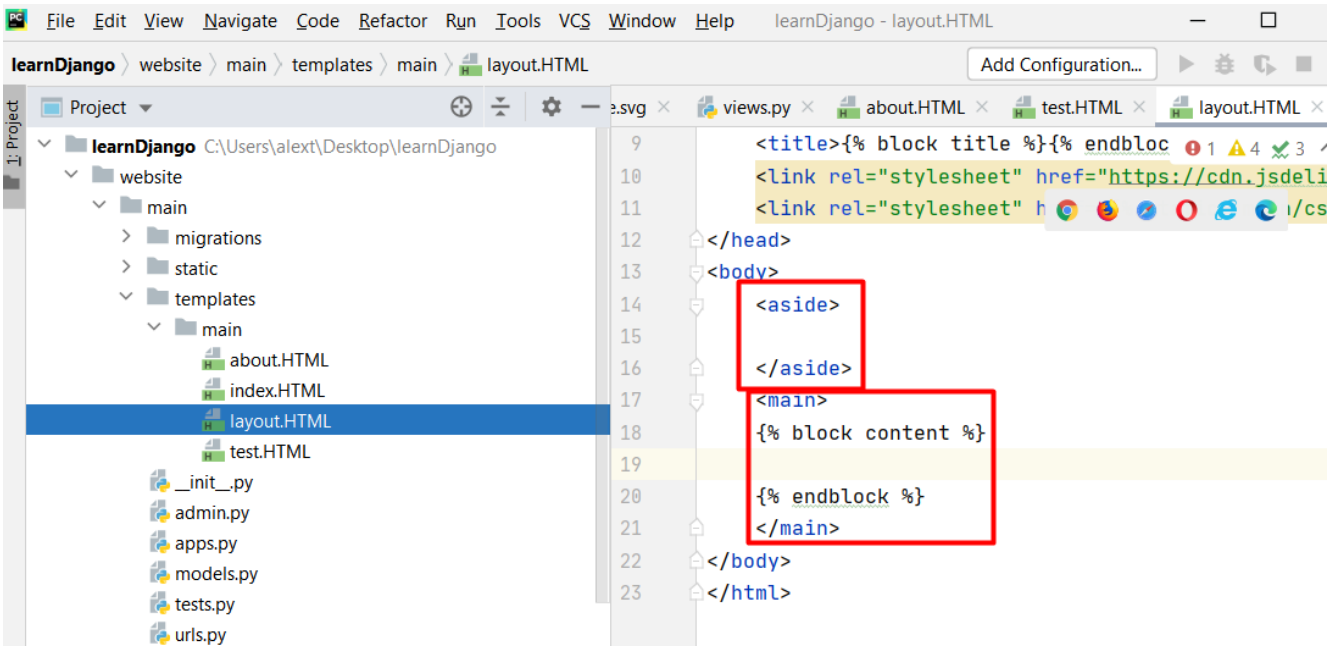


Рисунок 2.3

Підключимо картинку з логотипом на бічну панель. Для цього всередині тегів <aside> впишемо саму картинку та додатковий опис – слово Лого.

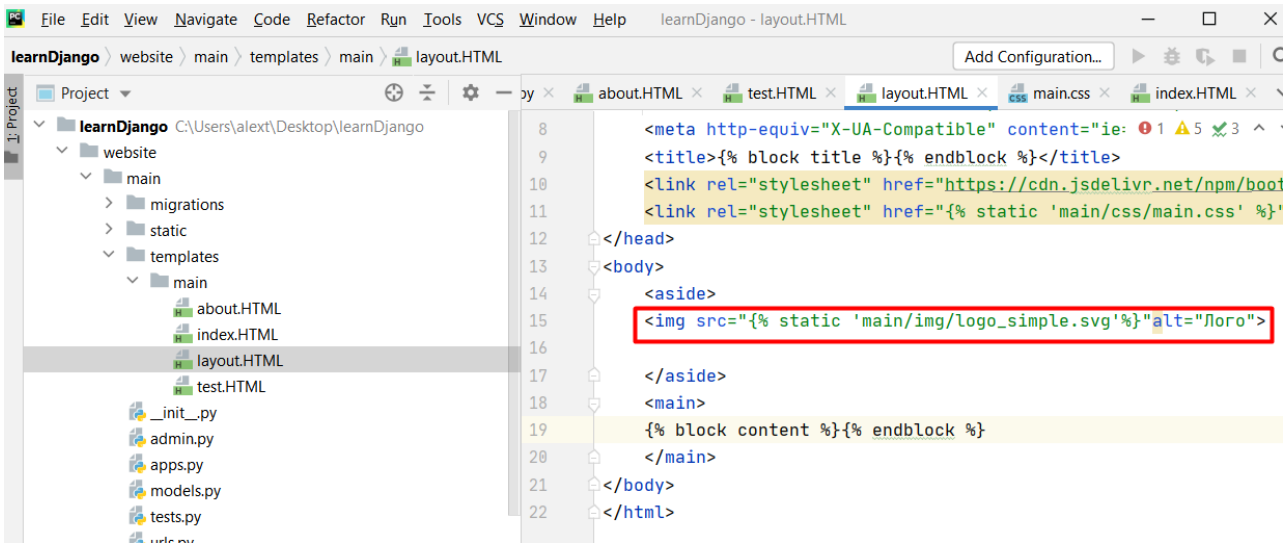


Рисунок 2.4

Створимо навігацію.

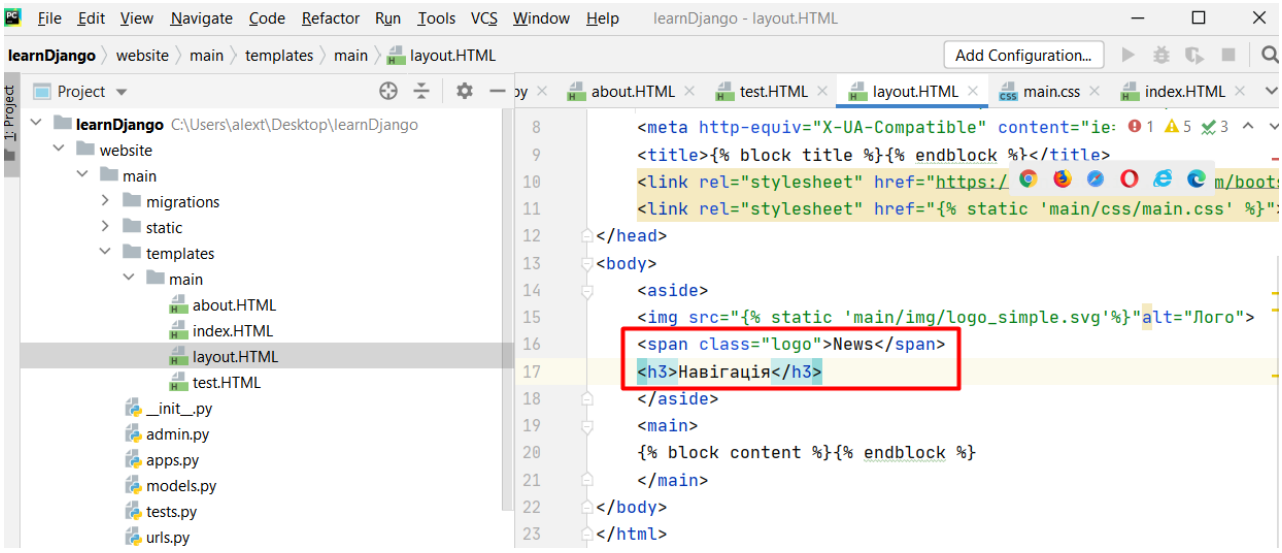


Рисунок 2.5

Впишемо конструкцію ul>li*3. Ми створюємо тег ul, а в ньому 3 штуки тегів li. І натискаємо клавішу табуляції.

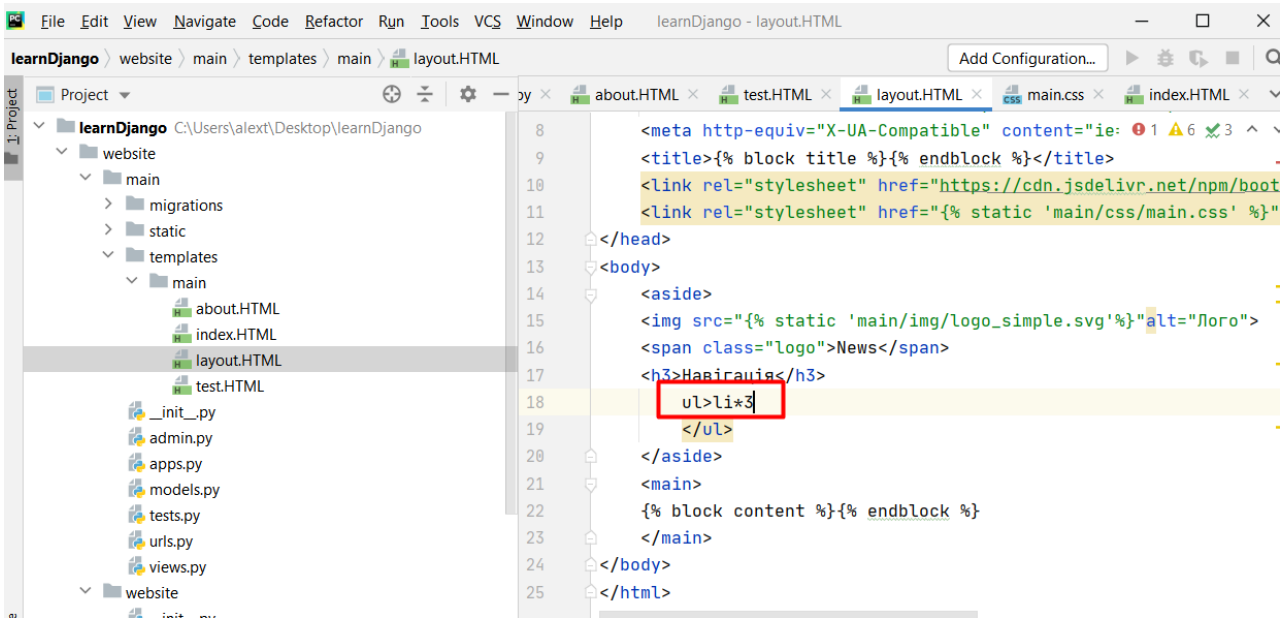


Рисунок 2.6

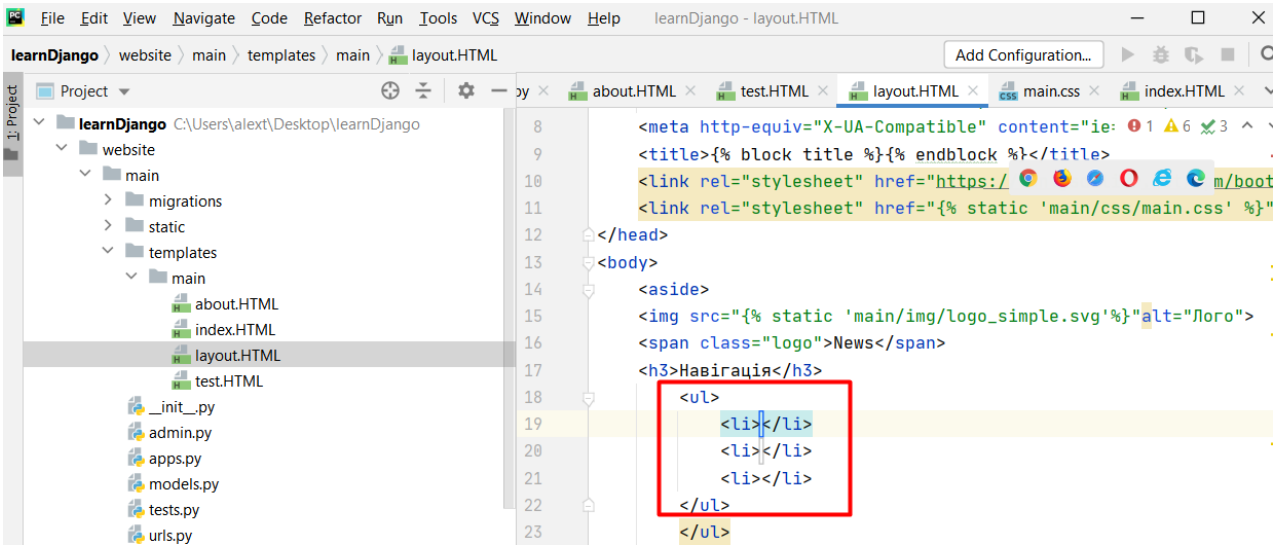


Рисунок 2.7

Вставимо код навігації.

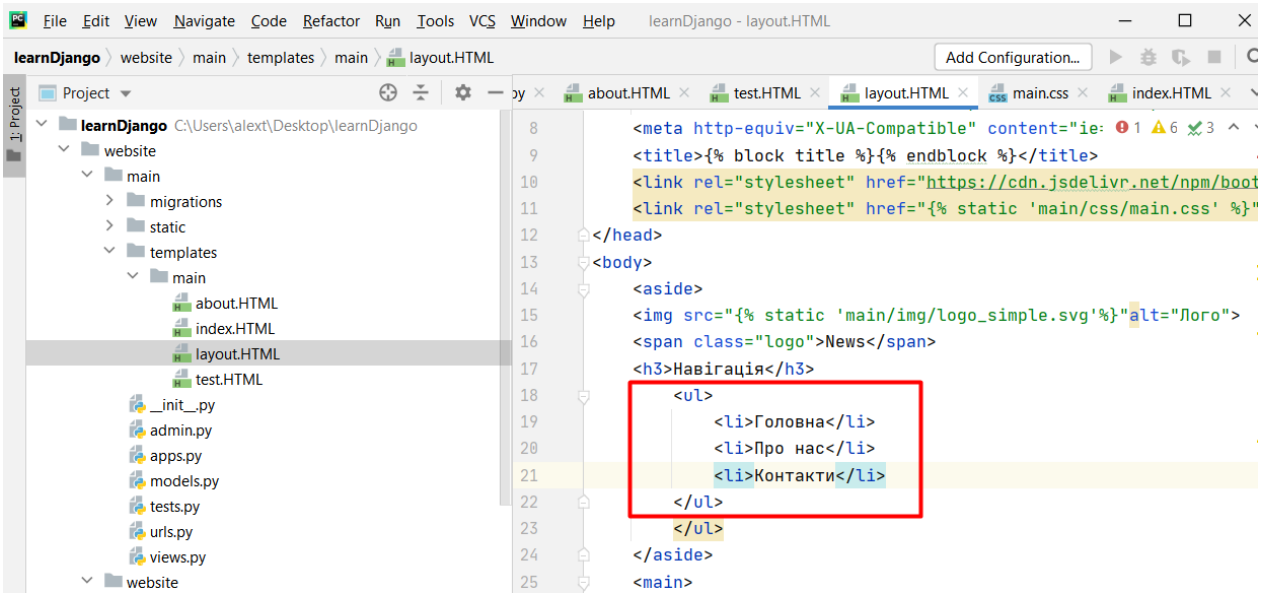


Рисунок 2.8

Вставимо іконки біля назв навігацій. Для цього повторимо:

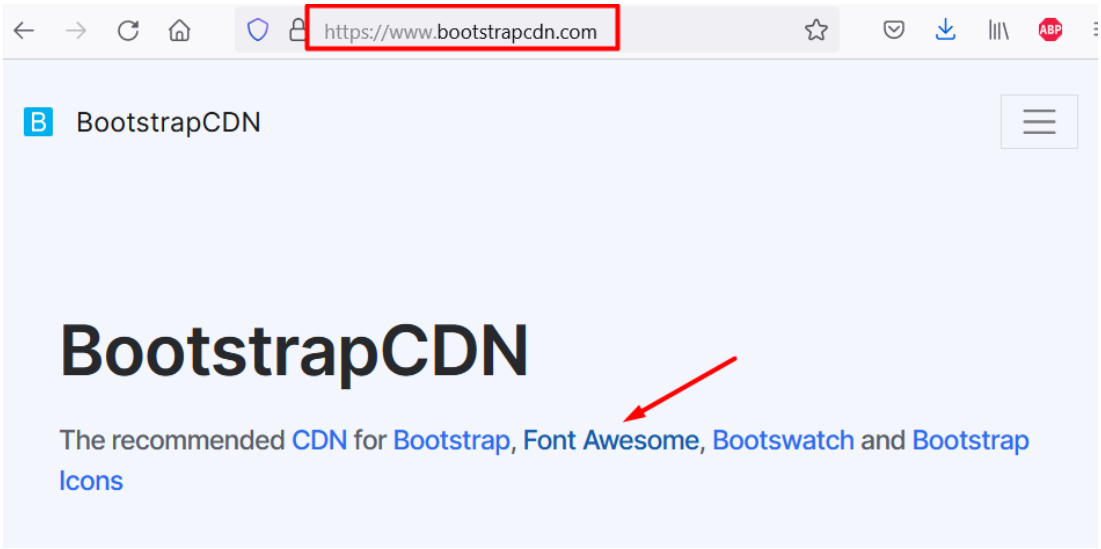


Рисунок 2.9

Зареєструйтеся на сайті Font Awesome та отримайте власний набір іконок. Для цього:

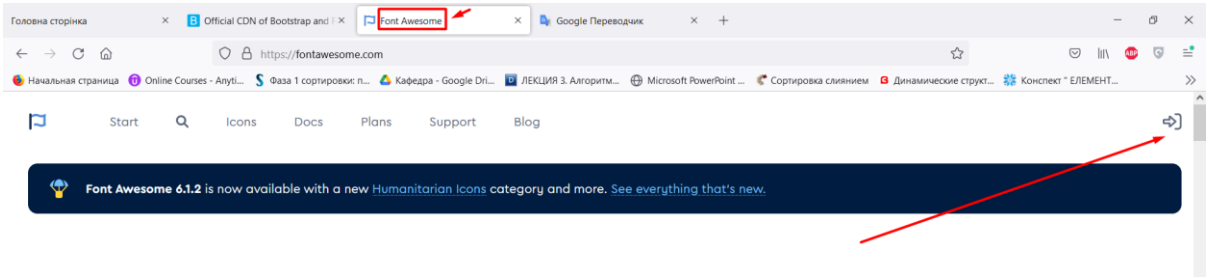


Рисунок 2.10

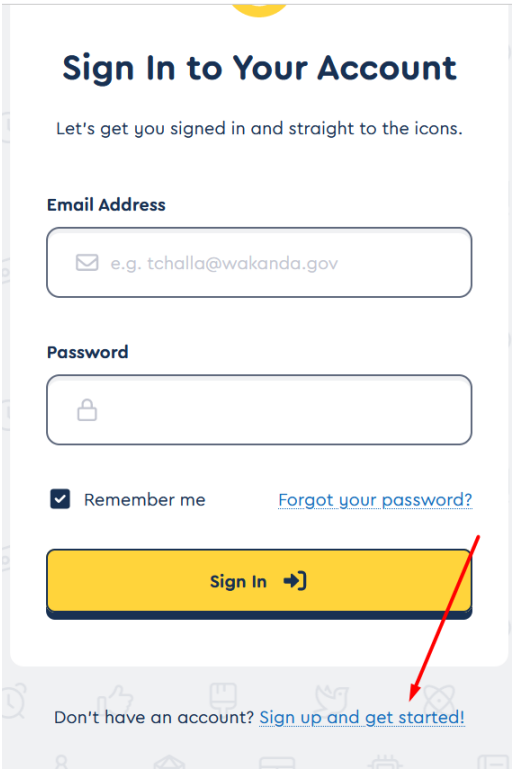


Рисунок 2.11

Впишіть власну електронну пошту та натисніть на кнопку "Відправити код набору".

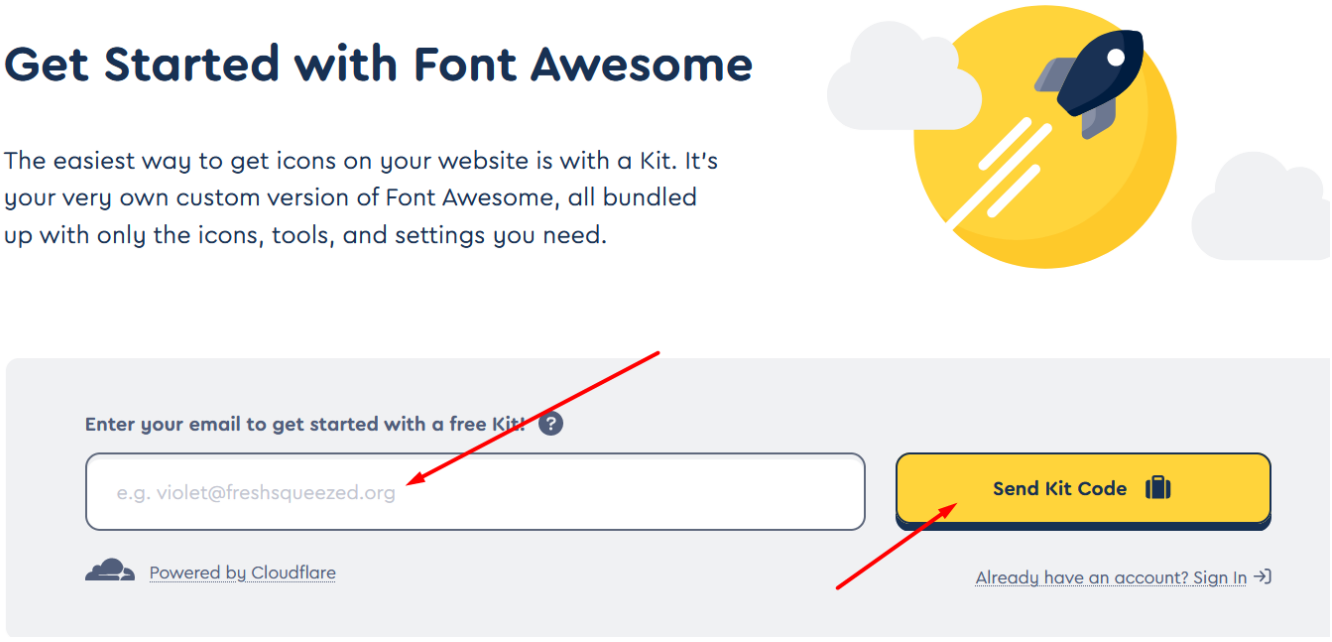


Рисунок 2.12

Після чого на пошті буде ваше посилання.

Вписуємо власне посилання для роботи з іконками.

Рисунок 2.13

Зайдіть на сторінку з пошуком іконок, введіть у пошук arrow-alt-circle-down і натисніть на знайдену іконку.

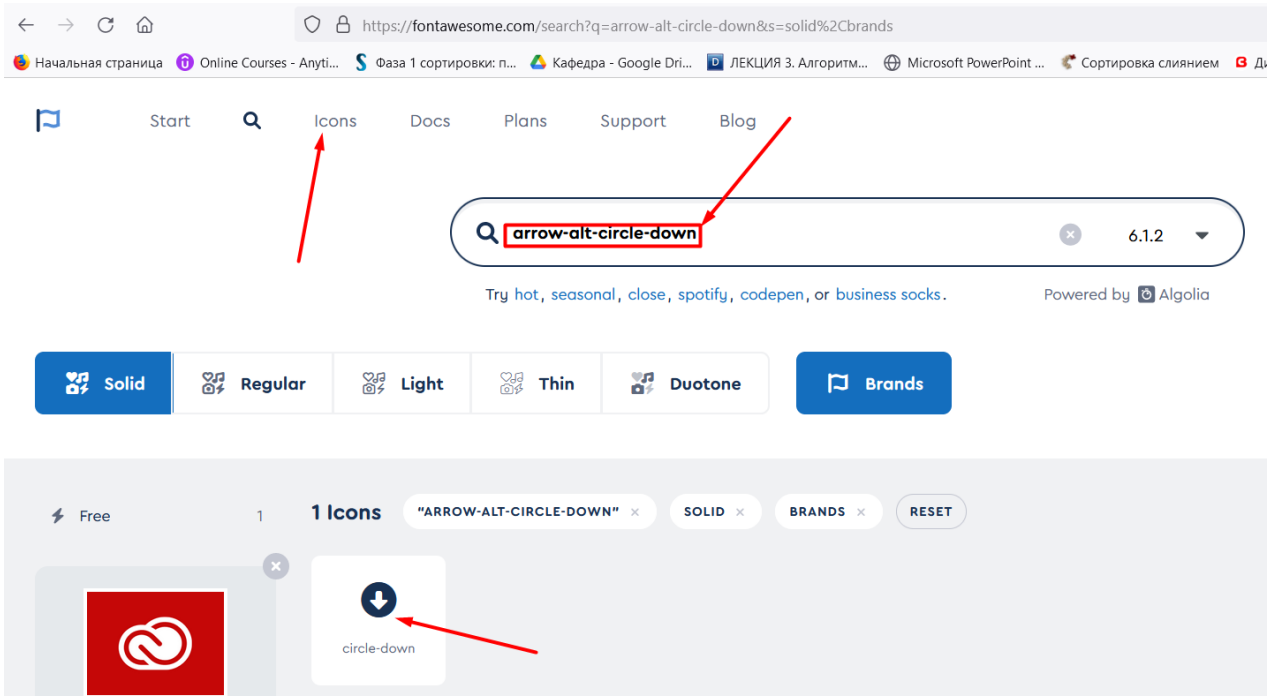


Рисунок 2.14

Скопіюйте код іконки.

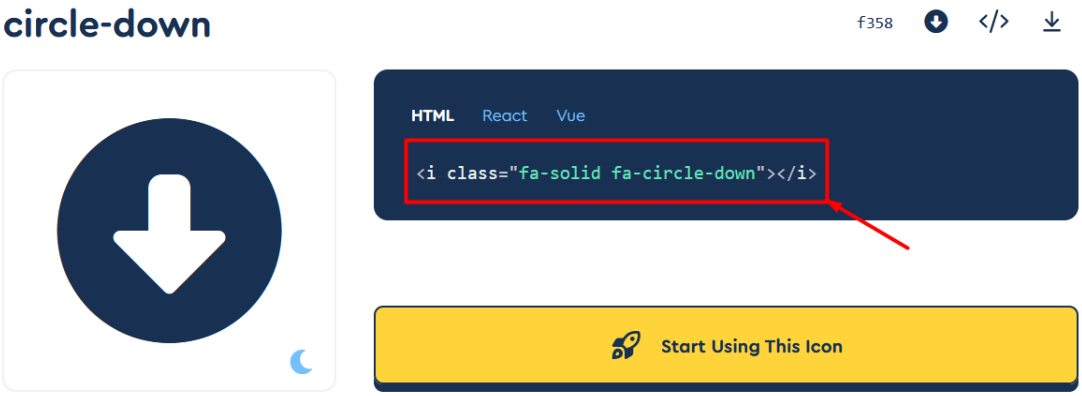


Рисунок 2.15

Так само зробіть і для іконки з назвою angles-right та apple-whole.

Перед назвою вставте посилання на іконку.

```
<a href=""><li><i class="fa-solid fa-angles-right"></i>Головна</li></a>

<a href=""><li><i class="fa-solid fa-circle-down"></i>Про нас</li></a>

<a href=""><li><i class="fa-solid fa-apple-whole"></i>Контакти</li></a>
```

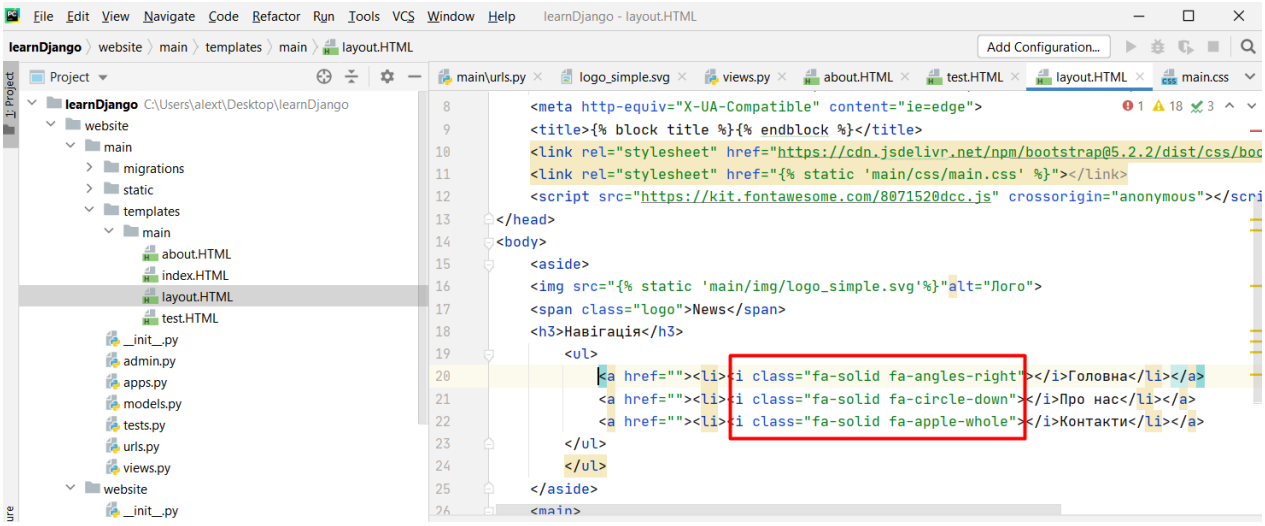


Рисунок 2.16

Перевірте роботу сторінки.

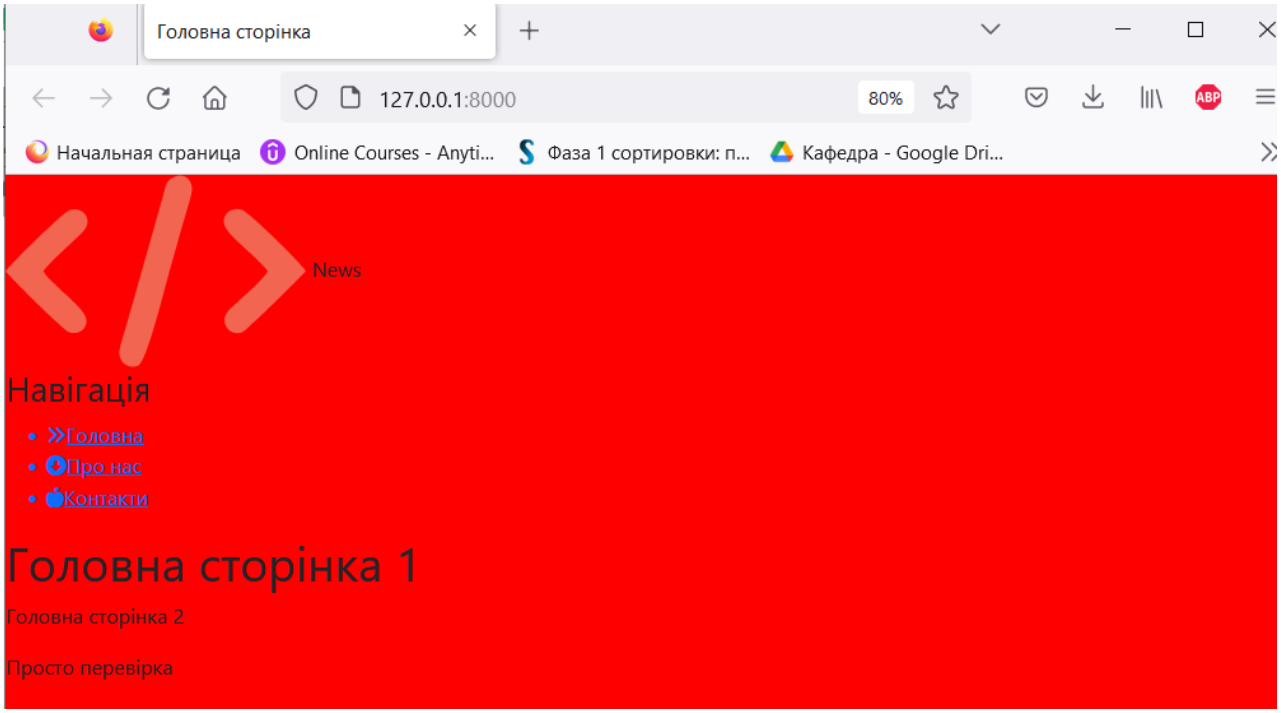


Рисунок 2.17

Почнемо роботу зі стилями. Повернемося і підключимо заднє тіло для цього допишемо в `<body class="main">`.

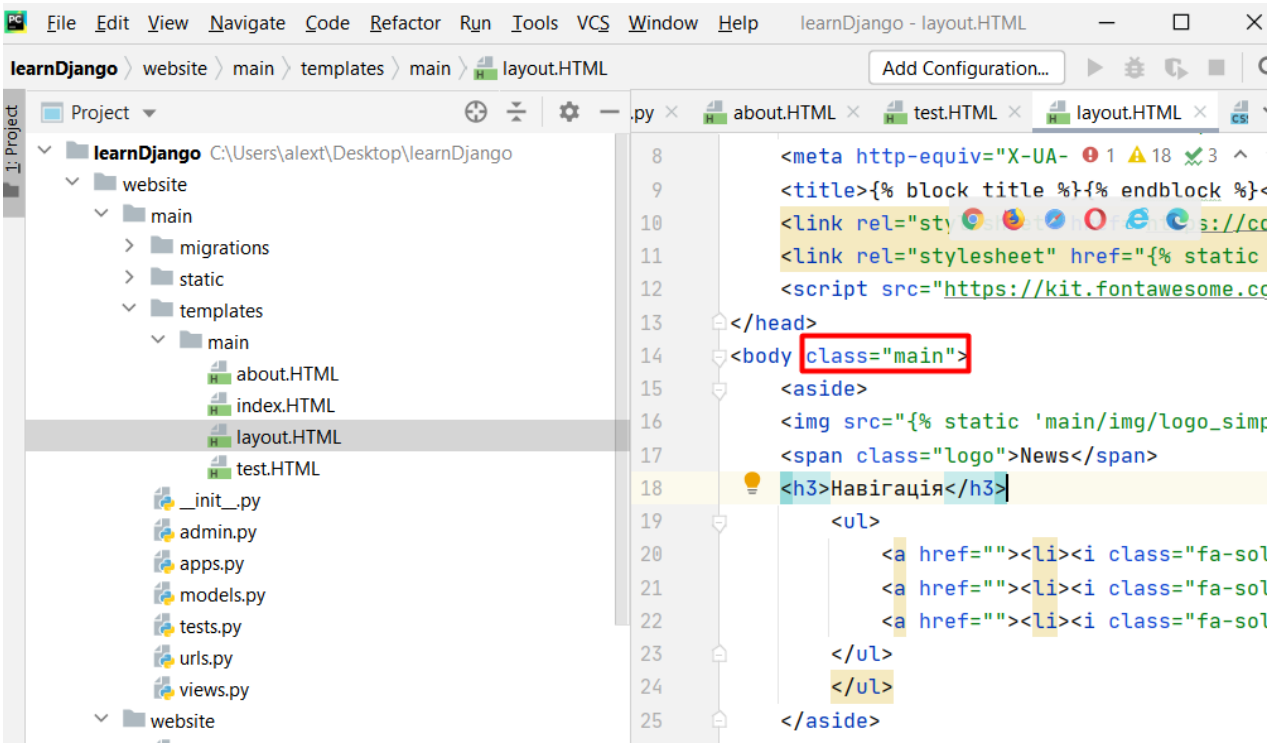


Рисунок 2.18

Пропишемо у файлі **main.css** задній фон - сірий.

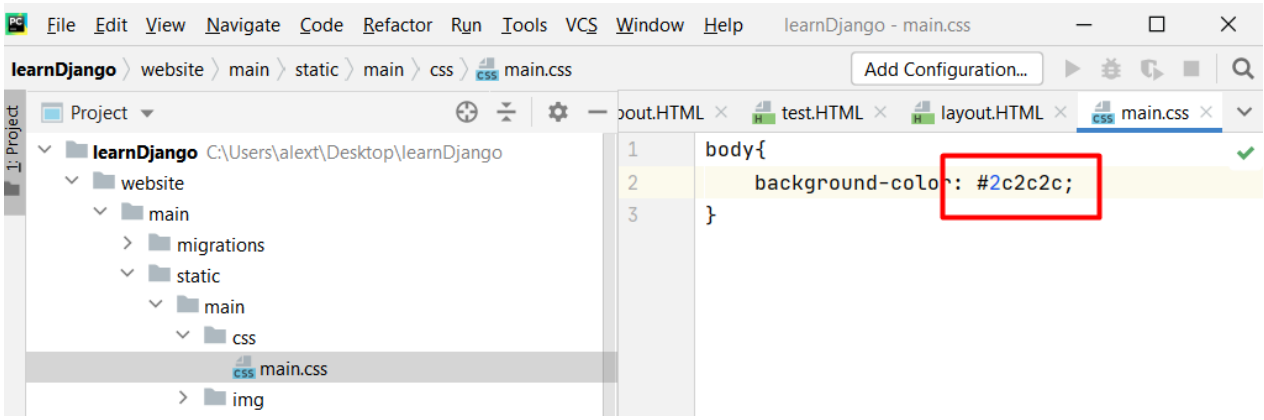


Рисунок 2.19

Підберемо стилі для тега aside: 1. Він має бути ліворуч. 2. Заднє тіло має бути вже інше. 3. Ширина блоку 20%. 4. Товщина лінії 2,5% з усіх боків. 5. Висота 100% висоти екрану. 6. Весь текст всередині блоку буде білим. 7. Обведення праворуч 5 пікселів суцільне зі своїм кольором.

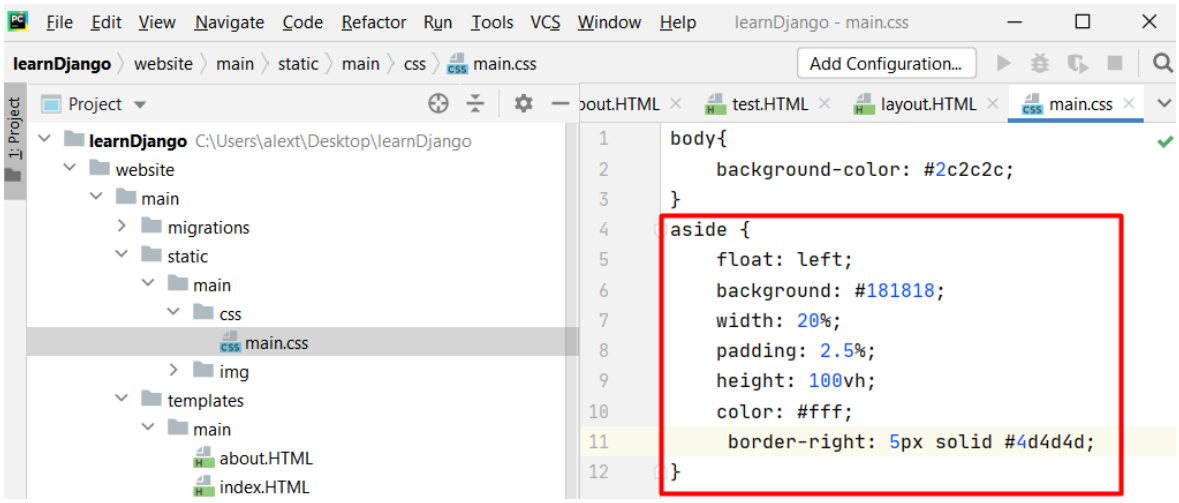


Рисунок 2.20

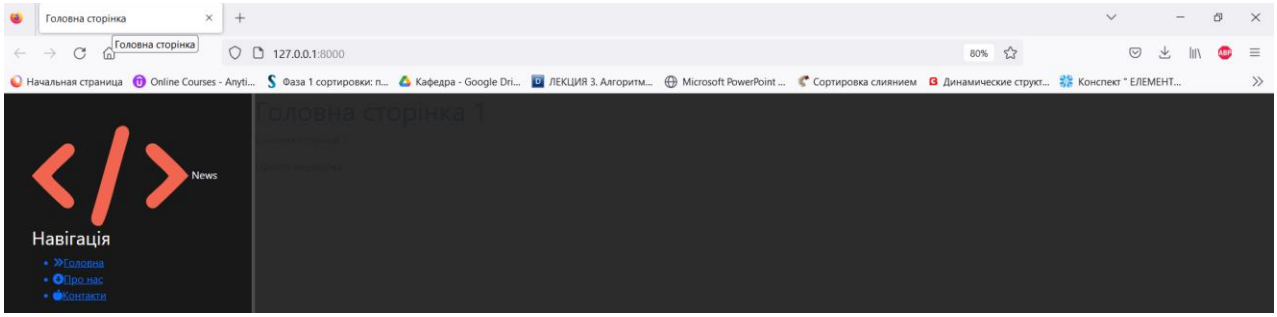


Рисунок 2.21

Зменшимо ширину та висоту картинки. Пропишемо їй ширину 70 пікселів. Розташована має бути зліва.

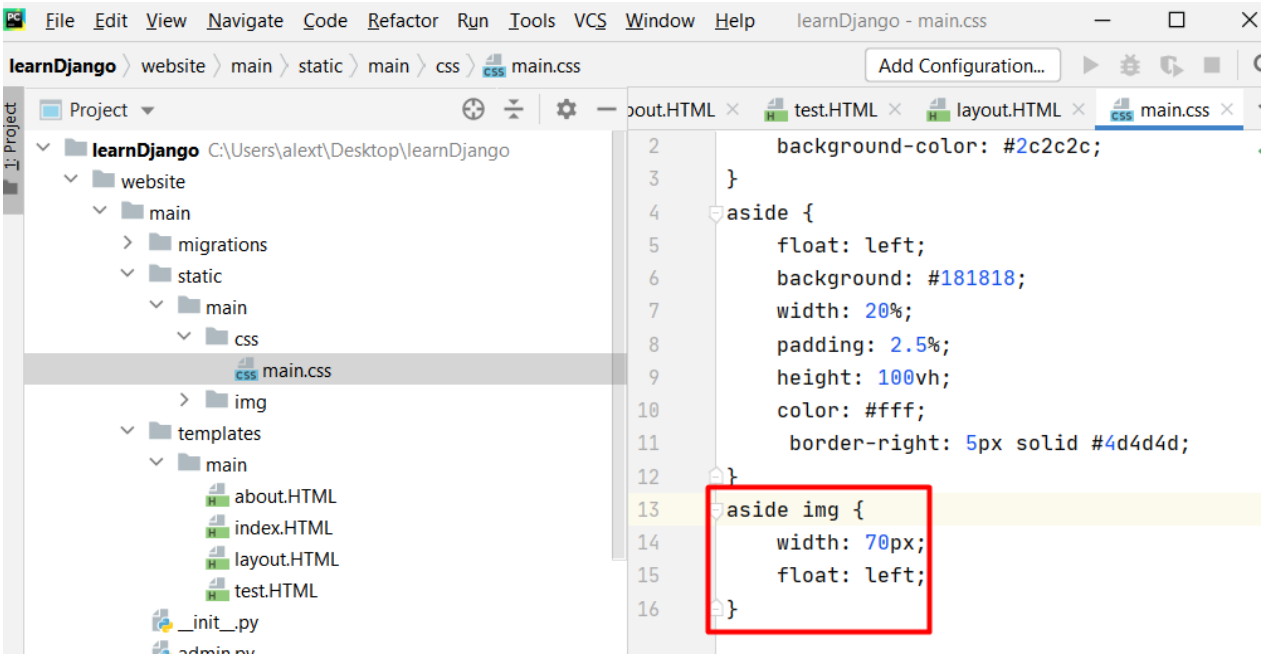


Рисунок 2.21

Отримаємо:

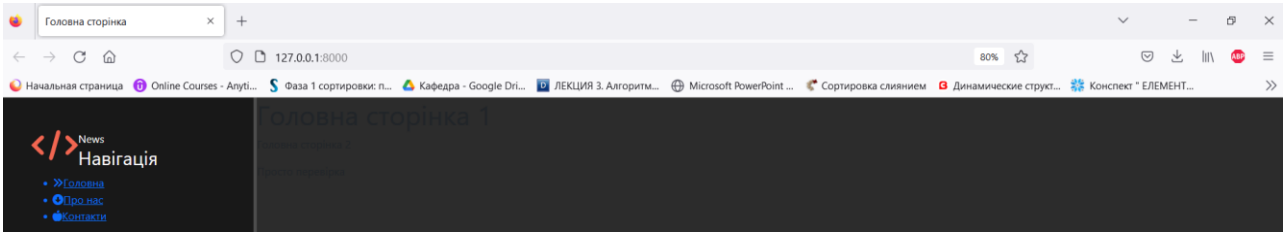


Рисунок 2.22

Літера 30 пікселів **front-size: 30px;**

Відступ з лівого боку 10 пікселів **margin-left: 10px;**

Шрифт жирний **font-weight: bold;**

font-weight може приймати один із дев'яти числових варіантів насиченості:

- 100: Thin;
- 200: Extra Light (Ultra Light);
- 300: Light;
- 400: Normal;

- 500: Medium;
- 600: Semi Bold (Demi Bold);
- 700: Bold;
- 800: Extra Bold (Ultra Bold);
- 900: Black (Heavy).

Всі ці числові значення задають ступінь товщини шрифту від найтоншого до товстого. Але в більшості системних шрифтів все одно є лише два варіанти товщини: звичайний normal (400) та жирний bold (700). Тому й інші значення якості застосовуються рідше. Крім перерахованих вище числових значень у font-weight може бути ще два відносні значення: bolder і lighter. Ці значення роблять шрифт жирнішим і тоншим, ніж поточне або успадковане значення.

Позиція піднята на 5 пікселів:

position: relative;

top: -5px;

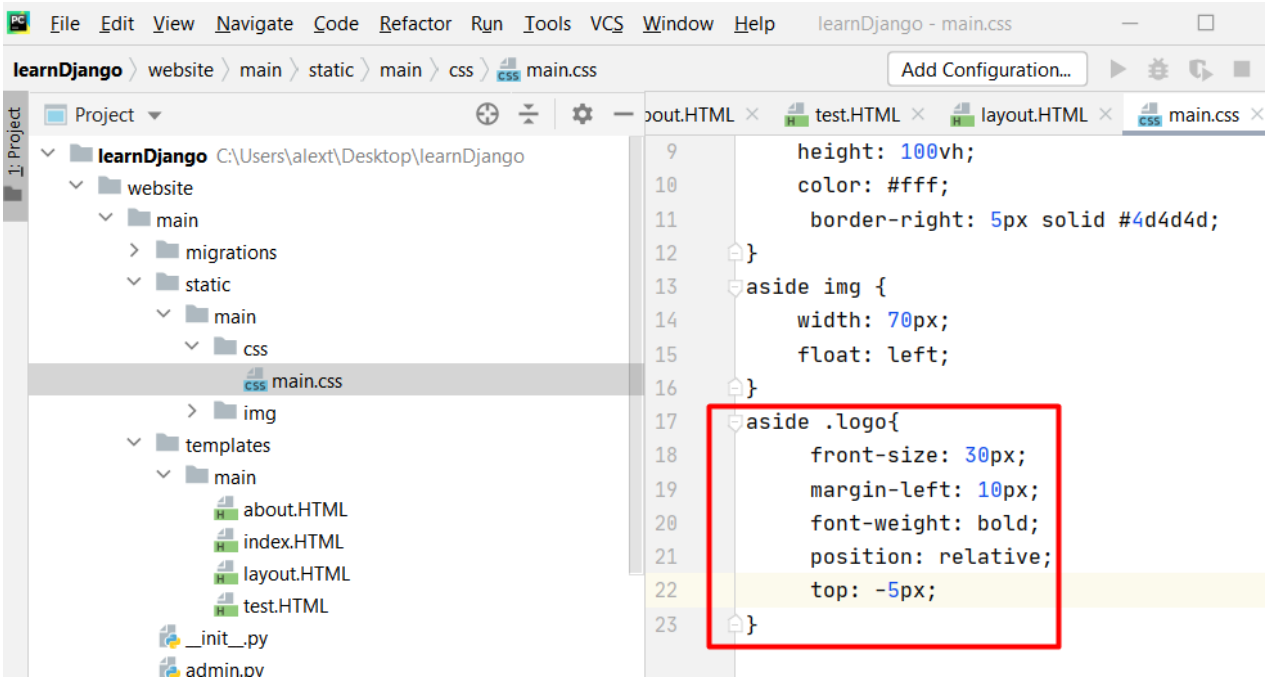


Рисунок 2.23

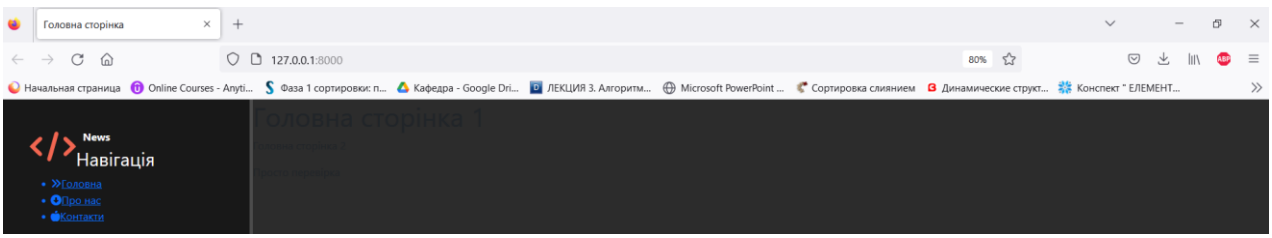


Рисунок 2.24

Переходимо на наступний тег h3:

1. Пропишемо відступ зверху 50 пікселів;

2. Висота літер 28 пікселів;

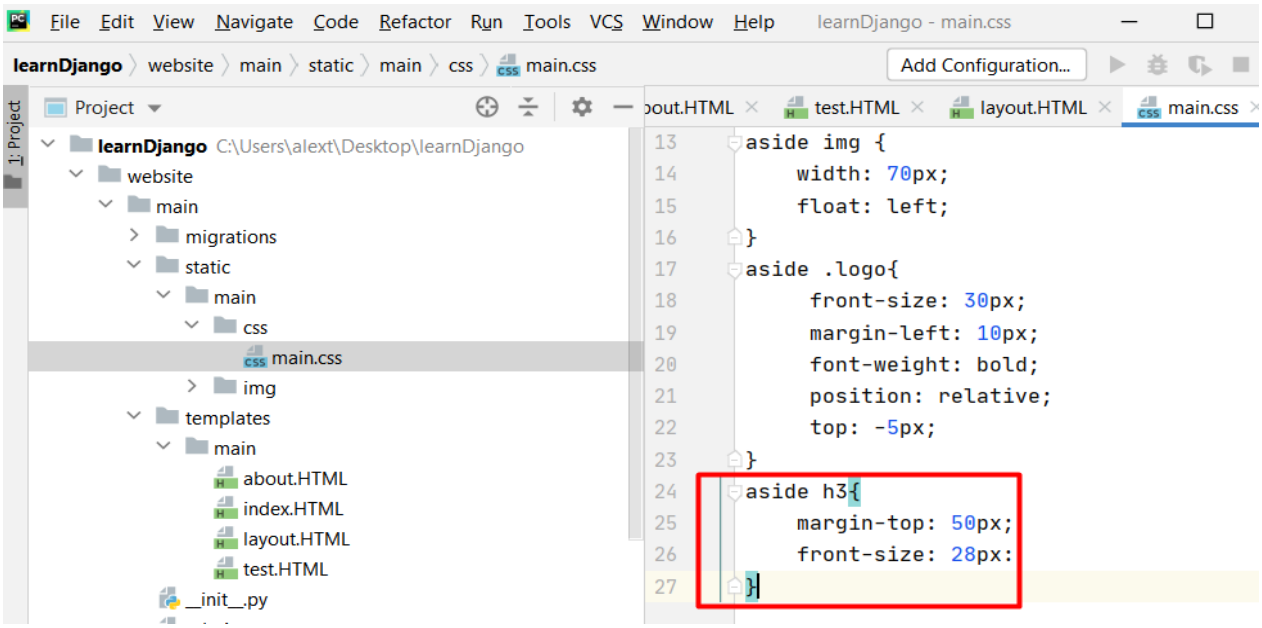


Рисунок 2.25

Отримаємо:

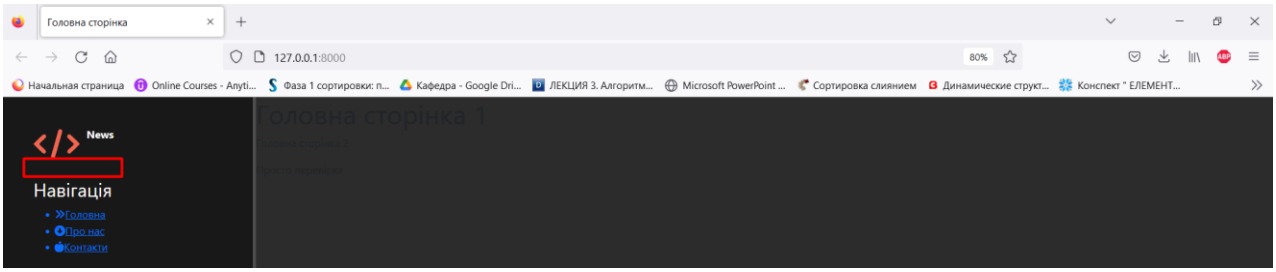


Рисунок 2.26

Додамо тепер загальні стилі всіх наших заголовків.

Приберемо стилі списку.

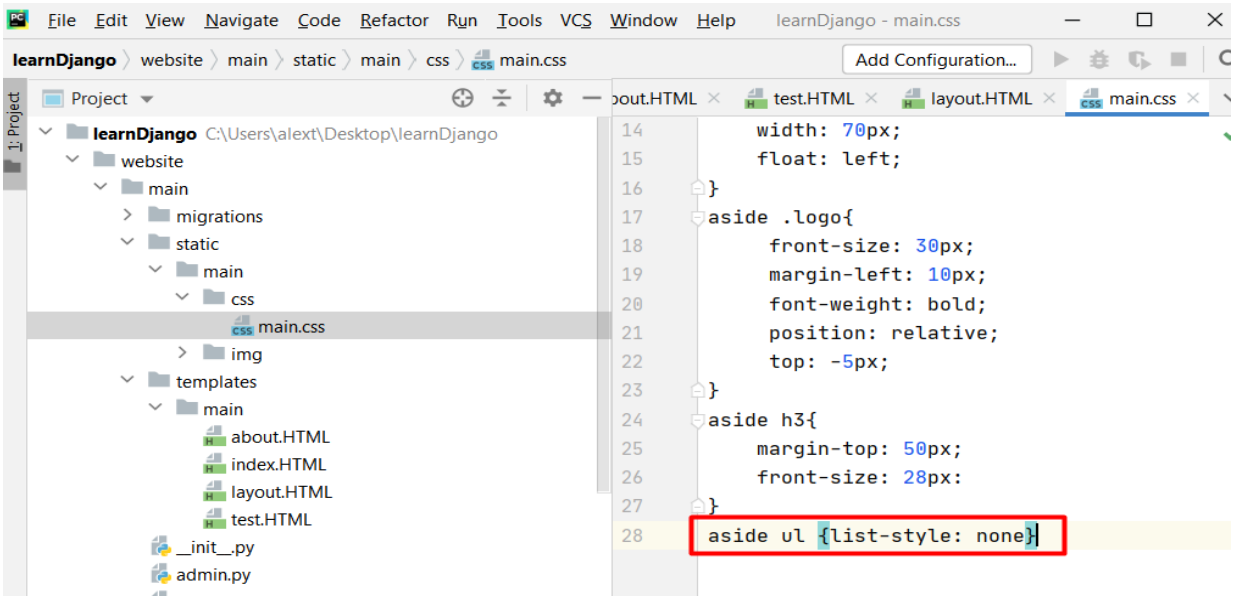


Рисунок 2.27

Отримаємо:

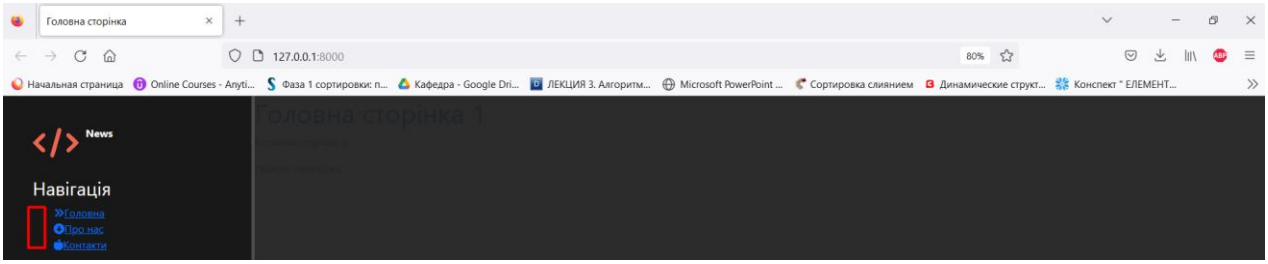


Рисунок 2.28

Тепер звернемося до кожного окремого елемента списку:

1. Колір у кожного елемента списку має бути білий.
2. Кожен елемент має бути блоковим.
3. Відступ зверху 20 пікселів.

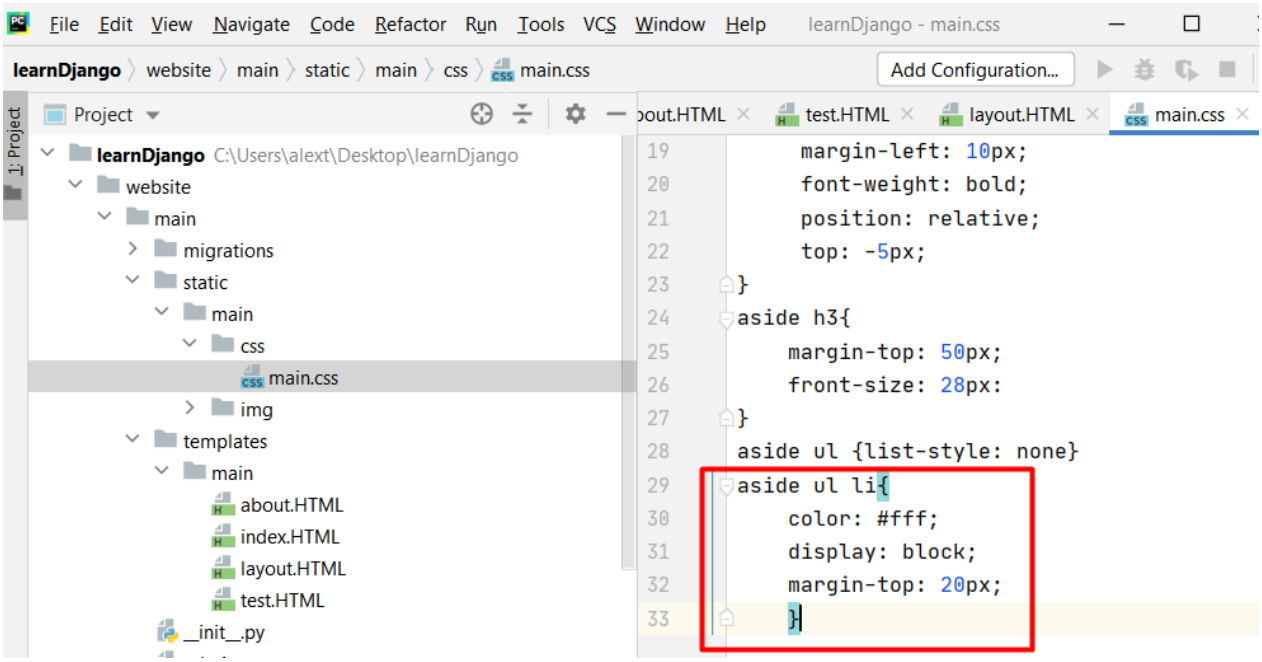


Рисунок 2.29

Отримаємо:

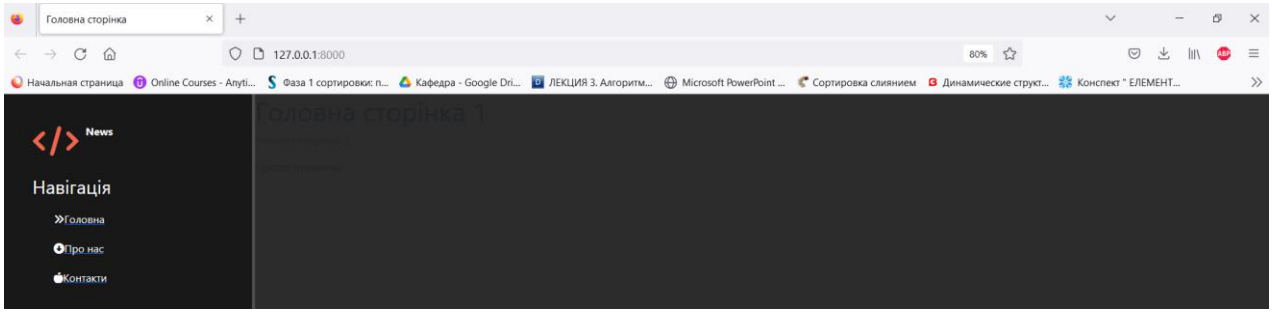


Рисунок 2.30

Додамо ефект при наведенні стрілки миші на тег li та на тег a. Працюємо через псевдо клас hover.

Змінюю колір при наведенні миші **color: #eb5959;**

Забираємо нижнє підкреслення **text-decoration: none;**

І за наведенні текст збільшуватиметься у 1.1 разу тобто на 10%. Звертаюся до властивості **transform transform: scale(1.1);**

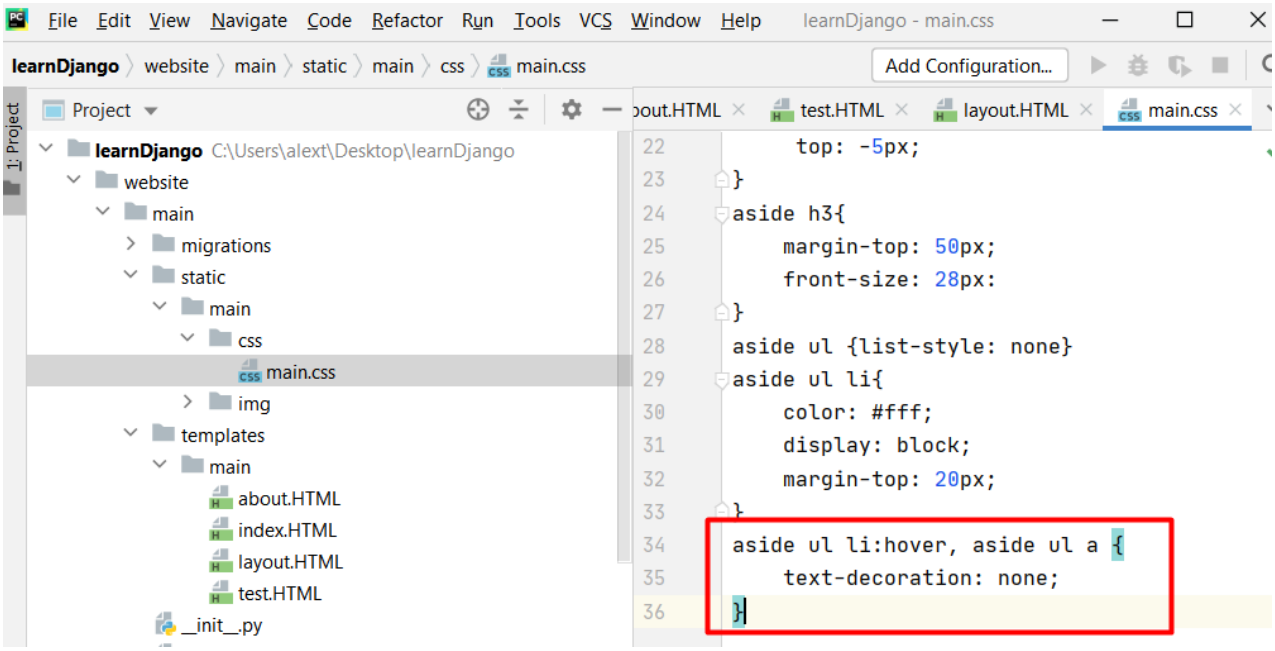


Рисунок 2.31

Запускаємо програму. При наведенні курсору на "Про нас" - шрифт виділяється червоним кольором та збільшується на 10%.

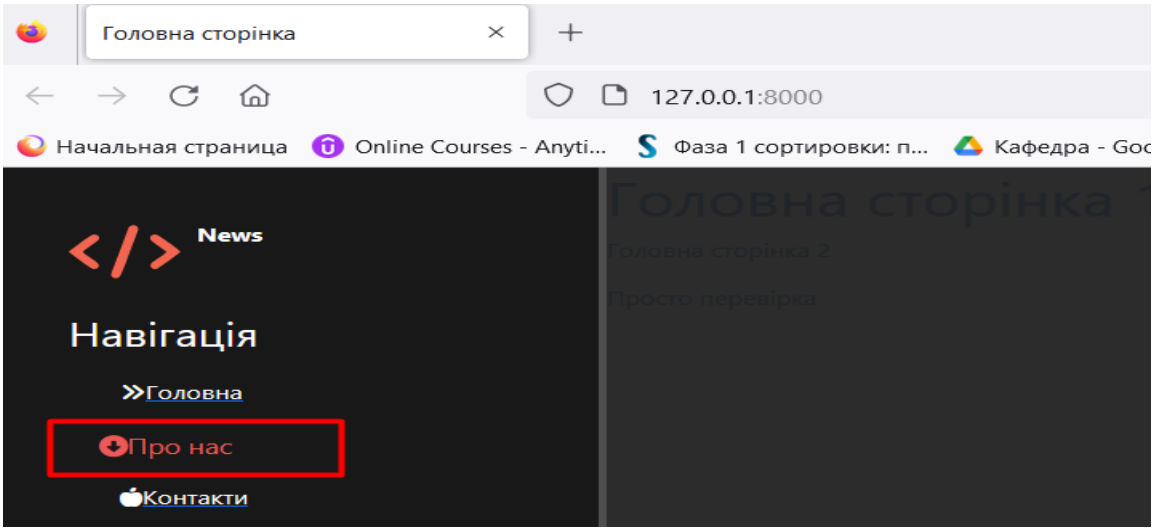


Рисунок 2.32

Для того, щоб збільшення на 10% відбувалося плавно, додамо ефект згладжування для всіх стилів 600мс з ефектом **ease**.

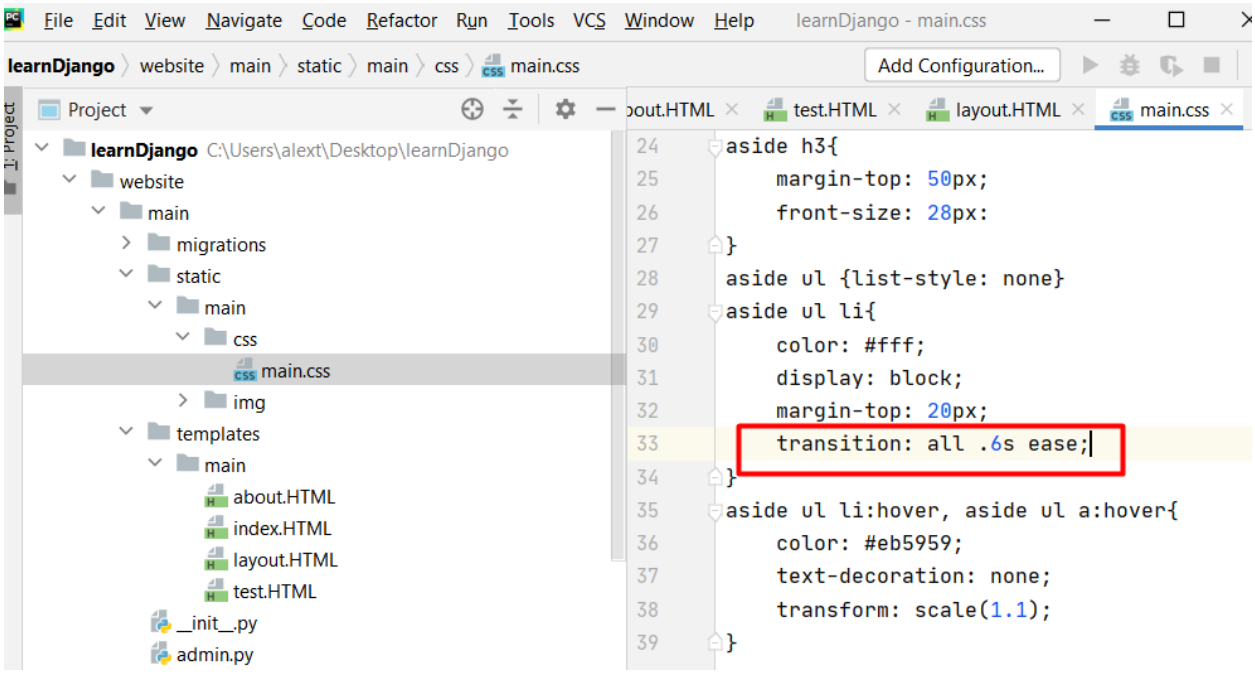


Рисунок 2.33

Створимо посилання. Заходимо в папку main **urls.py,** додаємо ім'я для головного посилання та для посилання about.

Рисунок 2.34

Заходимо у **layout.HTML** та прописуємо:

Рисунок 2.35

Зробимо красиві стилі для основної сторінки. Зайдемо у файл і видалимо те, що там було записано. Додамо новий div і утворимо клас features і кнопку використовуючи стилі Bootstrap.

Рисунок 2.36

Тепер додамо стилі до тексту, який не бачимо. Переходимо у файл **main.css**, підключаємося к тегу **main** та класу **features**.

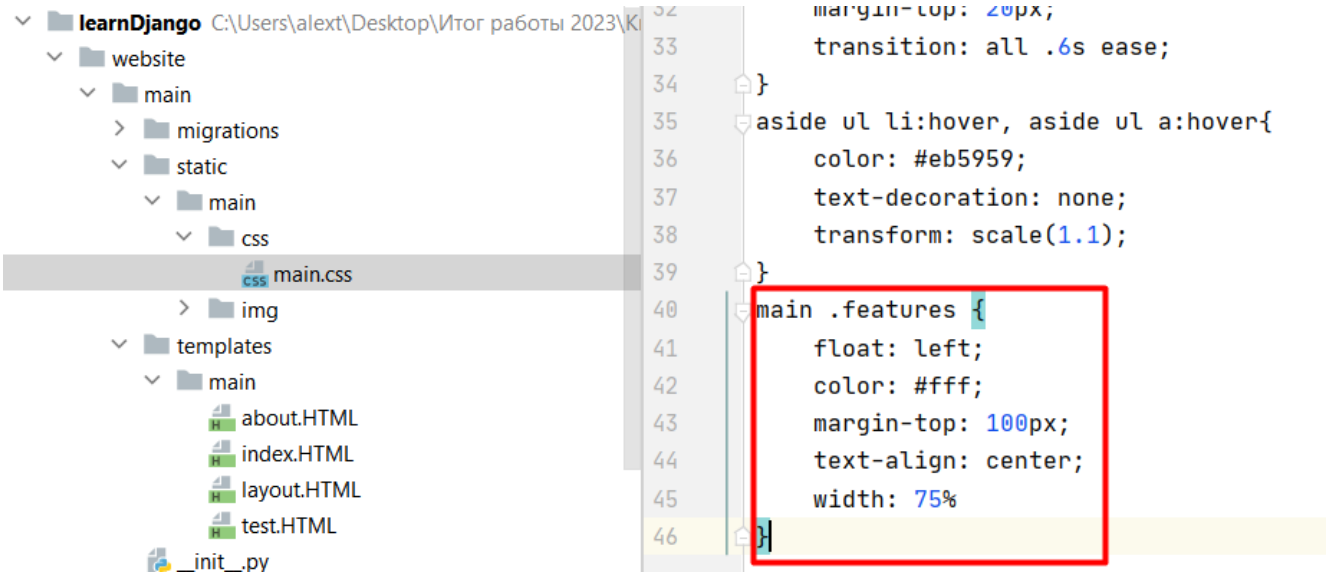


Рисунок 2.37

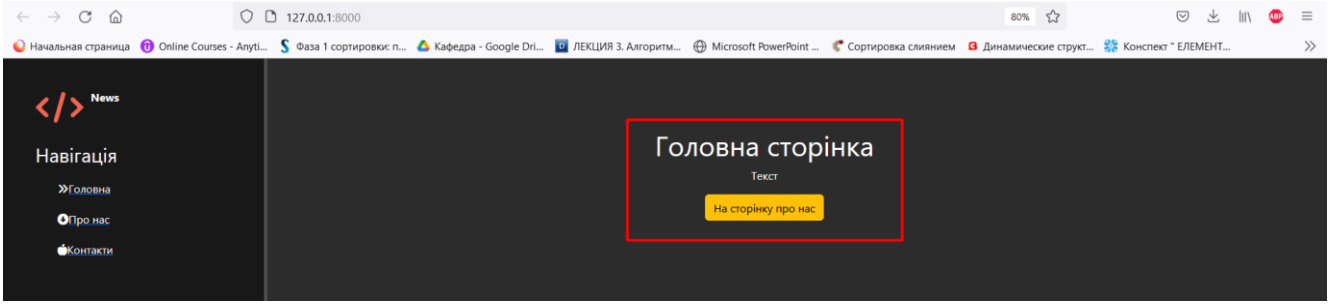


Рисунок 2.38

Додаємо ще стилей.

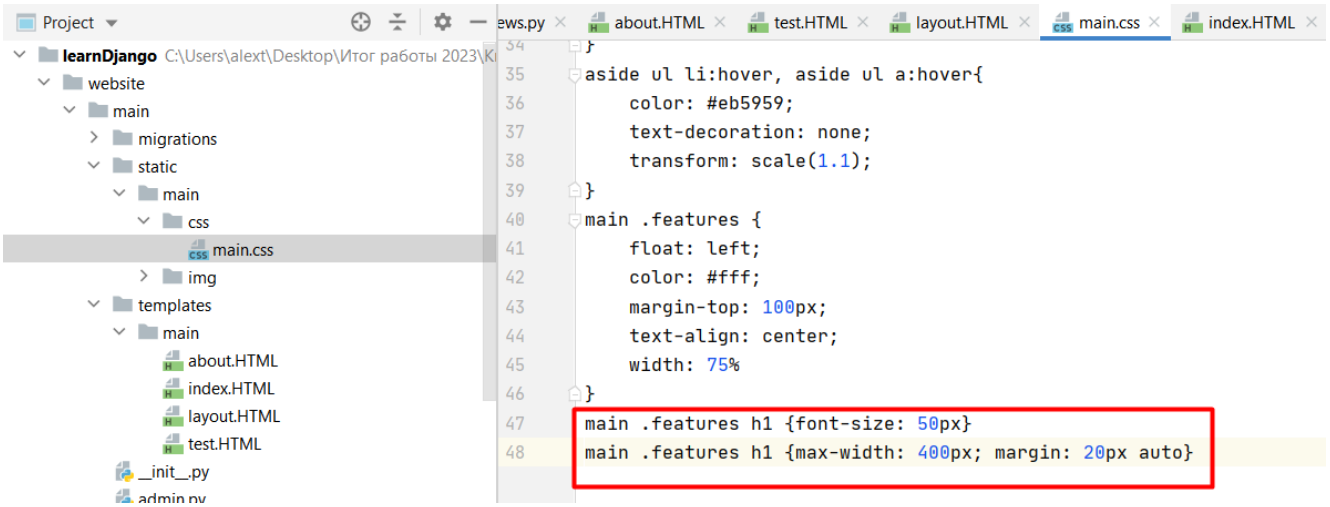


Рисунок 2.39

Перенесемо інформацію з файлу **index.HTML** у файл **about.HTML**.

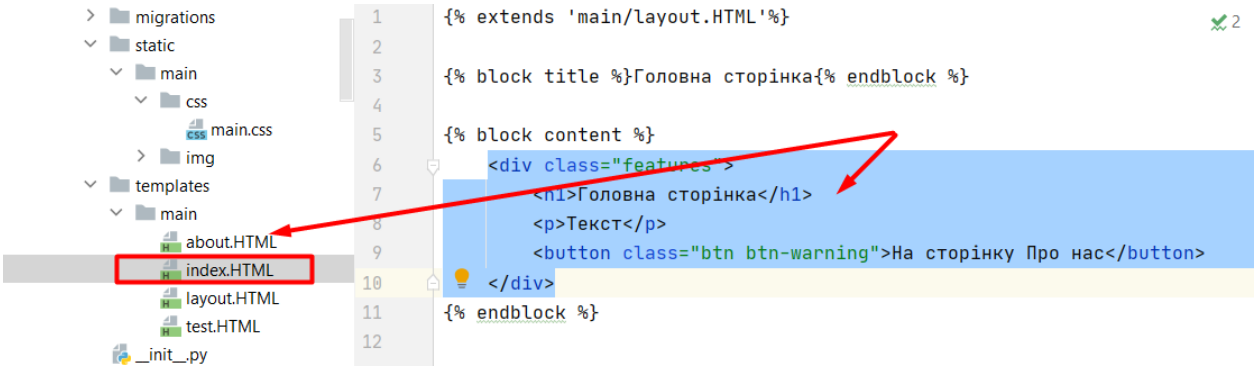


Рисунок 2.40

Отримаємо:

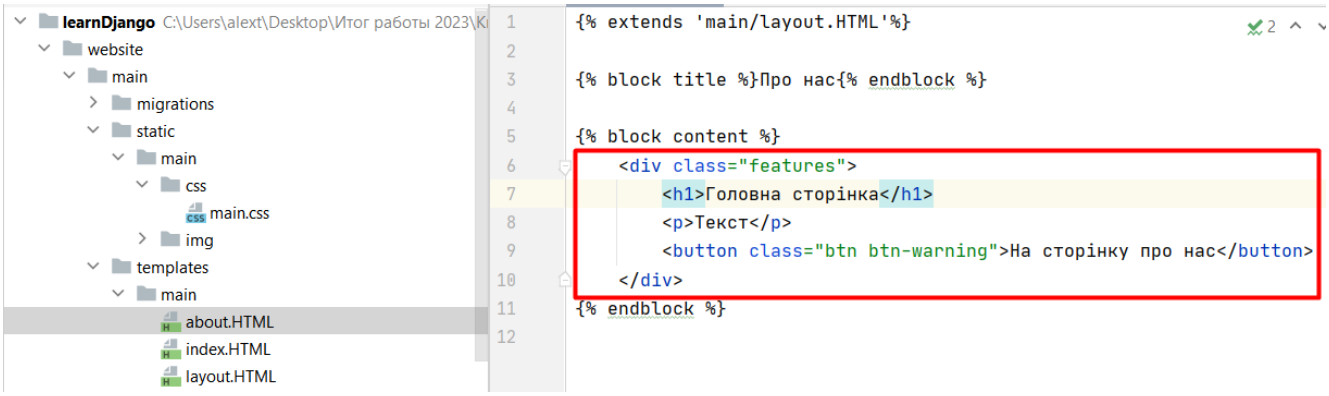


Рисунок 2.41

Рисунок 2.42

Отримаємо:

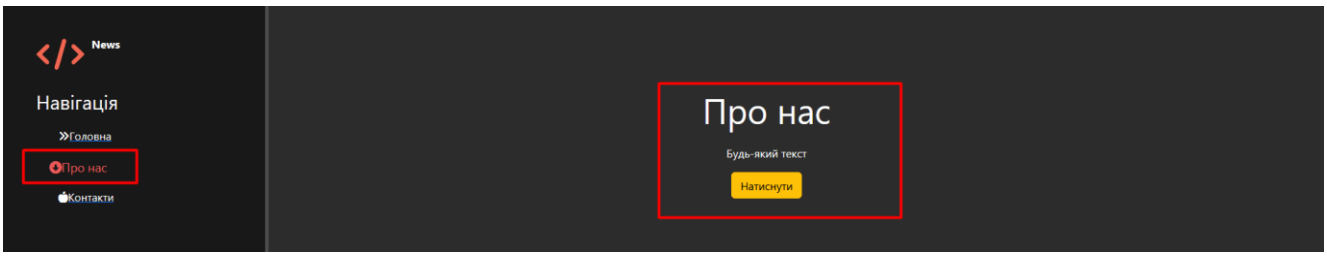


Рисунок 2.43

Передача даних у HTML-шаблон. Відкриємо файл **views.py**.

Для цього всередину функції **render** треба передати дані, які будуть представлені у вигляді словника. Ключу **title** передається фраза Головна сторінка.

Рисунок 2.44

У файлі текст (“Головна сторінка”) змінюємо на ключ **title** у фігурних дужках.

Рисунок 2.45

І цю ж змінну виводитимемо на початку сторінки.

Рисунок 2.46

Напишемо по-іншому. Створимо окремий словник. Всередині пропишемо, що нам треба.

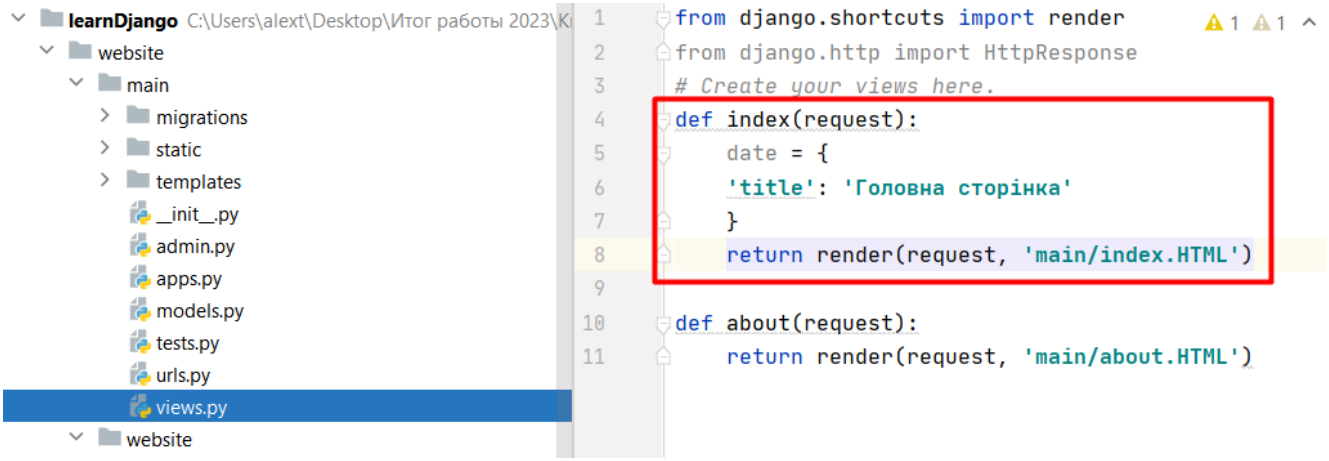


Рисунок 2.47

А замість того параметра передаватимемо раніше створений словник **date**.

Рисунок 2.48

Додамо до словника ще й список, який буде всередині шаблону.

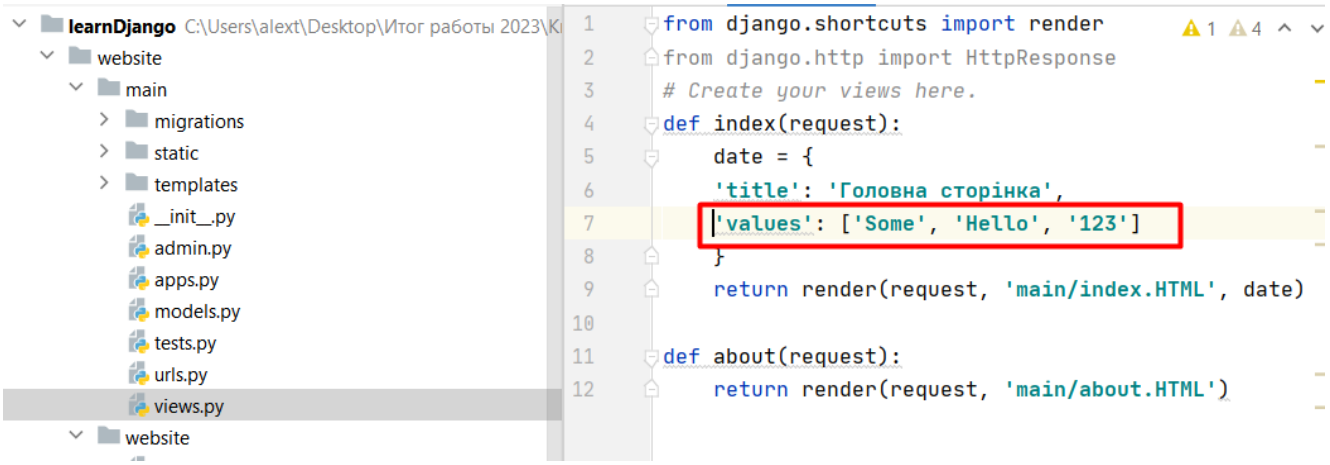


Рисунок 2.49

Щоб перебирати список та виводити кожен елемент списку на екран, треба звернутися до файлу **index.HTML**.

Для перебору списку використовується цикл зі змінною el.

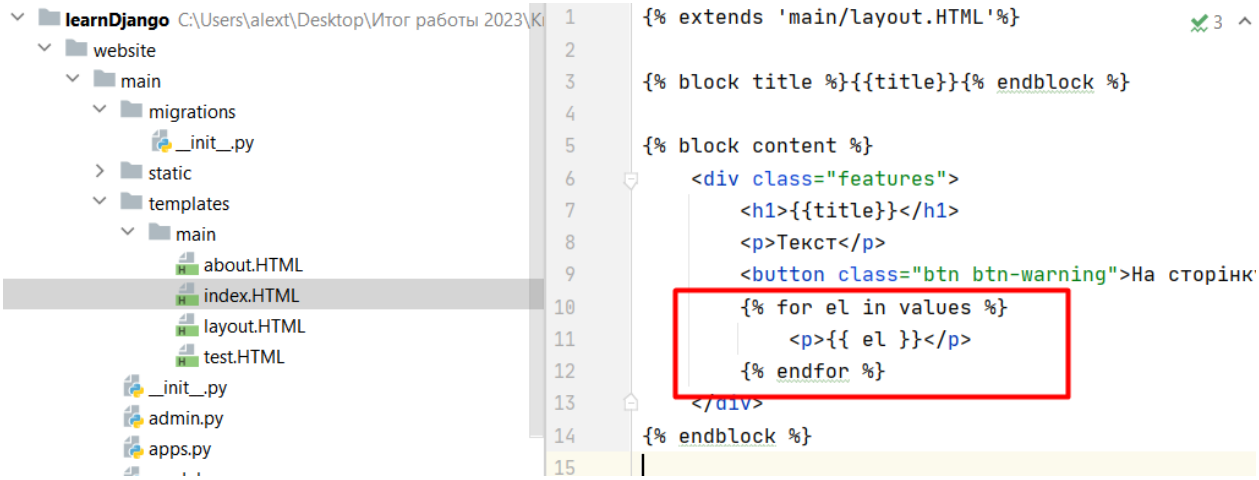


Рисунок 2.50

Отримаємо:

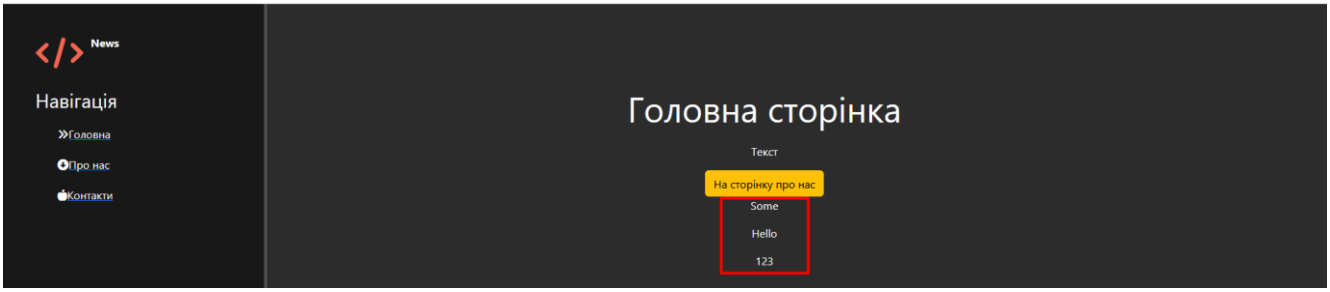


Рисунок 2.51

Всередині шаблону ми можемо передавати і словники.

Рисунок 2.52

Заходимо у файл **index.HTML** та вписуємо:

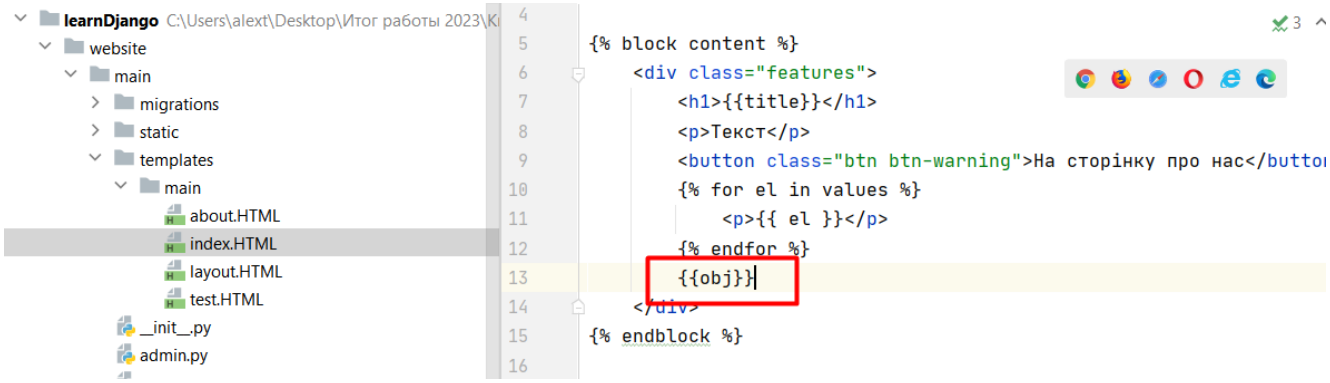


Рисунок 2.53

Запускаємо програму.

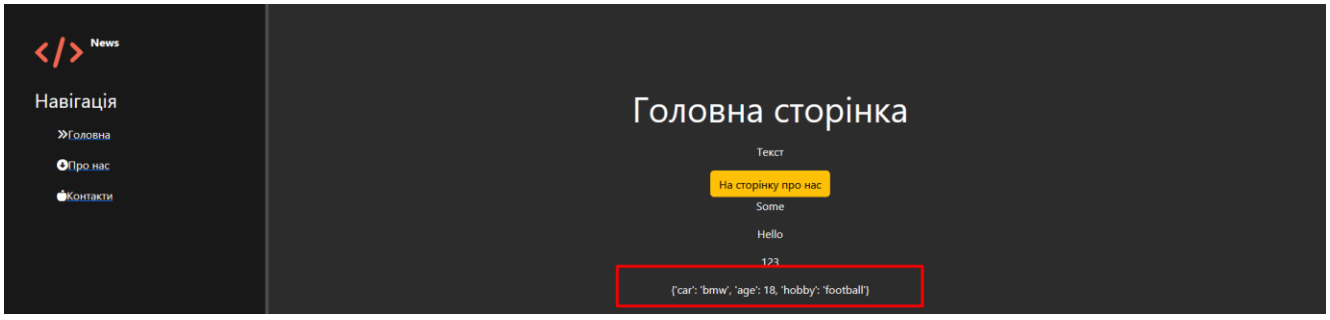


Рисунок 2.54

Якщо бажаємо вивести тільки значення 18, прописуємо:

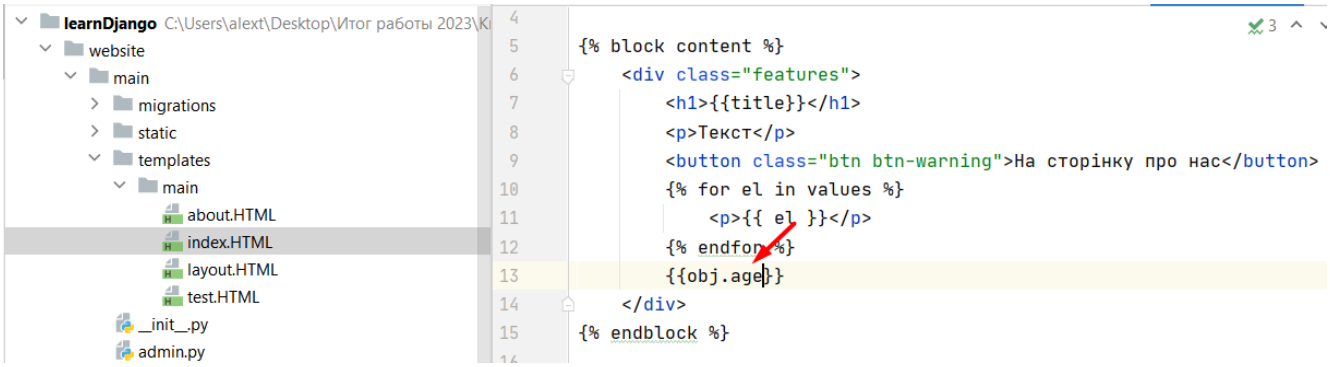


Рисунок 2.55

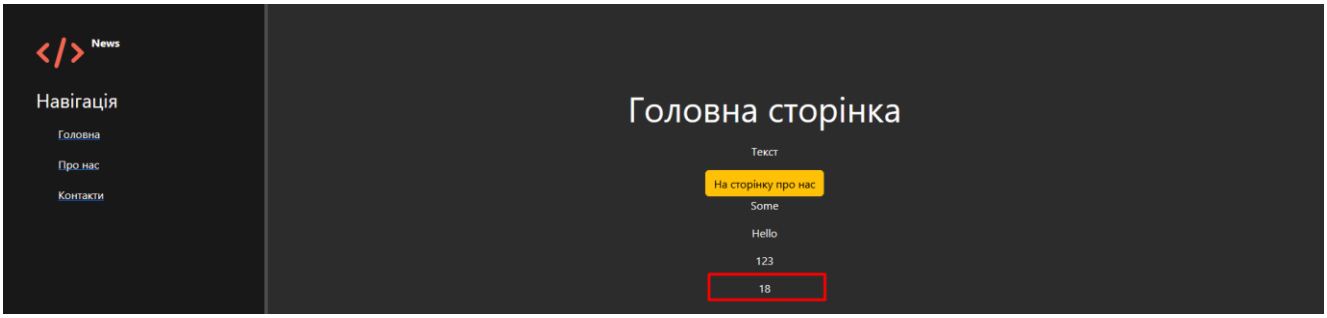


Рисунок 2.56

Для перебору по об’єкту використовуємо цикл.

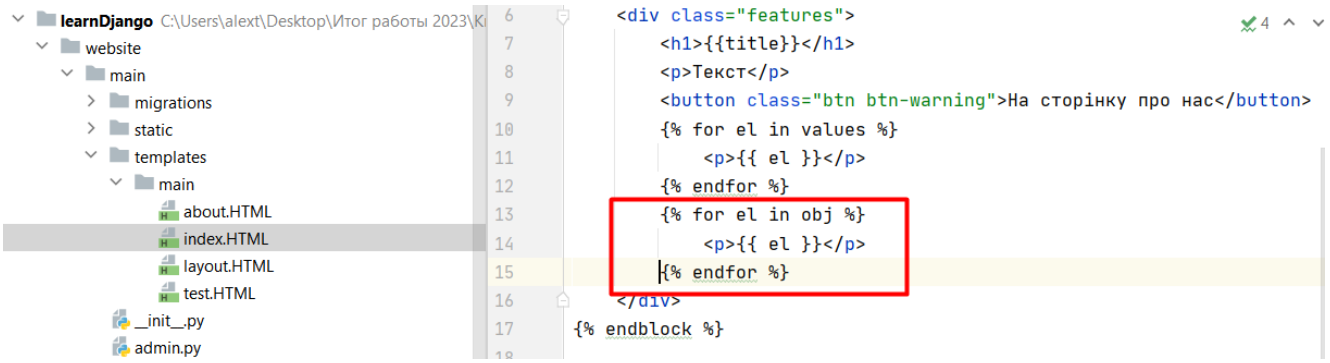


Рисунок 2.57

Результат роботи:

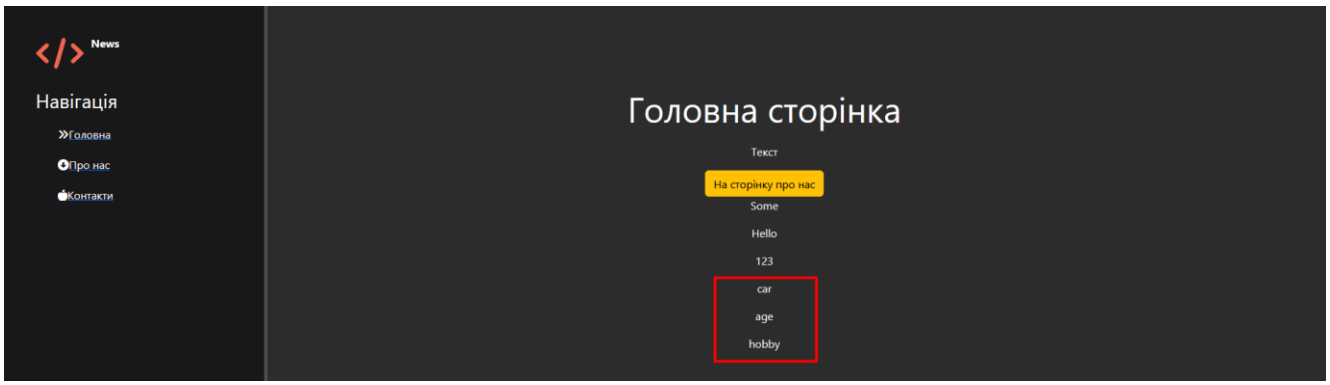


Рисунок 2.58

Крім використання циклів та умов у Django можна використовувати фільтри. Наприклад, усі літери зробити більшими або великими.

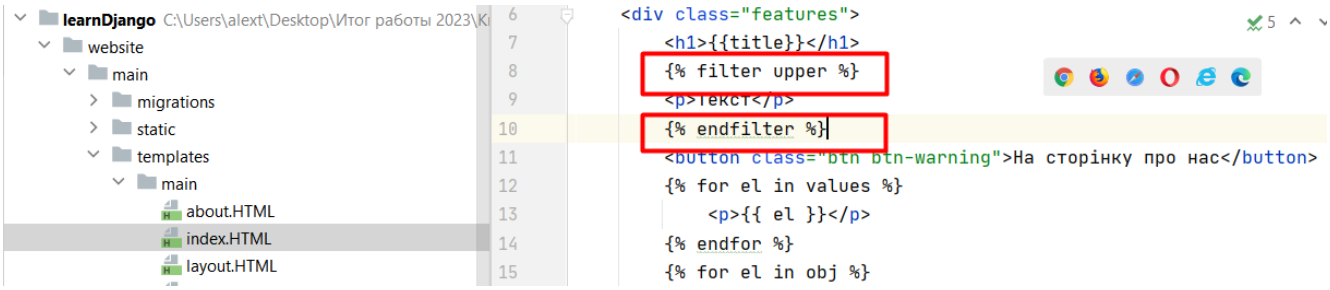


Рисунок 2.60

Результат:

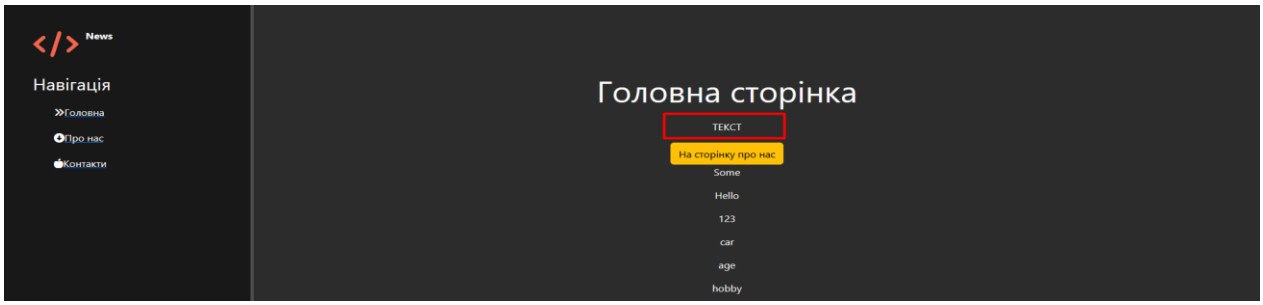


Рисунок 2.61

Практичне завдання:

1.Повторити хід роботи.

2. Отримати результат.

Практична робота №3

Тема. Фреймворк Django (Джанго). Робота з базами даних. Створення панелі адміністратора. Частина 3.

Мета роботи. Робота з базами даних

Зміст.

4. Оформлення кнопок сайту. Робота з базами даних. Створення панелі адміністратора.

5. Виконання роботи.

6. Отримання результату.

Хід роботи.

Тепер для кнопки "На сторінку про нас":

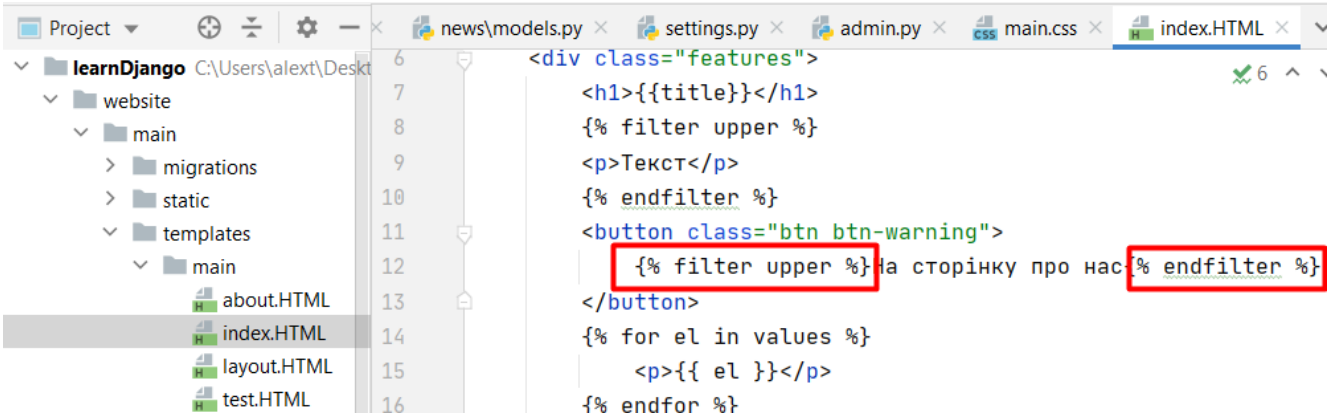


Рисунок 3.1

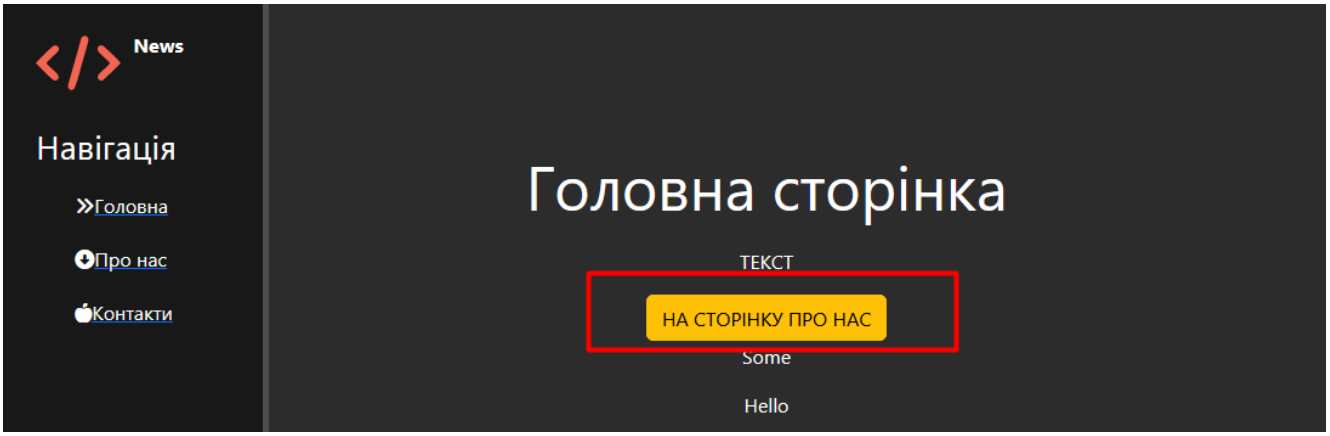


Рисунок 3.2

Тепер розпочинаємо роботу з базами даних. Створимо нову сторінку та нову таблицю у базі даних. Заповнимо таблицю різними записами.

Зараз всередині нашого проекту є один додаток **main**, який обробляє два посилання - головну сторінку і **about**.

Рисунок 3.3

Створимо окремий додаток і в цьому додатку оброблятимемо сторінки з додавання нового запису до бази даних, сторінки з виведення записів з бази даних.

Створимо новий додаток **news**. У терміналі пропишемо.

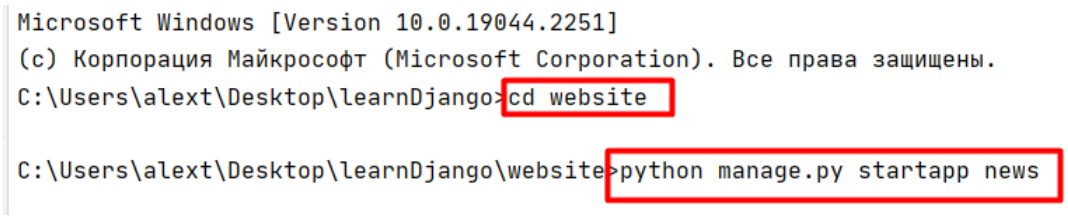


Рисунок 3.4

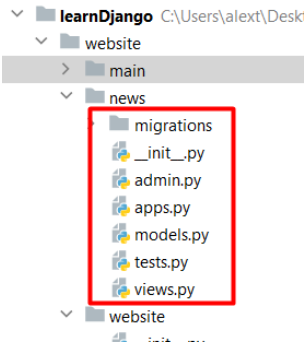


Рисунок 3.5

Зареєструємо його у файлі **settings.py**.

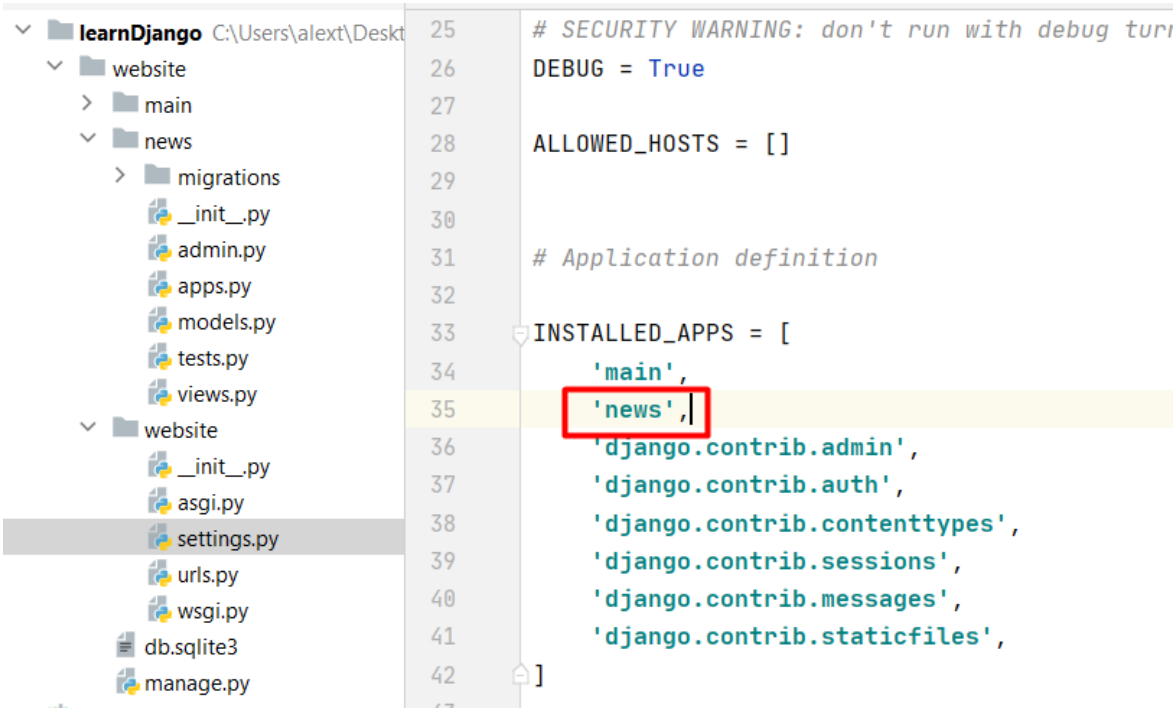


Рисунок 3.6

Додамо обробник події. Додамо його до файлу urls.py всього проекту.

Рисунок 3.7

Відкриємо новий файл у новій папці.

Рисунок 3.8

Копіюємо туди код з аналогічного файлу з минулого додатку.

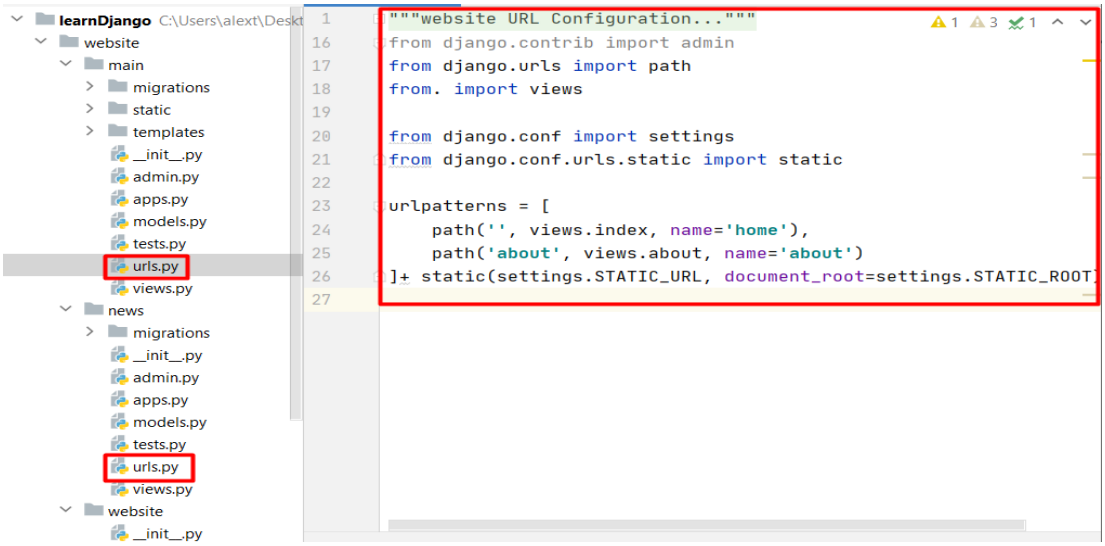


Рисунок 3.9

Всередині цього файлу нам необхідно змінити url адреси, які ми будемо відстежувати. Ми будемо викликати нову функцію **News_home** для url адреси.

Рисунок 3.10

Зайдемо у файл **views.py** нового додатка і створимо новий метод News_home з одним обов'язковим параметром request і будемо повертати якийсь HTML-шаблон. У якості першого параметру вкажемо request, а як другий параметр шлях до папки.

Рисунок 3.11

Запустимо локальний сервер. Для цього в терміналі напишемо runserver.

```
C:\Users\alext\Desktop\learnDjangoUA\website>python manage.py runserver
```

Отримаємо:

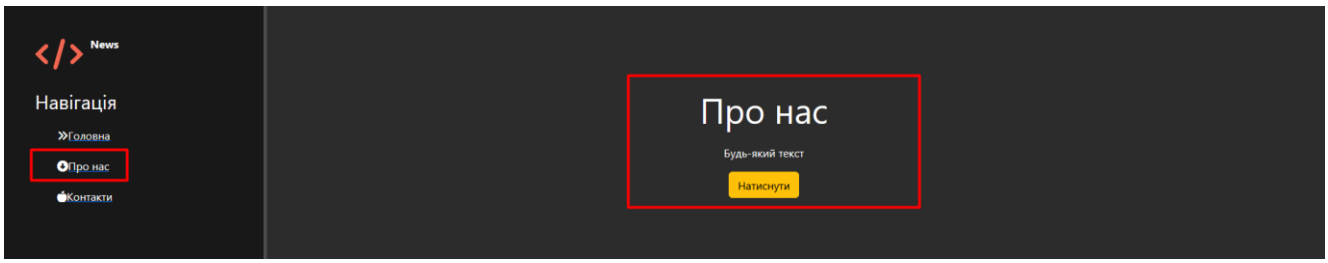


Рисунок 3.12

І ми перейшли на сторінку по нас.

Шаблон будемо зберігати у новому додатку. Для цього папці **News** створимо папку **templates**. В середині цієї папки створимо ще одну папку **news** та файл.

Рисунок 3.13

Копіюємо із файлу **index.HTML** у наш файл весь вміст.

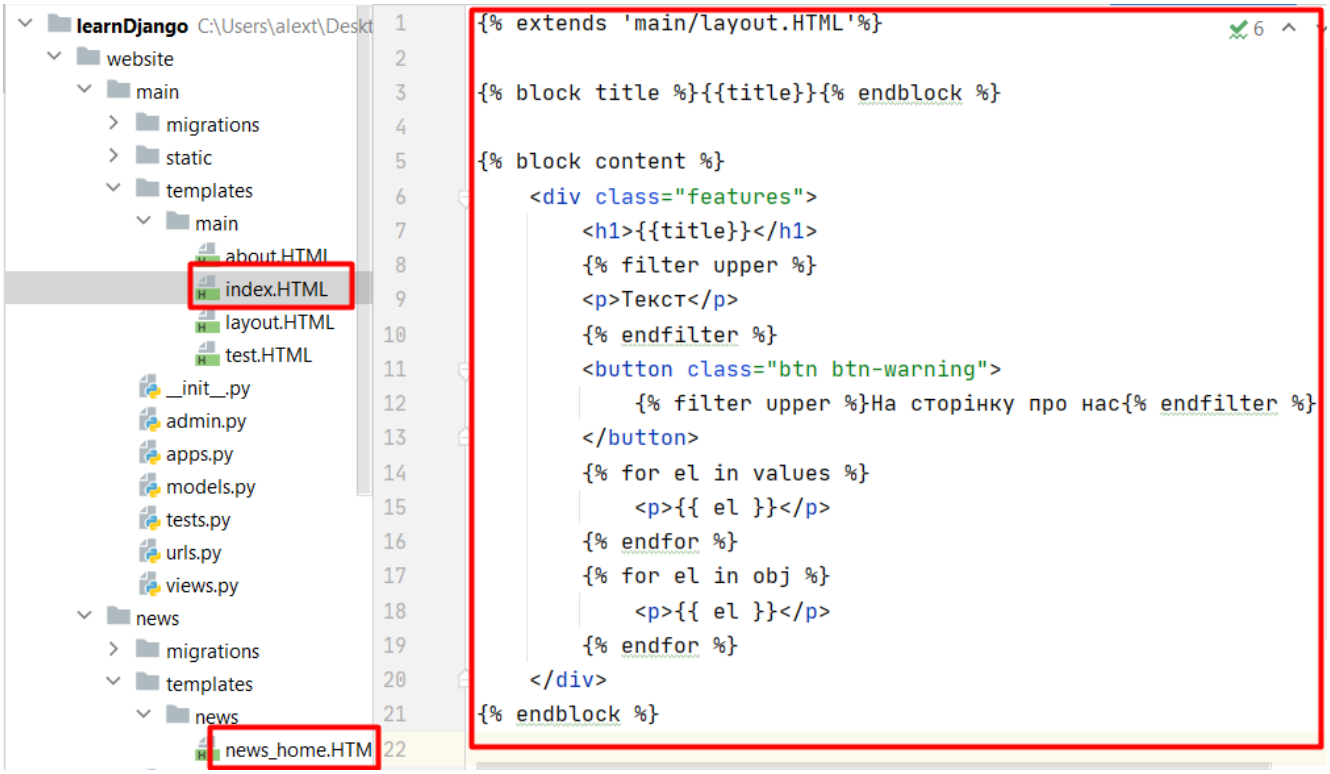


Рисунок 3.14

Змінимо вміст файлу. Виводитимемо наступний заголовок – “Новини на сайті”.

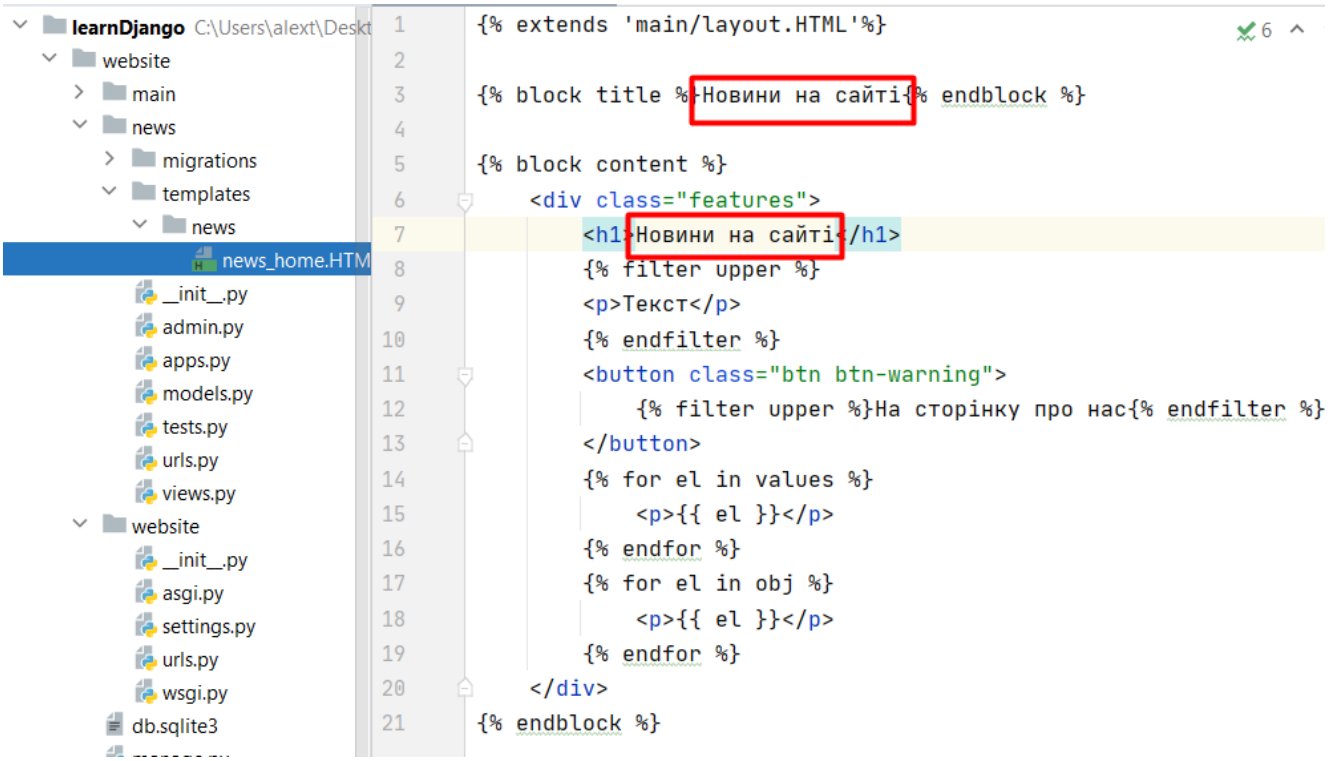


Рисунок 3.15

Отримаємо:

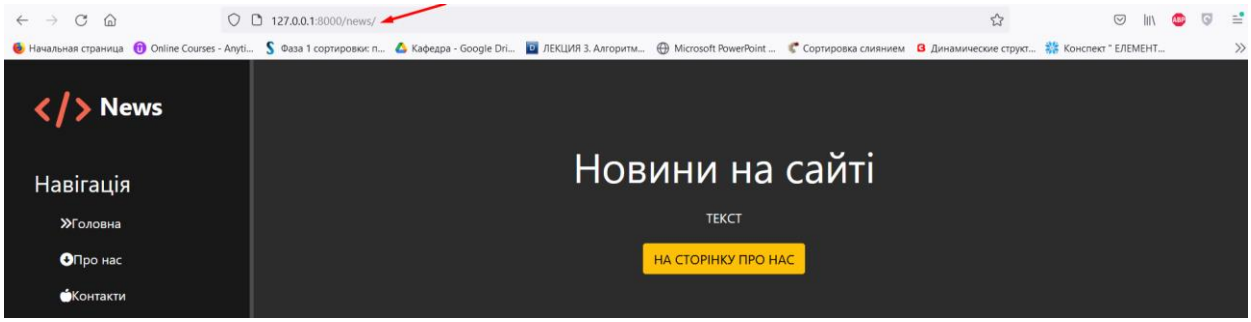


Рисунок 3.16

Зробимо кнопку для новин.

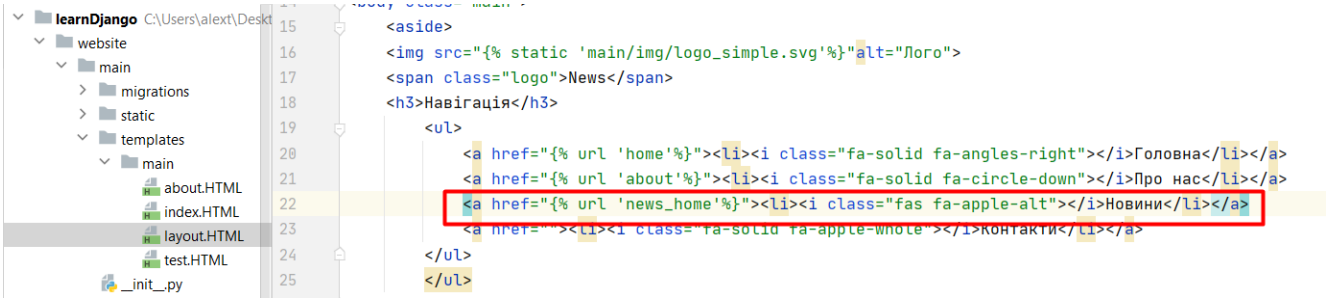


Рисунок 3.17

Попрацюємо із базами даних. Всередині таблиці ми будемо зберігати всі наші записи новин. Потім з таблиці ми братимемо і відображатимемо на сторінці новини.

Створимо таблицю усередині бази даних.

Відкриємо файл **models.py** у папці **news**.

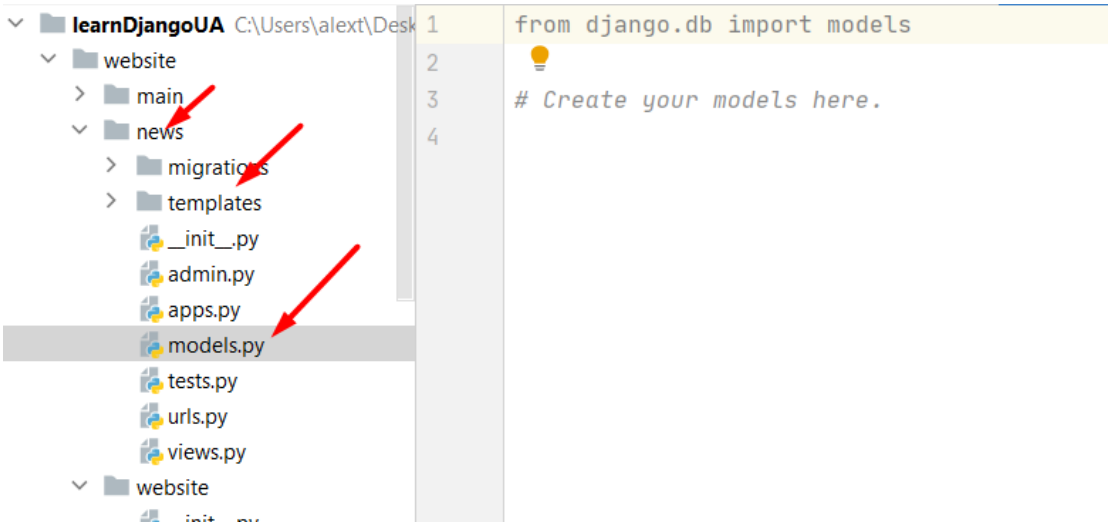


Рисунок 3.17

Щоб створити таблицю в базі даних, потрібно створити класи.

Клас Articles буде успадковуватися від класу Model. Пропишемо поля всередині таблиці. Буде 4 поля: назва, анонс, повний текст та дата. Перше поле “Назва” з типом даних Char (до 200 символів тексту), в якому буде встановлена максимальна довжина рядку у 50 символів.

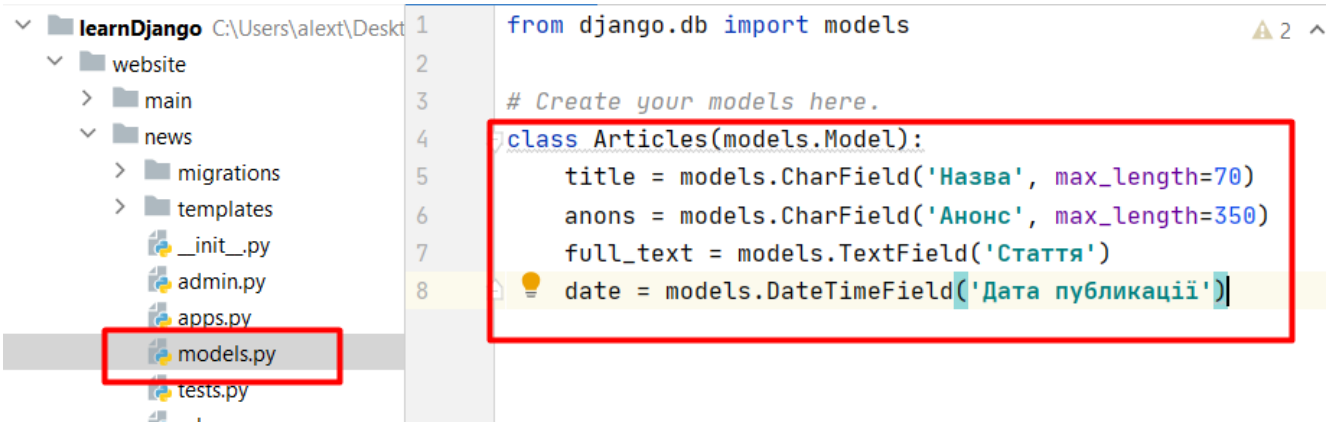


Рисунок 3.18

Створимо метод, який повертатиме назву цього об'єкту.

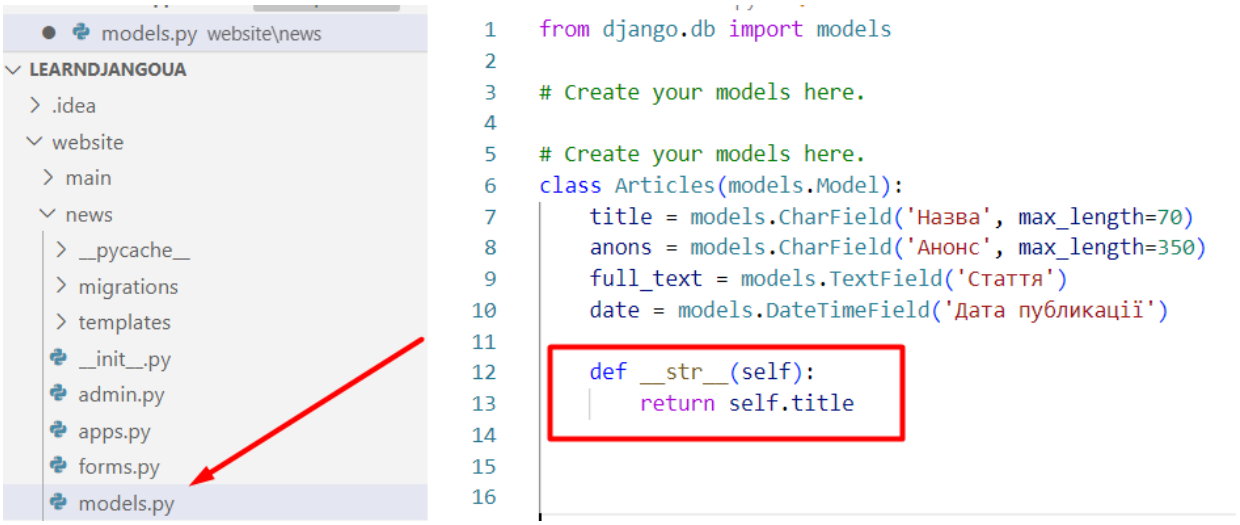


Рисунок 3.19

Синхронізуємо наш проект з базою даних.

У терміналі напишемо команду python manage.py makemigrations, перебуваючи в директорії, що містить файл **manage.py**.

```
Microsoft Windows [Version 10.0.19044.2251]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\alext\Desktop\learnDjango>cd website

C:\Users\alext\Desktop\learnDjango\website>python manage.py makemigrations
```

Рисунок 3.20

Отримали новий файл - файл міграції. Він визначає таблицю у базі даних.

News - migration - init.py.

Рисунок 3.21

Тут буде створено таблицю, назвою Articles. І в ній буде п'ять полів. Поле **"id"** завжди створюється – це порядковий номер полів.

Проведемо міграцію у терміналі. Введемо наступну команду:

```
C:\Users\alext\Desktop\learnDjango>cd website
C:\Users\alext\Desktop\learnDjango\website>python manage.py makemigrations
Migrations for 'news':
  news\migrations\0001_initial.py
    - Create model Articles
C:\Users\alext\Desktop\learnDjango\website>python manage.py migrate
```

Рисунок 3.22

Результат роботи

```
C:\Users\alext\Desktop\learnDjangoUA\website>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, news, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
```

Рисунок 3.23

Запускаемо сервер

```
C:\Users\alext\Desktop\learnDjangoUA\website>python manage.py runserver
```

Переходимо по адресу

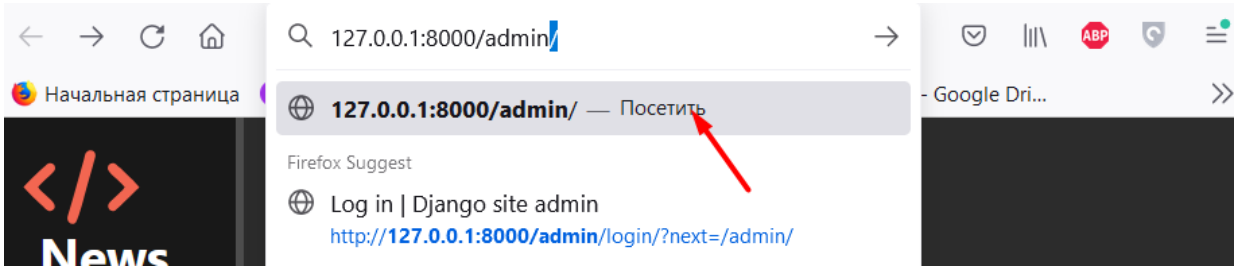


Рисунок 3.24

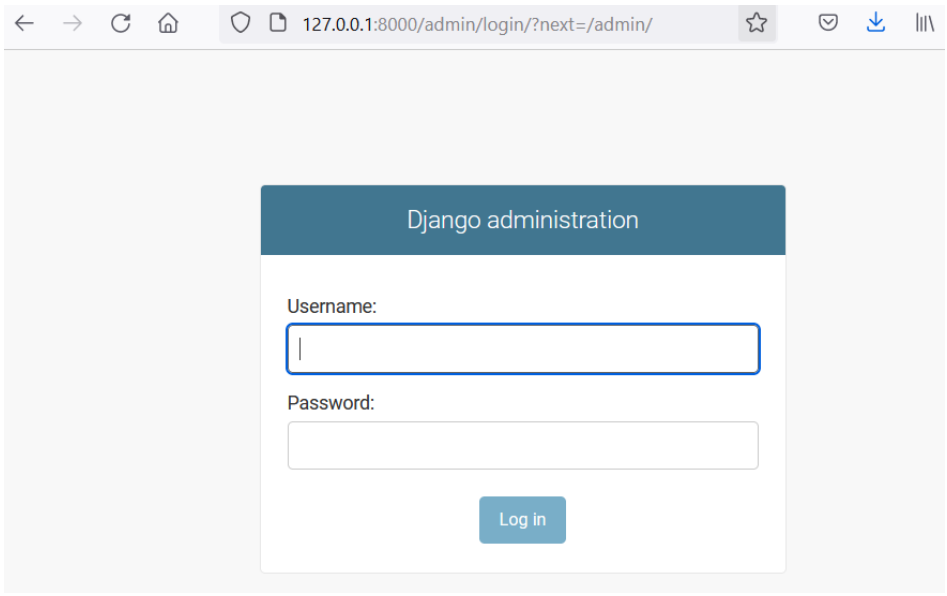


Рисунок 3.25

Перекладаємо панель адміністратора на українську. Знаходимо папку.

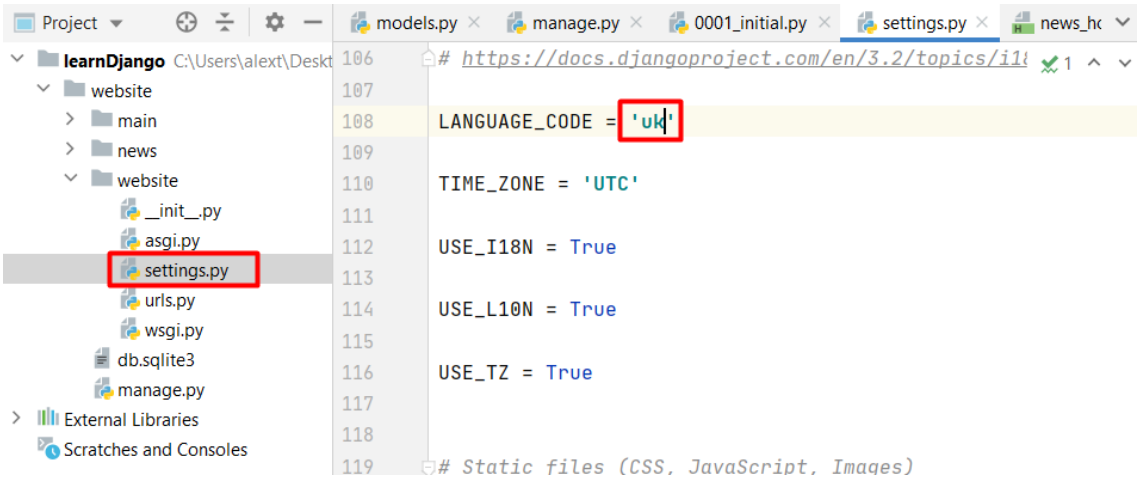


Рисунок 3.26

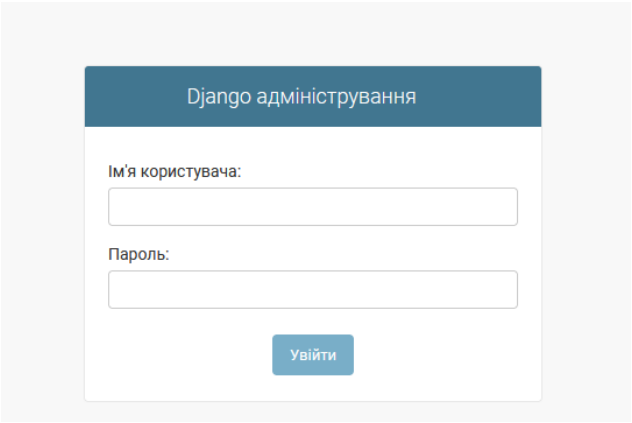


Рисунок 3.27

Зареєструємося, щоб можна було заходити в панель адміністратора.

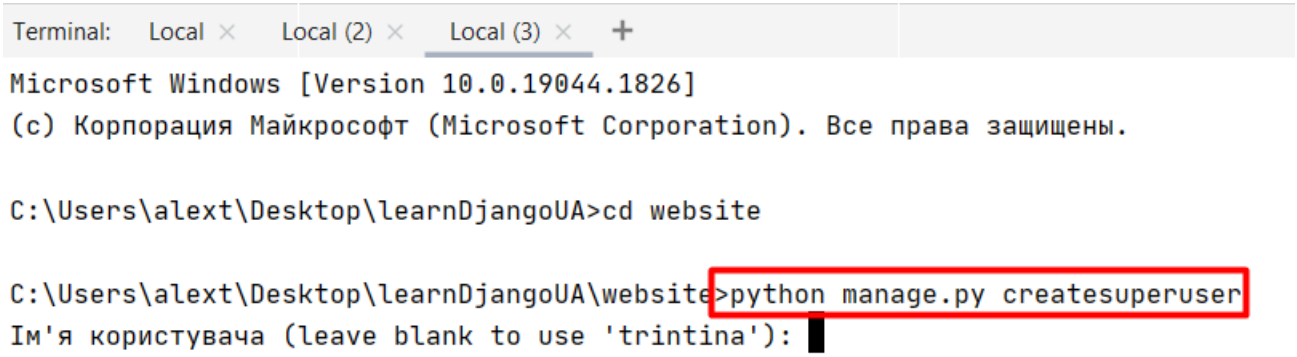


Рисунок 3.28

```
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

Пароль оберемо для простоти 12345, ім'я користувача - admin:

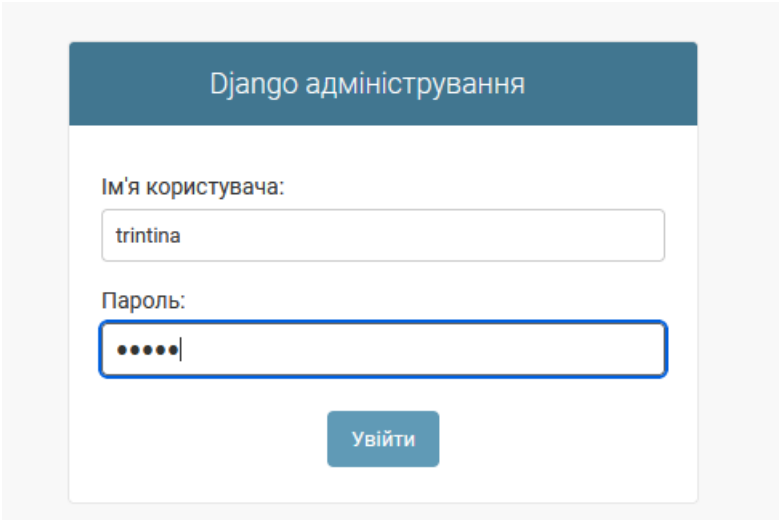


Рисунок 3.29

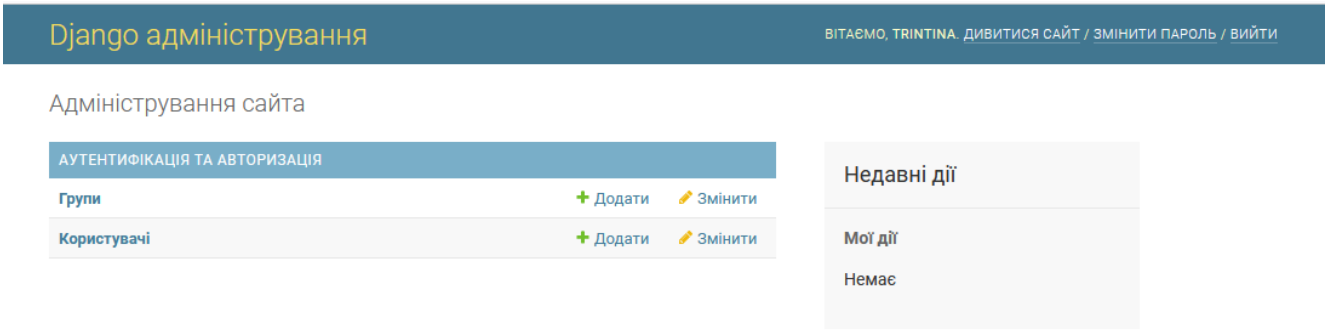


Рисунок 3.30

Тепер можна додавати та видаляти користувачів та дописувати їх дані.

Але це треба також зареєструвати. Заходимо в папку **news**, а потім в **admin.py.**

З models імпортуємо articles і прописуємо реєстрацію моделі в панелі адміністратора.

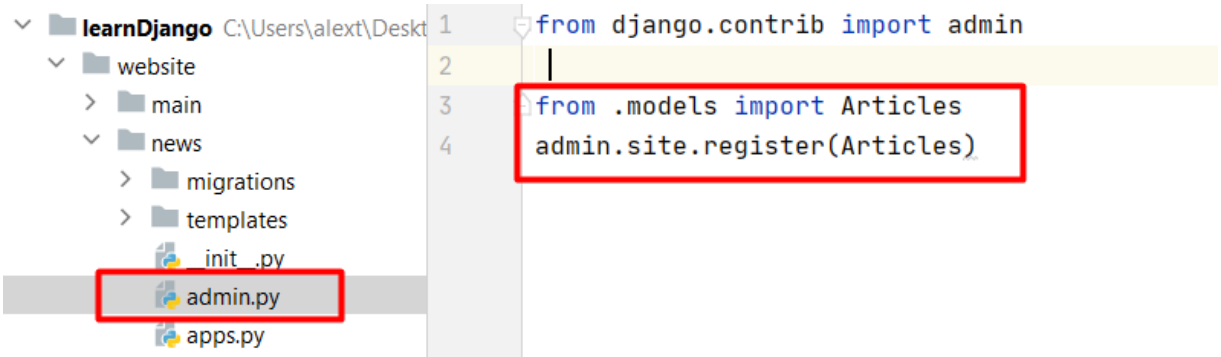


Рисунок 3.31

Оновлюємо панель адміністратора.

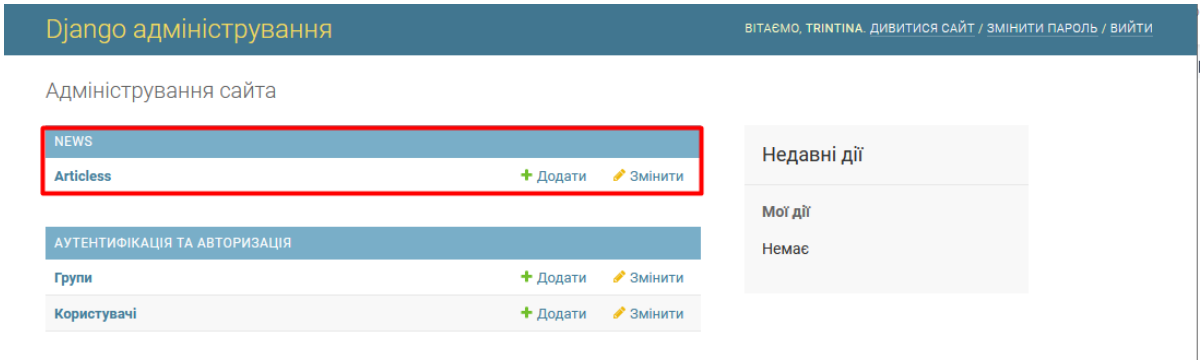


Рисунок 3.32

Перейменуємо панель. Для цього входимо до файлу **models.py** та дописуємо вкладений клас Meta та вказуємо назву для таблиці. У нас буде назва “Новина” або “Новини” для множини.

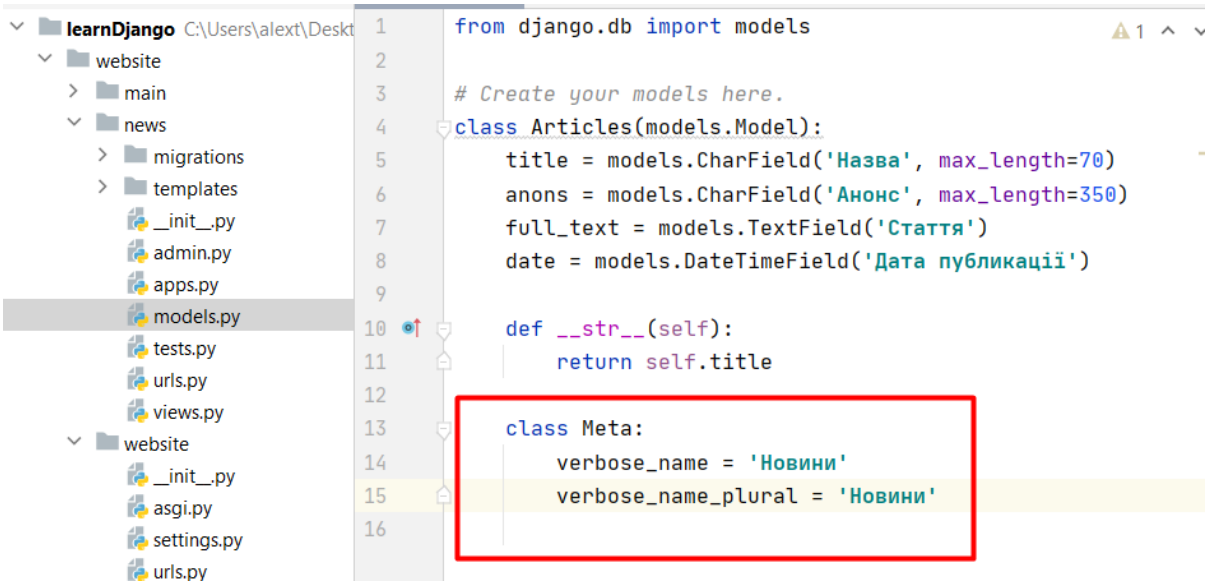


Рисунок 3.33

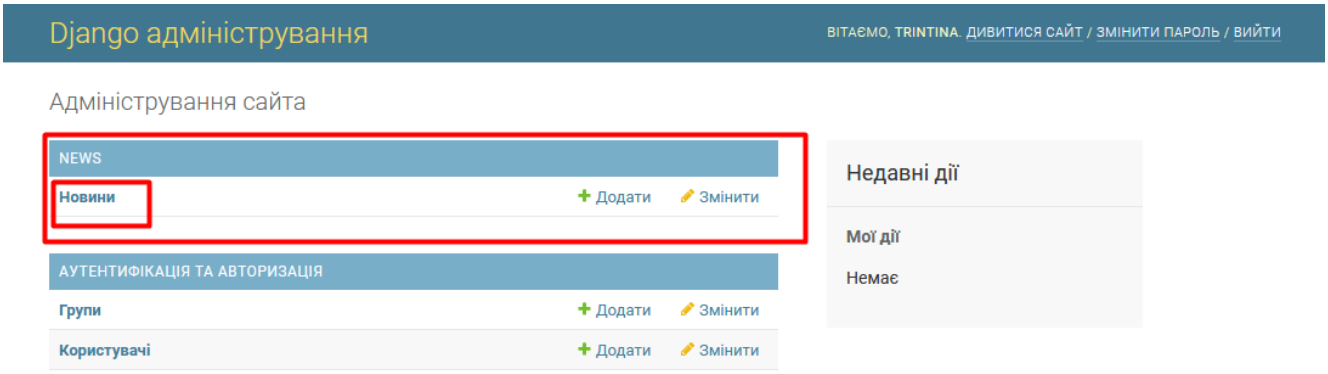


Рисунок 3.34

Тепер додамо новину. Натиснемо на таблицю та впишемо.

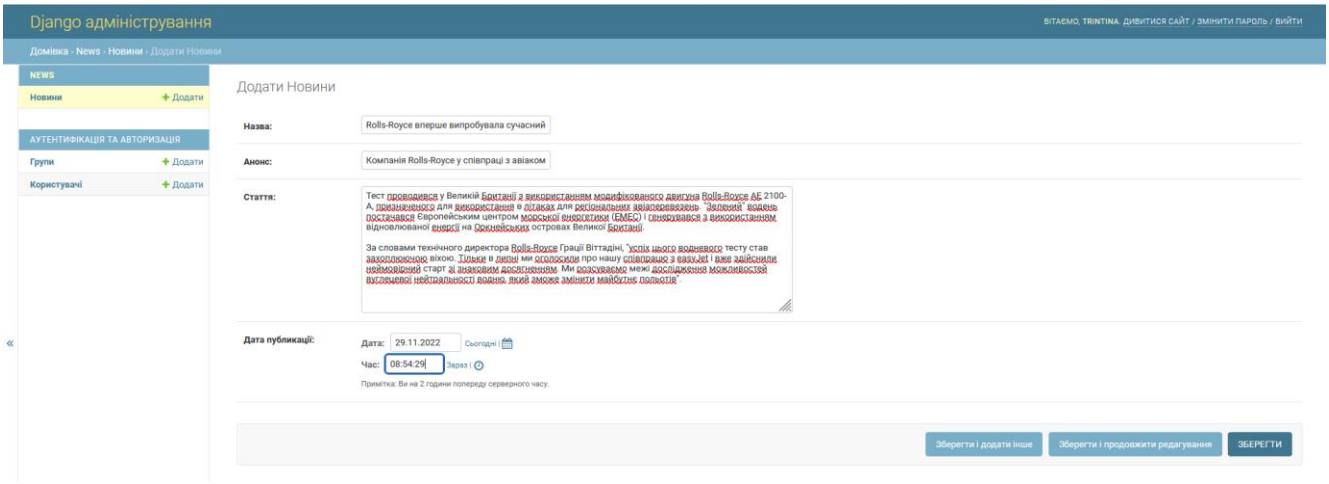


Рисунок 3.35

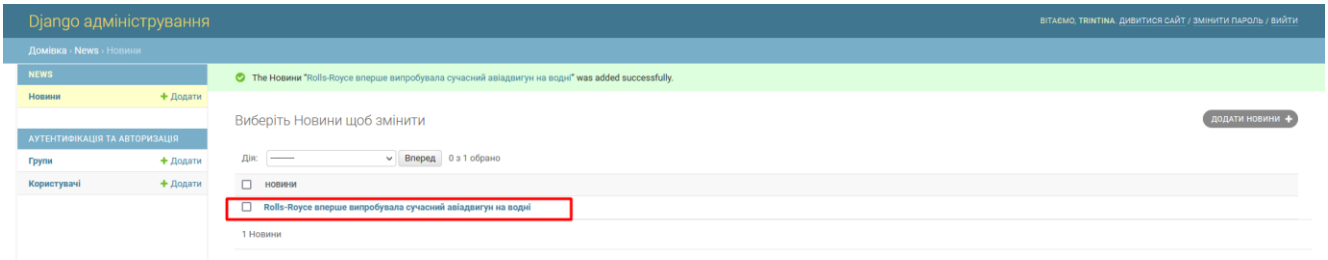


Рисунок 3.36

Спробуємо отримати дані з таблиць баз даних та відобразити їх. Відкриємо файл news - views.py. Імпортуємо ту модель з файлу models, з якою працюватимемо, а саме клас Articles.

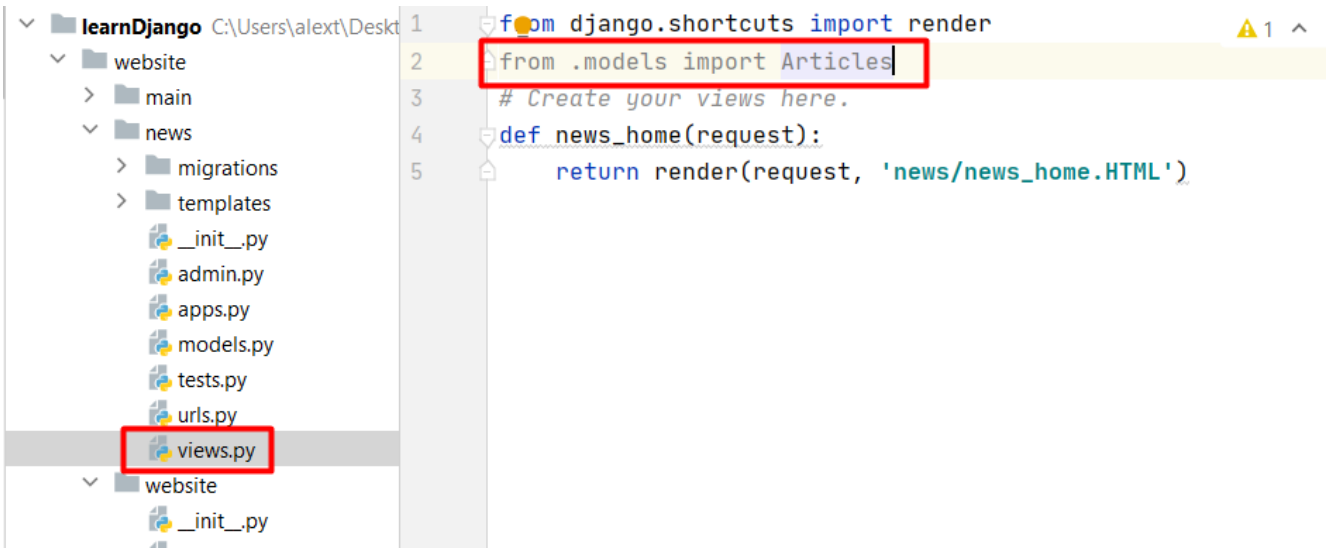


Рисунок 3.37

Створюємо об'єкт news і звертаємось до класу Articles та до всіх об'єктів.

І передаємо як третій параметр.

Рисунок 3.38

Переходимо до шаблону та прописуємо звернення до news:

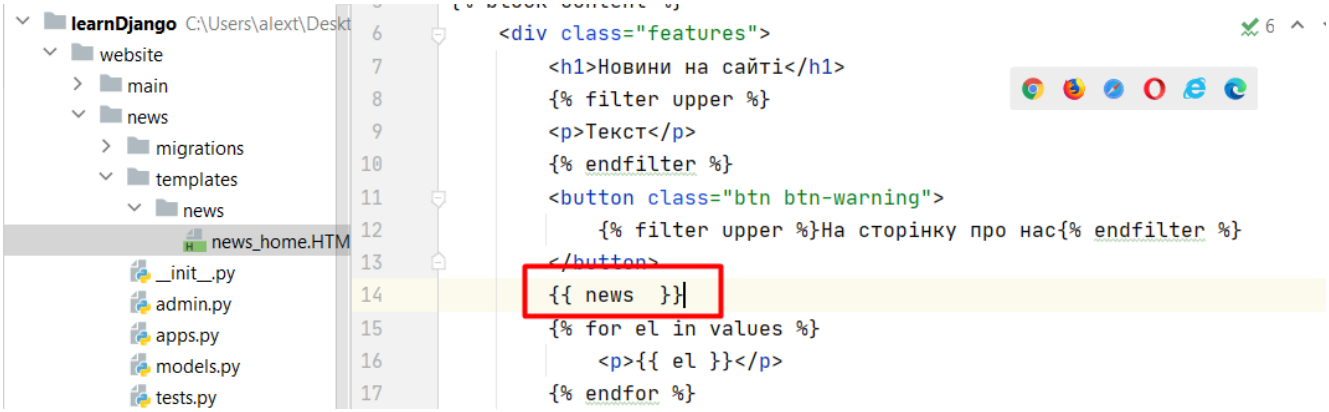


Рисунок 3.39

Чотири новини наводяться за назвою:

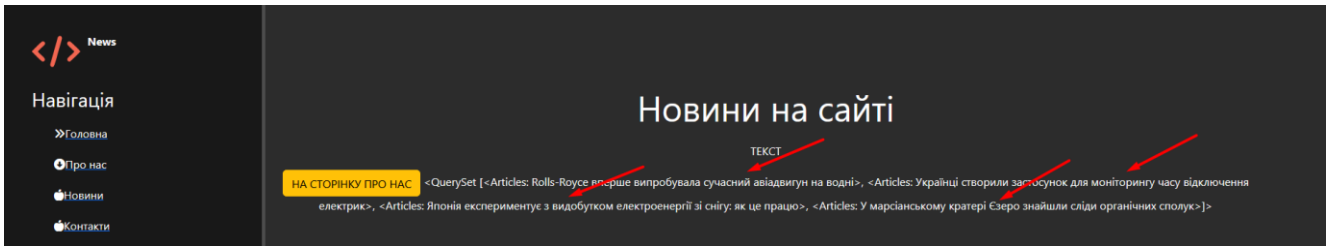


Рисунок 3.40

Новини виглядають не презентабельно, тому виведемо їх через цикл і додатково виведемо анонс статті.

Рисунок 3.41

Пропишемо для краси у шаблоні нові стилі для блоку alert-warning.

Рисунок 3.42

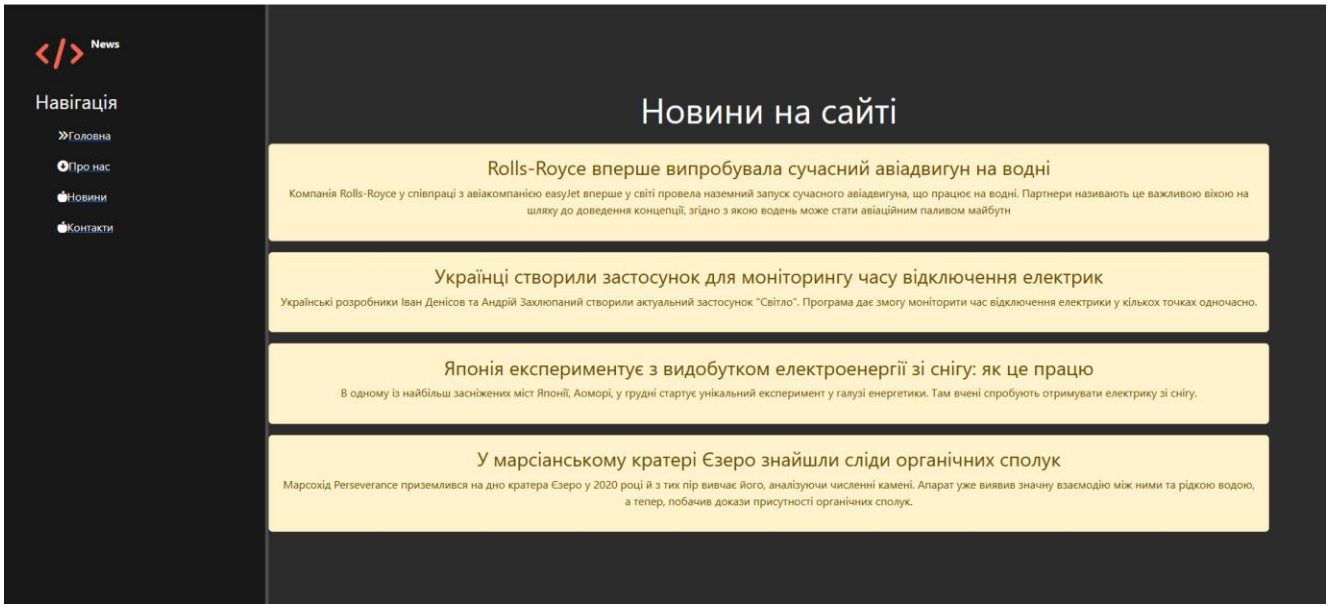


Рисунок 3.43

Можна отримувати новини відсортованими (наприклад за датою). Для цього входимо у **views.py** та прописуємо date.

Рисунок 3.44

Метод order_by() повертає масив об’єктів. Ми можемо обрати лише частину з цих об’єктів. Наприклад, оберемо перші два елементи масиву.

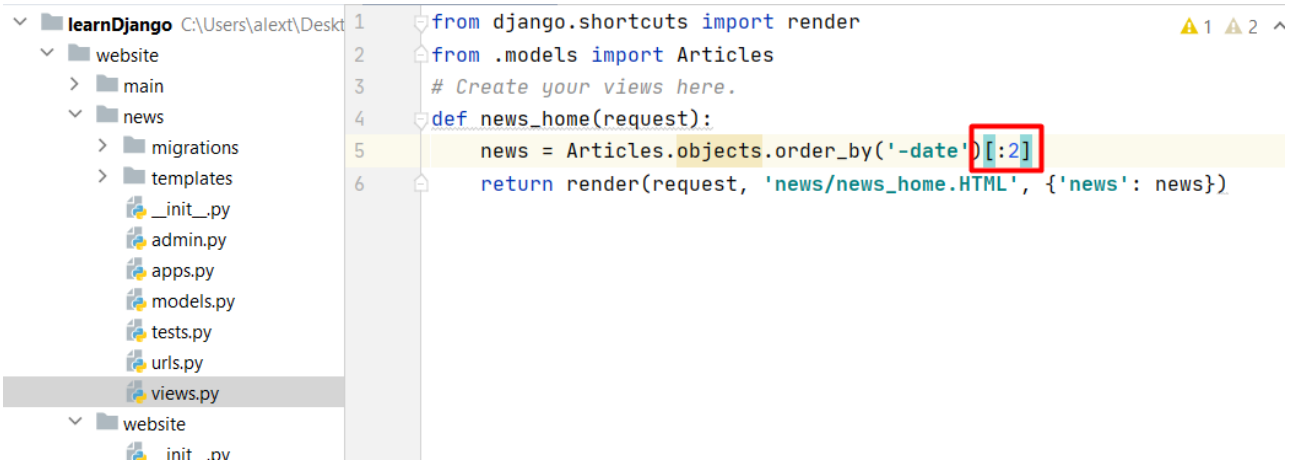


Рисунок 3.45

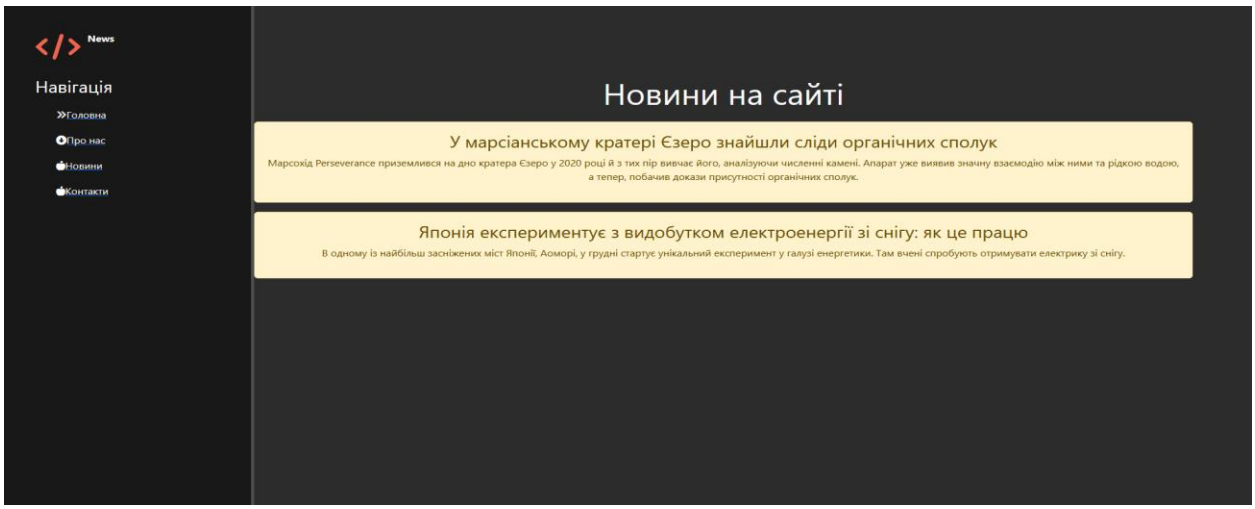


Рисунок 3.46

Практичне завдання:

1. Повторити хід роботи.
2. Отримати результат.

Практична робота №4

Тема. Фреймворк Django (Джанго). Налагодження роботи панелі адміністратора. Частина 4.

Мета роботи. Налагодження роботи панелі адміністратора.

Зміст.

1. Налагодження роботи панелі адміністратора.
2. Виконання роботи.
3. Отримання результату.

Хід роботи

Заходимо у файл і додамо функцію, яка виконує наступну задачу: якщо немає жодних записів, то з'являтиметься текст – “У вас немає записів”.

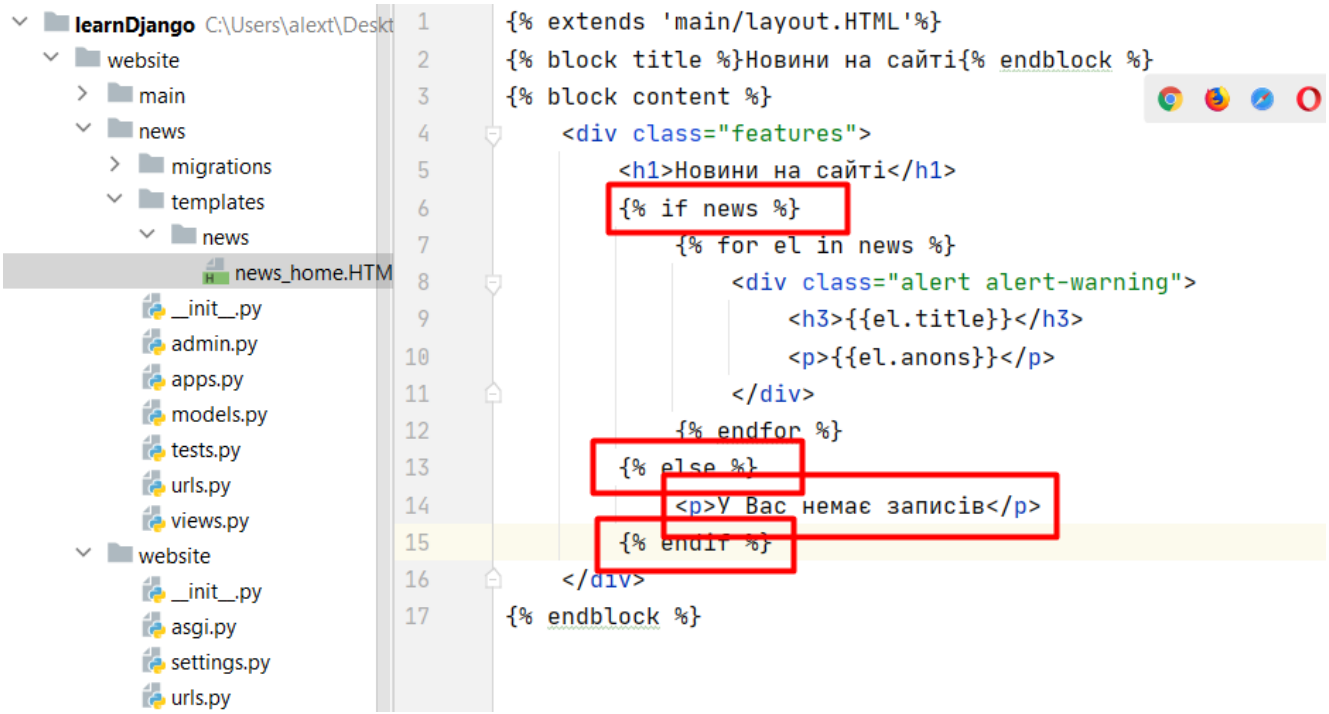


Рисунок 4.1

Створимо окрему сторінку з формою додавання нових записів.

Створимо посилання.

Додаємо посилання.

Запускаємо програму.

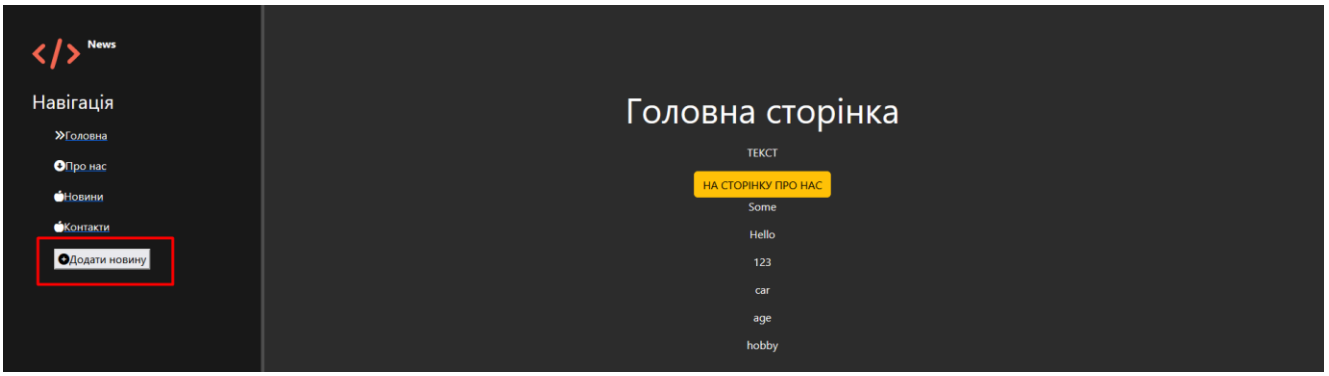


Рисунок 4.2

Додамо іменоване посилання:

Рисунок 4.3

Створювати посилання потрібно в додатку **news**.

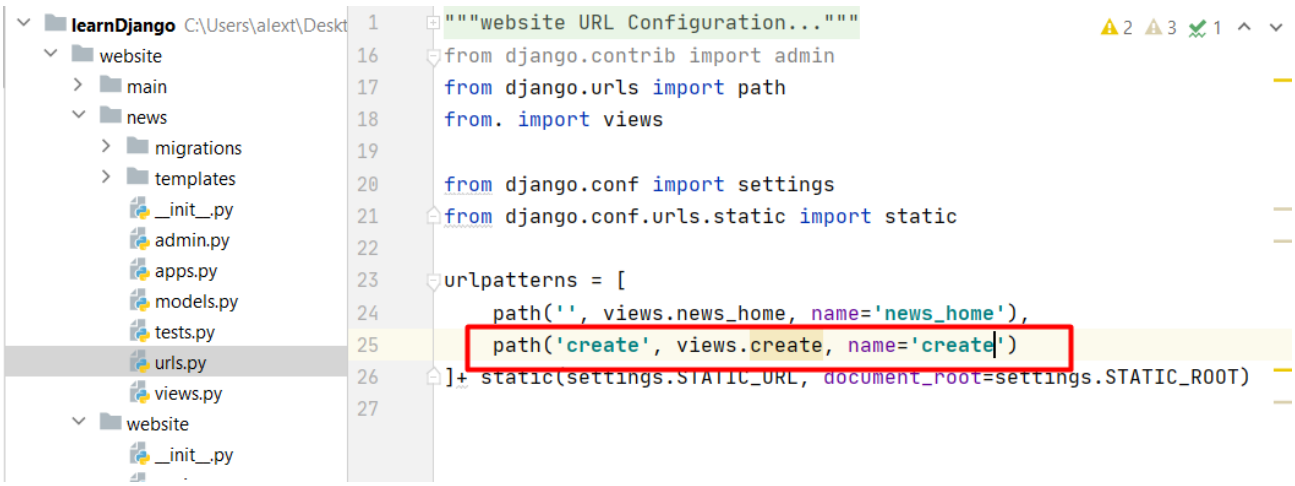


Рисунок 4.4

Переходимо у файл і створимо функцію create з параметром request.

Рисунок 4.5

Створимо шаблон:

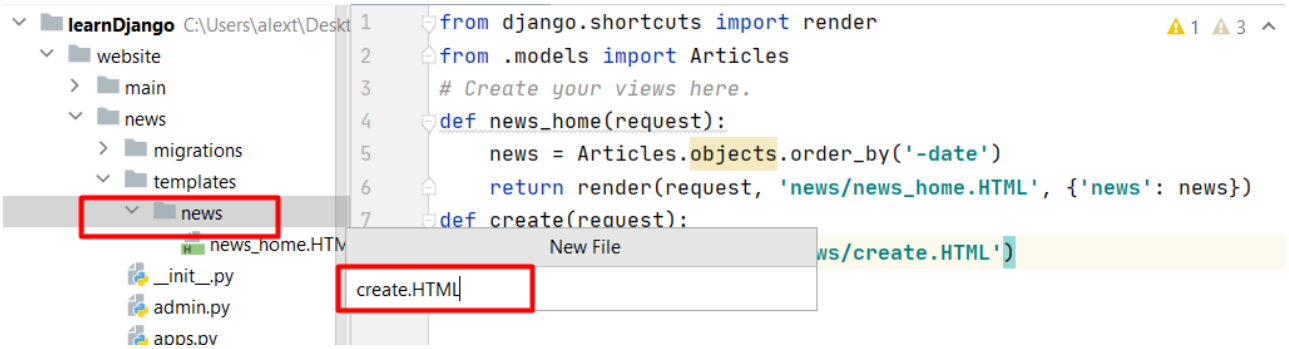


Рисунок 4.6

Вписуємо шаблон, як у попередньому випадку, змінюємо лише текст.

Додаємо тег form з методом передачі даних post.

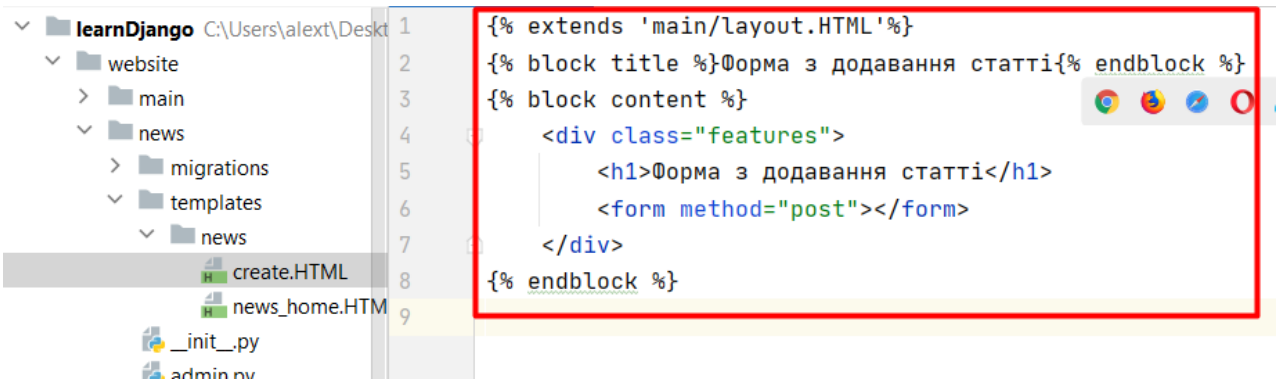


Рисунок 4.7

Пропишемо поля для назви статті та клас для зовнішнього вигляду.

Рисунок 4.8

Отримаємо:

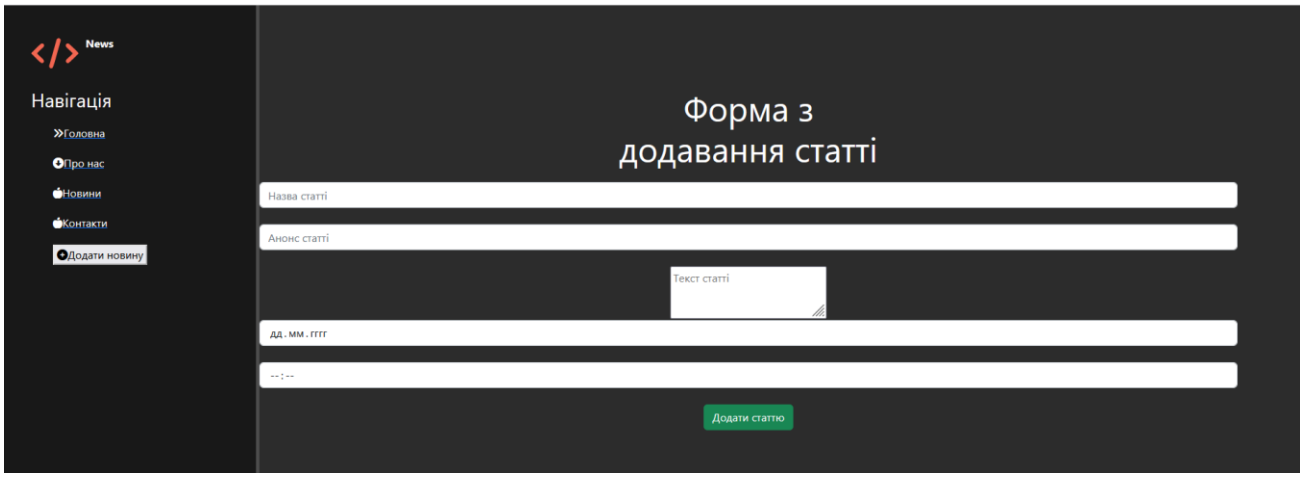


Рисунок 4.9

Допишемо:

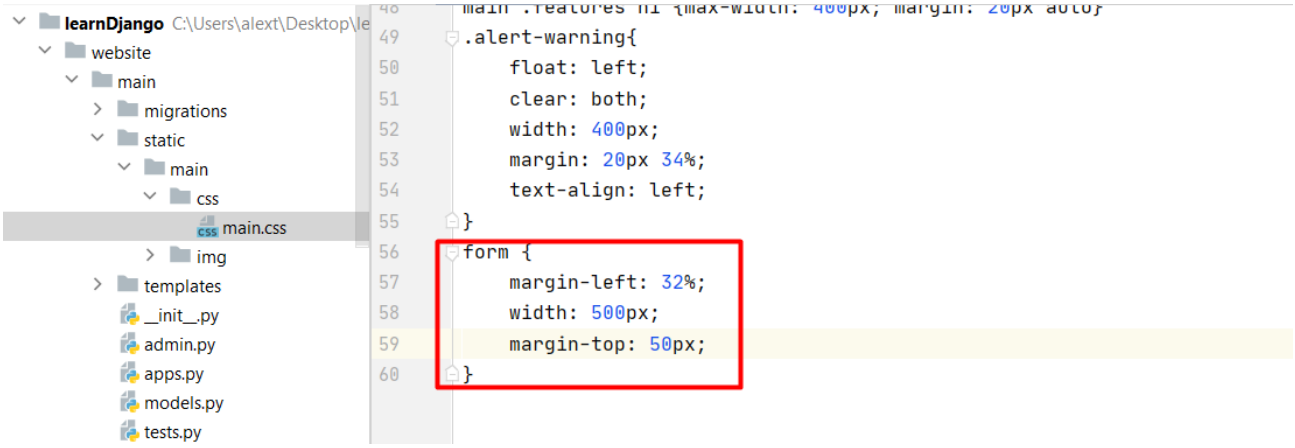


Рисунок 4.10

Отримаємо:

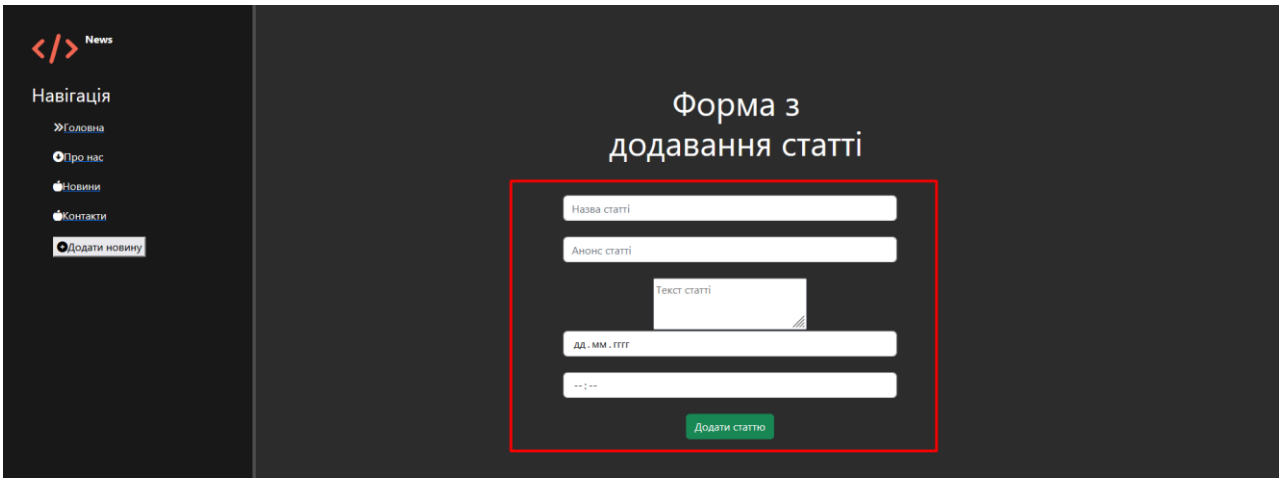


Рисунок 4.11

Додамо функціональну частину до цієї форми. Для цього підключаємо csrf token.

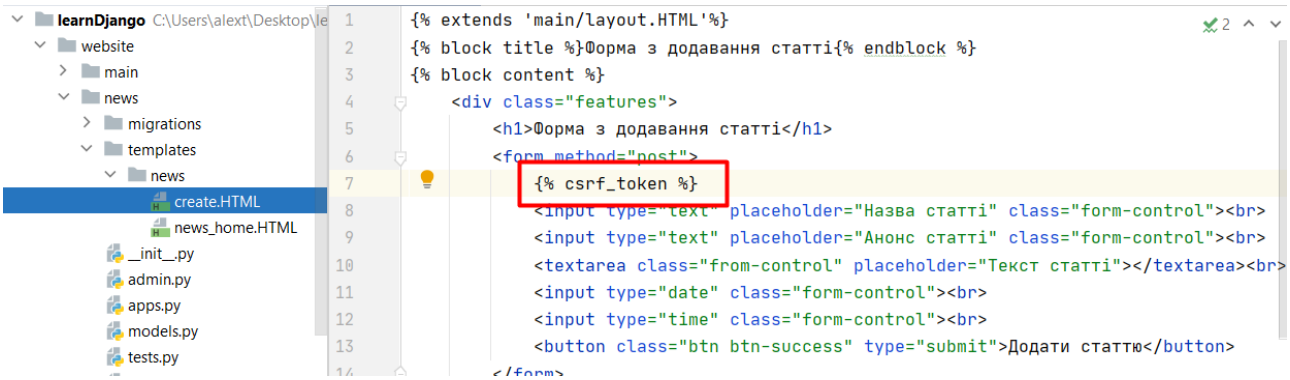


Рисунок 4.12

Зв'яжемо наш новий шаблон з таблицею Articles. Для цього створимо новий файл **forms.py**.

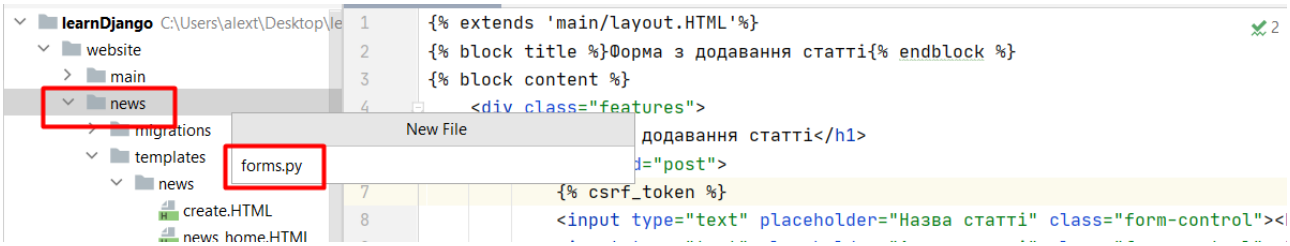


Рисунок 4.13

Імпортуємо туди Articles. А також із папки jango.forms імпортуємо клас ModelForm.

Створюємо ArticlesForm, який все успадковує від ModelForm. Усередині створюємо клас Meta. І пишемо його характеристики. Ми працюємо з моделлю Articles. Враховуємо поля. Виводитимемо назву, анонс, текст та дату.

Рисунок 4.14

Тепер створимо об'єкт на ім'я класу Articles. Передамо його до шаблону.

Відкриваємо views.py і імпортуємо потрібний клас ArticlesForm. Створюємо об'єкт на основі цього класу **def create(request)**, створюємо словник data та його прописуємо **return render(request, 'news/create.HTML', data) .**

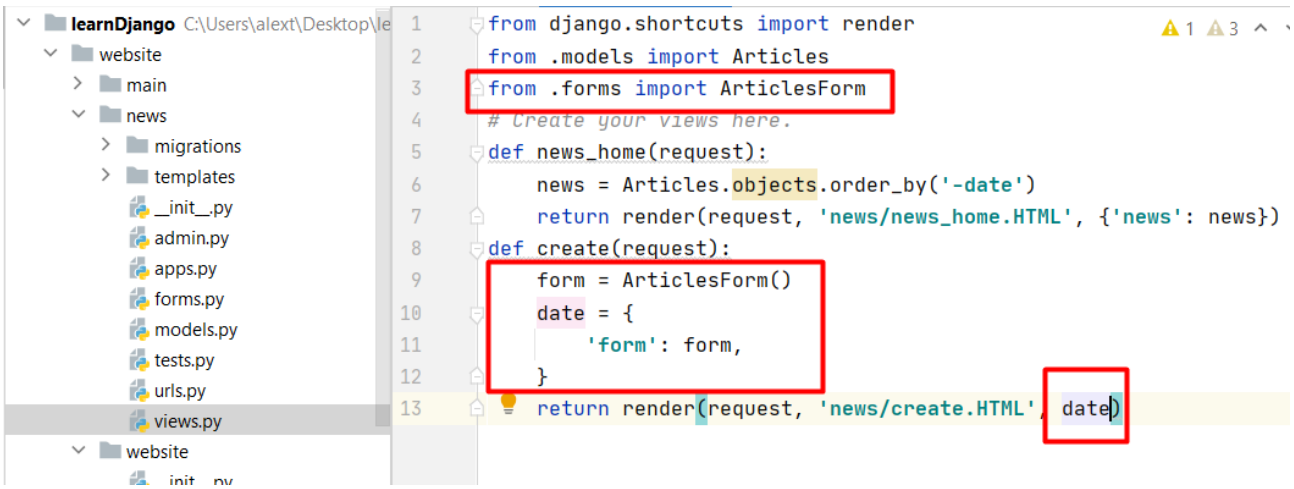


Рисунок 4.15

Переходимо у create.HTML.

Виведемо весь об'єкт form.

Рисунок 4.16

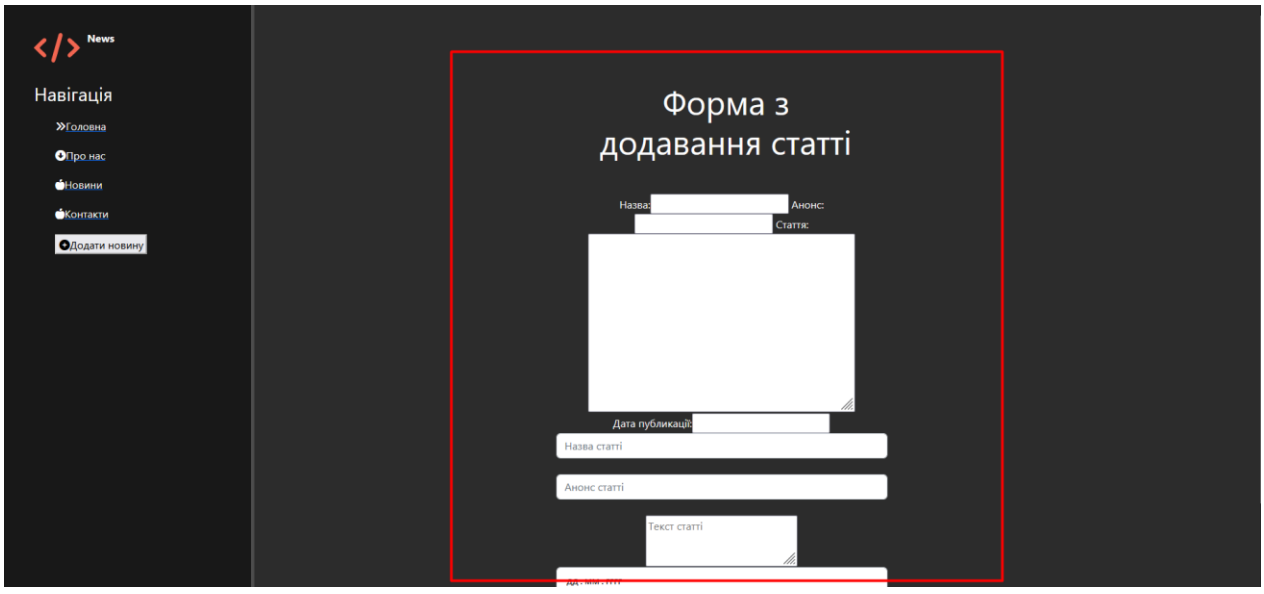


Рисунок 4.17

Ми бачимо, що у новому вигляді додано нову форму. Приведемо форму до ладу. Допишемо до файлу:

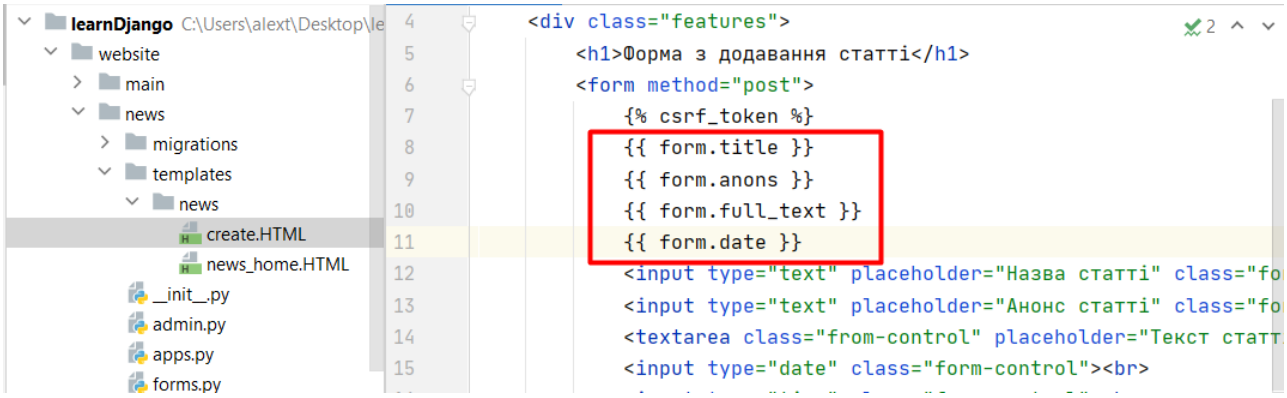


Рисунок 4.18

Зникли написи:

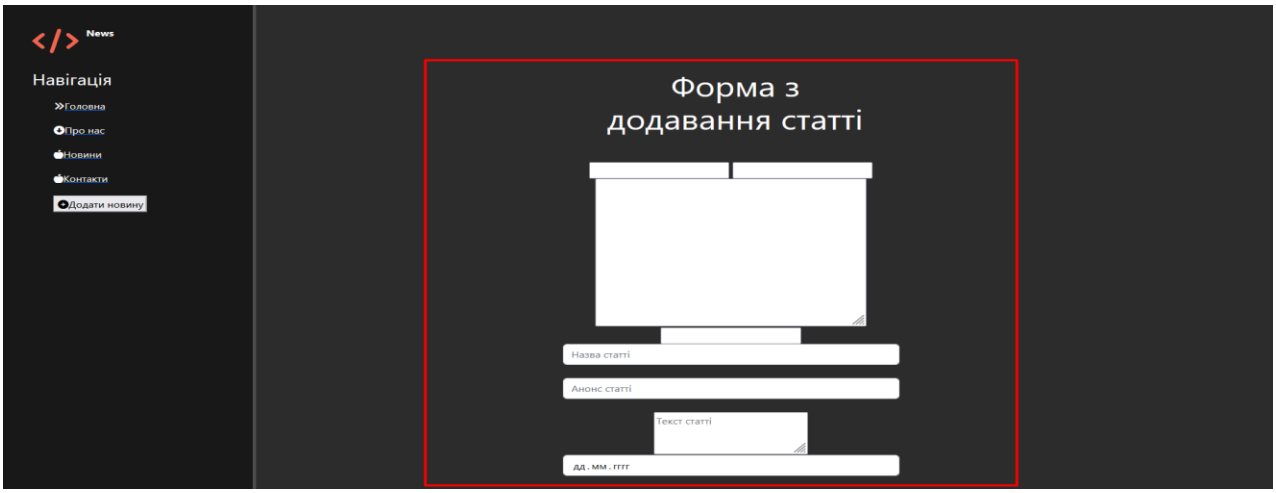


Рисунок 4.19

Додамо атрибути до полів. Для цього зайдемо в **forms.py.**

Запишемо словник – опис кожного поля. Імпортуємо клас TextImput, DateTimeInput, Textarea.

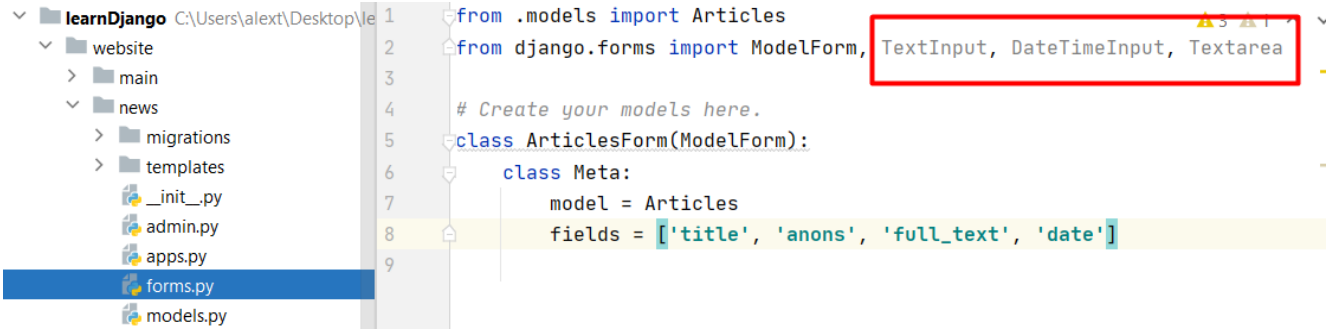


Рисунок 4.20

Пропишемо:

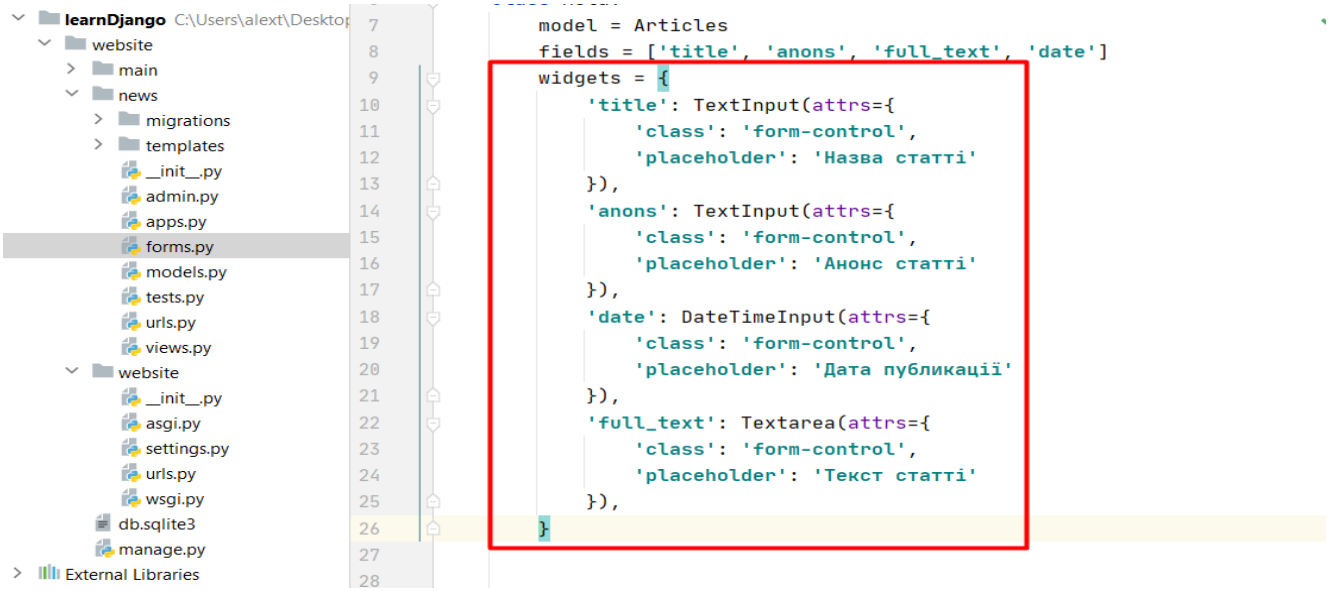


Рисунок 4.21

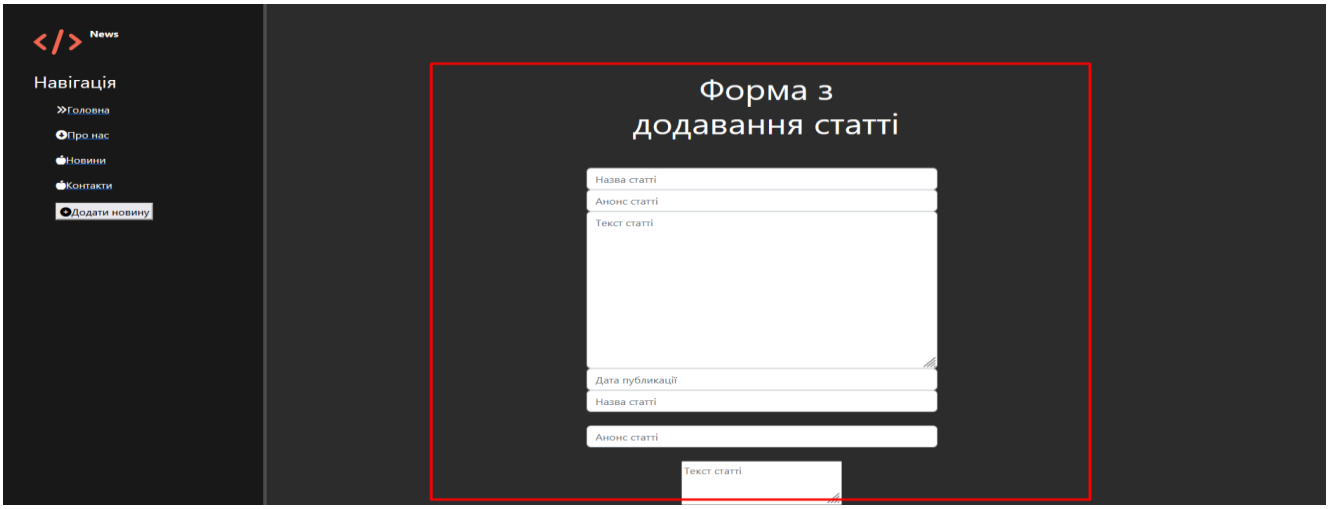


Рисунок 4.22

Додамо відстань між табличками.

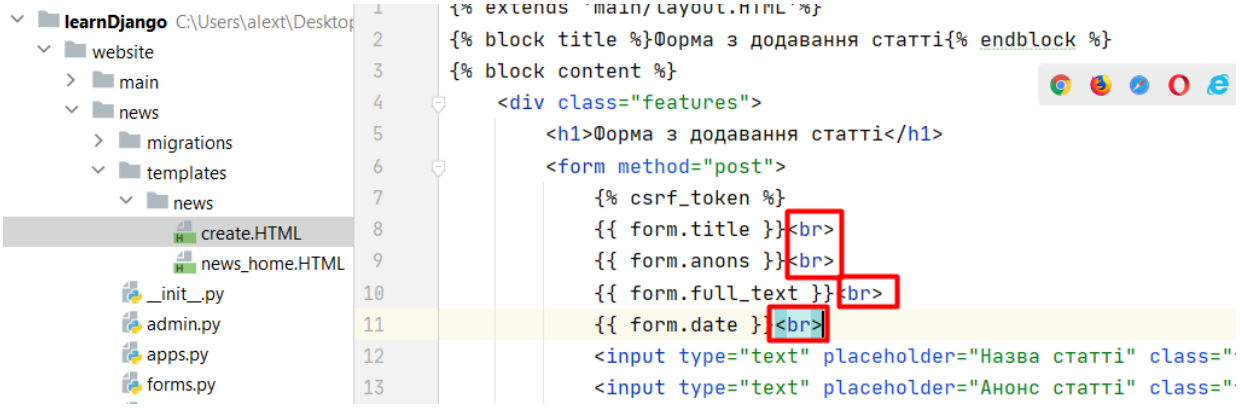


Рисунок 4.23

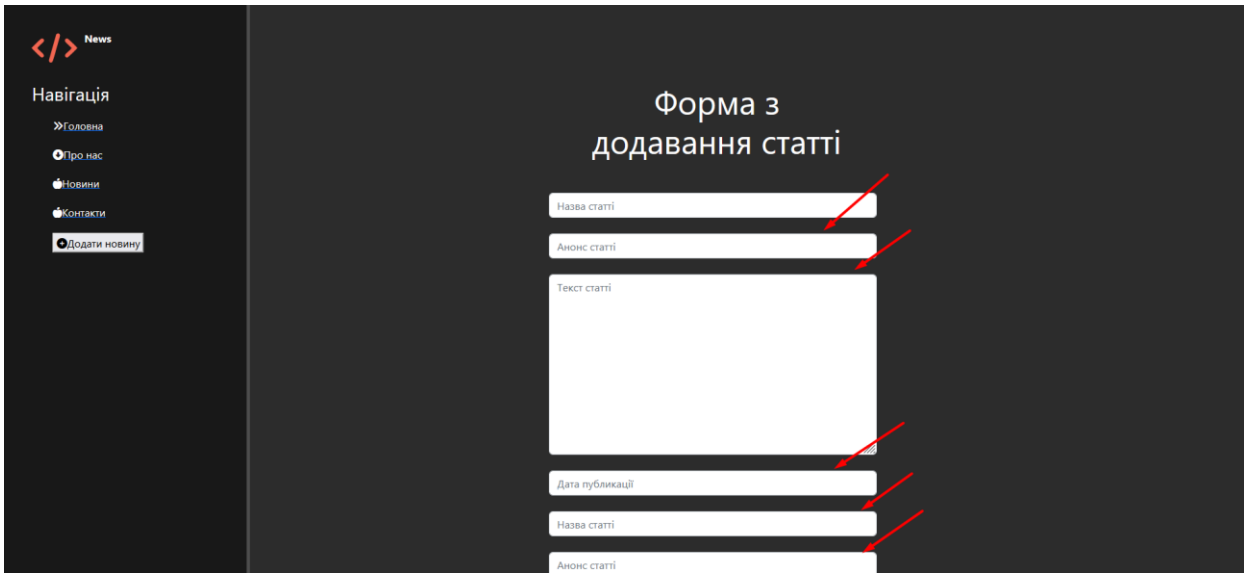


Рисунок 4.24

Пропишемо обробку даних, отриманих із форми. Пропишемо отримання даних із форми. Вона відбуватиметься на тій самій сторінці, де й сама форма.

Заходимо до **views.py.**

Пропишемо, отримання даних з форми, їх обробку, показ помилок або запис нових даних в таблицю баз даних.

Пропишемо умову перевірки методу передачі. Якщо йде метод передачі даних, такий як POST - знатимемо, що дані відправляються з форми, т.к. у самій формі ми вказали метод.

Пропишемо умову if request.method == 'POST'.

Створимо об'єкт form на основі класу ArticlesForm і вкажемо параметр request.POST (Form = ArticlesForm(request.POST)).

Для перевірки даних використовуємо звернення до об'єкта, і звертаємося до методу is_valid(), він дозволяє перевірити коректність заповнення даних.

1. Якщо дані правильно заповнені – зберігаємо нові записи у табличці Articles, звертаємось до об'єкта form та методу save.

2. Якщо не правильно заповнені дані то створюємо змінну error = ''

і всередину впишемо “Форма була невірною”.

У шаблон передаватимемо текст помилки 'form': form, 'error': error - по ключу error передаватимемо змінну, яка називається error.

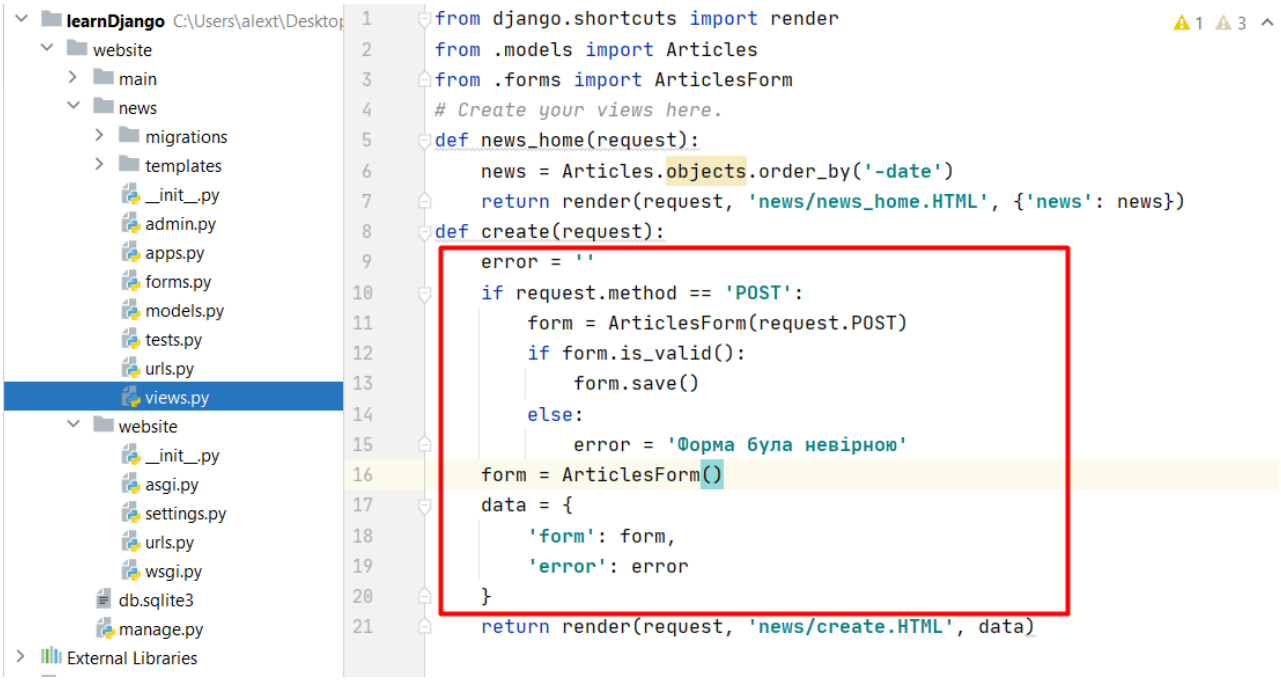


Рисунок 4.25

Виводимо помилку. Якщо помилки немає, змінна порожня і блок не видно, якщо ні, то бачимо помилку.

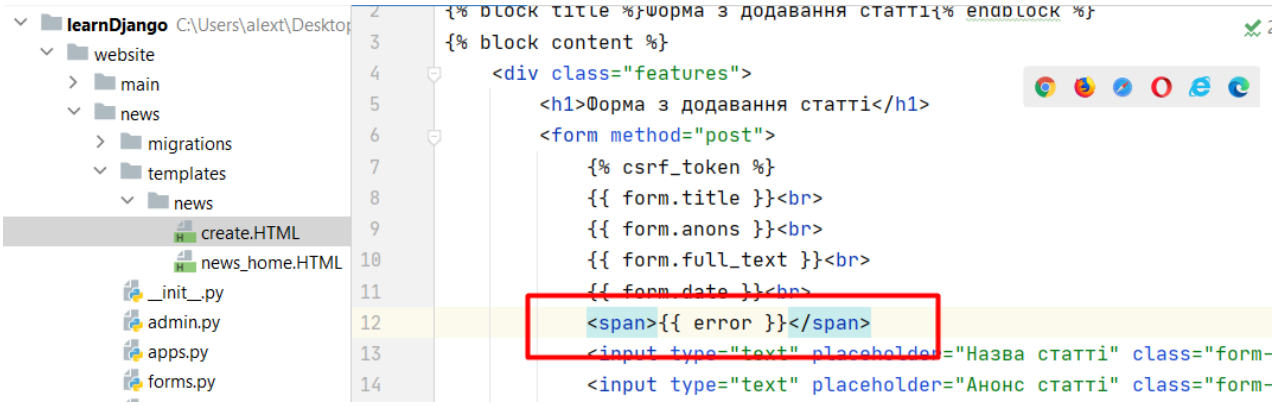


Рисунок 4.26

Припустимо, що текст введено правильно. Проведемо переадресацію на іншу сторінку, наприклад головну. Треба вписати return.

Підключаємо метод redirect.

Рисунок 4.27

Вводимо дані у форму новин та натискаємо "додати новину".

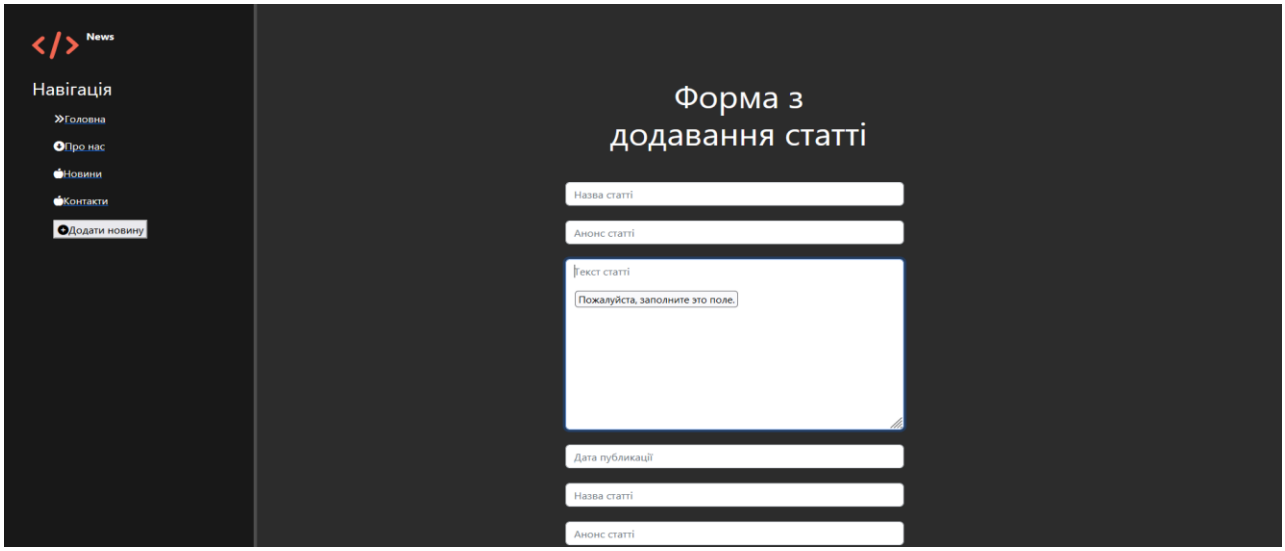


Рисунок 4.28

Переходимо в панель адміністратора:

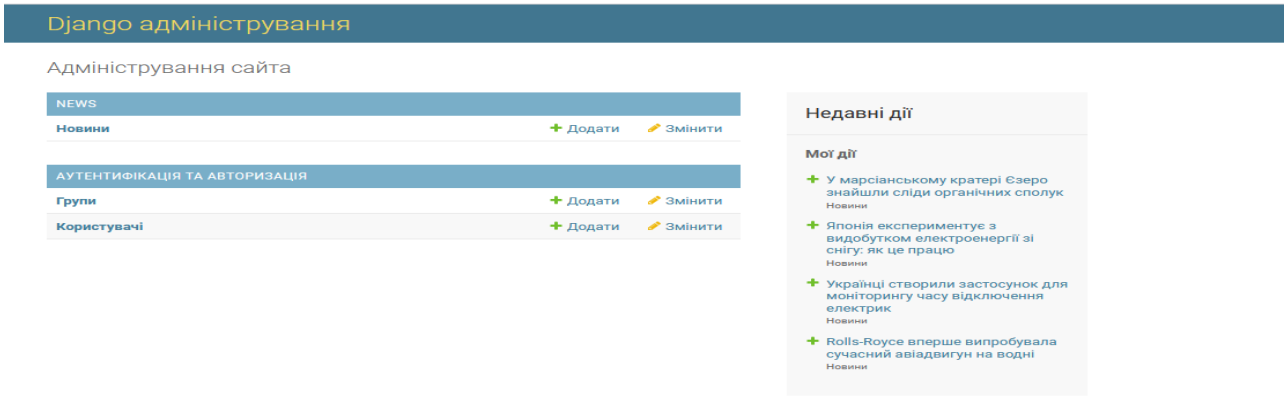


Рисунок 4.29

Є чотири новини. Для зміни новини достатньо натиснути "Змінити новину".

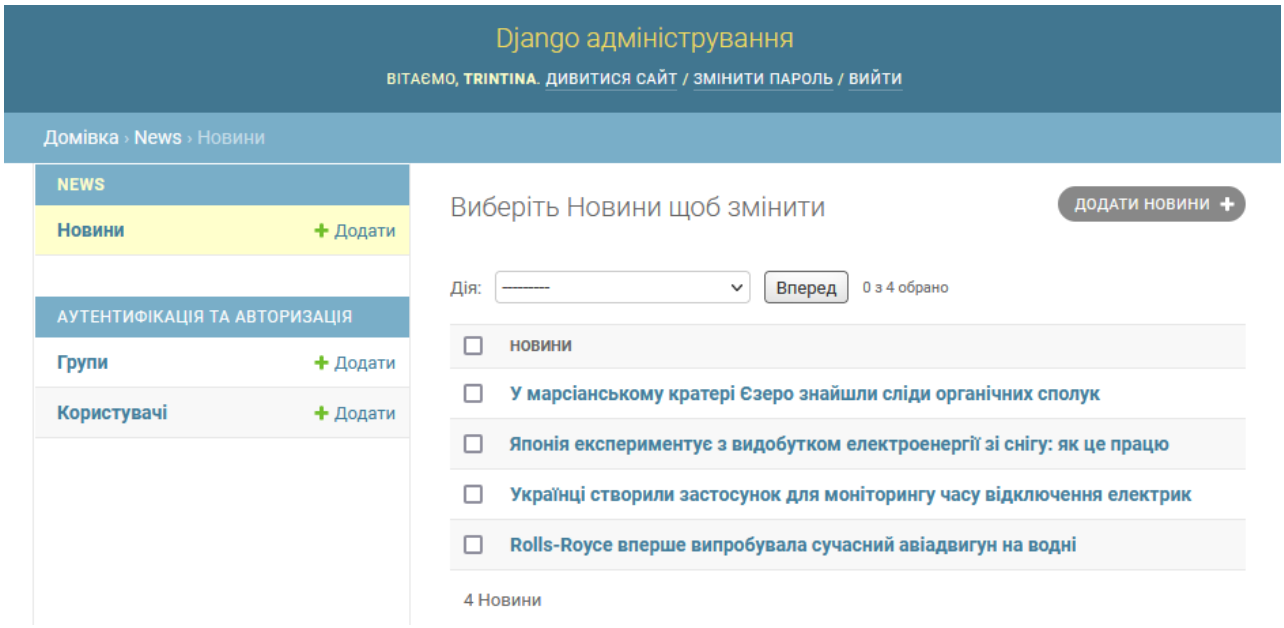


Рисунок 4.30

Створимо динамічні сторінки для відображення кожної окремої статті, які будуть доступні за адресою url.

Створимо клас, який успадковуватиме все від вбудованого класу і всередині класу опишемо деякі характеристики для динамічної сторінки.

Створимо клас. Для цього імпортуємо вбудований клас з views з папки generic. За допомогою команди from django.views.generic import імпортуємо необхідний клас.

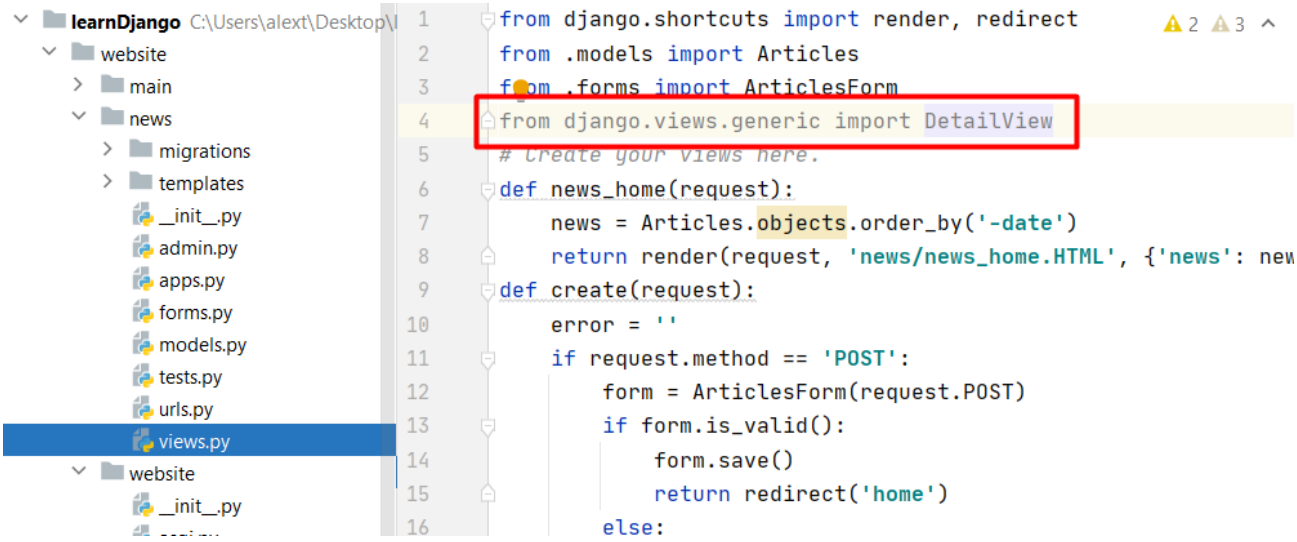


Рисунок 4.31

Пропишемо створення нового класу class NewsDetailView(DetailView), який успадковує клас DetailView. Будемо працювати з моделлю model=Articles. Вказуємо змінну template_name, в якій вказуємо шаблон, який оброблятиме сторінку, з якою ми працюємо template_name = 'news/details_view.html'. Шаблон news/details_view.html потім зробимо.

Створимо змінну context_object_name - назва ключа за яким будемо передавати запис всередину шаблону з назвою article.

context_object_name = 'article'

Рисунок 4.32

Заходимо у папку urls.py та напишемо відстеження адреси. Для відстеження динамічного параметра вводимо '<int:pk>'.

int - ціле число, а потім його назва pk.

Звертаємось до файлу views і викликаємо клас NewsDetailView і звернемося до методу as_view(). Інакше буде помилка. Оскільки ми працюємо з іменованими адресами, то назвемо news-detail.

path('<int:pk>', views.NewsDetailView.as_view(), name='news-detail')

Рисунок 4.33

Створимо новий шаблон у папці з назвою **details_view.html**

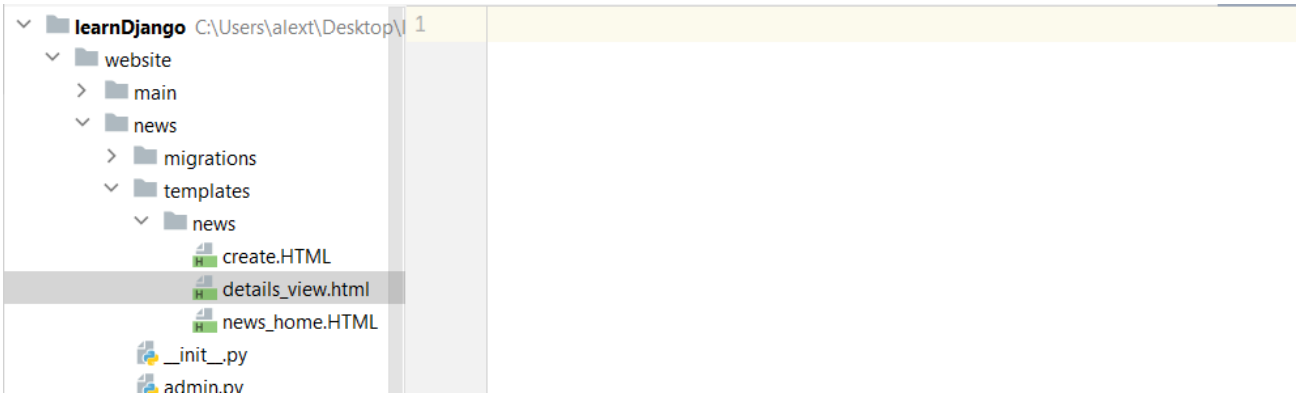


Рисунок 4.34

Пропишемо новий код. Звернемося до об'єкта article і звертаємося до поля, яке хочемо вивести.

Наприклад:

назва - <h1>{{ article.title }}</h1>

дата публікації статті - {{ article.date }}

повний текст статті - {{article.full_text}}

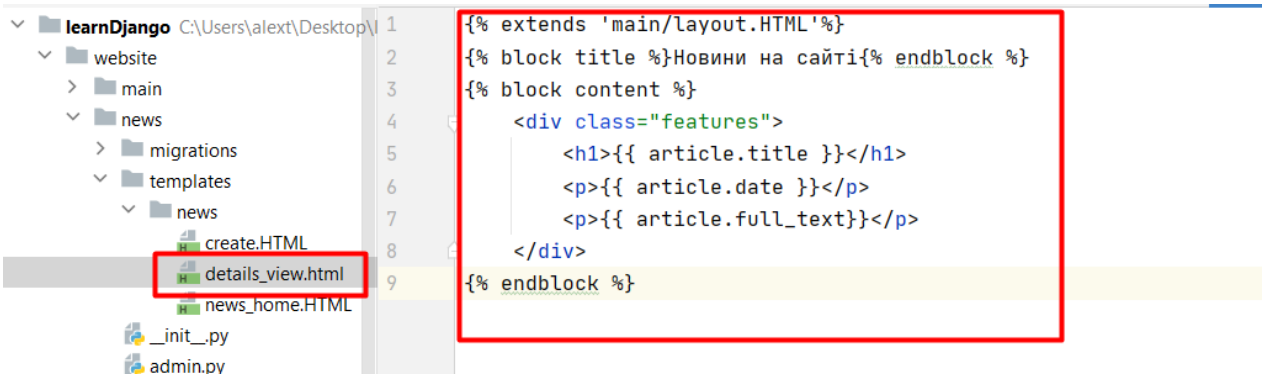


Рисунок 4.35

Побачимо, що вийшло. Переходимо за адресою http://127.0.0.1:8000/news/1

Видно, що це працює правильно.

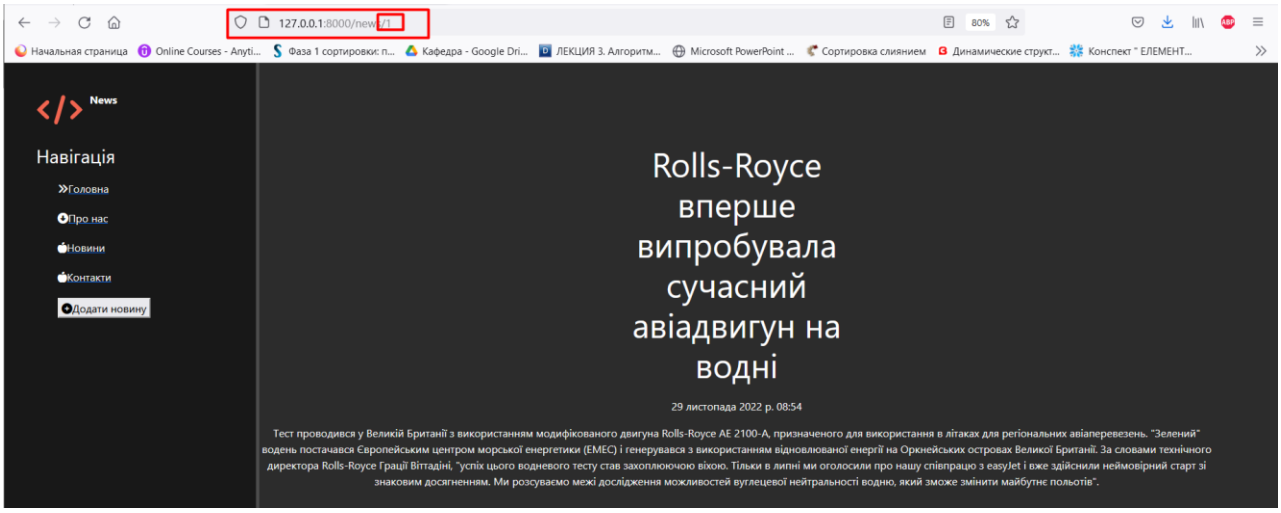


Рисунок 4.36

Тепер додамо кнопку на детальніший текст статті, яка перекладатиме нас за адресою:

http://127.0.0.1:8000/news/1

http://127.0.0.1:8000/news/2

http://127.0.0.1:8000/news/3

Переходимо у шаблон news_home.HTML. Створимо нове посилання зі стилем btn btn-warning. На кнопці буде виведено “Читати докладніше”.

Вкажемо url-адресу за допомогою іменованих посилань url 'news-detail'.

News-detail у нас приймає деякий динамічний параметр, а далі через пропуск вказуємо "{% url 'news-detail' el.id%}".

Читати докладніше

Рисунок 4.37

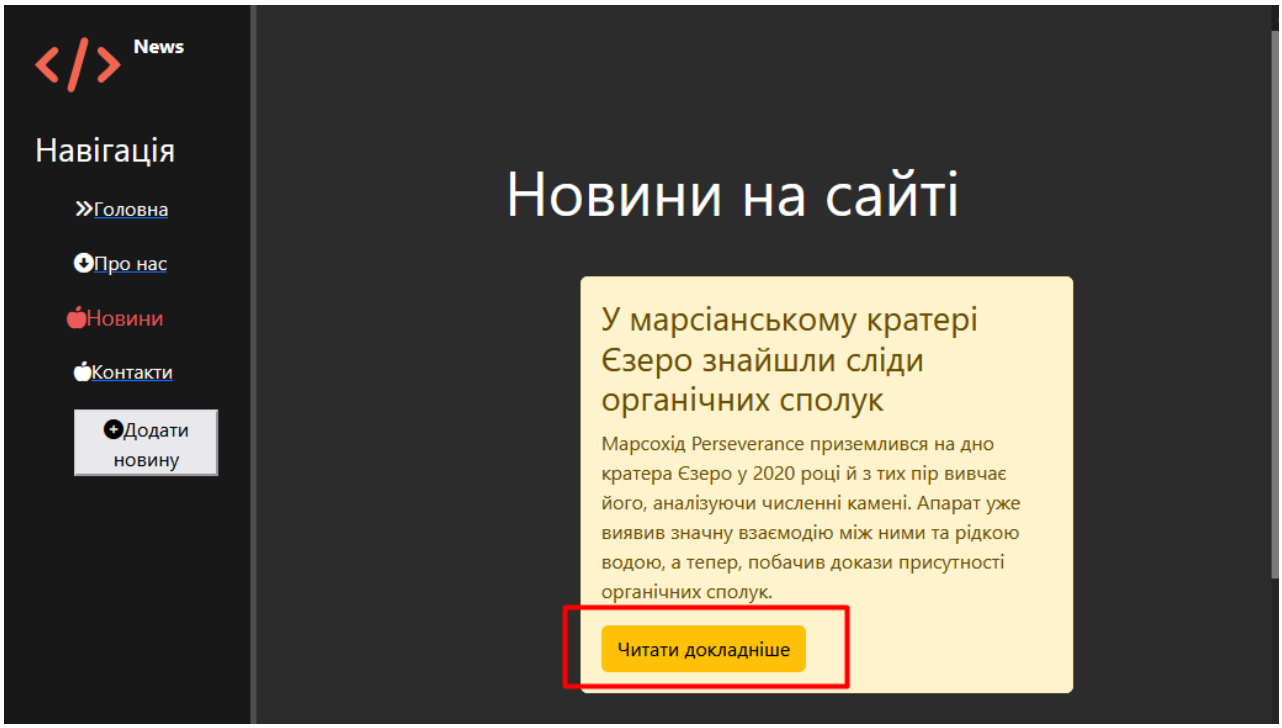


Рисунок 4.38

Тепер поміняємо назву для кожної сторінки.

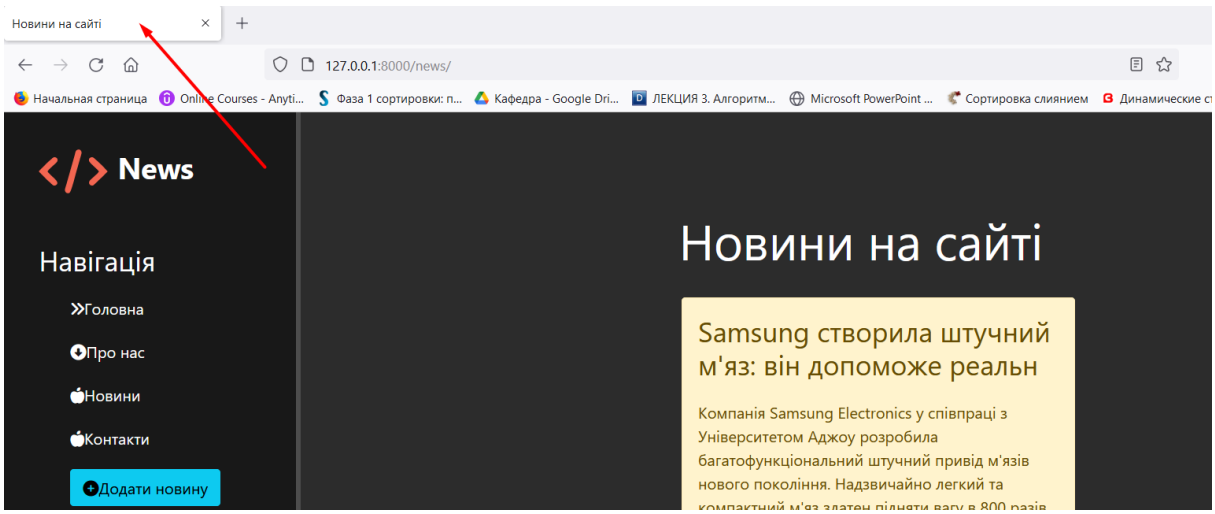


Рисунок 4.39

Для цього заходимо у details_view.html

Нам необхідно title передавати у форматі article.title.

Замість новини на сайті вписуємо article.title.

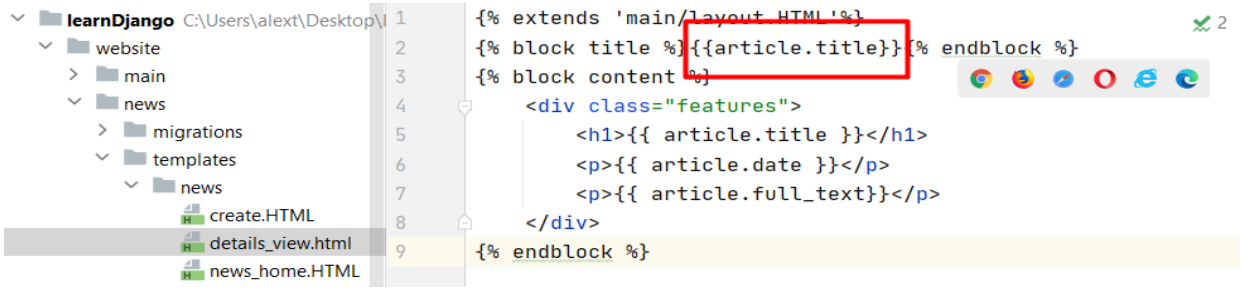


Рисунок 4.40

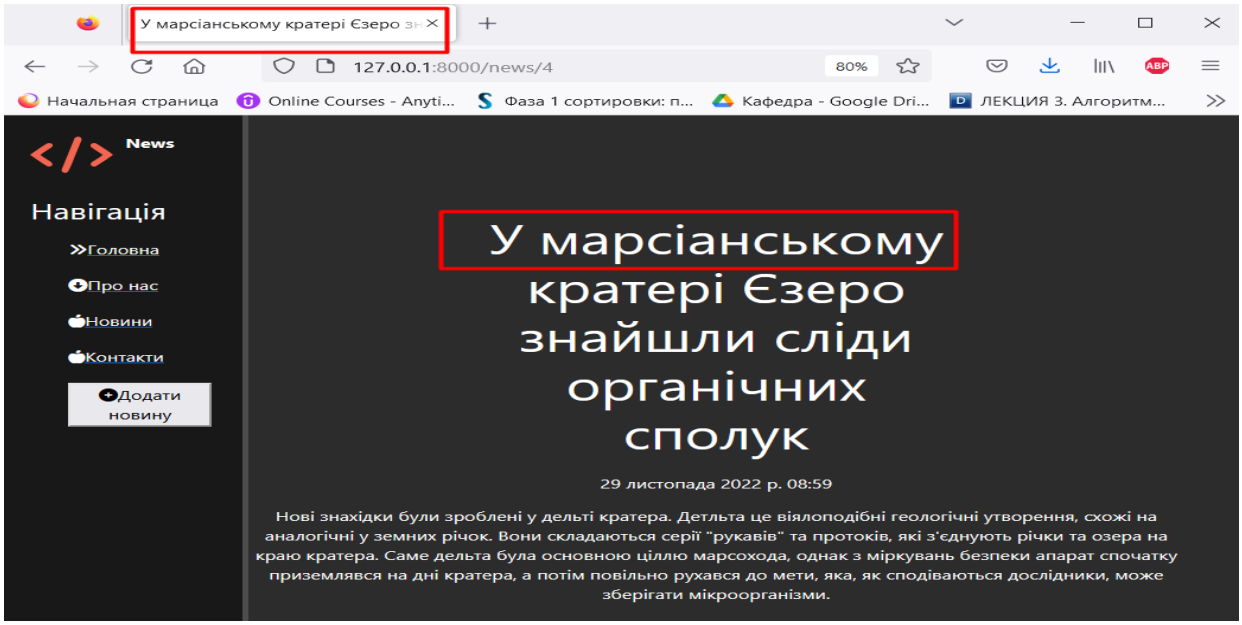


Рисунок 4.41

Реалізуємо сторінку по оновленню якогось запису в базі даних.

Наша сторінка буде доступна за адресою url.

Пропишемо обробник url адреси. Для цього заходимо до файлу urls.py.

Прописуємо:path('<int:pk>/update', views.NewsUpdateView.as_view(), name='news-update')

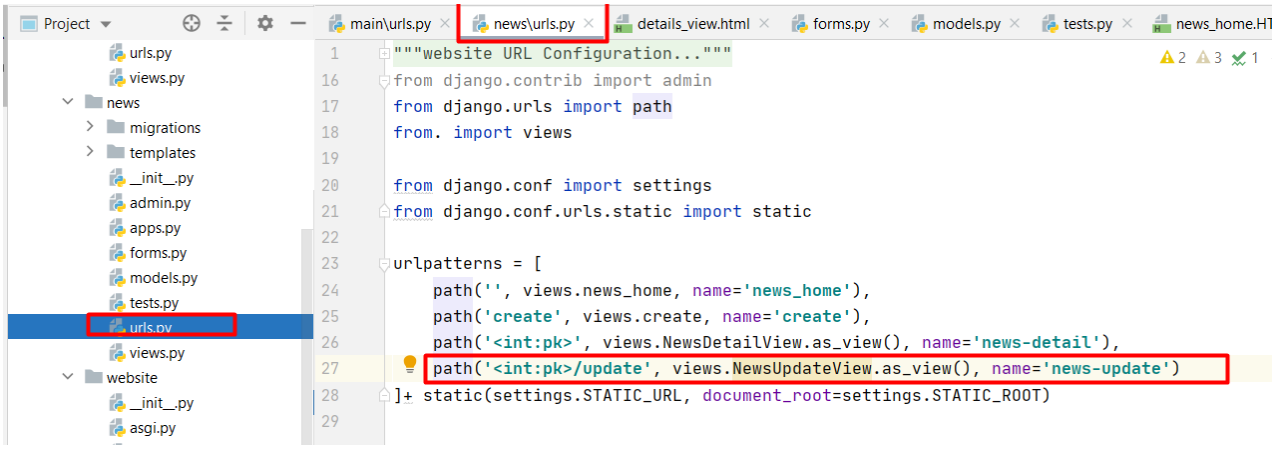


Рисунок 4.42

Наш адрес динамічний параметр <int:pk> зі словом ‘news-update’.

Будемо викликати клас (ще не створили його) NewsUpdateView.

Наше посилання тепер називається news-update.

Створюємо клас NewsUpdateView.

Заходимо до news\views.py.

Підключаємо клас UpdateView.

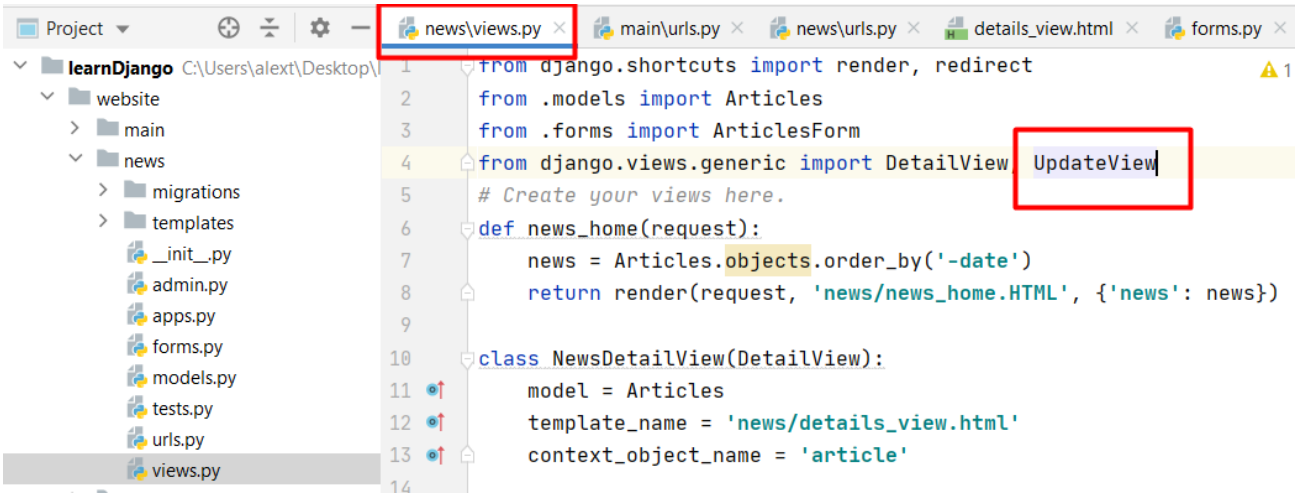


Рисунок 4.43

Створюємо новий клас NewsUpdateView, він буде успадковувати все від класу UpdateView.

Працюватимемо з моделлю model = Articles. Вкажемо той самий шаблон, що ми вже створили та використовували всередині методу create. Вкажемо всі поля, які мають бути у формі 'title', 'anons', 'ful_text', 'date'. Для цього вкажемо клас, який використовується для відображення форми. У класі вже прописані характеристики для відображення форми.

form_class = ArticlesForm

Рисунок 4.44

Видалимо наступні рядки:

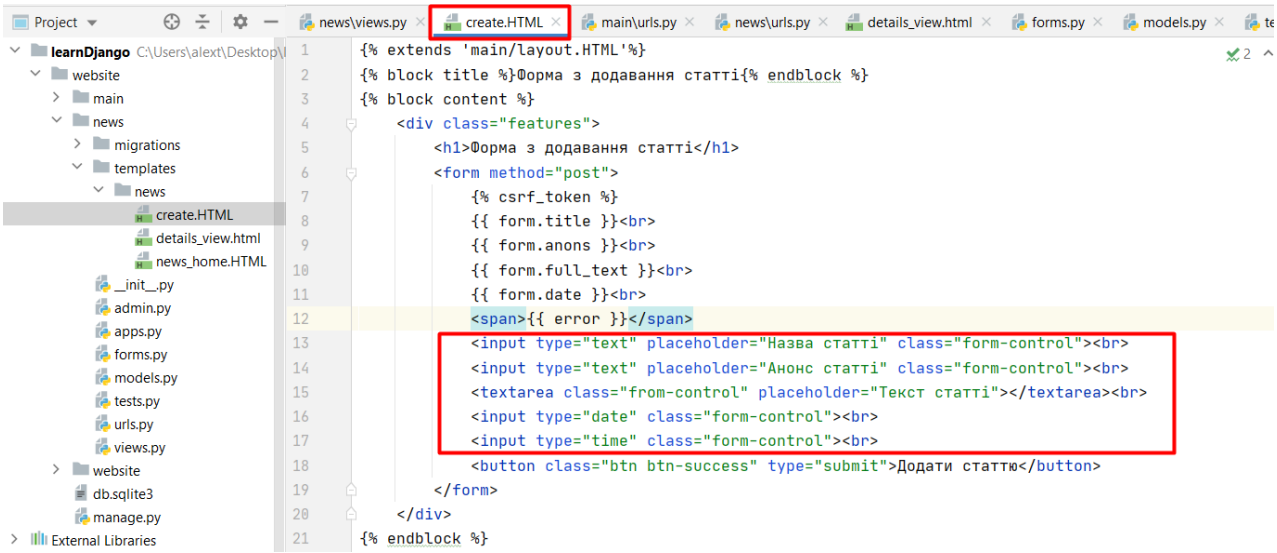


Рисунок 4.45

Отримаємо:

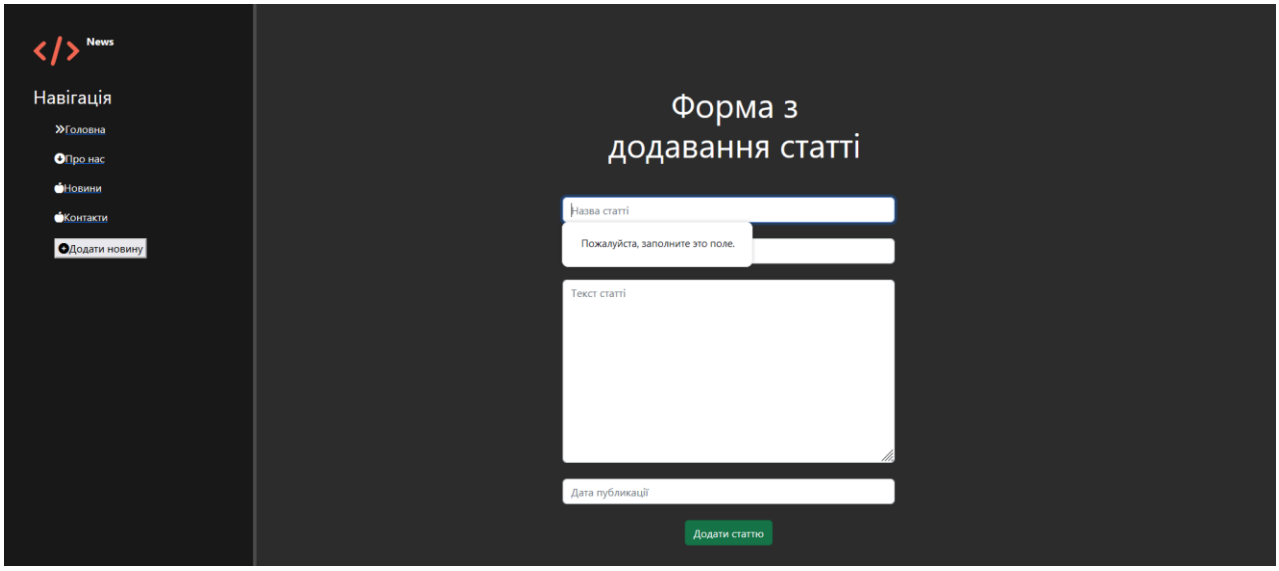


Рисунок 4.46

Для того, щоб можна було виправляти вміст статті треба перейти в всередині методу get_absolute_url. Вказується сторінка, куди переадресовувати користувача після того, як оновимо статтю, переадресовуємо користувача на сторінки користувача - return f'/news/{self.id}'

Щоб отримати оновлені записи пропишемо self.id.

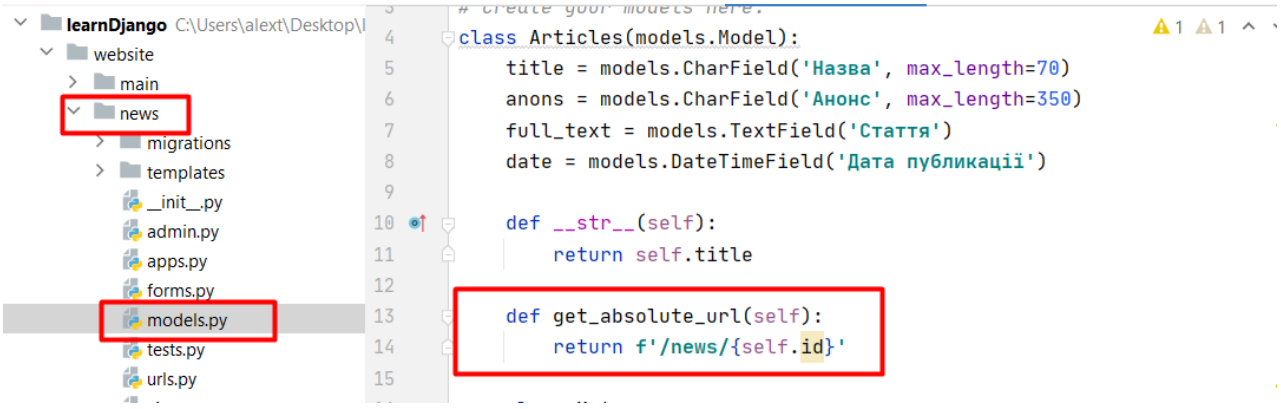


Рисунок 4.47

Створимо сторінку для видалення запису.

Пропишемо відстеження адреси.

Додаємо новий елемент path('<int:pk>/delete', views.NewsDeleteView.as_view(), name='news-delete').

Наш адресс динамічний параметр <int:pk> зі словом ‘news-delete’.

Будемо викликати клас (ще не створили його) NewsDeleteView.

Наше посилання тепер називається news-delete.

Рисунок 4.48

Створюємо клас NewsDeleteView.

Заходимо до **views.py**.

Рисунок 4.49

Імпортуємо вбудований клас DeleteView.

Створюємо новий клас NewsDeleteView, який успадковує клас DeleteView та використовує новий шаблон news-delete.html. Впишемо переадресацію користувача після видалення запису на новини.

success_url = 'news'.

Рисунок 4.50

Створимо новий шаблон: права клавіша миші, файл news-delete.html.

Заповнимо:

Рисунок 4.51

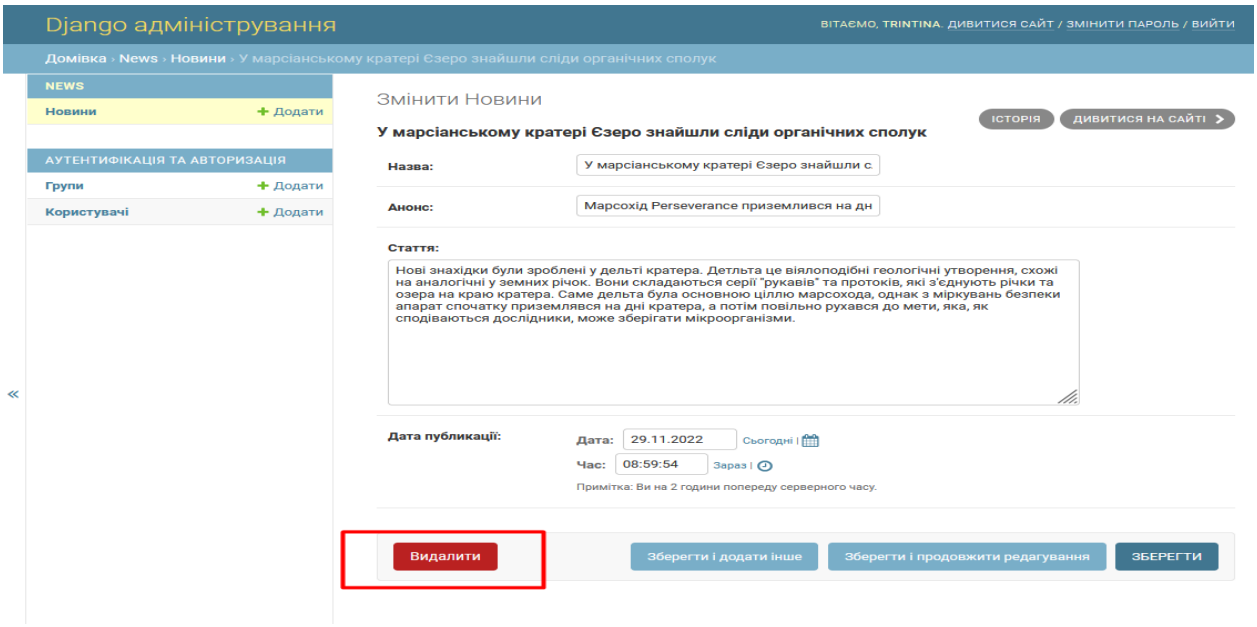


Рисунок 4.52

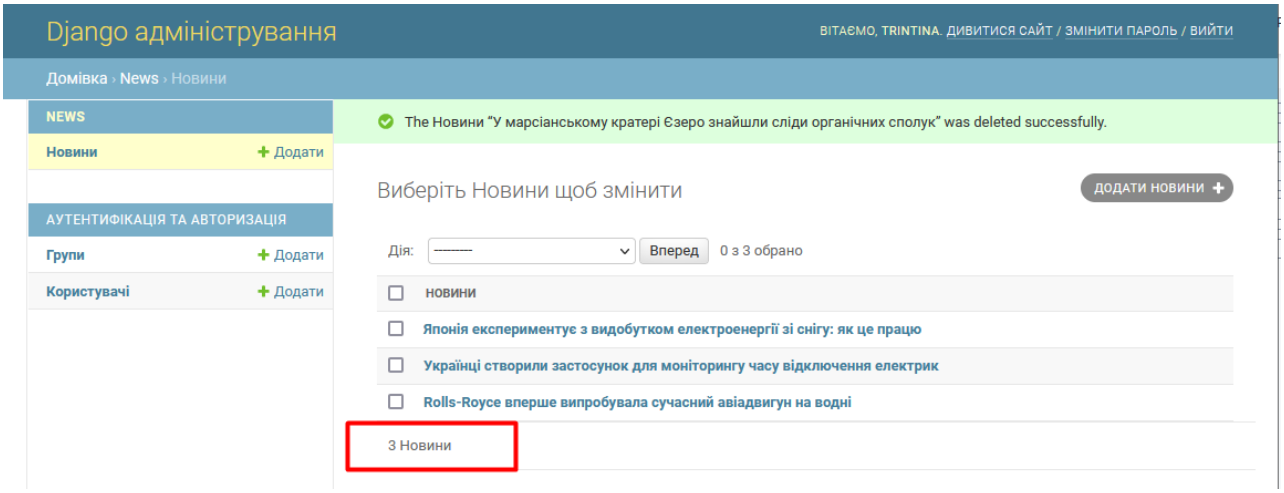


Рисунок 4.53

Кнопка видалити працює. Залишилось три статті.

Додамо дві кнопки на редагування та видалення.

Заходимо в **details_view.html.** Створимо два посилання, в яких ми переходимо в одному випадку на url адресу news-delete, в іншому випадку на news-update. За допомогою article.id ми передаємо динамічний параметр.

Рисунок 4.54

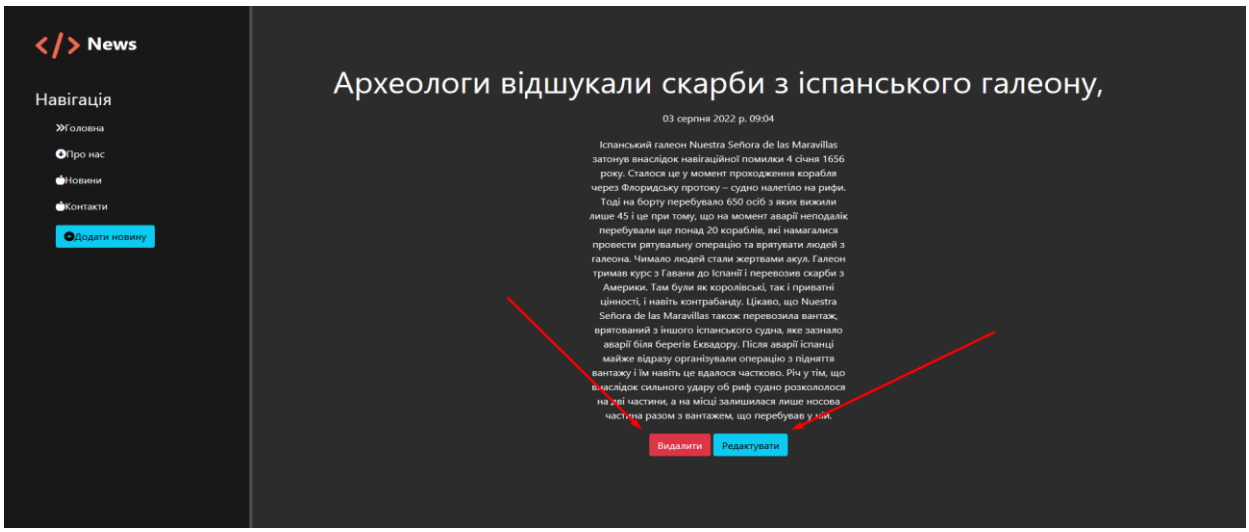


Рисунок 4.55

Натисніть на кнопку редагування.

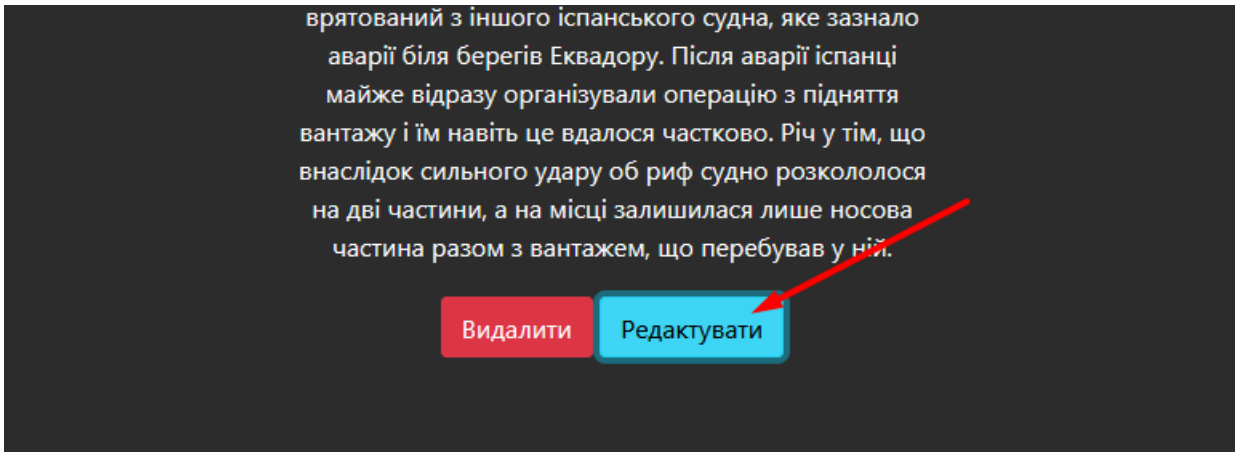


Рисунок 4.56

Отримаємо:

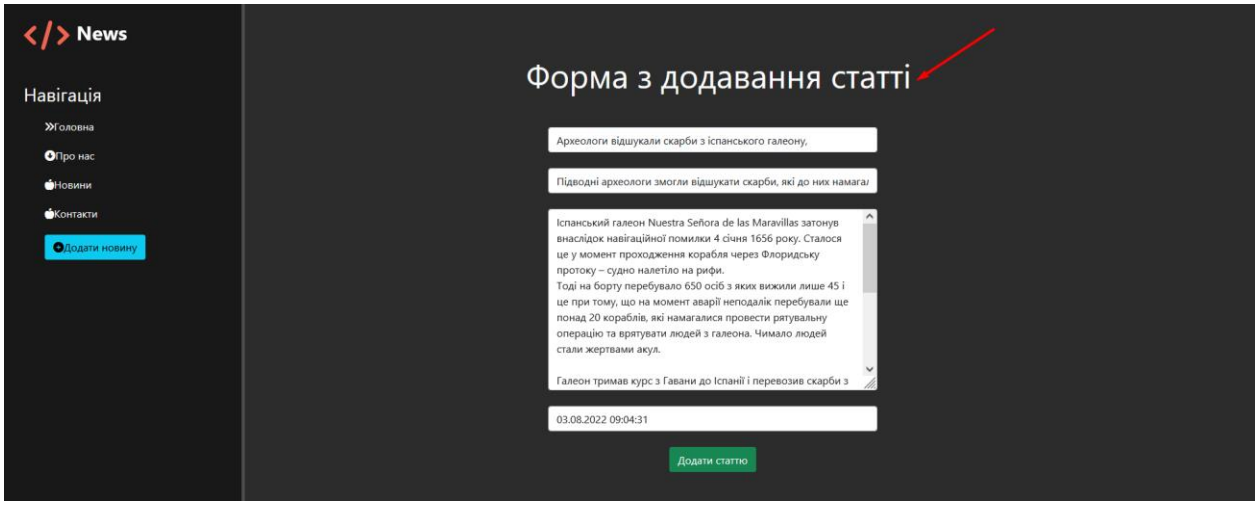


Рисунок 4.57

Натисніть кнопку видалення статті.

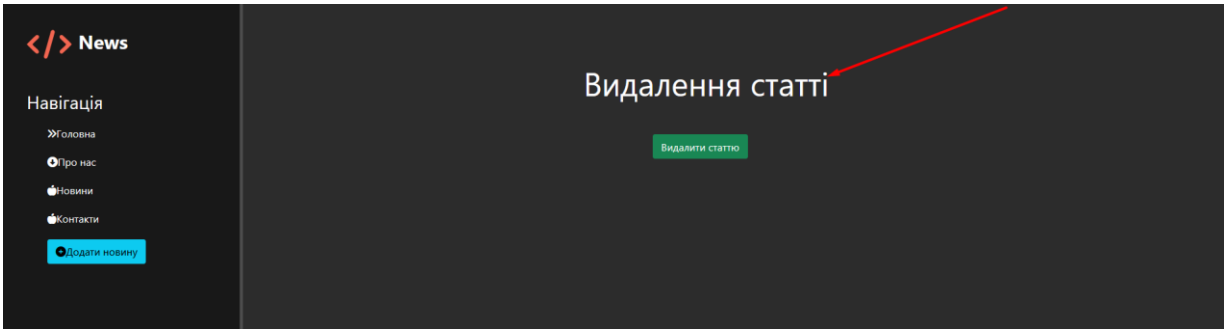


Рисунок 4.58

Програма працює коректно.

Практичне завдання

1. Повторити хід роботи.
2. Отримати результат.

Практична робота №5

Тема. Бібліотека NumPy на мові програмування Python.

Мета роботи. **Опанувати бібліотеку NumPy на мові програмування Python.**

Зміст.

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

Хід роботи

Пакет NumPy використовується для аналізу даних, у машинному навчанні та наукових обчисленнях. Він суттєво полегшує обробку векторів та матриць. Деякі провідні пакети Python використовують NumPy як основний елемент своєї інфраструктури. До них відносяться Scikit-learn, SciPy, Pandas і Tensorflow. Вміння працювати з NumPy дає значну перевагу при налагодженні складніших сценаріїв бібліотек. Дуже багато типів даних можуть бути представлені як n-мірні

масиви. Аудіофайл — це одномірний масив семплів. Кожен семпл являє собою число, яке є крихітним фрагментом аудіосигналу. Аудіо CD-якості можуть містити 44 100 семплів в секунду, кожен з яких є цілим числом у проміжку між -32767 і 32768. Це означає, що десятисекундний WAVE-файл CD-якості можна помістити в масив NumPy довжиною 10 * 44 100 = 441 000 семплів. Хочете витягти першу секунду аудіо? Завантажте файл у масив NumPy під назвою audio, після чого отримайте частину масиву audio[:44100]. Фрагмент аудіофайлу виглядає так:

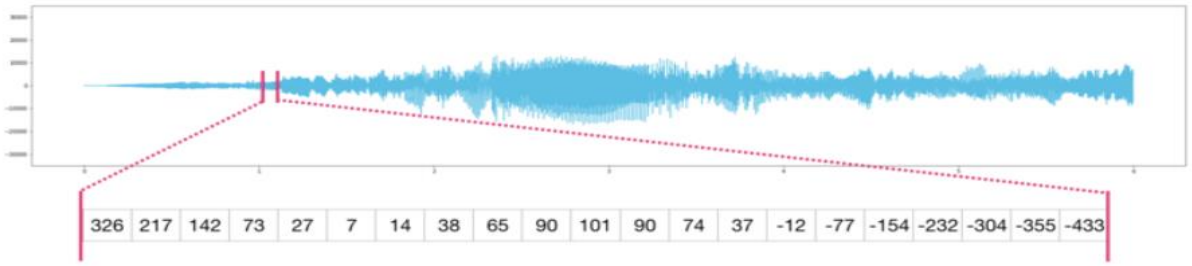


Рисунок 5.1

Це саме стосується даних часових рядів, наприклад, зміни вартості акцій з часом.

Зображення є матрицею пікселів за розміром (висота x ширина). Якщо зображення чорно-біле, тобто представлене в півтонах, кожен піксель може бути представлений як єдине число. Зазвичай між 0 (чорний) і 255 (білий). Хочете обрізати квадрат розміром 10 x 10 пікселів у верхньому лівому куті картинки? Просто введіть в NumPy image[:10, :10].

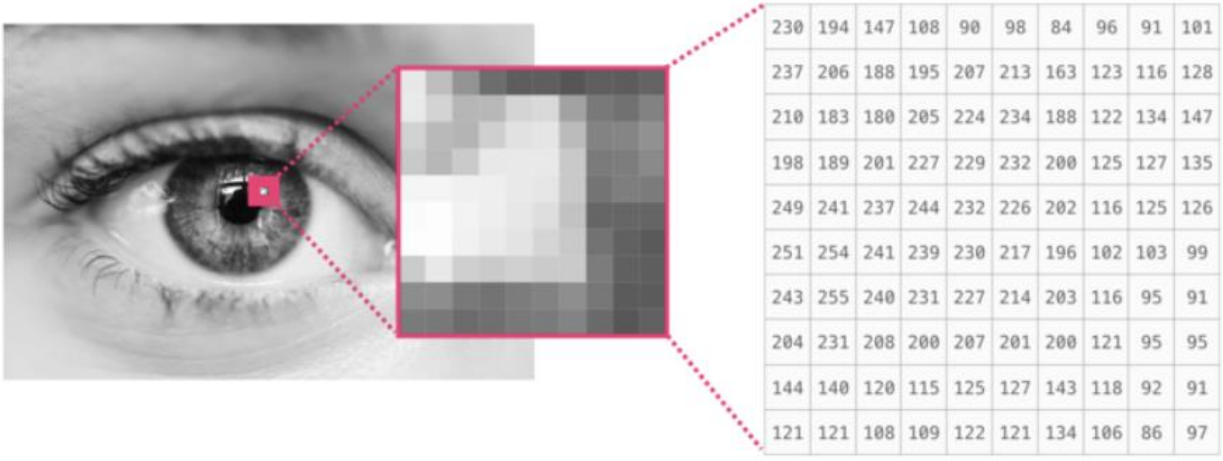


Рисунок 5.2

Якщо зображення кольорове, кожен піксель представлений трьома числами. Тут за основу береться кольорова модель RGB — червоний (R), зелений (G) і синій (B). У цьому випадку нам знадобиться третя розмірність, так як кожна клітина вміщує тільки одне число. Таким чином, кольорова картинка буде представлена масивом ndarray з розмірами: (висота x ширина x 3).

Рисунок 5.3

Технічна документація: https://numpy.org/doc/stable/reference/ufuncs.html#:~:text=A%20universal%20function%20(or%20ufunc,and%20several%20other%20standard%20features.

Встановлення бібліотеки

Для встановлення бібліотеки numpy треба відкрити термінал та використати систему керування пакетами pip.

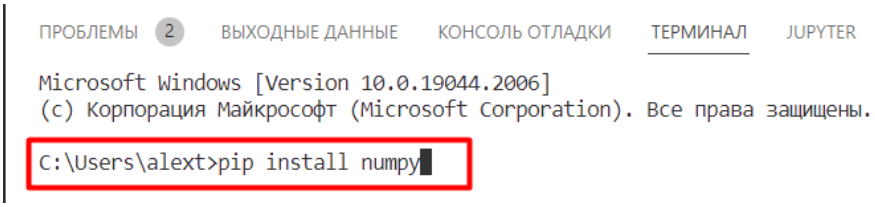


Рисунок 5.4

Створення масивів NumPy

1. Підключаємо бібліотеку NumPy за допомогою команди

```
import numpy as np
```

2. Найпростіше масив NumPy створити зі списку Python.

```
myPythonList = [1,9,8,3]
```

3. Перетворення списку на масив NumPy здійснюється за допомогою об'єкта np.array.

```
numpy_array_from_list = np.array(myPythonList)
```

4. Для виведення на екран результату просто наберемо:

```
print(numpy_array_from_list)
```

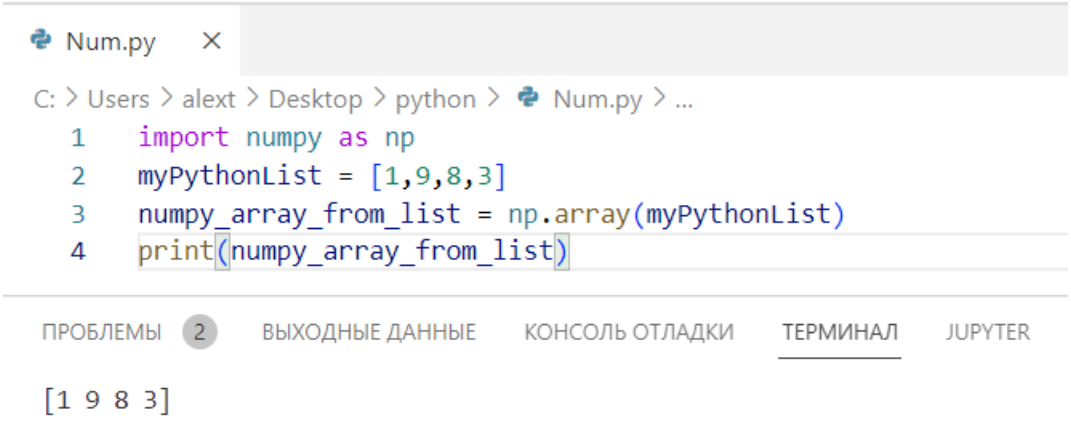


Рисунок 5.5

Насправді ці операції можна поєднати в одну:

```
a = np.array([1,9,8,3])
```

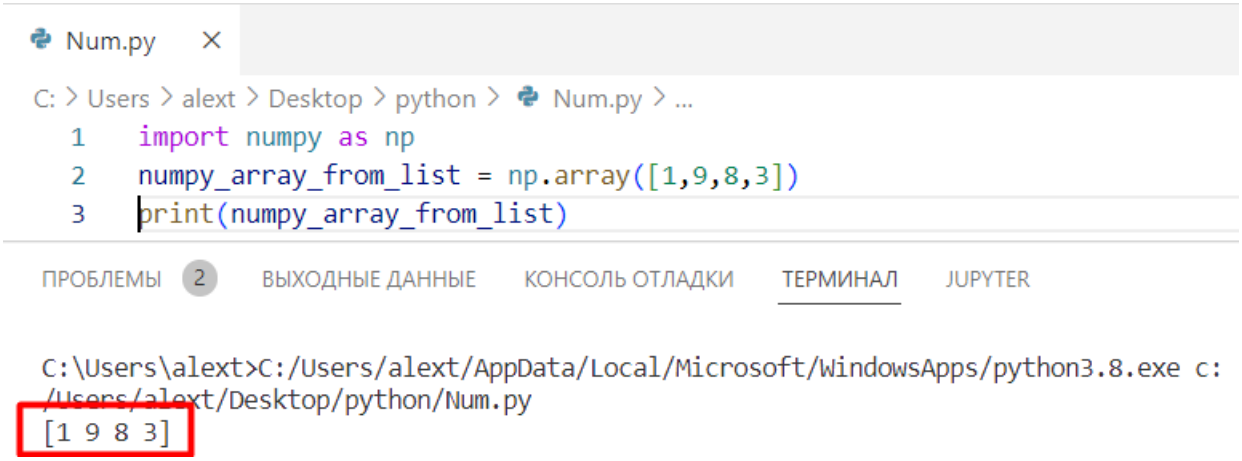


Рисунок 5.6

Примітка: у документації Python для створення масиву використовується метод np.ndarray(). Однак описаний метод також є рекомендованим. Так само масив NumPy може бути створений і з кортежу.

Математичні операції з масивами

Для масивів NumPy визначено всі основні арифметичні операції: додавання, множення, віднімання та поділ. Синтаксис теж звичайний: ім'я масиву, а за ним оператор (+, *, -, /).

Приклад:

np.add(x1, x2) - додає поелементно

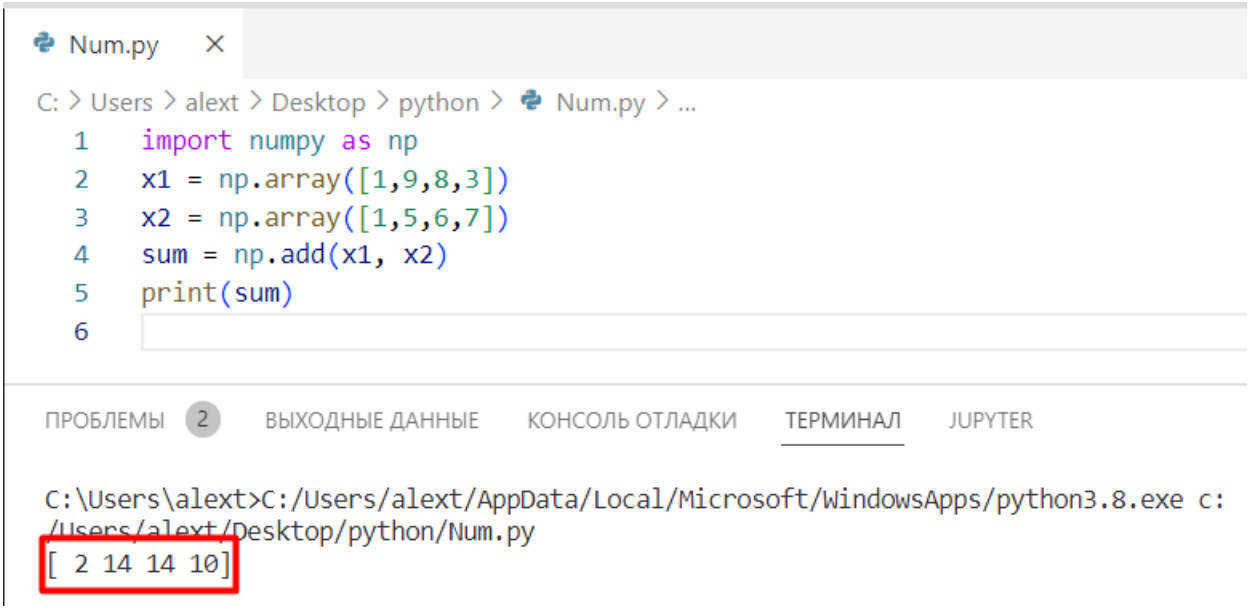


Рисунок 5.7

Приклад:

np.subtract(x1, x2) - віднімає поелементно

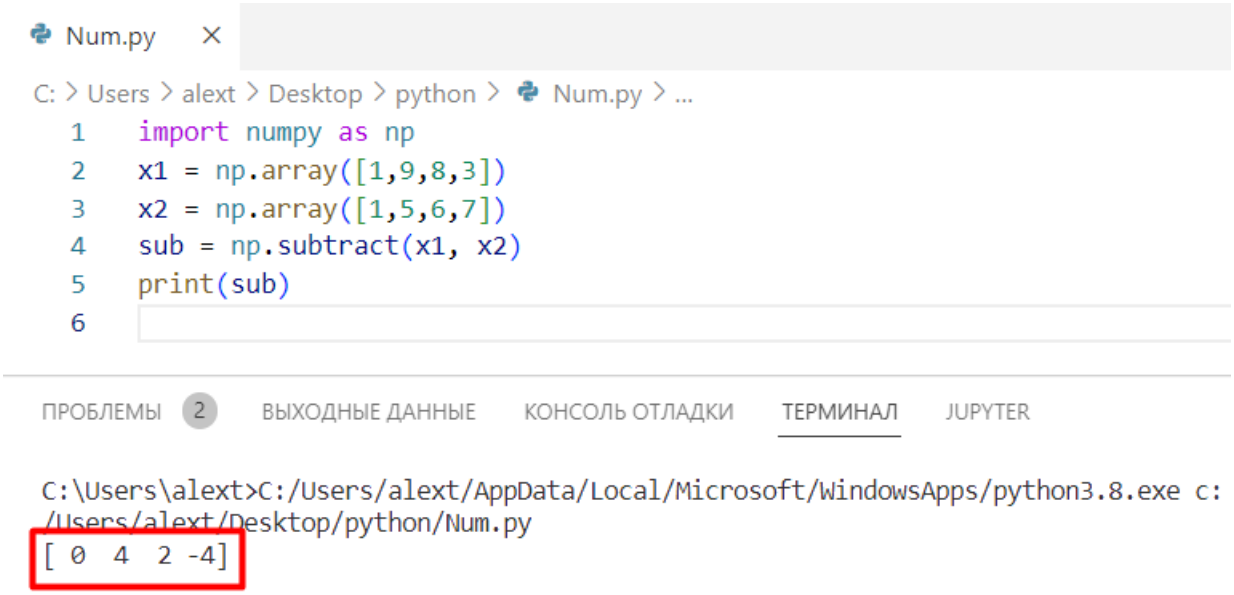


Рисунок 5.8

Приклад:

np.multiply(x1, x2) - помножує поелементно.

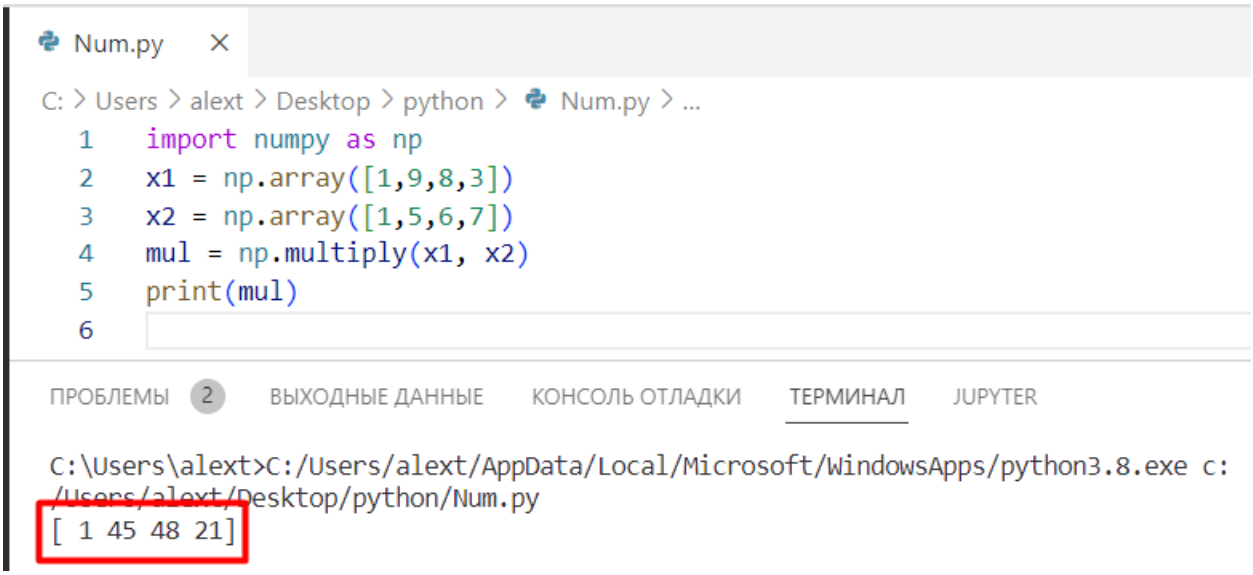


Рисунок 5.9

Приклад:

np.matmul(x1, x2) - Матричний добуток двох масивів.

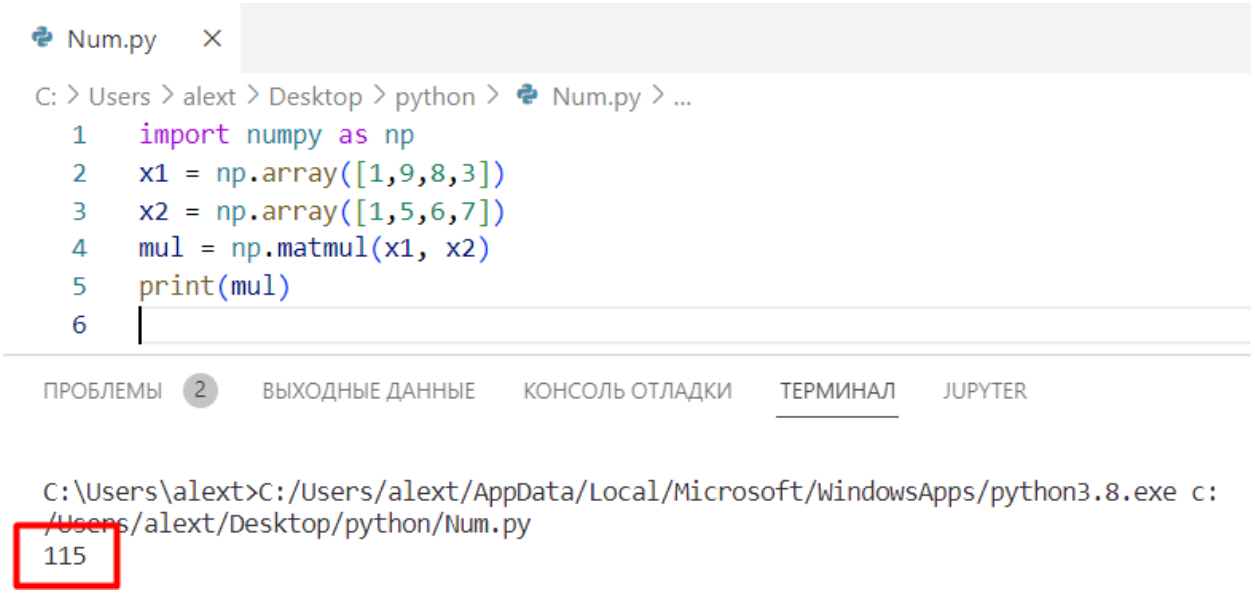


Рисунок 5.10

Приклад:

numpy_array_from_list + 5

В результаті цієї операції до кожного елемента масиву додалося число 5.

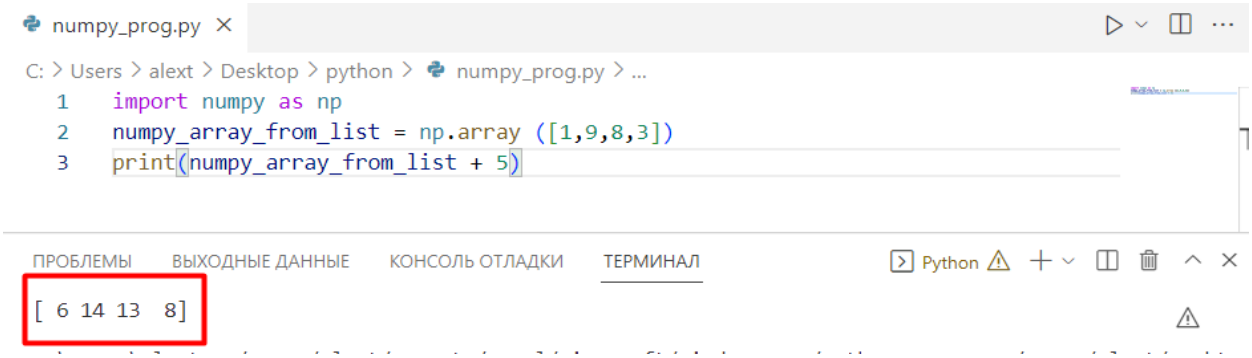


Рисунок 5.11

Розмірність масивів

Розмір масиву можна визначити за допомогою об'єкта shape, написавши його відразу після імені масиву. Так само, за допомогою dtype, можна визначити тип масиву.

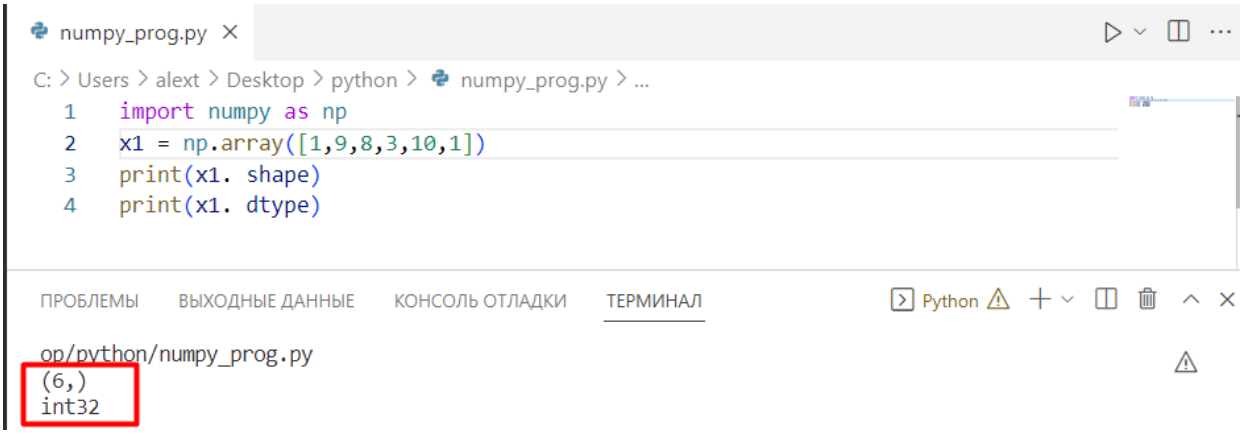


Рисунок 5.12

Тип integer не може містити значень із плаваючою комою. Тому, якщо ви створюєте масив із такими значеннями, його тип буде float.

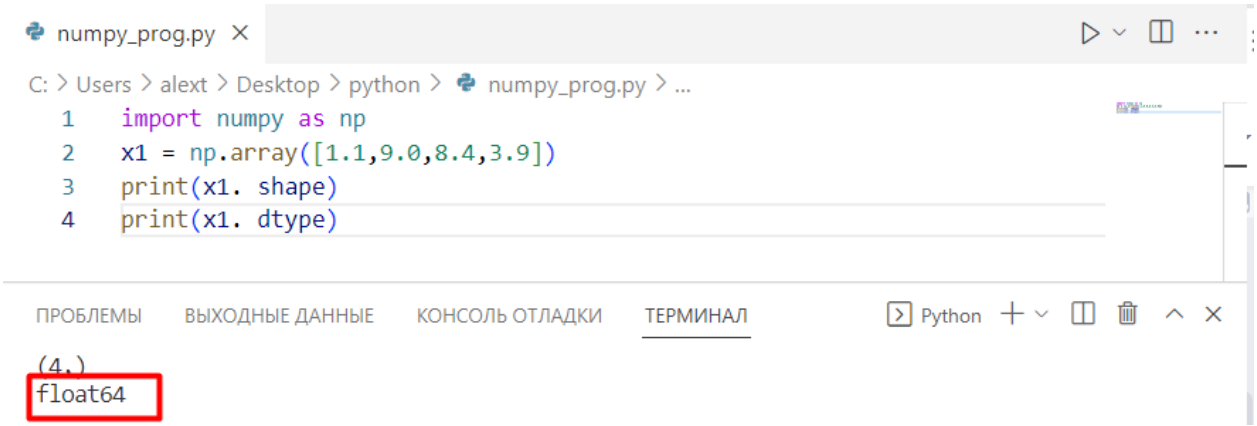


Рисунок 5.13

Двовимірні масиви

Розмір можна додати за допомогою коми. Зверніть увагу, що це має бути у квадратних дужках [].

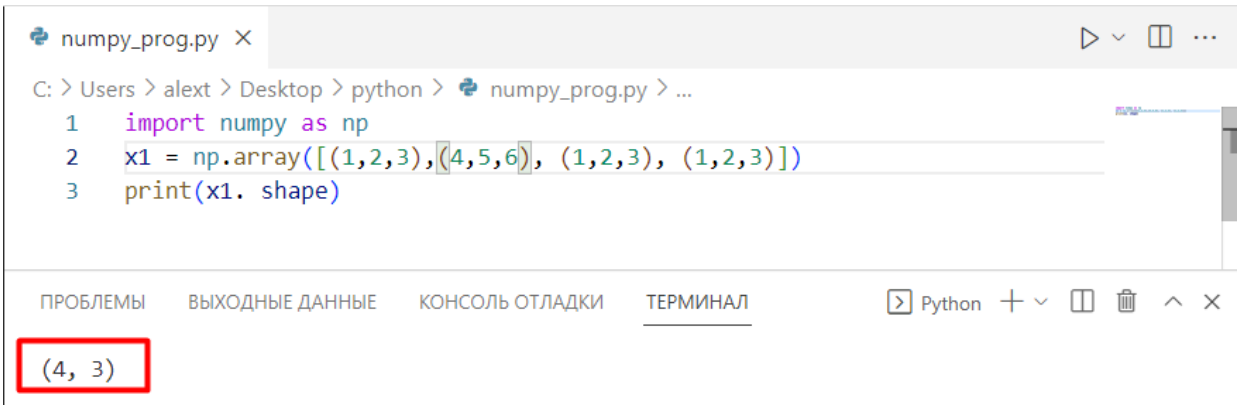


Рисунок 5.14

Тривимірні масиви

Масиви вищих розмірностей можна створювати так:

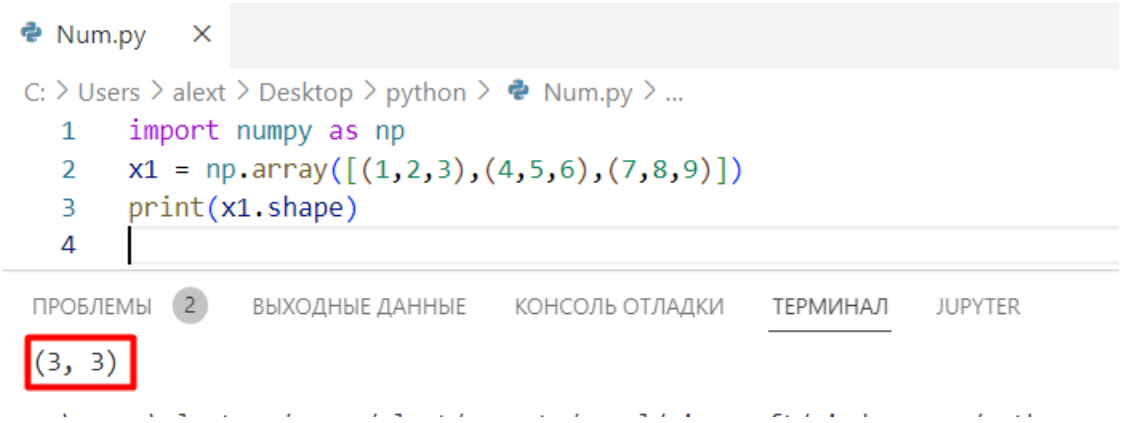


Рисунок 5.15

Виведемо матрицю.

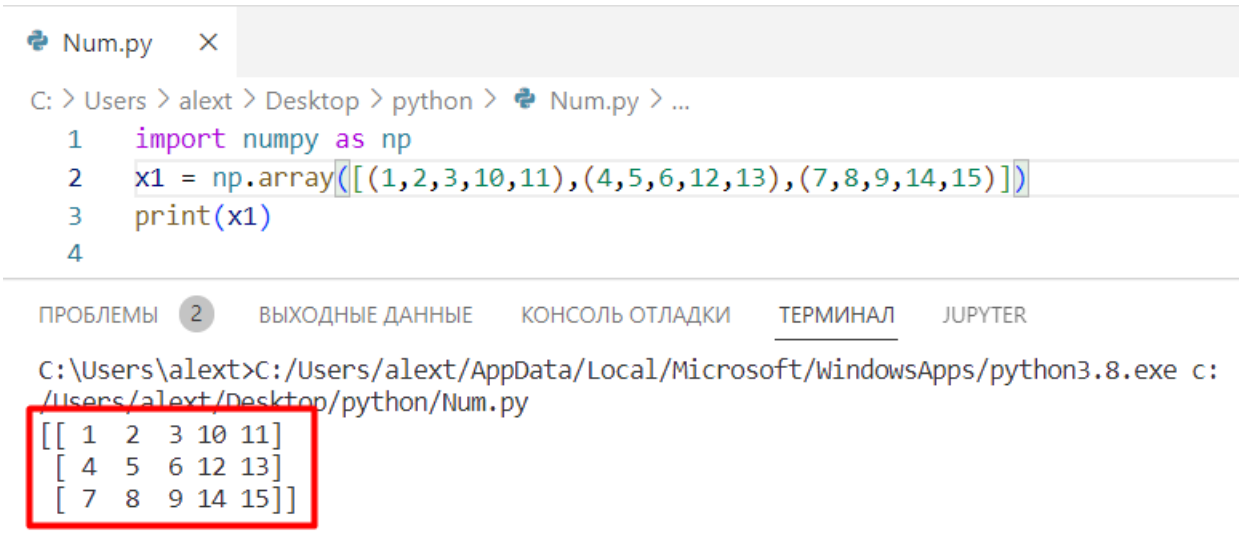


Рисунок 5.16

Функції numpy.zeros() та numpy.ones().

Функція numpy.zeros() використовується для створення нульової матриці. Також вона може бути використана, коли ви в процесі першої ітерації ініціалізуєте вагові коефіцієнти TensorFlow а також у різних статистичних завданнях.

Синтаксис буде наступним:

numpy.zeros(shape, dtype=float, order='C')

Тут shape - розмірність масиву, dtype - тип даних (зазначається опціонально, за умовчанням заданий тип float64), a order визначає порядок зберігання масиву в пам'яті. Order вказується опціонально, за умовчанням заданий параметр C, що означає построкове зберігання. Можна поміняти на F і зберігати по колонках.

Приклад:

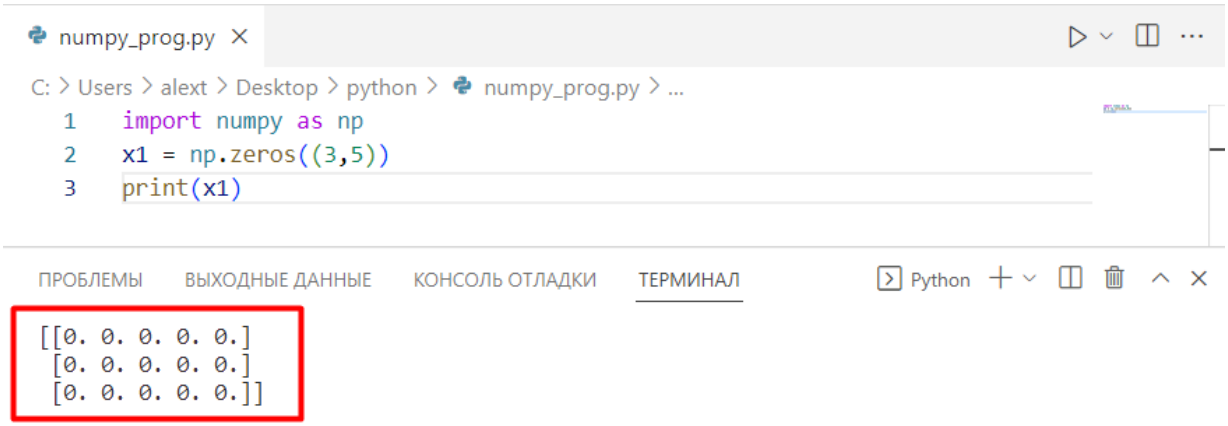


Рисунок 5.17

Приклад використання із зазначенням типу даних:

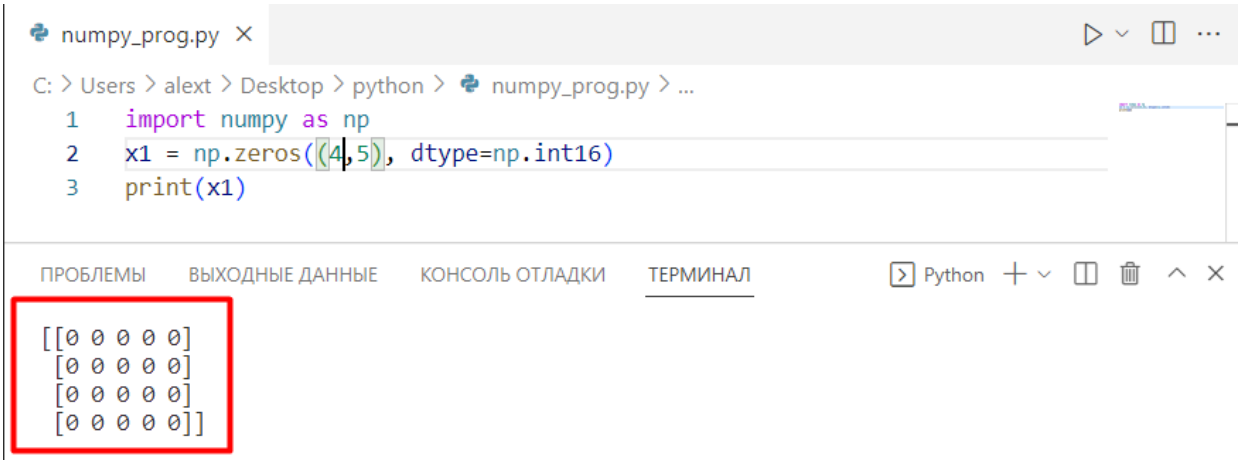


Рисунок 5.18

Функція numpy.ones() застосовується для створення матриці, що складається з одних одиниць. Крім того, ви можете скористатися цією функцією у різних статистичних завданнях.

Синтаксис буде наступним:

np.ones(shape, dtype=float, order='C')

У цьому синтаксисі shape - розмірність масиву, dtype - тип даних (зазначається опціонально, за замовчуванням заданий тип float64), order визначає порядок зберігання масиву в пам'яті. Order вказується опціонально; за умовчанням заданий параметр 'C', що означає построкове зберігання. Можна поміняти на F і зберігати по колонках.

Приклад:

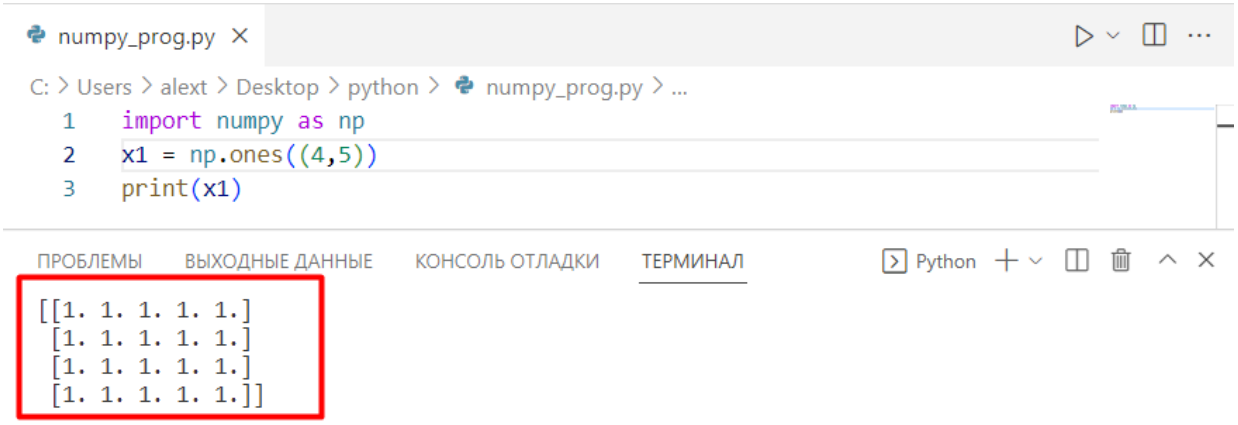


Рисунок 5.19

Функції numpy.hstack() та numpy.vstack()

За допомогою функції hstack() ви можете приєднувати дані горизонтальної осі. Це дуже зручна функція в NumPy. Давайте розберемо її на прикладі:

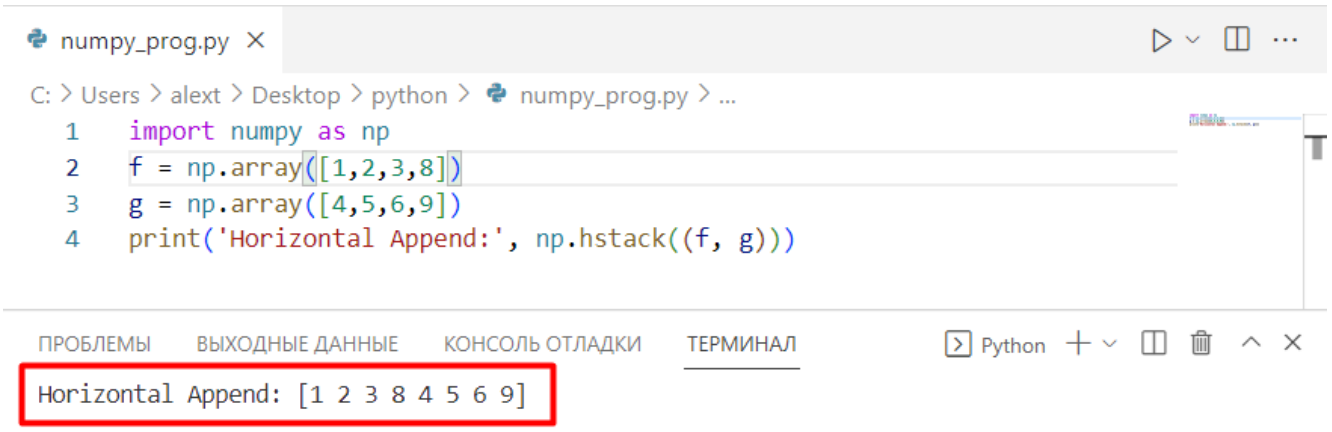


Рисунок 5.20

За допомогою функції vstack() ви можете приєднувати дані вертикальної осі. Давайте розглянемо наступний приклад:

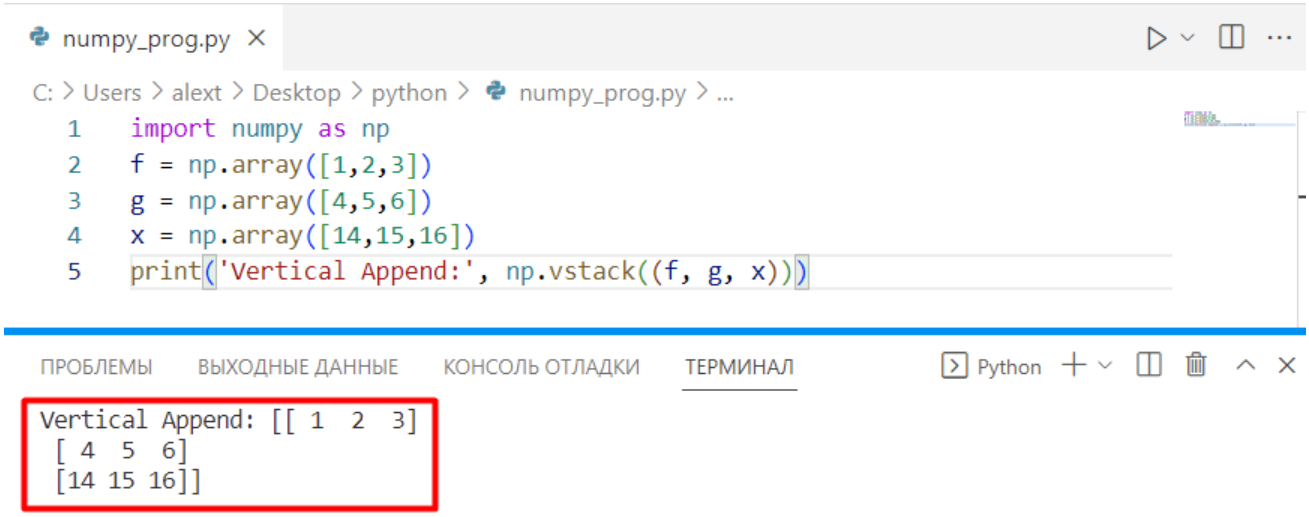


Рисунок 5.21

Генерація випадкових чисел

Для генерації випадкових чисел з розподілом Гауса використовується функція numpy.random.normal(loc, scale, size). loc – це середнє, математичне очікування, scale – стандартне відхилення, size – розмір згенерованого масиву.

Приклад:

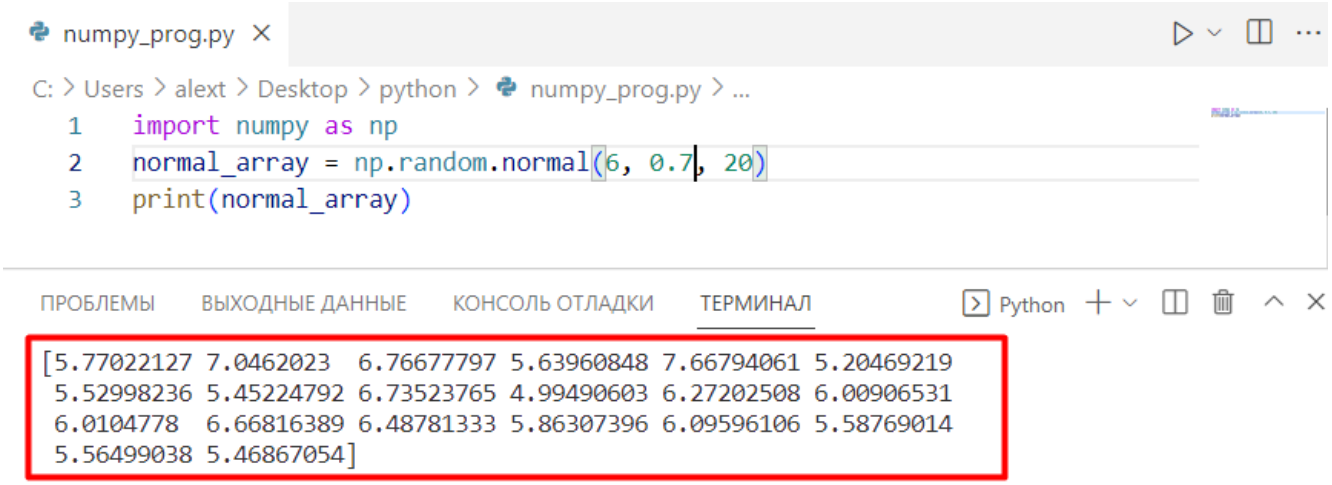


Рисунок 5.22

На графіку цей розподіл матиме такий вигляд:

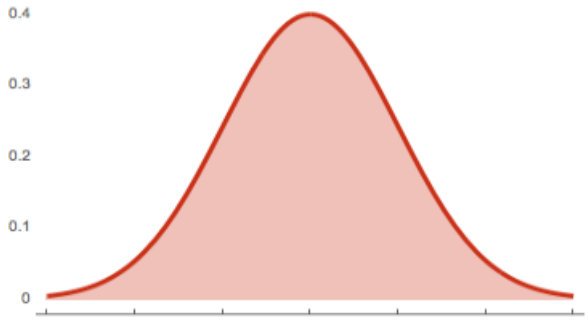


Рисунок 5.23

Функції numpy.reshape() та numpy.flatten().

У деяких випадках вам необхідно змінити форму масиву даних. Для цього використовується функція reshape(). Синтаксис буде наступним:

numpy.reshape(a, newShape, order='C')

У цьому синтаксисі:

a - масив, форму якого ви хочете змінити,

newShape - бажана форма,

order - визначає порядок зберігання масиву в пам'яті (вказується опціонально). За замовчуванням встановлено параметр 'C', що означає построкове зберігання. Можна змінити на 'F' і зберігати по колонках, у стилі Fortran.

Приклад:

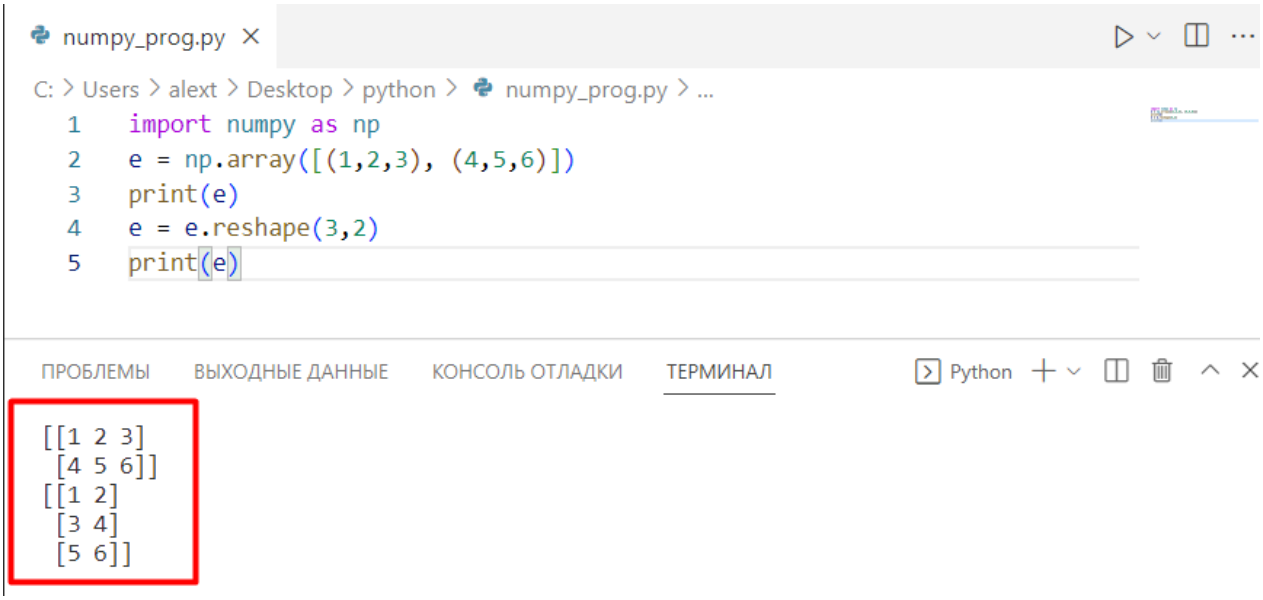


Рисунок 5.24

Функція flatten().

Працюючи з нейронними мережами, наприклад з згортковими мережами, часто потрібно перетворювати масиви в одномірний вектор. Для цього можна використовувати функцію flatten(). Синтаксис у неї наступний:

np.flatten(order='C')

Тут order – опціональний параметр. За замовчуванням задається 'C', при цьому перетворення на вектор буде проводитися рядково.

Приклад:

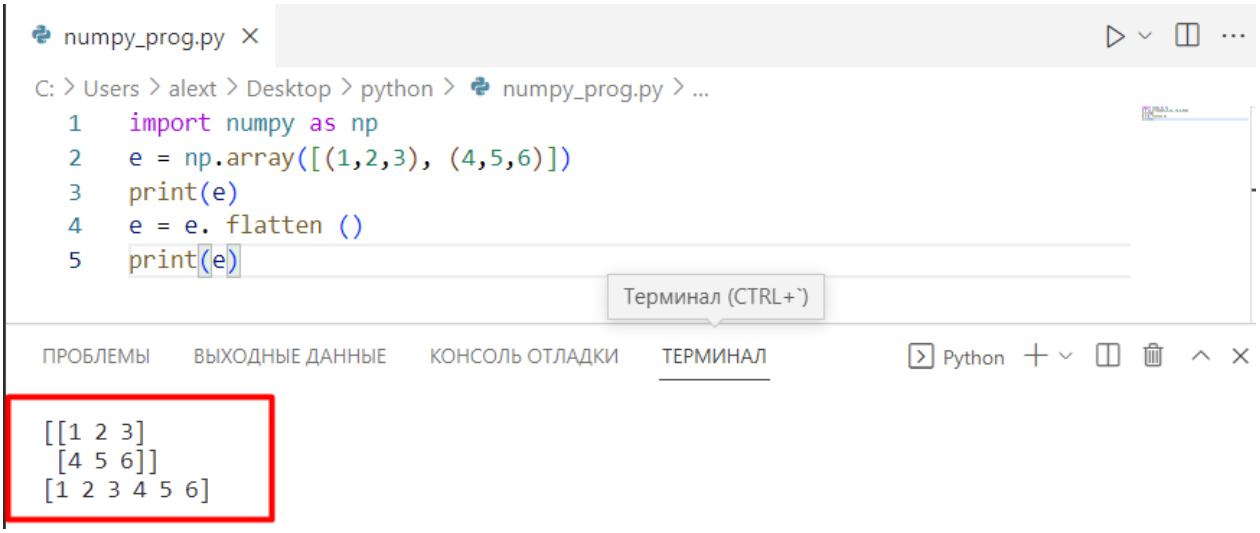


Рисунок 5.25

Функція numpy.arange()

np.arange() — це вбудована в бібліотеку NumPy функція, яка повертає об'єкт типу ndarray, що містить рівномірно розташовані значення всередині заданого інтервалу. Наприклад, щоб створити значення від одного до десяти, ви можете скористатися функцією np.arange().

Синтаксис:

np.arange(start, stop, step)

Тут start – початок інтервалу, stop – кінець інтервалу, step – інтервал між значеннями (за замовчуванням дорівнює 1, у прикладі 3).

Приклад:

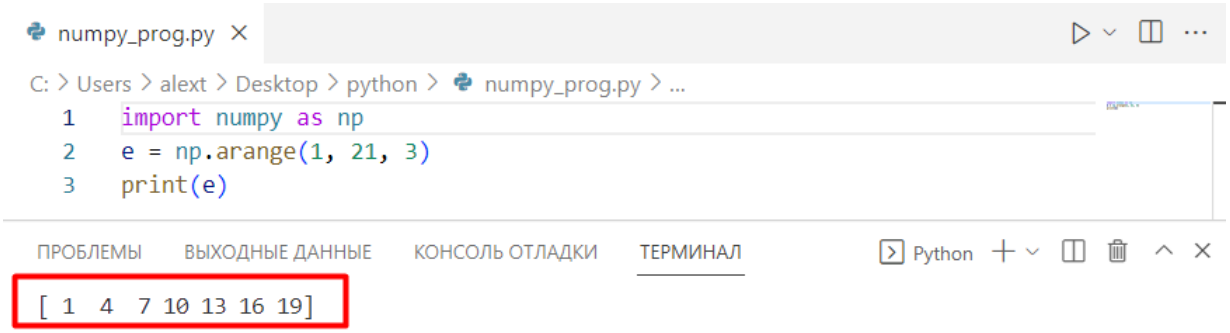


Рисунок 5.26

Функція numpy.asarray().

Функція asarray() використовується, коли необхідно конвертувати вхідні дані в масив. Вхідними даними можуть бути списки, кортежі, масиви numpy (ndarray) та ін.

Синтаксис наступний:

np.asarray(data, dtype=**None**, order=**None**)[source]

У цьому синтаксисі: data — дані, які потрібно конвертувати, dtype — необов'язковий аргумент для вказання типу даних (якщо не заданий, то тип даних встановлюється виходячи з вхідних даних), order визначає порядок зберігання масиву у пам'яті. Вказується опціонально: за умовчанням заданий параметр 'C', що означає построкове зберігання. Можна змінити на 'F' і зберігати по колонках, у стилі Fortran.

Приклад:

Розглянемо двовимірну матрицю розміром 5X5, заповнену одиницями.

Якщо ви забажаєте змінити значення цієї матриці, у вас нічого не вийде. Тому що копію змінити неможливо.

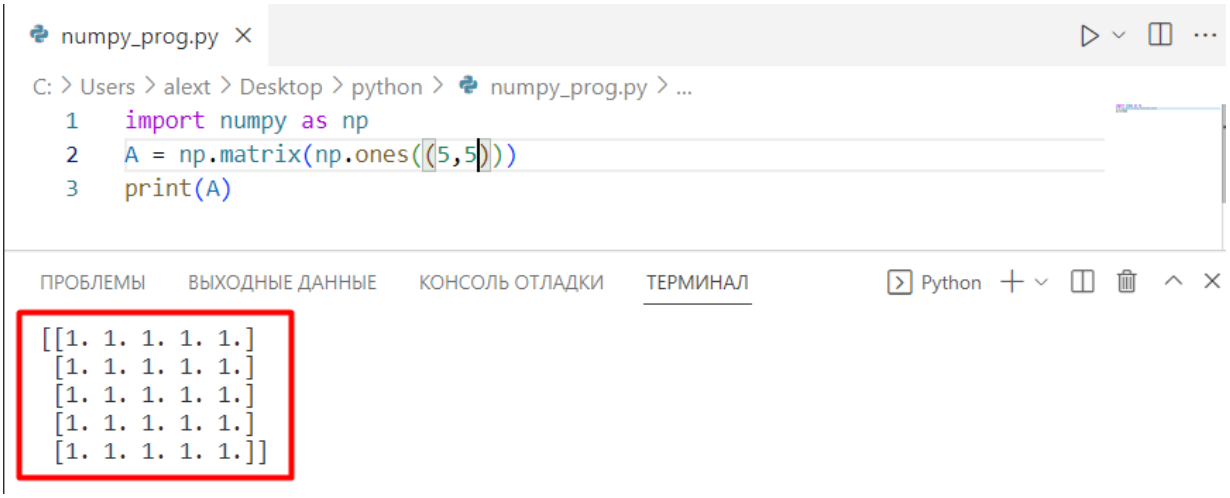


Рисунок 5.27

Матриця - це об'єкт незмінного типу. Для внесення змін можна використовувати функцію asarray(). Давайте подивимося, чи зміниться дані, якщо ми, наприклад, замінимо значення даних другого рядку значеннями 2.

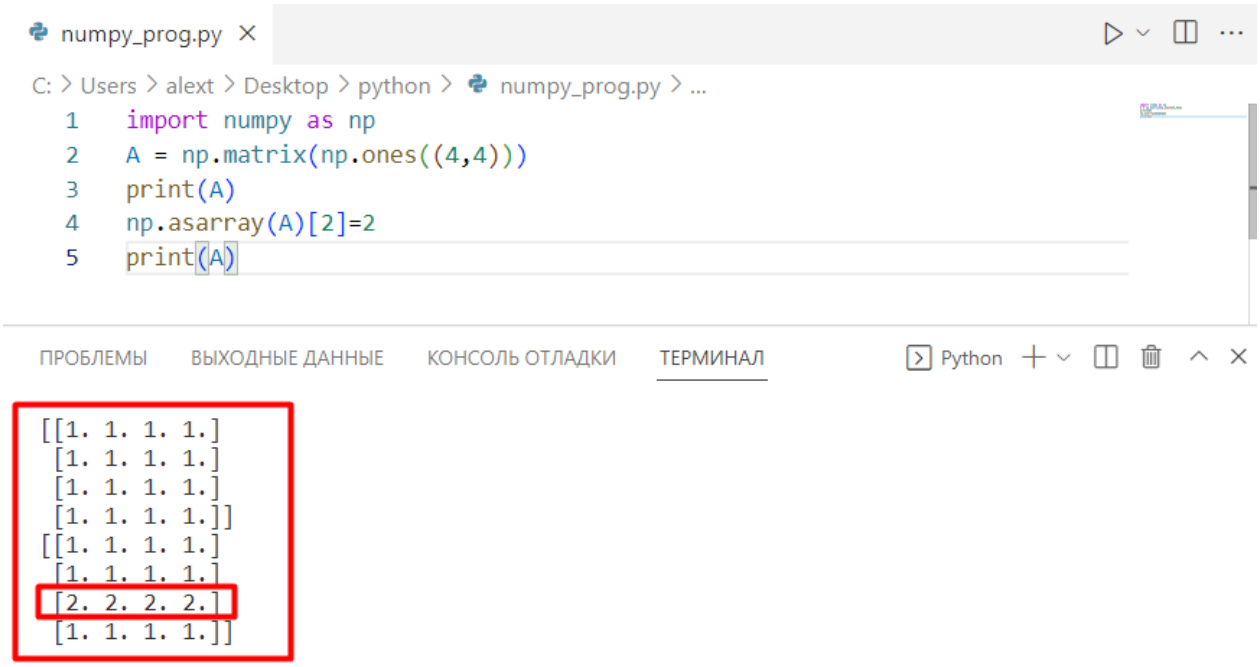


Рисунок 5.28

Пояснення до коду:

np.asarray(A)-конвертуємо матрицю A в масив;

[2]-обираємо третій ряд.

Функції numpy.linspace() та numpy.logspace()

Ця функція створює послідовність даних, рівномірно розташованих на числовій прямій заданому інтервалі.

Синтаксис:

np.linspace(start, stop, num, endpoint)

У цьому синтаксисі: start - початок послідовності, stop - кінець послідовності, num - кількість даних у вибірці (значення за замовчуванням - 50), endpoint - визначення кінця вибірки (якщо True (за замовчуванням), то вибірка закінчується на останньому значенні, якщо False - останнє значення у вибірку не включається).

Приклад

Створимо послідовність із 15 чисел, рівномірно розташованих в інтервалі від 1 до 6.

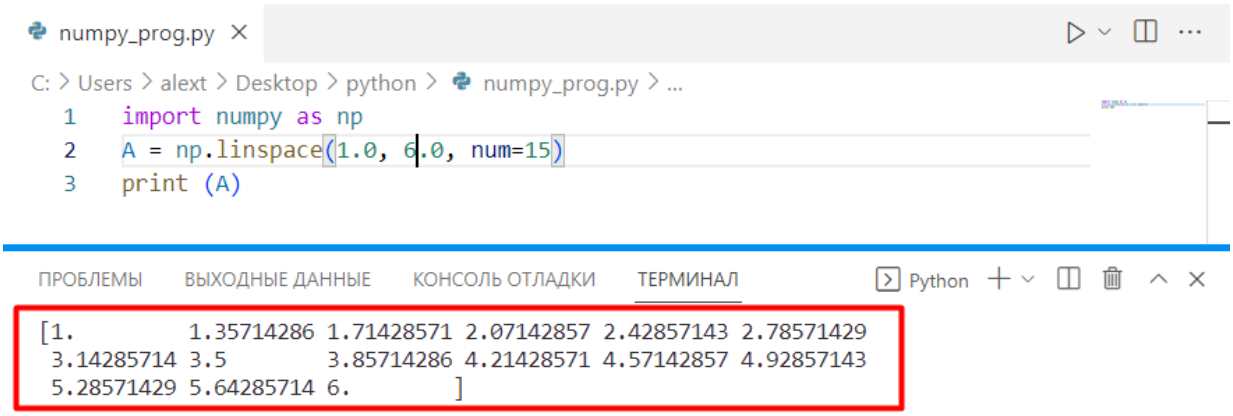


Рисунок 5.29

Функція numpy.logspace()

Функція numpy.logspace() повертає послідовність даних, рівномірно розміщених у заданому інтервалі на числовій прямій у логарифмічному масштабі. Параметри цієї функції точно такі ж, як і у numpy.linspace().

Синтаксис:

np.logspace(start, stop, num, endpoint)

Приклад

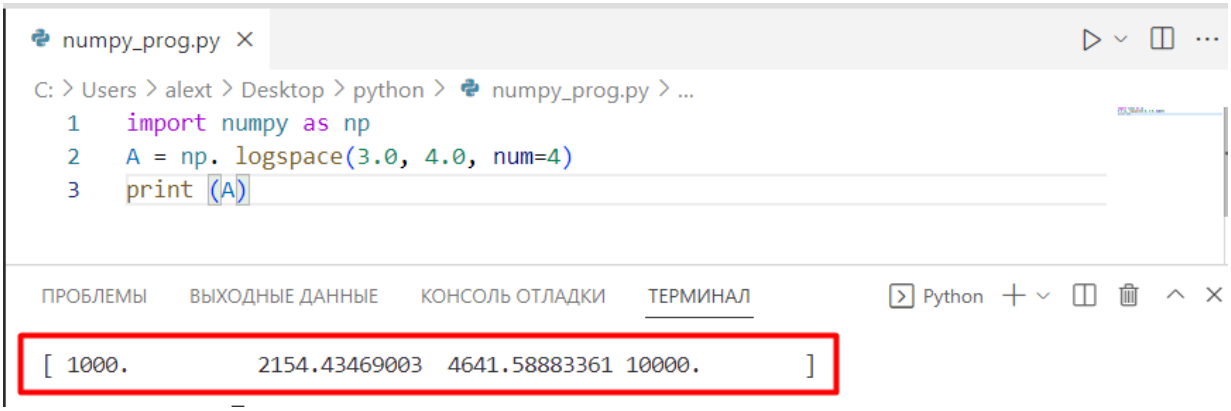


Рисунок 5.30

І ще, якщо вам потрібно дізнатися розмір масиву в байтах, можна використовувати атрибут itemsize.

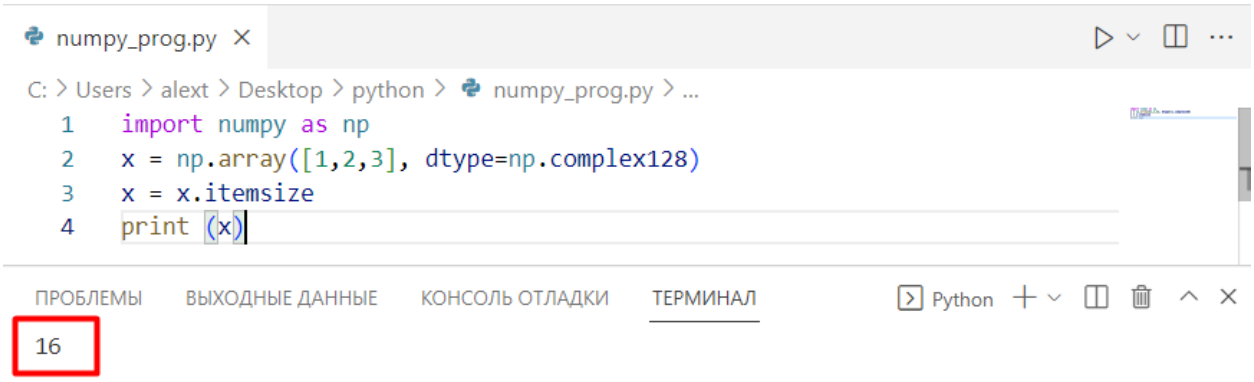


Рисунок 5.31

Елемент x має розмір 16 байтів.

Індекси та зрізи у масивів NumPy.

Робити зрізи даних у NumPy дуже просто. Зараз ми братимемо зрізи матриці 'e'. Зауважимо, що для того, щоб отримувати окремі ряди або колонки, нам потрібно буде використовувати квадратні дужки.

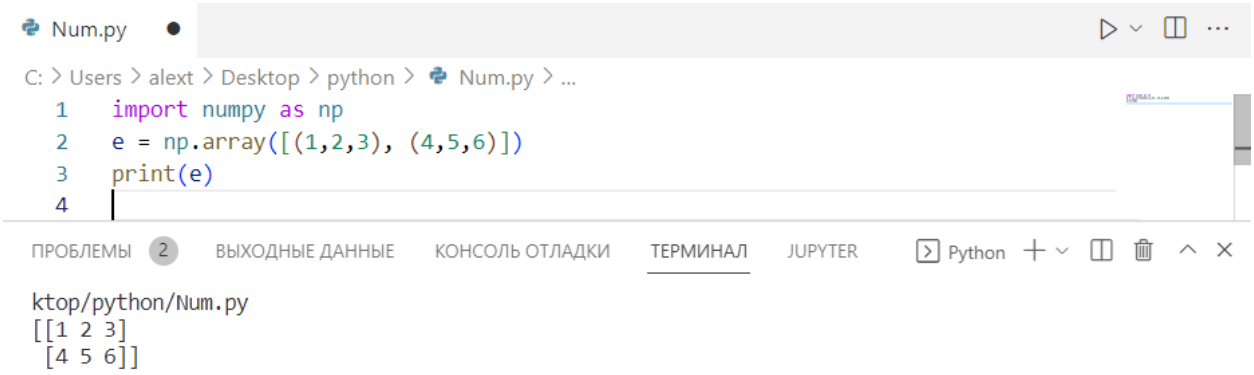


Рисунок 5.32

У NumPy перший елемент масиву/рядка/колонки починається з 0. Перший рядок - print('First row:', e[0]). Другий рядок - print('Second row:', e[1]).

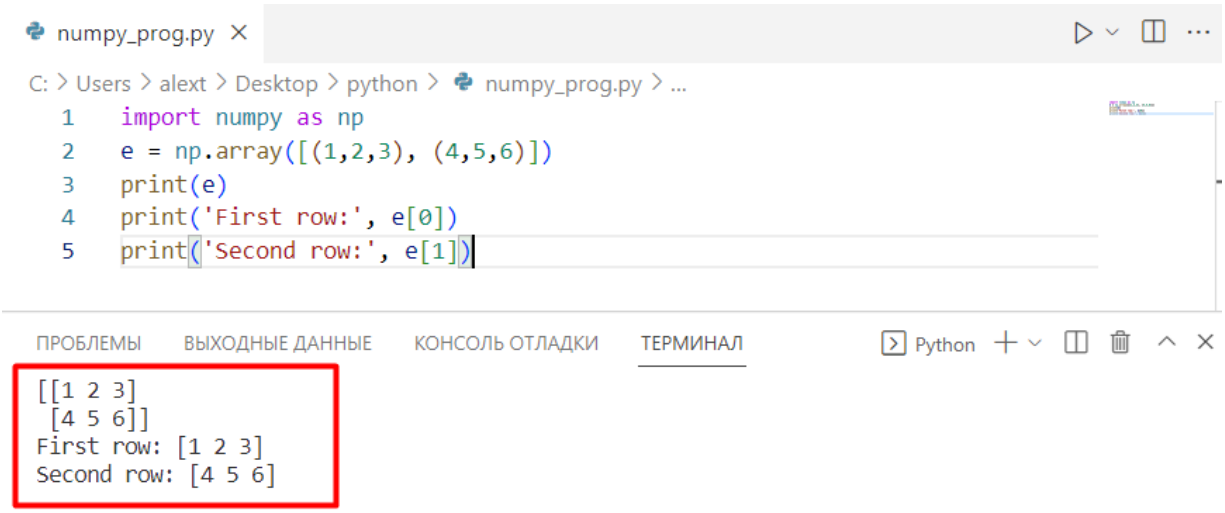


Рисунок 5.33

У Python, як і в багатьох інших мовах:

1. значення перед комою відноситься до рядів;
2. значення після коми відноситься до колонок;
3. якщо ви хочете вибрати всю колонку, вам треба додати: перед індексом; двокрапка означає, що ви хочете вибрати всі ряди з цієї колонки.

Приклад

print('Second column:', e[:,1])

Second column: [2 5]

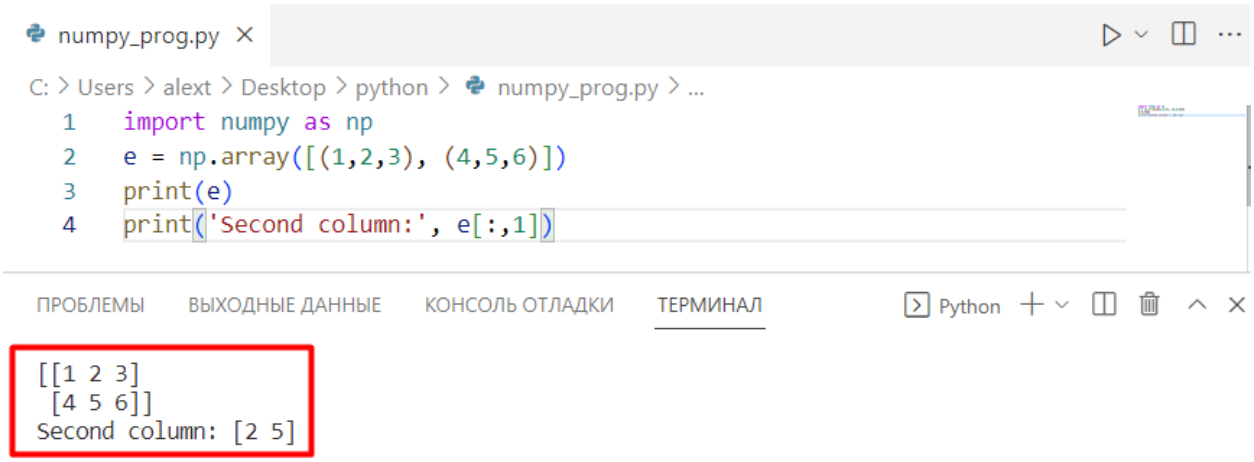


Рисунок 5.34

Візьмемо перші два значення третього ряду. Ми використовуємо “:”, щоб отримати дані з усіх колонок до другої.

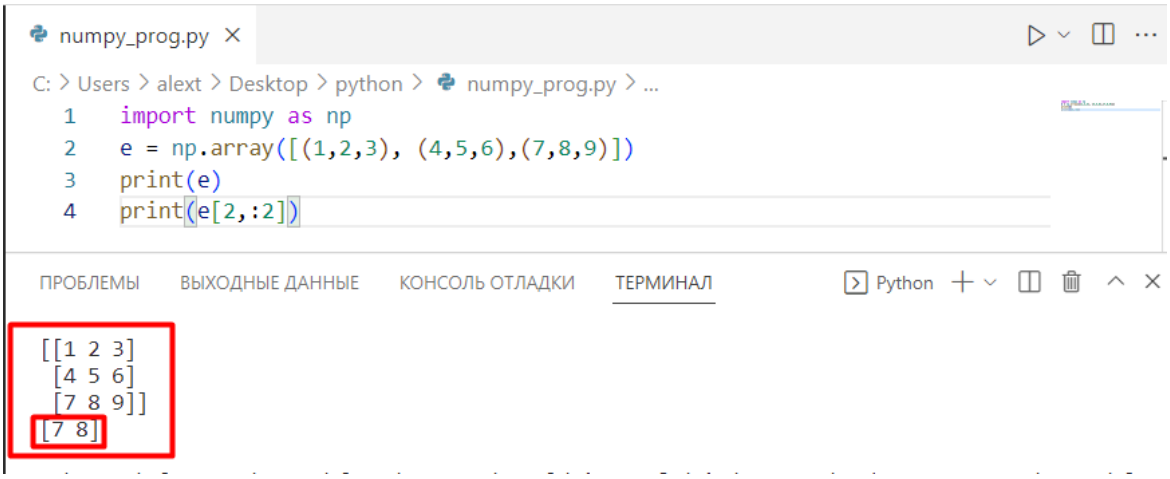


Рисунок 5.35

Статистичні функції в NumPy з прикладами.

У NumPy є багато корисних статистичних функцій. Наприклад, функції максимуму, мінімуму, стандартного відхилення та інші. Нижче наведено список цих функцій.

Функція мінімуму - np.min()

Функція максимуму - np.max()

Функція середнього - np.mean()

Функція медіани - np.median()

Функція стандартного відхилення - np.std()

Зверніть увагу на наступний масив:

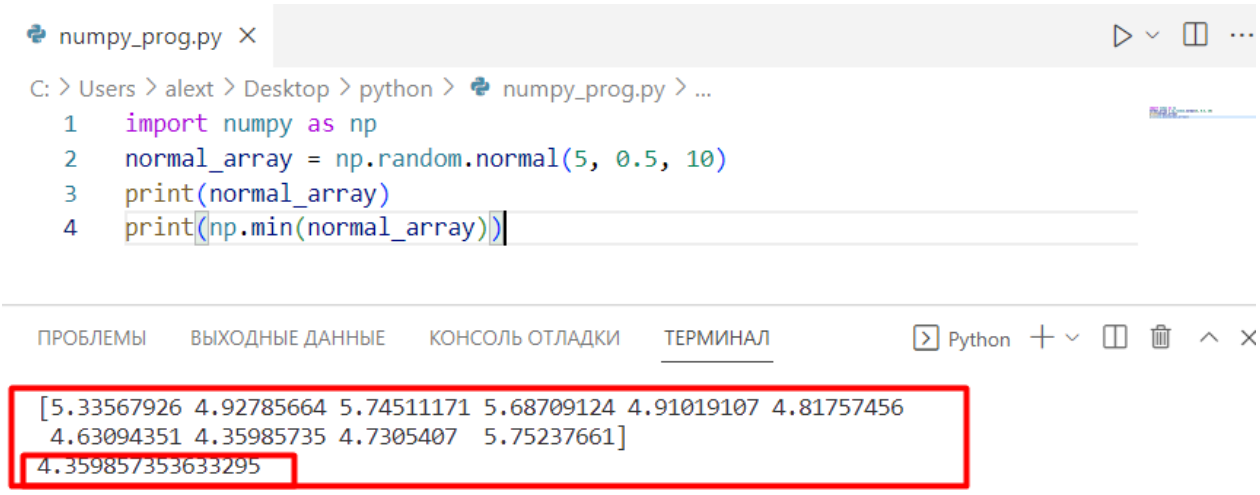


Рисунок 5.35

Приклади роботи статистичних функцій

Min це print(np.min(normal_array))

Max це print(np.max(normal_array))

Mean це print(np.mean(normal_array))

Median це print(np.median(normal_array))

Sd це print(np.std(normal_array))

Функція numpy.dot(): скалярний добуток у NumPy

numpy.dot() – це потужний набір функцій для матричних обчислень. Наприклад, за допомогою numpy.dot() ви можете обчислити скалярний добуток двох векторів. Крім цього, numpy.dot() може працювати з двовимірними масивами і виробляти матричне множення.

Синтаксис:

np.dot(x, y, out=None)

Тут x, y - вхідні масиви (вони можуть бути як одновимірними, так і двовимірними), a out - тип вихідних даних (вказується опціонально). Якщо на вході одновимірні масиви, то на виході скаляр. Якщо ж масиви двовимірні, то на виході теж двомірний масив (прим. перекладача: зазвичай цей параметр не задається, але його можна використовувати з метою покращення продуктивності).

Приклад:

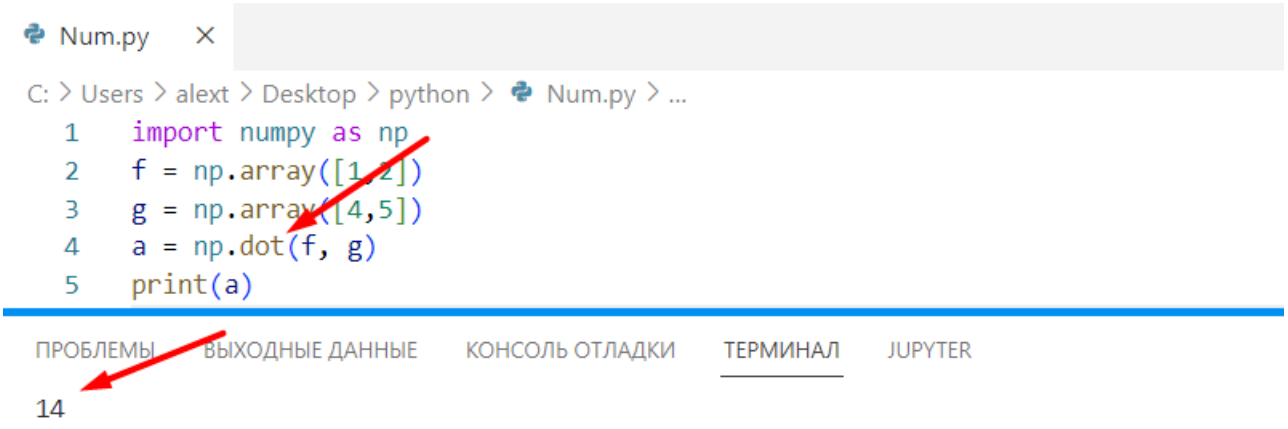


Рисунок 5.36

Матричне множення

Для матричного множення двовимірних масивів використовується функція numpy.matmul(). Працює вона так:

У разі двовимірних масивів проводиться звичайне матричне множення.

Якщо розмірність більша за 2, то множення інтерпретується як поєднання двох масивів. Якщо один з масивів одномірний, його розмірність підвищується до двовимірного, а потім проводиться матричне множення.

np.matmul(x, y, out=None)

У цьому синтаксисі x, y - вхідні масиви (скалярні величини неприпустимі), а out - опціональний параметр (зазвичай тип вихідних даних визначається як ndarray).

Приклад:

Отже, помножимо дві матриці за допомогою np.matmul().

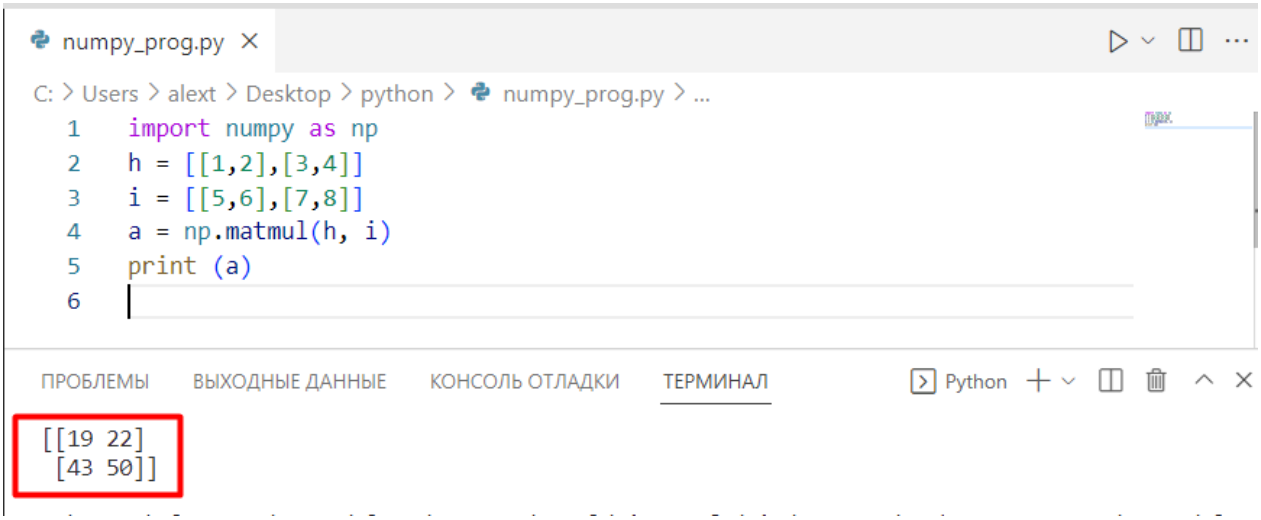


Рисунок 5.37

Детермінант.

І останнє, але важливе. Якщо вам потрібно визначити детермінант матриці, то ви можете скористатися функцією np.linalg.det(). Зверніть увагу, що в цій функції перевіряться розмірність (детермінант можна обчислити лише у квадратної матриці).

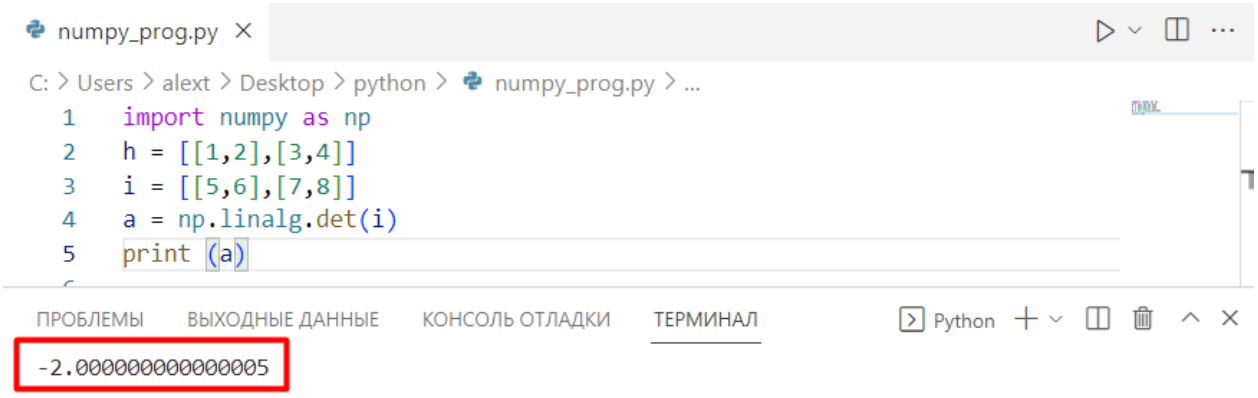


Рисунок 5.38

Практичне завдання.

1. Напишіть програму генерації матриці 5 на 5 виду

$$\begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{vmatrix}$$

та виріжіть з неї:

a. Другу строку матриці.

b. Третій стовбець матриці.

c. Матрицю $\begin{vmatrix} 14 & 15 \\ 19 & 20 \\ 24 & 25 \end{vmatrix}$.

2. Згенеруйте двовимірну матрицю розміром 5X5, заповнену одиницями.

a. Замінить значення даних третього ряду цифрою 2.

b. Вивести третій рядок.

c. Вивести другий стовбець.

3. Створіть послідовність з 20 чисел, рівномірно розташованих в інтервалі від 1 до 10 та виведіть на екран.

4. Створіть послідовність з 20 чисел рівномірно розміщених у заданому інтервалі від 1 до 10 на числовій прямій у логарифмічному масштабі.

5. Згенеруйте масив зі 30 чисел від 1 до 100.

a. Вивести максимальне число.

b. Вивести мінімальне число.

6. Знайдіть детермінант матриці $\begin{vmatrix} 1 & 2 & 4 & 7 \\ 8 & 10 & 12 & 16 \\ 6 & 7 & 8 & 1 \\ 5 & 3 & 9 & 2 \end{vmatrix}$.

Практична робота №6

Тема. Бібліотека Matplotlib.

Мета роботи. **Опанувати бібліотеку Matplotlib на мові програмування Python.**

Зміст.

Вивчення відомостей про бібліотеку Matplotlib.

Виконання роботи.

Отримання результату.

Хід роботи

Бібліотека Matplotlib у Python. Бібліотека matplotlib у Python допомагає відображати дані на графіках у простішому вигляді. Matplotlib допомагає будувати графіки та візуалізувати фігури для представлення даних.

Встановлення бібліотеки

Для встановлення бібліотеки Matplotlib треба відкрити термінал та використати систему керування пакетами pip.

pip install matplotlib

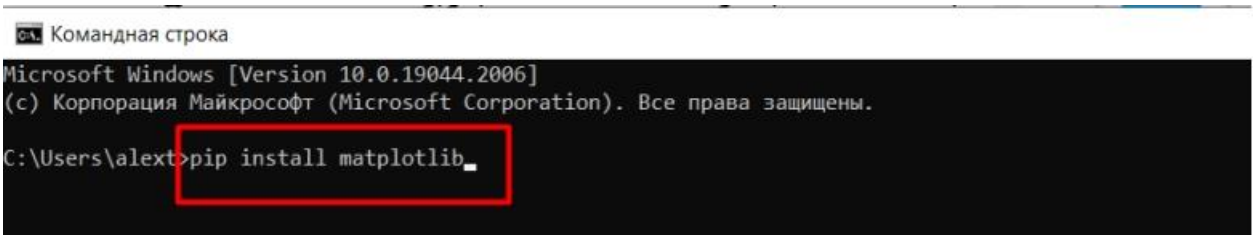


Рисунок 6.1

Найпростіші графічні команди:

plt.scatter() – маркер або точкове малювання;

plt.plot() - ламана лінія;

plt.text() - нанесення тексту;

Будемо використовувати два списки Python у якості джерела даних для точкової графіки.

Приклад:

Num.py

C: > Users > alext > Desktop > python > Num.py > ...

```
1  import matplotlib.pyplot as plt
2  year = [1950, 1975, 2000, 2018, 2022]
3  population = [2.12, 3.681, 5.312, 6.981, 8.015]
4  plt.plot(year, population)
5  plt.show()
6
```

Рисунок 6.2

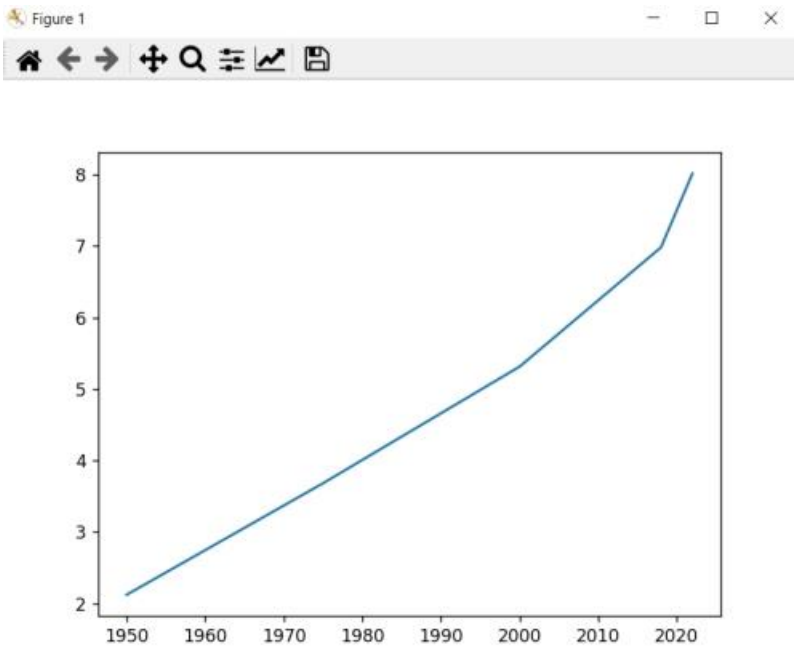


Рисунок 6.3

Зверніть увагу на кнопки з піктограмами у верхньому рядку вікна графіка. Справа в тому, що саме вікно інтерактивно, і, поки воно не закрите, можна масштабувати графік на власний розсуд. Після натискання на кнопку

з'являється курсор у формі хреста, за допомогою якого на полі графіка можна лівою клавішею миші виділити прямокутну область, яка буде збільшена. Після натискання на  за допомогою миші можна зрушувати зображення графіка. За допомогою кнопки  можна повернутися до вихідного вигляду графіка. Клавіші  використовуються для руху в історії створених видів. Клавіша  відкриває вікно налаштування розташування поля графіка у графічному вікні. І, нарешті, клавіша  викликає діалог збереження зображення у файл на диску у вибраному форматі (png, pdf, eps, svg та ін.).

За допомогою команд: plt.xlabel('Year'), plt.ylabel('Population') та plt.title('World Population') надписуємо осі та назву графіку.

Num.py

C: > Users > alext > Desktop > python > Num.py > ...

```
1  import matplotlib.pyplot as plt
2  year = [1950, 1975, 2000, 2018, 2022]
3  population = [2.12, 3.681, 5.312, 6.981, 8.015]
4  plt.xlabel('Year')
5  plt.ylabel('Population')
6  plt.title('World Population')
7  plt.plot(year, population)
8  plt.show()
9
```

Рисунок 6.4

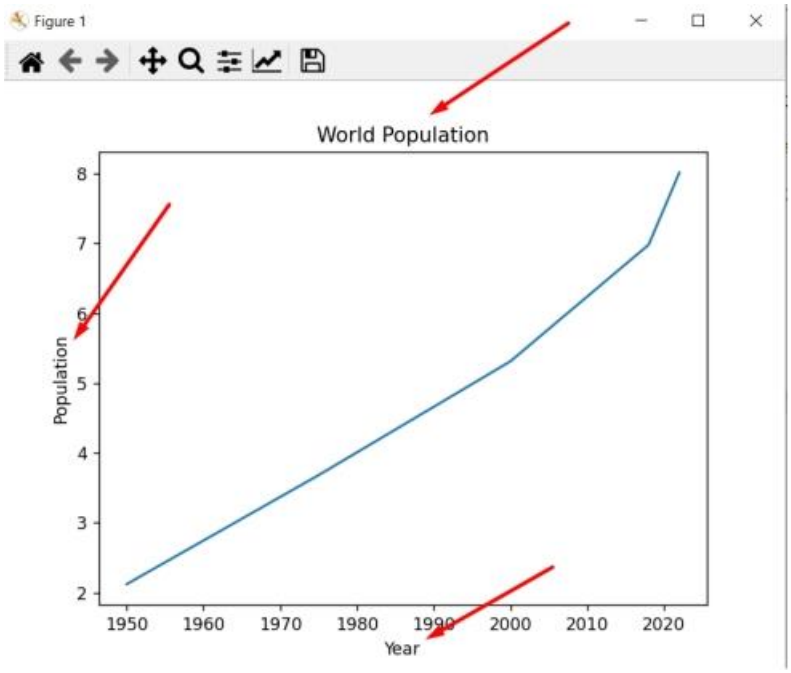


Рисунок 6.5

Діаграма розсіювання.

Графік показував точки, які фактично не були передані в масиві, оскільки він показує лінію. Що, якщо ми хочемо бачити тільки фактичні точки на графіку, використовуємо команду plt.scatter(year, population).

Num.py

C: > Users > alext > Desktop > python > Num.py > ...

```
1  import matplotlib.pyplot as plt
2  year = [1950, 1975, 2000, 2018, 2022]
3  population = [2.12, 3.681, 5.312, 6.981, 8.015]
4  plt.xlabel('Year')
5  plt.ylabel('Population')
6  plt.title('World Population')
7  plt.scatter(year, population)
8  plt.show()
9
```

Рисунок 6.6

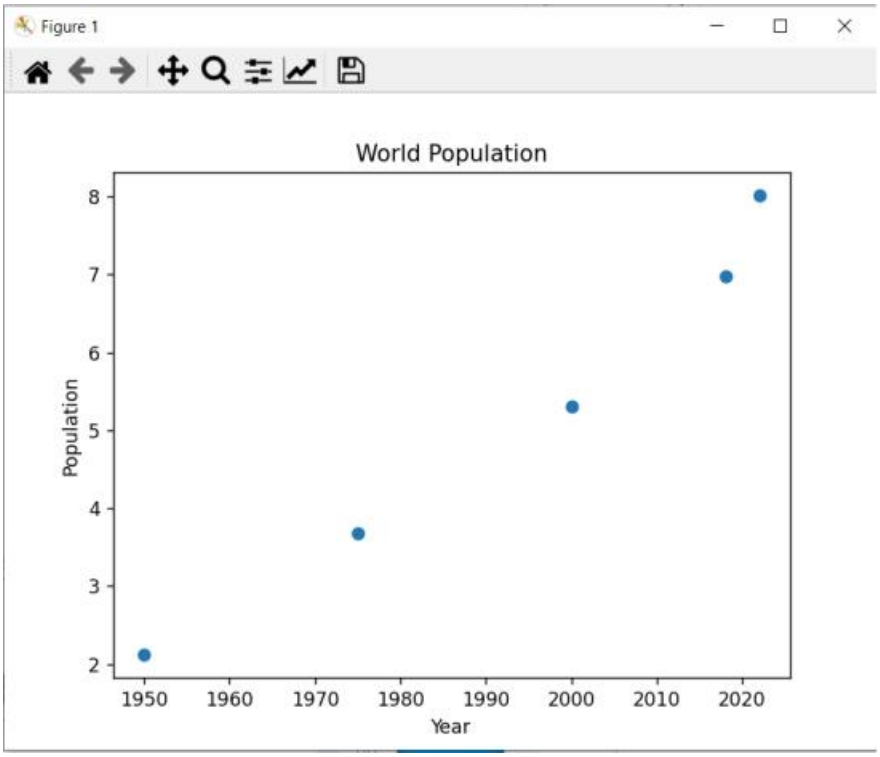


Рисунок 6.7

Гістограмми та діаграми.

plt.bar(), plt.barh(), plt.barbs(), broken_barh() - стовпчаста діаграма;

plt.hist(), plt.hist2d(), plt.hlines - гістограма;

plt.pie() – кругова діаграма;

plt.boxplot() - "ящик з вусами" (boxwhisker);

plt.errorbar() - оцінка похибки, "вуса".

В цьому розділі ми познайомилися з гістограмами. В той час як графіки інформують нас про те, як змінюються наші дані, гістограма описує, як наші дані розраховуються. Чим більше значень у діапазоні, тим вище смуга діапазону. Ми використовуємо функцію hist() для побудови гістограми. У нього є 2 важливі параметри: список значень для побудови; кількість діапазонів для розподілу цих точок. Продемонструємо це за допомогою фрагмента коду:

Num.py

C: > Users > alext > Desktop > python > Num.py > ...

```
1  import matplotlib.pyplot as plt
2  values = [0, 1.2, 1.3, 1.9, 4.3, 2.5, 2.7, 4.3, 1.3, 3.9]
3  plt.hist(values, bins = 4)
4  plt.show()
5
```

Рисунок 6.8

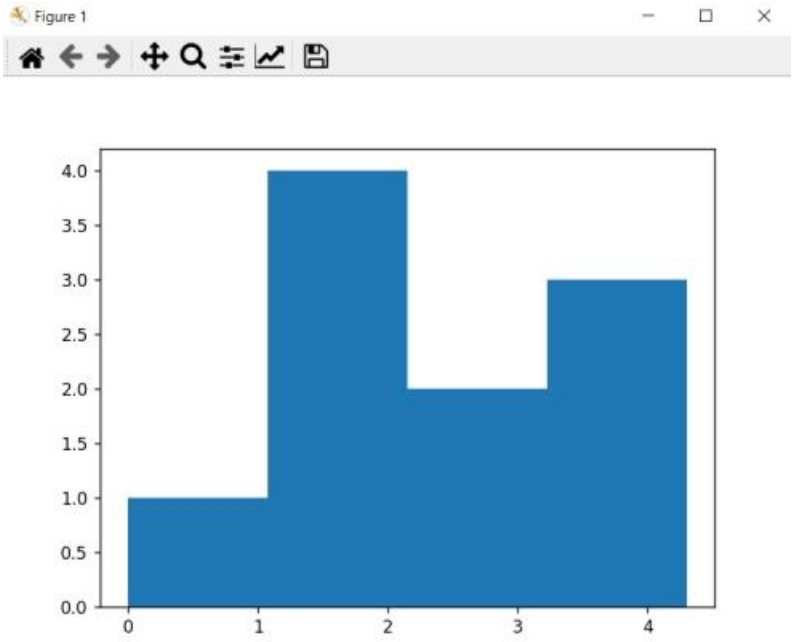


Рисунок 6.9

Num.py ● гра 2 — копия.py ●

C: > Users > alext > Desktop > python > Num.py > ...

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  x = [1, 2, 3, 4, 5]
4  z1 = [10, 17, 24, 16, 22]
5  fig = plt.figure()
6  plt.bar(x, z1)
7  plt.show()
```

Рисунок 6.10

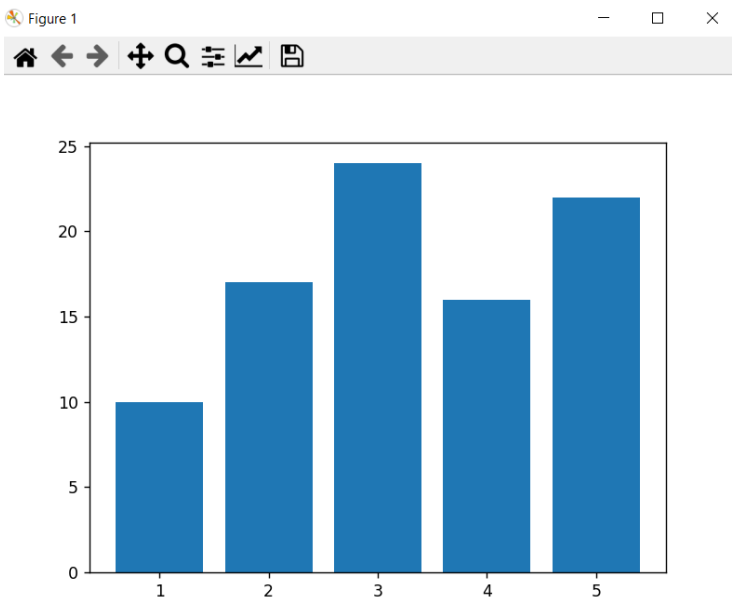


Рисунок 6.11

Малювання кількох кривих

Цілком поширене малювання кількох кривих на одному графіку для порівняння.

Num.py

C: > Users > alext > Desktop > python > Num.py > ...

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
4  cos, sin = np.cos(X), np.sin(X)
5  plt.plot(X, cos)
6  plt.plot(X, sin)
7  plt.show()
8
```

Рисунок 6.12

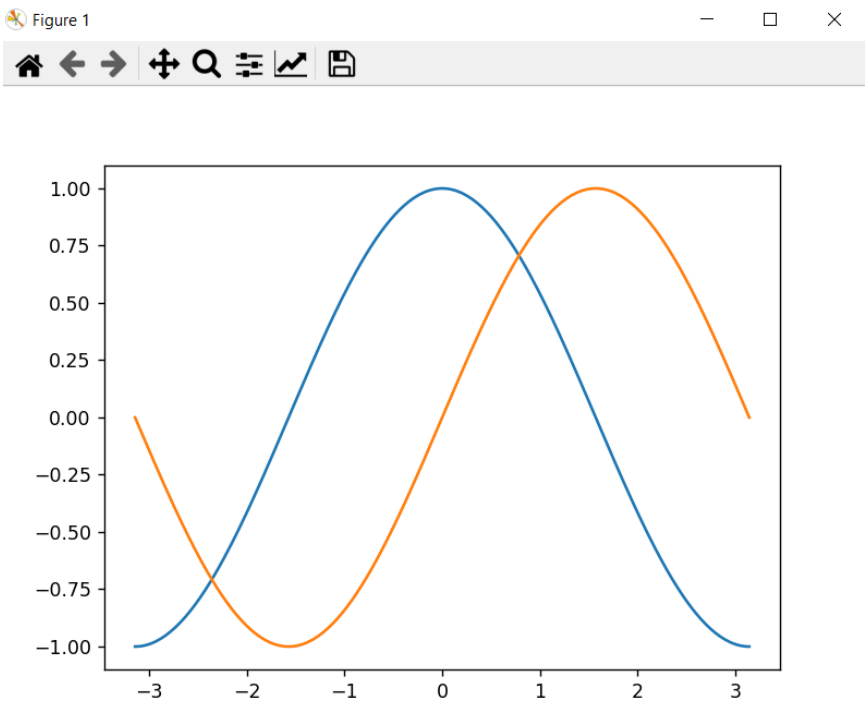


Рисунок 6.13

Якщо ми хочемо змінити їх колір і показати, що представляє кожен колір, треба додати команди: plt.plot(X, cos, color='blue', label="cosine"), plt.plot(X, sin, color='red', label="sine"), plt.legend(loc='upper left', frameon=False).

Num.py

C: > Users > alext > Desktop > python > Num.py > ...

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
4  cos, sin = np.cos(X), np.sin(X)
5  plt.plot(X, cos, color='blue', label="cosine")
6  plt.plot(X, sin, color='red', label="sine")
7  plt.legend(loc='upper left', frameon=False)
8  plt.show()
9
```

Рисунок 6.14

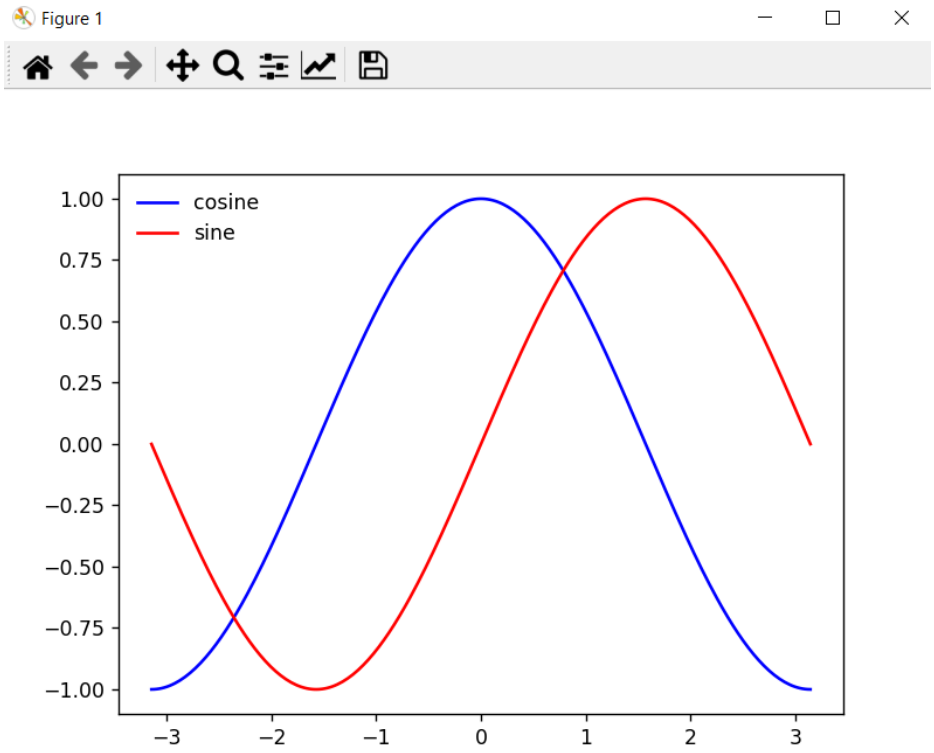


Рисунок 6.15

Змінено колір кривих, щоб спростити порівняння; додано в рамку написи, в яких вказано, які кольори представляють функції. Це полегшує читання метаданих на графіку.

Створення кругової діаграми

Ми можемо створювати привабливі кругові діаграми за допомогою простого фрагмента коду:

numpy_prog.py

C: > Users > alext > Desktop > python > numpy_prog.py > ...

```
1  import matplotlib.pyplot as plt
2  names = ('Nata', 'Dick', 'Alex', 'Jill', 'Meredith', 'George')
3  speed = [10, 7, 32, 4, 3, 2]
4  colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'red', 'blue']
5  explode = (0.1, 0, 0, 0, 0, 0) # explode 1st slice
6  plt.pie(speed, explode=explode, labels=names, colors=colors,  autopct='%1.1f%%', shadow=True, startangle=140)
7  plt.axis('equal')
8  plt.show()
```

Рисунок 6.16

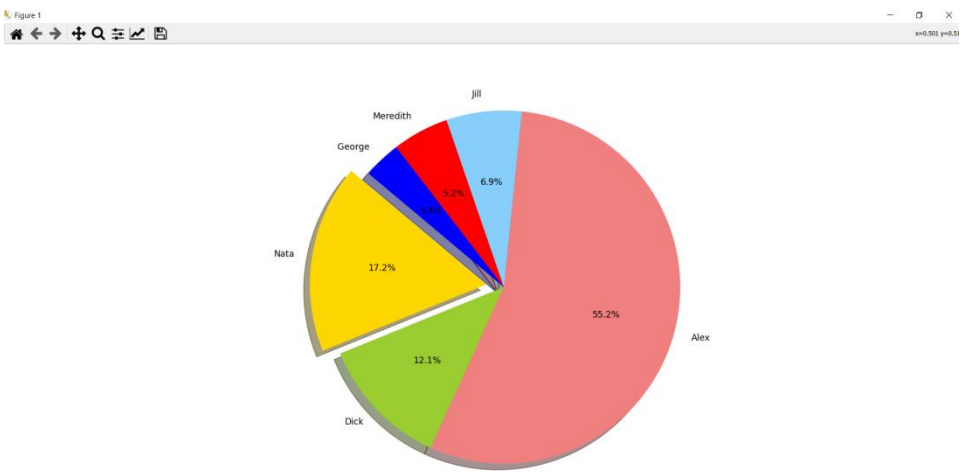


Рисунок 6.17

Створення теплових карт

Графіки – це круто, але коли справа доходить до візуалізації географічної інформації, немає нічого кращого, ніж теплова карта:

Num.py

C: > Users > alext > Desktop > python > Num.py > ...

```
1   import numpy as np
2   import numpy.random
3   import matplotlib.pyplot as plt
4   temperature = np.random.randn(4096)
5   anger = np.random.randn(4096)
6   heatmap, xedges, yedges = np.histogram2d(temperature, anger, bins=(64,64))
7   extent = [xedges[0], xedges[-1], yedges[0], yedges[-1]]
8   plt.clf()
9   plt.ylabel('Anger')
10  plt.xlabel('Temp')
11  plt.imshow(heatmap, extent=extent)
12  plt.show()
13
```

Рисунок 6.17

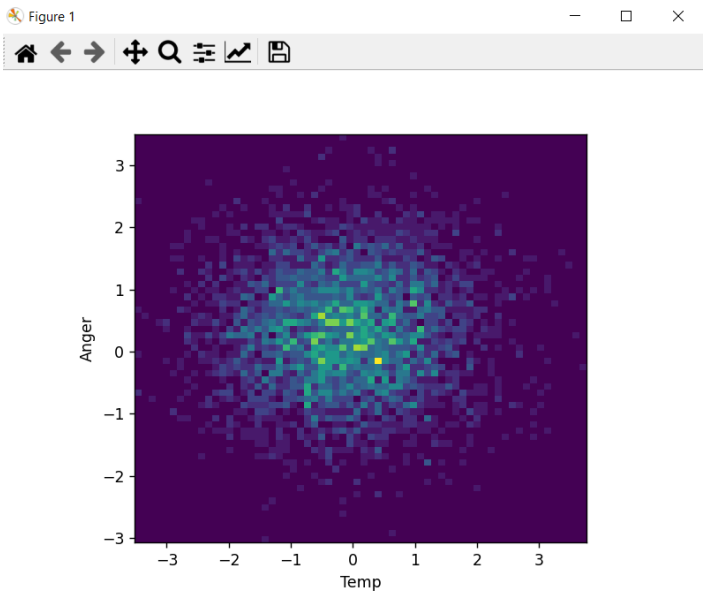


Рисунок 6.18

Нижче представлено найпростішу програму, яка будує графік функції sin(x)/x на інтервалі від –10.

Num.py

C: > Users > alext > Desktop > python > Num.py > ...

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  X = np.arange(-10, 10, 0.1)
4  Y = np.sin(2*X)/X
5  plt.plot(X, Y)
6  plt.show()
7
```

Рисунок 6.19

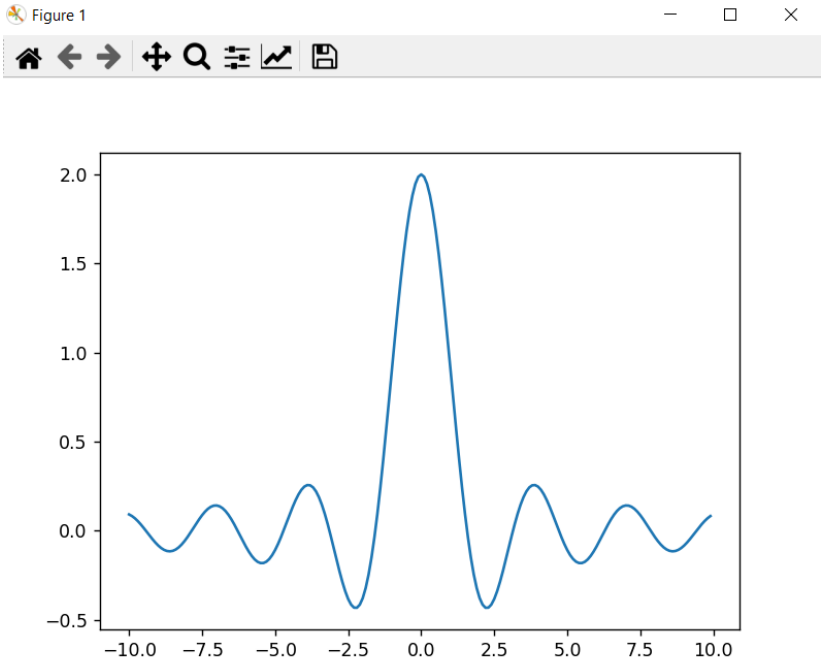


Рисунок 6.20

У рядках 1 та 2 імпортуються бібліотеки matplotlib та numpy. У рядку 3 створюється одномірний numpy-масив з значень, що монотонно збільшуються від –10 до 10 (не включаючи верхню межу) з кроком 0.1. Тобто, масив буде таким: [-10.0, -9.9, -9.8, -9.7, …, 9.7, 9.8, 9.9]. Він містить 200 елементів. У Python є функція range, яка створює перелічення за заданими умовами. Однак, range працює лише з цілими числами, що для цього прикладу неприйнятно. Далі масив Y – це значення нашої функції. У рядку 4 обчислюємо sin(X)/X всім значень X. Саме для такого простого запису нам знадобилася бібліотека numpy. У рядку 5 викликається метод plot з двома аргументами X та Y. Це єдина команда, яка самостійно формує графічне вікно – поле виведення графіка, розраховує масштаб по осях координат та виводить графік функції. Усі налаштування вікна виконані в даному випадку за умовчуванням. Якщо ж ви працюєте у середовищі IDLE Python або в командному рядку Python обов'язково знадобиться рядок 6.

Зауважимо, що бібліотека matplotlib дозволяє використовувати дуже різноманітні способи графічного уявлення даних, у тому числі створювати діаграми різного виду, тривимірні анімовані графіки, по-різному позиціонувати кілька графіків на одному полі виведення, керувати відображенням осей

координат, та ін. Поекспериментуємо з відображення графіка з використанням синтаксису, типового для MATLAB. Змінимо рядок 5 так: plt.plot(X, Y, 'r').

Num.py

C: > Users > alext > Desktop > python > Num.py > ...

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  X = np.arange(-10, 10, 0.1)
4  Y = np.sin(2*X)/X
5  plt.plot(X, Y, 'r')
6  plt.show()
7
```

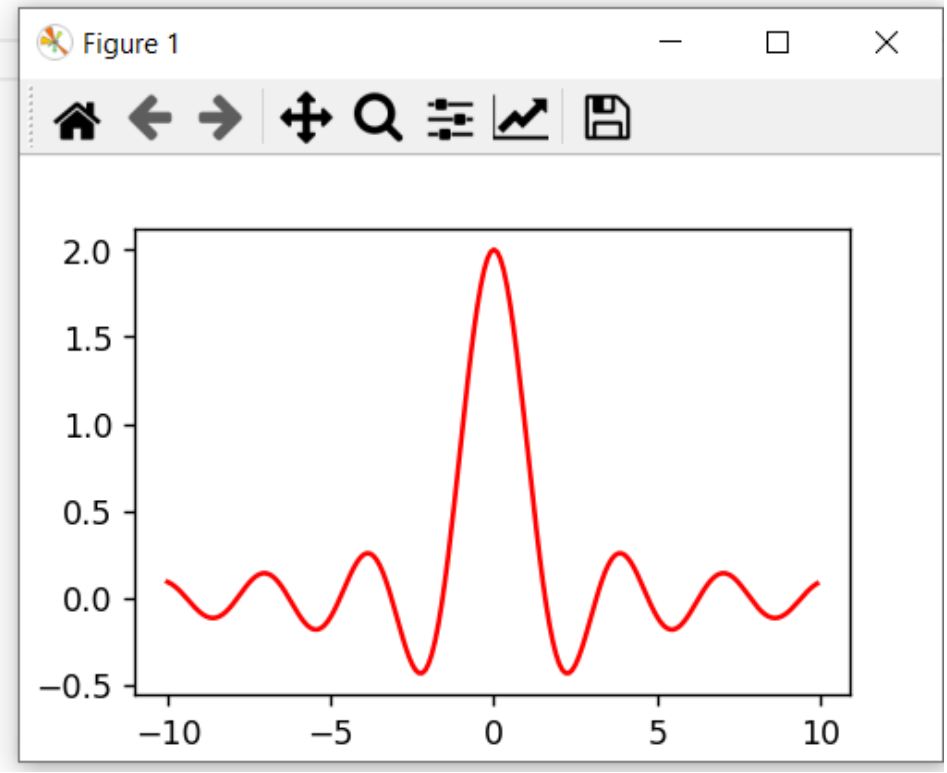


Рисунок 6.21

Отримали криву червоного кольору. Як третій аргумент можливі варіанти y, m, c, r, g, b, k.

Введемо маркери для точок графіка:

plt.plot(X, Y, 'k+')

Лінія зникла, але з'явилися маркери чорного кольору у вигляді хрестиків.

Num.py

C: > Users > alext > Desktop > python > Num.py > ...

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  X = np.arange(-10, 10, 0.1)
4  Y = np.sin(2*X)/X
5  plt.plot(X, Y, 'k+')
6  plt.show()
7
```

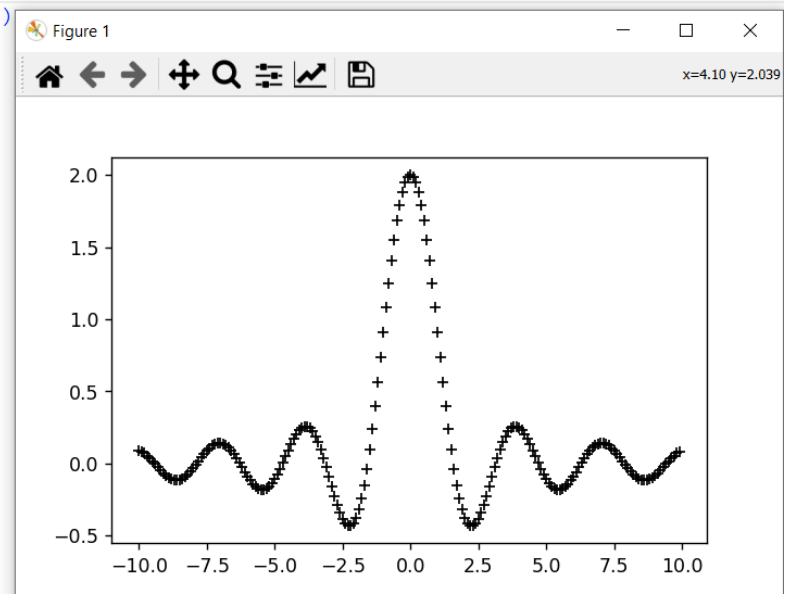


Рисунок 6.22

Можливі варіанти ., o, x, +, *, s.

Пурпурна штрихова лінія може бути задана так:

plt.plot(X, Y, 'm--')

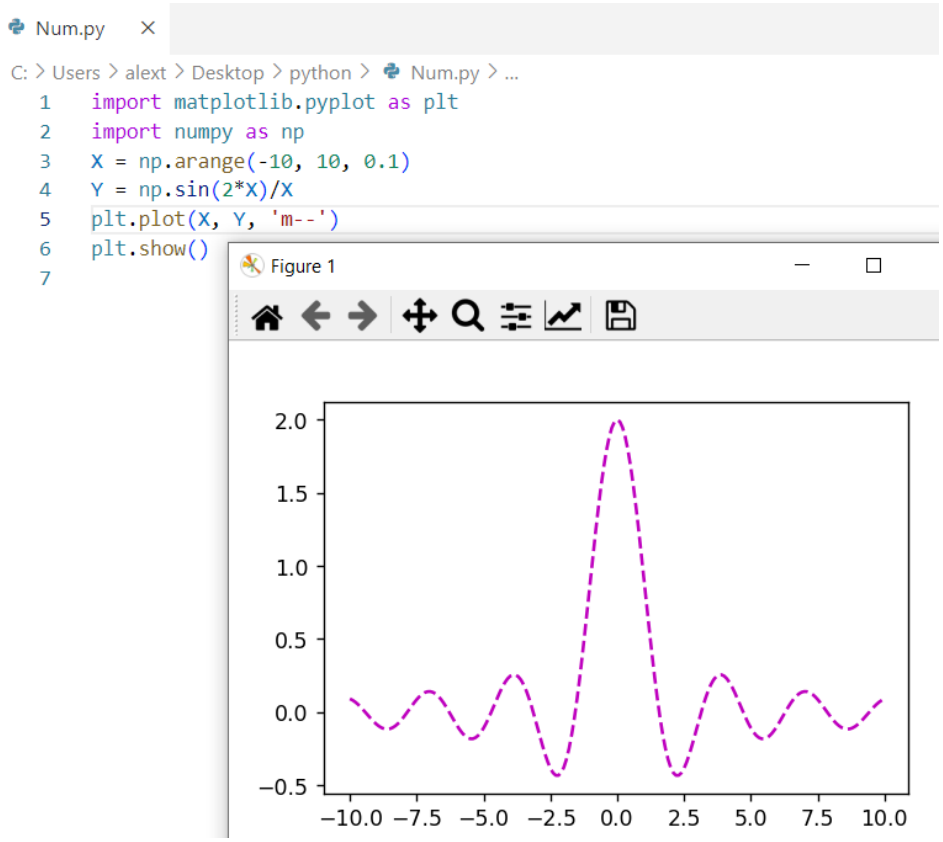


Рисунок 6.23

Можливі варіанти: -, :, -., --. І ось ще один приклад:

plt.plot(X, Y, 'k:o')

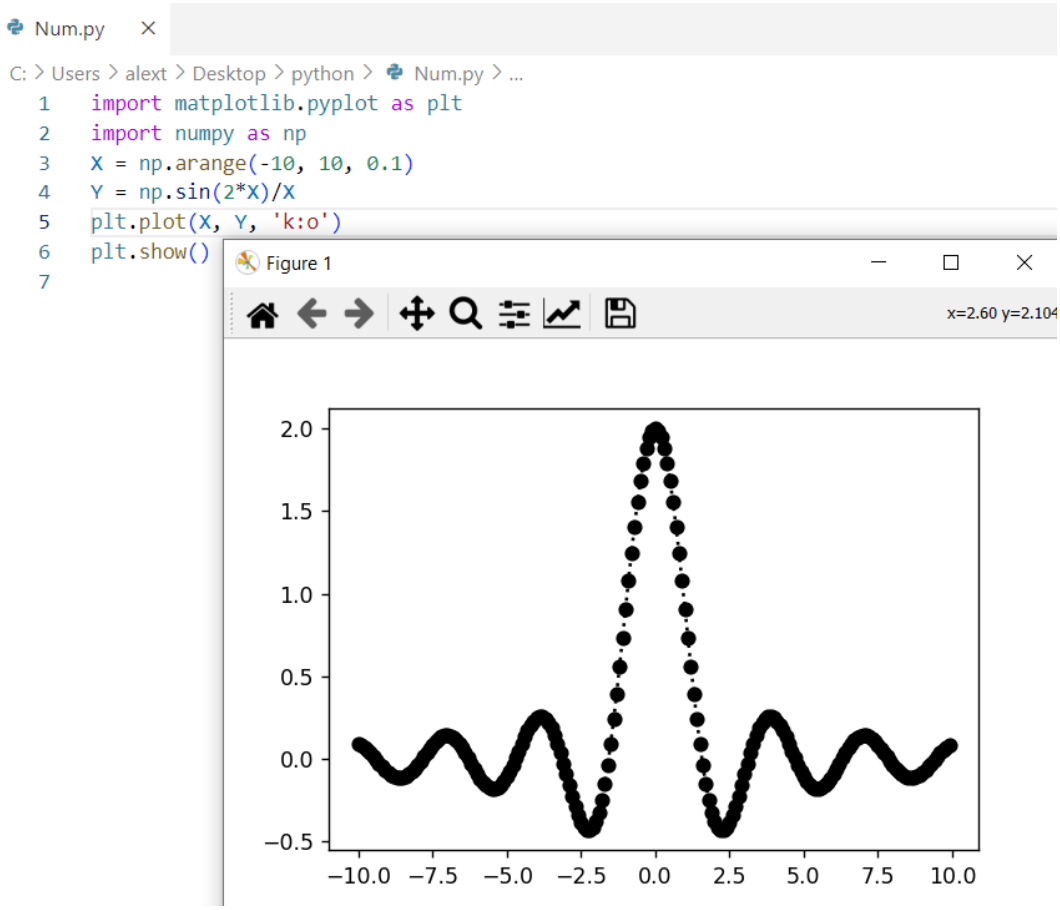


Рисунок 6.24

Третій параметр використовує рядковий формат виду "[колір] [лінія] [маркер]". Результат (з використанням інтерактивних можливостей графіка) наведено на рис.

З точки зору MATLAB можливості дизайну кривих, що відображаються, були б вичерпані вищезазначеними прикладами, але не для Python. Чудова особливість цієї мови полягає у можливості передавати значення у функцію з явним зазначенням імен параметрів. Таким чином, попередній рядок програми можна було переписати так:

plt.plot(X, Y, color='black', marker='o',
linestyle='dotted')

Num.py

C: > Users > alext > Desktop > python > Num.py > ...

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  X = np.arange(-10, 10, 0.1)
4  Y = np.sin(2*X)/X
5  plt.plot(X, Y, color='red', marker='o', linestyle='dotted')
6  plt.show()
```

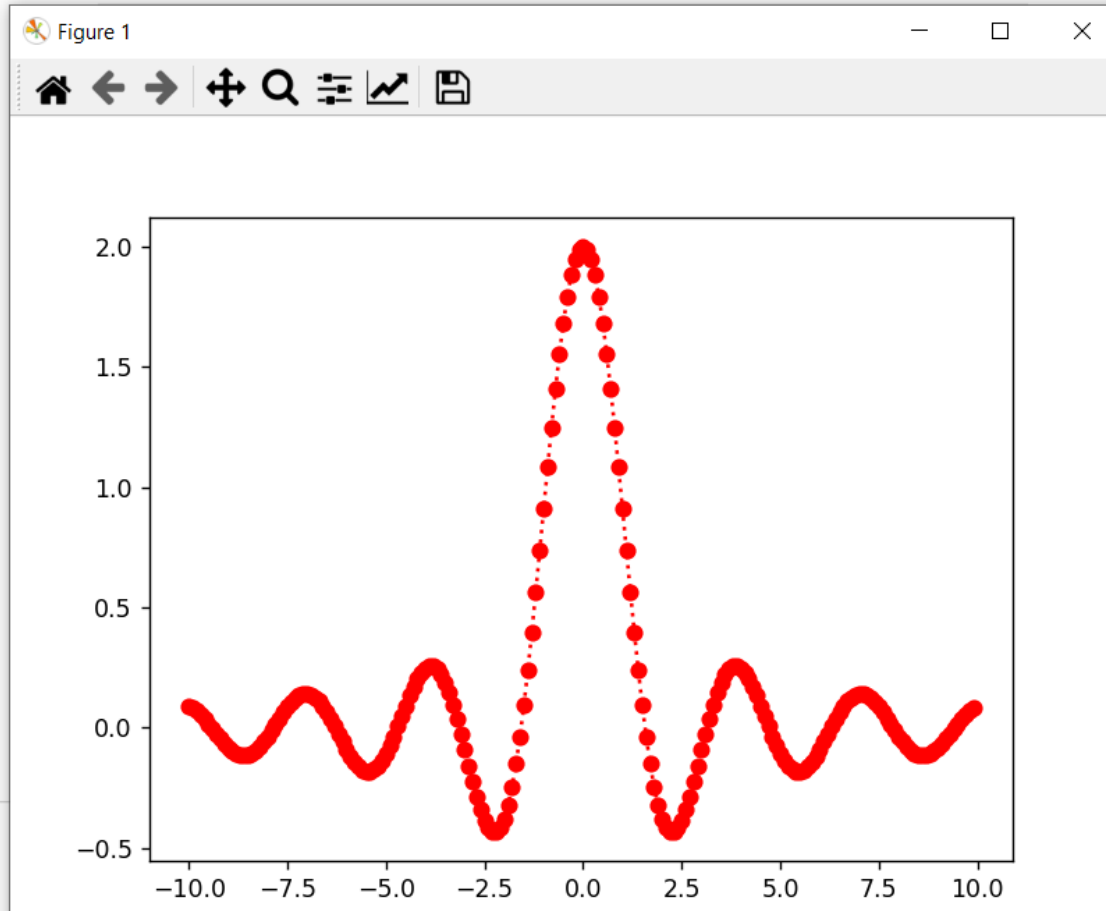


Рисунок 6.25

Порядок перерахування імен параметрів та їх кількість не є важливими. Спробуємо змінити дизайн кривої так, щоб це була зелена штрихова лінія завтовшки 1.2 pt. з червоними ромбоподібними маркерами з тонкою синьою окантовкою. Отже:

plt.plot(X, Y, linestyle='dashed', color='green',
linewidth = 1.2, marker='d', markersize = 6,
markeredgecolor = 'blue', markerfacecolor = 'red',
markeredgewidth = 0.5)

```
Num.py
C: > Users > alext > Desktop > python > Num.py > ...
1  import matplotlib.pyplot as plt
2  import numpy as np
3  X = np.arange(-10, 10, 0.1)
4  Y = np.sin(2*X)/X
5  plt.plot(X, Y, linestyle='dashed', color='green',
6  linewidth = 1.2, marker='d', markersize = 6,
7  markeredgecolor = 'blue', markerfacecolor = 'red',
8  markeredgewidth = 0.5)
9  plt.show()
```

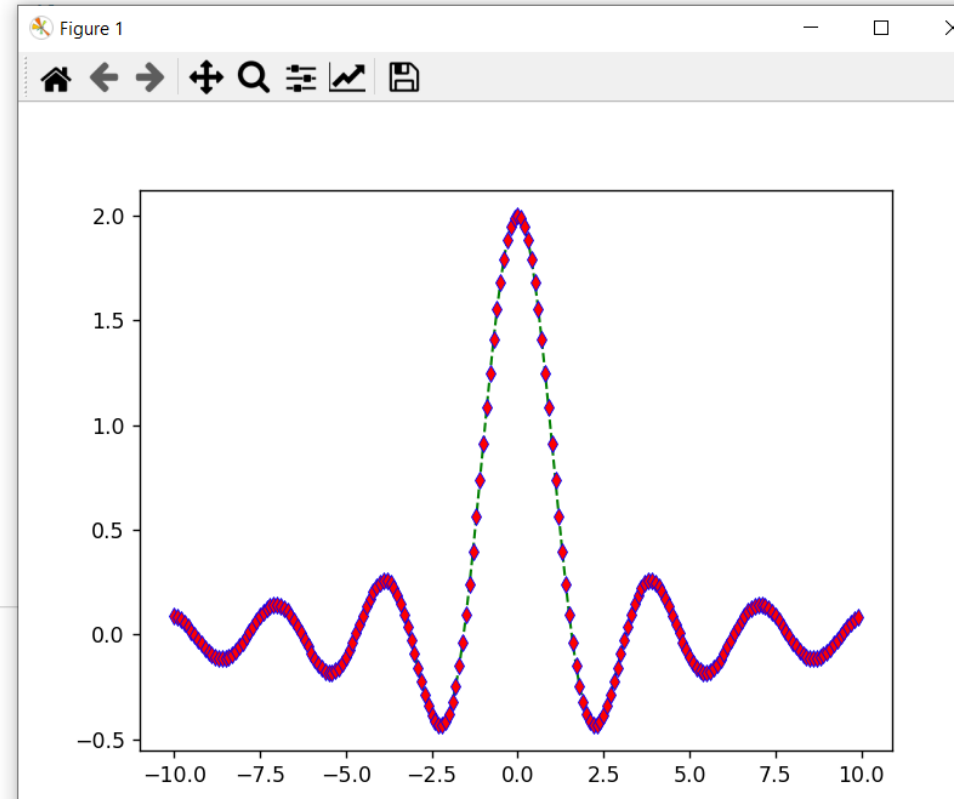


Рисунок 6.26

Деякі варіанти ліній та маркерів представлені нижче.

Таблиця 1

Значення	Колір
y	жовтий
m	бузковий
c	блакитний
r	червоний
g	зелений
b	синій

w	білий
k	чорний

Таблиця 2

Значення	Тип лінії
-	суцільна
:	точки
- .	штрих-пунктир
- -	штрихова

Таблиця 3

Значення	Тип точки маркера
.	Крапка
0	Коло
X	Перехрестя
+	Плюс
*	Зірочка
s	Квадрат
d	Ромб
V	Трикутник вершиною вниз
Λ	Трикутник вершиною вгору
<	Трикутник вершиною вліво
>	Трикутник вершиною вправо
P	П’ятикутник
h	Шестикутник

Оцінка похибок вимірювань, безумовно, є важливою складовою експериментальних досліджень. Розглянемо на прикладі, як бібліотеки numpy і matplotlib справляються з такими рутинними обчислювальними завданнями.

Зобразимо графік усереднених значень деякого набору даних з вказівкою помилки вимірювань у кожній точці .

У цьому прикладі ми емулюємо експериментальні дані, обчислюючи sin(x) з додаванням випадкової похибки (рядок 9). Кількість точок, що обчислюються - 20, число повторень обчислень для кожної точки – 10. В результаті у нас виходить numpy-матриця результатів, для якої ми можемо застосовувати необхідні статистичні функції. Для розрахунку середньоарифметичного значення $\bar{Y}_i = \frac{1}{N}\sum_{k=1}^{N} Y_k(X_i)$ для N вимірів у кожній і точці використовуємо метод mean (рядок 10). Для розрахунку стандартного відхилення в кожній точці $s_i = \sqrt{\frac{1}{N-1}\sum_{k=1}^{N}(Y_k(X_i) - \bar{Y}_i)^2}$ Візьмемо спосіб std (рядок 11). Параметр axis=1 вказує, що середнє береться за рядками. Параметр ddof визначає статистичну схему розрахунку - значення 1 відповідає незміщеній оцінці дисперсії (тобто у знаменнику стоїть значення N – 1). Таким чином, Y_mean та Y_std – це одновимірні масиви, які міститимуть середні значення і помилки для всіх точок. Залишилось відобразити ці результати на графіку, що з успіхом робить метод errorbar (рядок 13), який виводить графік середніх значень (зелена крива) разом із інтервалами помилок (червоні лінії) для кожної точки. Крім параметра yerr, можна використовувати і xerr для малювання. інтервалу помилок за шкалою абсцис, але для нашого завдання це не потрібно. Для відображення графіка вихідної функції (синя крива) використовуємо звичайний метод plot (рядок 16).

Для того щоб показати вихідний набір значень масиву Y, на основному графіку ми використовували врізання, яка була реалізована за допомогою методу axes (рядок 18). Параметрами методу виступає список із чотирьох значень: лівий нижній кут врізання по осях X і Y, та розміри врізання dX, dY в умовних одиницях від 0 до 1 (одиниця – максимальна умовна довжина поля графіка). Встановивши нові координати за допомогою методу plot виводимо всю матрицю Y на полі врізаного графіка (рядок 19). Метод text виводить напис, позицію лівого нижнього кута якої слід задати у списку аргументів методу у поточній системі координат (рядок 20). Для того, щоб такий складний графік коректно

відображався, до початку всіх малювань задаємо необхідний розмір поля виведення (рядок 12).

Num.py ● гра 2 — копия.py ●

C: > Users > alext > Desktop > python > Num.py > ...

```
1   import matplotlib.pyplot as plt
2   import numpy as np
3
4   n_rows = 20 #  число "експериментальних" точок
5   n_cols = 10 # число вимірювань на кожній точці
6   X = np.linspace(0, 5, n_rows) # значенНя по  абсцисі
7   Y = np.zeros((n_rows, n_cols)) # вихідна нульова матриця
8   for i in range(n_cols): # заповнюємо матрицю
9       Y[:,i] = np.sin(X) + 0.3 * np.random.randn(20)
10      Y_mean = Y.mean(axis=1) # средне по кожної строці
11      Y_std = Y.std(axis=1, ddof=1) # стандартне відхилення
12      plt.figure(figsize=(9, 6)) # розмір поля виводу
13      plt.errorbar(X, Y_mean, yerr=Y_std, fmt='o-g',
14      capsize=4, elinewidth=1, ecolor='red',
15      label='Спотворена функція')
16      plt.plot(X, np.sin(X), 'b', label = 'Справжня функція')
17      plt.legend()
18      plt.axes([.18, .18, .35, .25]) # врізання графіка
19      plt.plot(X,Y) # результати модельного експерименту
20      plt.text(0, -1, 'початкові дані')
21  plt.show()
```

Рисунок 6.27

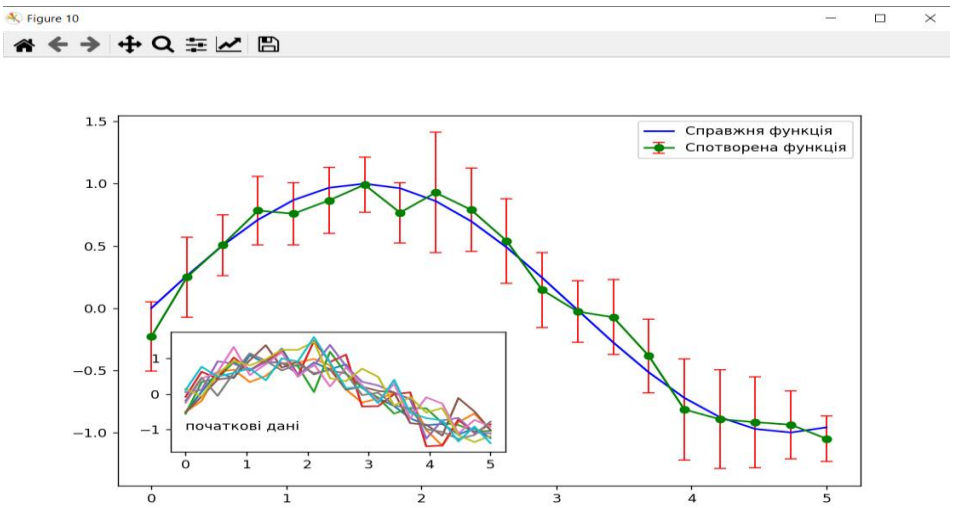


Рисунок 6.28

Малюємо тривимірні графіки функцій. Приклад найпростішої реалізації наступний.

Параметри контейнера для виведення графіка задаються у рядку 6. Аналогічно MATLAB для розрахунку значень функції Z(X,Y) для всього набору значень аргументів (рядки 9 та 10) попередньо треба розширити вектори X і Y в матриці, чим займається метод meshgrid(). Для малювання графіка будемо використовувати метод plot_surface із параметрами, заданими за замовчуванням. Результат наведено на рис.6.30.

Num.py × гра 2 — копия.py ●

C: > Users > alext > Desktop > python > Num.py > ...

```
1  import matplotlib.pyplot as plt
2  from mpl_toolkits.mplot3d import Axes3D
3  import numpy as np
4  fig = plt.figure()
5  ax = fig.add_subplot(111, projection='3d')
6  X = np.arange(-5, 5, 0.25)
7  Y = np.array(X)
8  X, Y = np.meshgrid(X, Y)
9  Z = X**2 + Y**2
10 ax.plot_surface(X, Y, Z)
11 plt.show()
```

Рисунок 6.29

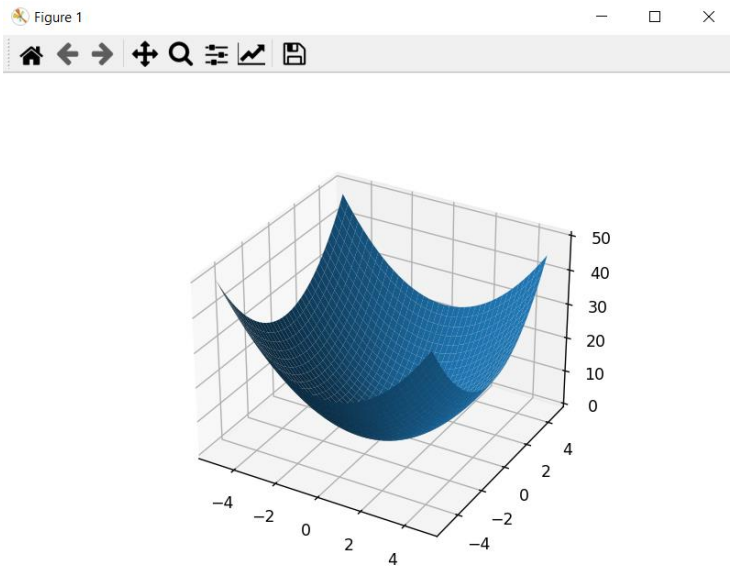


Рисунок 6.30

Вікно з графіком залишається інтерактивним – можна за допомогою миші повернути систему координат на будь-який кут. У якості експерименту з дизайном поверхні, можна спробувати замінити рядок 15 на наступну (обрана колірна палітра):

ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='viridis', edgecolor='none')

Num.py × гра 2 — копия.py ●

C: > Users > alext > Desktop > python > Num.py > ...

```
1   import matplotlib.pyplot as plt
2   from mpl_toolkits.mplot3d import Axes3D
3   import numpy as np
4   fig = plt.figure()
5   ax = fig.add_subplot(111, projection='3d')
6   # Підготовка даних
7   X = np.arange(-5, 5, 0.25)
8   Y = np.array(X)
9   X, Y = np.meshgrid(X, Y)
10  Z = X**2 + Y**2
11  # Будування  графика
12  ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
13  cmap='viridis', edgecolor='none')
14  plt.show()
```

Рисунок 6.31

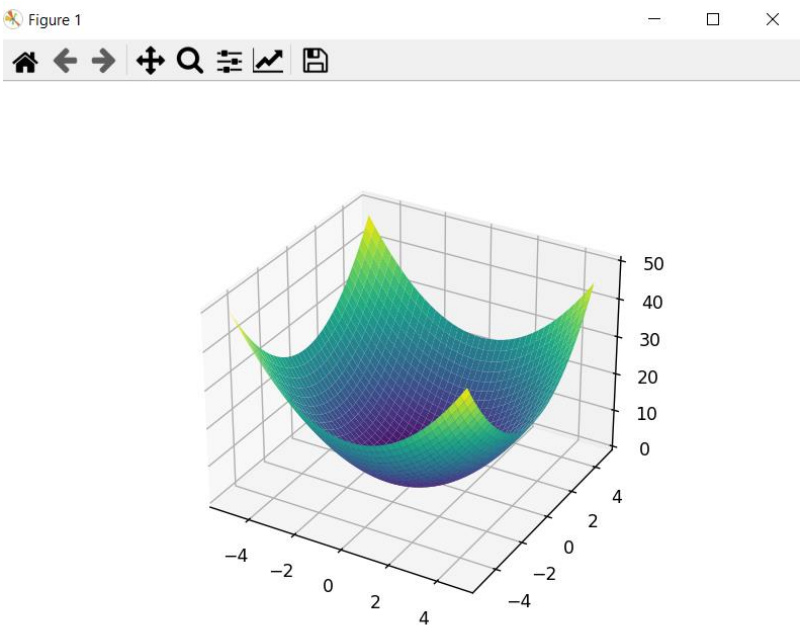


Рисунок 6.32

або на таку (поверхня як тривимірний контур):

ax.contour3D(X, Y, Z, 50, cmap='binary')

Num.py
гра 2 — копия.py
C: > Users > alext > Desktop > python > Num.py > ...

```
1   import matplotlib.pyplot as plt
2   from mpl_toolkits.mplot3d import Axes3D
3   import numpy as np
4   fig = plt.figure()
5   ax = fig.add_subplot(111, projection='3d')
6   # Підготовка даних
7   X = np.arange(-5, 5, 0.25)
8   Y = np.array(X)
9   X, Y = np.meshgrid(X, Y)
10  Z = X**2 + Y**2
11  # Будування  графика
12  ax.contour3D(X, Y, Z, 50, cmap='binary')
13  plt.show()
```

Рисунок 6.33

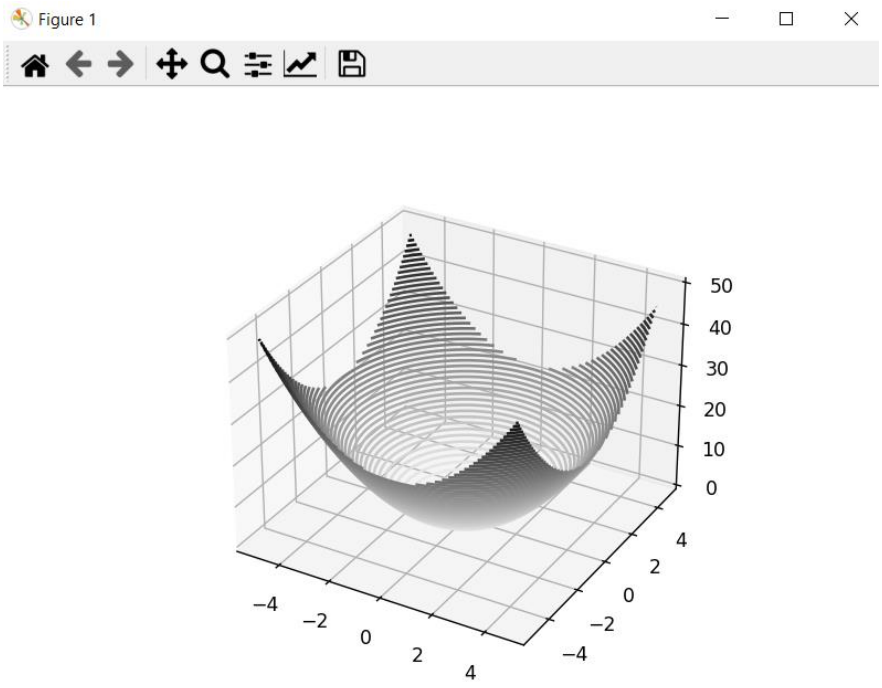


Рисунок 6.34

за допомогою методу scatter:

ax.scatter(X, Y, Z, marker='*')

Num.py | гра 2 — копия.py

C: > Users > alext > Desktop > python > Num.py > ...

```
1   import matplotlib.pyplot as plt
2   from mpl_toolkits.mplot3d import Axes3D
3   import numpy as np
4   fig = plt.figure()
5   ax = fig.add_subplot(111, projection='3d')
6   # Підготовка даних
7   X = np.arange(-5, 5, 0.25)
8   Y = np.array(X)
9   X, Y = np.meshgrid(X, Y)
10  Z = X**2 + Y**2
11  # Будування  графика
12  ax.scatter(X, Y, Z, marker='*')
13  plt.show()
```

Рисунок 6.35

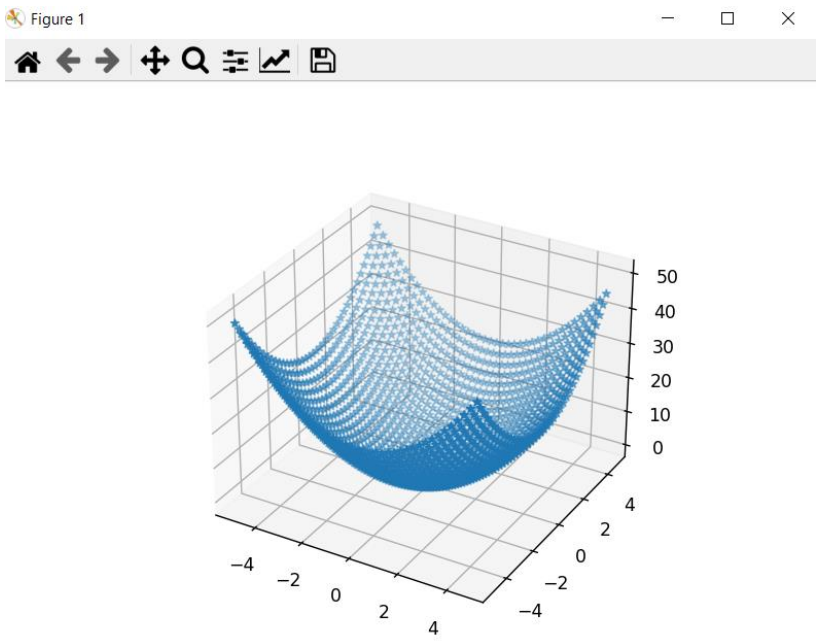


Рисунок 6.36

Заливка:

plt.fill() - заливання багатокутника;

plt.fill_between(), plt.fill_betweenx() - заливання між двома лініями;

Num.py × гра 2 — копия.py ●

C: > Users > alext > Desktop > python > Num.py > ...

```
1   import matplotlib.pyplot as plt
2   import numpy as np
3
4   x = np.arange(0, 4*np.pi+0.1, 0.1)
5   y = np.sin(x)
6   z = np.sin(2*x)
7
8   # fill()
9   fig = plt.figure()
10  plt.fill(x, y, 'r') # метод псевдографіки pcolor
11  plt.title('Simple fill')
12  plt.grid(True)
13
14  plt.show()
```

Рисунок 6.37

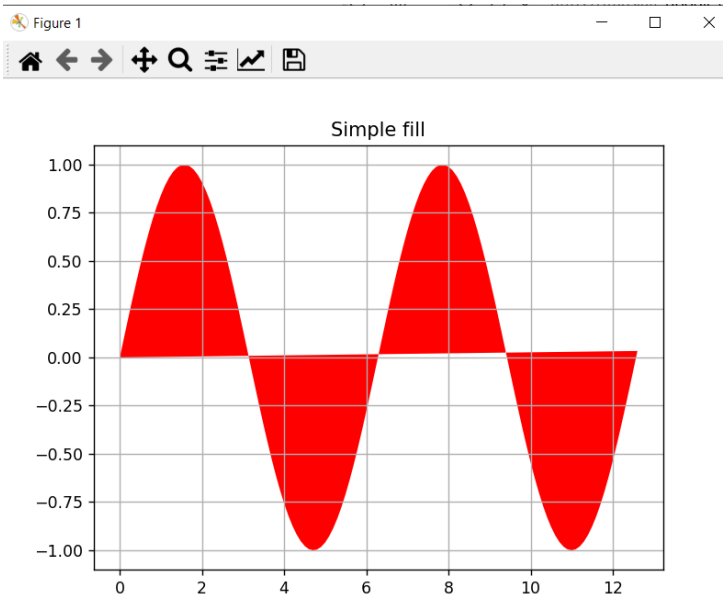


Рисунок 6.38

plt.fill_between()

Num.py
гра 2 — копия.py
C: > Users > alext > Desktop > python > Num.py > ...

```
1   import matplotlib.pyplot as plt
2   import numpy as np
3   x2 = np.arange(20)
4   y2 = -1.5*x2 + 2.33
5   z2 = 0.7*x2 - 8.5
6   # fill_between()
7   fig = plt.figure()
8   plt.plot(x2, z2, color='pink', linewidth=4.0)
9   plt.plot(x2, y2, color='g', linewidth=4.0)
10  plt.fill_between(x2, y2, z2)
11  plt.title('Simple fill_between')
12  plt.grid(True)
13  plt.show()
```

Рисунок 6.39

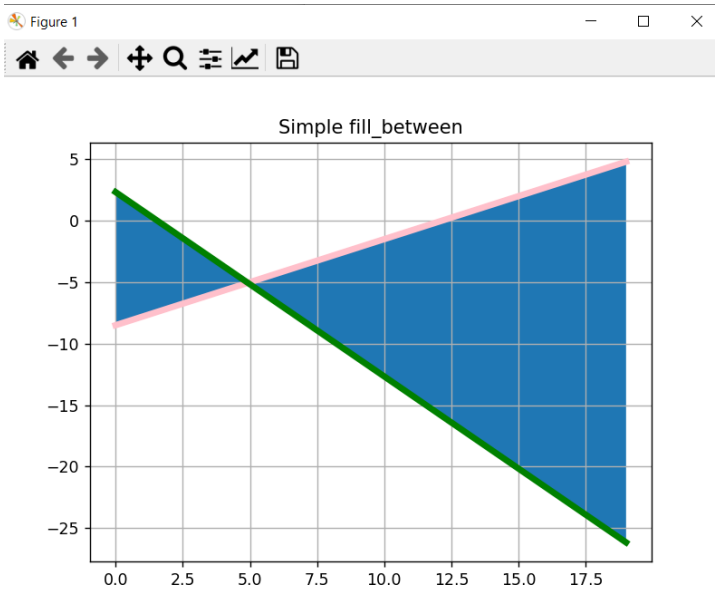


Рисунок 6.40

Num.py ● гра 2 — копия.py ●

C: > Users > alext > Desktop > python > Num.py > ...

```
1   import matplotlib.pyplot as plt
2   import numpy as np
3   x = np.arange(0, 4*np.pi+0.1, 0.1)
4   y = np.sin(x)
5   z = np.sin(2*x)
6   # fill_betweenx()
7   fig = plt.figure()
8   plt.plot(z, x, color='pink', linewidth=4.0)
9   plt.plot(z, x-1.0, color='g', linewidth=4.0)
10  plt.fill_betweenx(z, x, x-1.0, color='cyan')
11  plt.title('Simple fill_betweenx')
12  plt.grid(True)
13  plt.show()
```

Рисунок 6.41

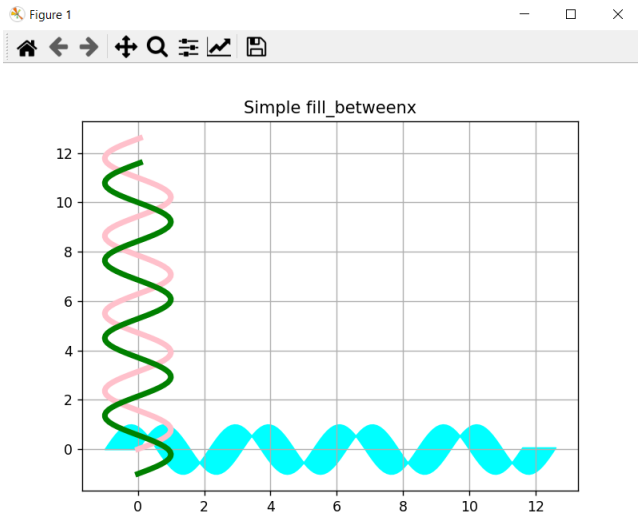


Рисунок 6.42

Практичне завдання :

1. Створити кругову діаграму з іменами 'Alex', 'Nick', 'Harry', 'Natali', 'Meredith', 'George' з параметрами [8, 7, 12, 4, 3, 2, 9] . Розмалювати частини діаграм у різні кольори. Відділити 'Natali'.

2. Побудуйте графік функції sin(x)*x на інтервалі від –10 до 10. Змінити дизайн кривої так, щоб це була червона штрихова лінія завтовшки 1.3 pt з зеленими маркерами типу пентагон з тонкою жовтою окантовкою.

3. Побудувати графік функції Z = 1/(X**2 + Y**2). По осі x та y від -5 до 5. Окрасити у сіро-білому тоні.

Практична робота №7

Розробка: студент гр.ПД 43 Швецов В.І.

Тема. Парсинг даних за допомогою бібліотеки "requests-html”

Мета роботи. Освоїти створення HTML-сесій, відправка Get-запитів та отримання HTML сторінки сайту, рендер динамічного JavaScript коду, робота з HTML-сторінкою.

Зміст.

1. Виконання роботи.
2. Отримання результату.

Хід роботи

Веб-скрапінг - перетворення у структуровані дані інформації з веб-сторінок, які призначені для перегляду людиною за допомогою браузера.

Веб-скрапінг включає в себе завантаження та вилучення. Спочатку завантажується сторінка (що робить браузер, коли ви переглядаєте сторінку), після цього можна добувати потрібну інформацію. Зміст сторінки може бути проаналізовано, переформатовано, його дані скопійовані в електронну таблицю тощо.

Веб-скрапер - це прикладний програмний інтерфейс для вилучення даних з веб-сайту.

GET-запит - метод передачі даних від клієнта до сервера з метою отримання статичної (незмінної) інформації.

Встановлення бібліотеки, створення HTML-сесії

Для встановлення бібліотеки через менеджер завантажень pip використовуємо команду:

pip install requests-html

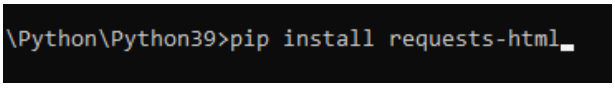


Рисунок 7.1. Встановлення бібліотеки через консоль

Для встановлення бібліотеки через IDE PyCharm заходимо до налаштувань проекту, вибираємо меню інтерпретаторів, вибираємо встановлену версію python та через “+” шукаємо та додаємо бібліотеку requests-html.

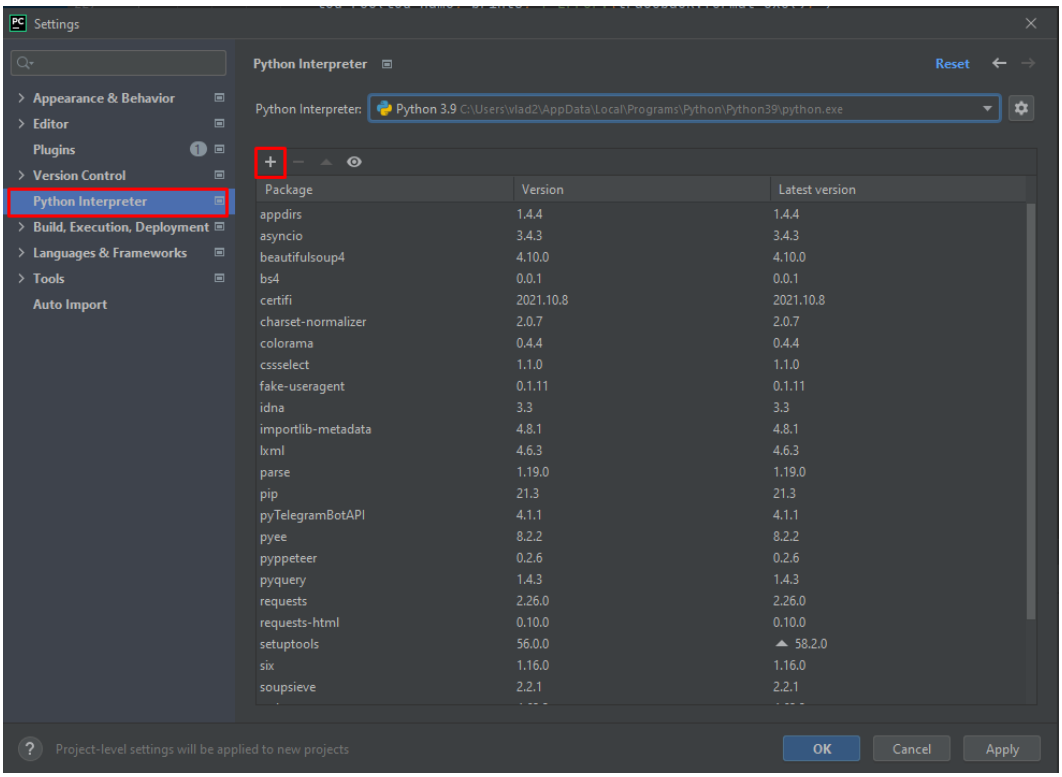


Рисунок 7.2. Встановлення бібліотеки через IDE PyCharm

Для початку роботи потрібно імпортувати з бібліотеки requests-html функцію HTMLSession. Далі створюємо нову HTML-сесію.

```
1  from requests_html import HTMLSession
2
3
4  session = HTMLSession()
5
6
```

Рисунок 7.3. Імпортування та створення нової сесії

Отримання та рендер HTML сторінки

Для навчання використаємо сайт університету та спробуємо отримати список усіх Кафедр за посиланням:

http://www.dut.edu.ua/ru/8-struktura-universiteta-pro-universitet

Для отримання веб-сторінки використовується Get-запит, для того щоб його відправити використовуємо метод get().

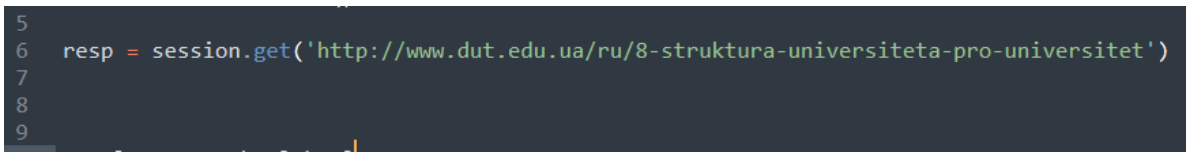


Рисунок 7.4. Get-запит

Після цієї команди у змінну resp запишеться об’єкт відповіді запиту, для того щоб перевірити вдалість запиту орієнтуються по параметру статусу коду. Після отримання відповіді потрібно вибрати з неї HTML-код нашої сторінки, для цього звертаємося до параметрів «resp.html.html». Для перевірки та подальшої роботи можемо записати нашу сторінку в HTML файл.

```python
from requests_html import HTMLSession

# створення сесії
session = HTMLSession()

# get
resp = session.get('http://www.dut.edu.ua/ru/8-struktura-universiteta-pro-universitet')

# перевірка статусу запиту
if resp.status_code == 200:
    print("get html page: success")

# отримання коду сторінки
result = resp.html.html

# запис сторінки до файлу
with open("result.html", "w", encoding="utf-8") as f:
    f.write(result)
```

```
get html page: success
[Finished in 872ms]
```

Рисунок 7.5. Код отримання сторінки

Запускаємо програму через інтерпретатор Python та бачимо результат. Перевіряємо код сторінки відкриваючи її через веб-браузер.

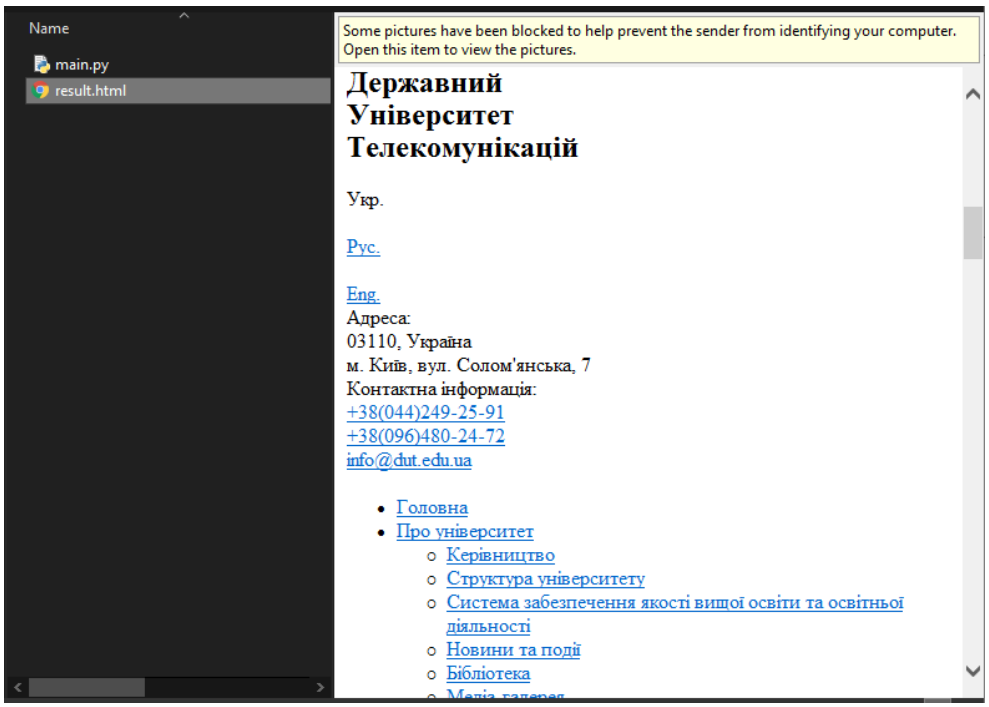


Рисунок 7.6. Створений Html файл з отриманою сторінкою

Вибірка даних за допомогою регулярних виразів

Після того, як ми отримали html сторінку, нам потрібно вибрати з неї потрібні нам дані, для цього можемо скористатися регулярними виразами.

Для початку встановлюємо (через pip або через pycharm, як робили раніше) та імпортуємо бібліотеку “re”.

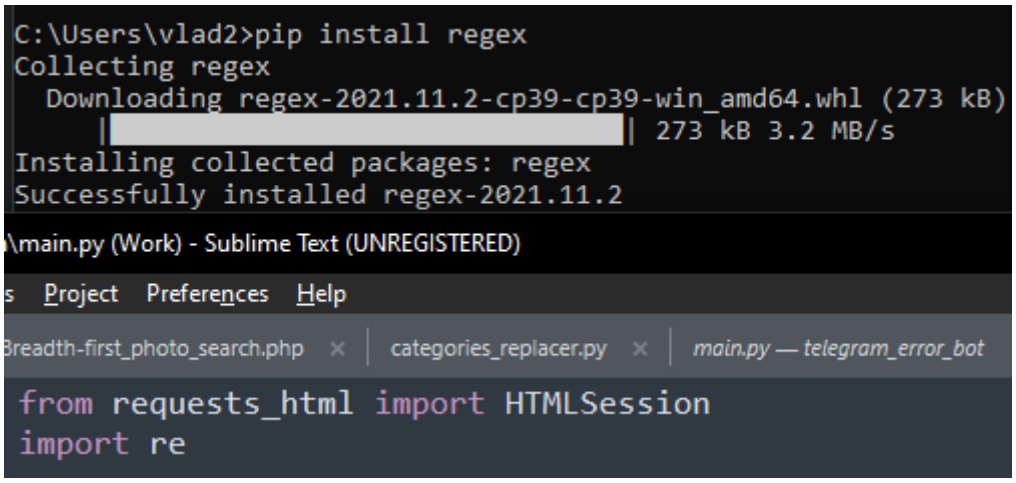


Рисунок 7.7. Встановлення та імпортування бібліотеки re

Після цього потрібно визначитися з регулярним виразом, яким ми будемо підбирати потрібну нам інформацію. Скористаємося сайтом regex101 за посиланням:

https://regex101.com

Для того щоб знайти назви всіх кафедр, нам потрібно дізнатися в яких html тегах вони знаходяться. Можемо їх знайти за допомогою інструментів розробника у браузері.

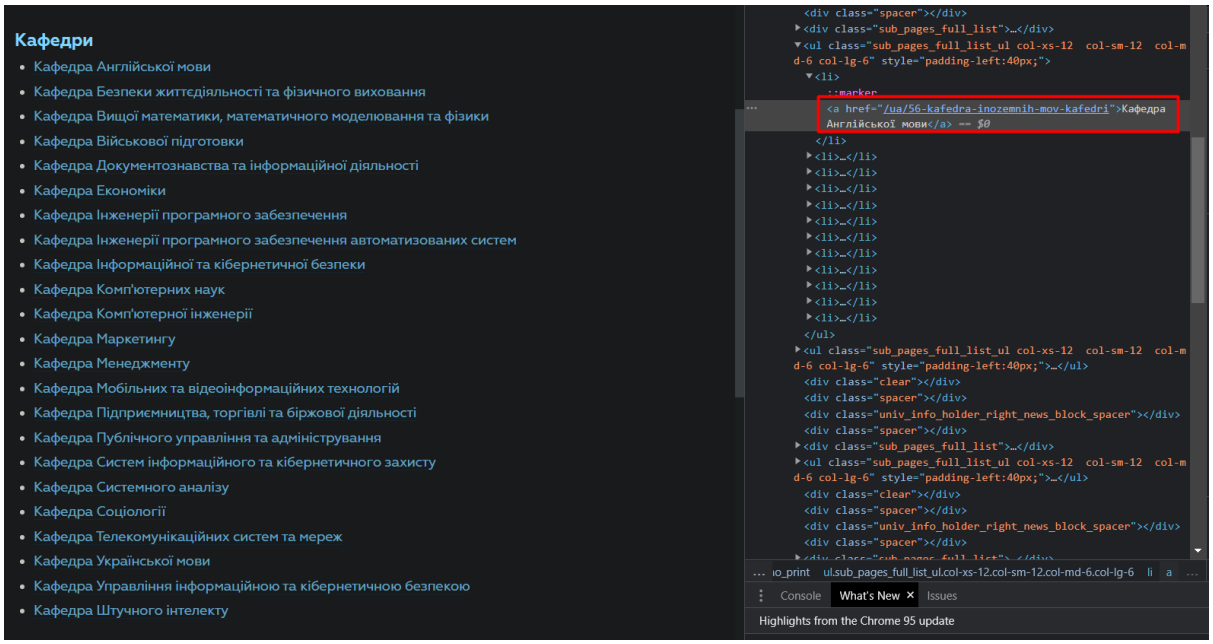


Рисунок 7.8. Html-тег з назвою кафедри

Далі потрібно написати сам регулярний вираз, знаючи що всі наші кафедри знаходяться у списках ul, та у посиланнях обов'язково буде “-kafedra-“.

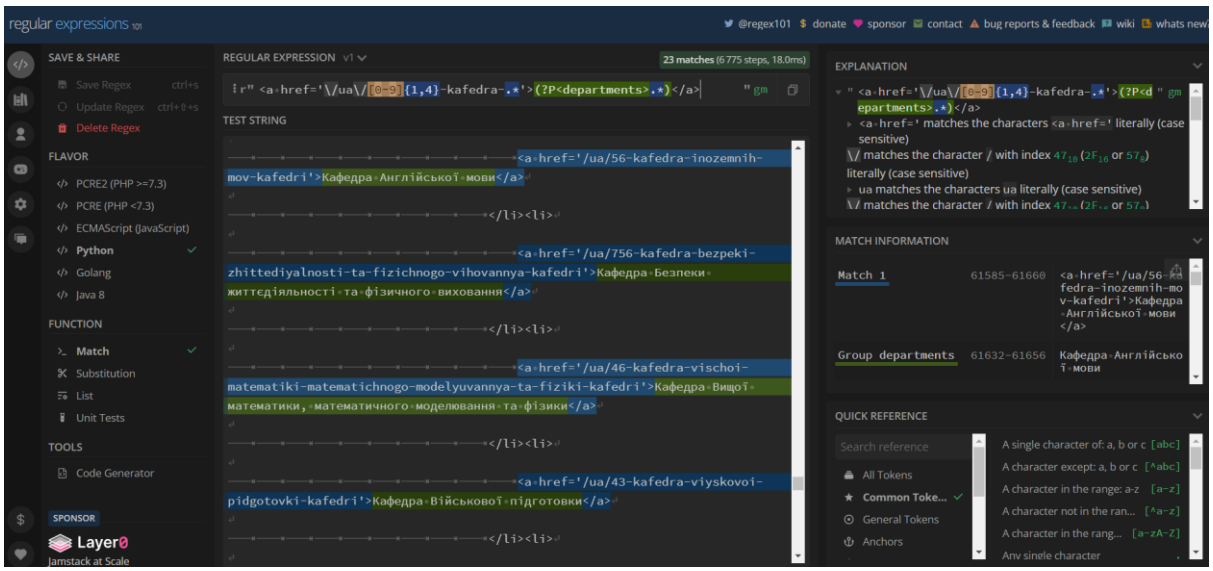


Рисунок 7.9. Регулярний вираз створений за допомогою сайту regex101

Далі для того щоб знайти всі кафедри нам потрібно використати функцію re.findall() яка знаходить всі співпадіння.

```
# https://regex101.com/r/O0eTKP/1/
regex = "<a href='\/ua\/[0-9]{1,4}-kafedra-.*'>(?P<departments>.*)</a>"
regex_result = re.findall(regex, result)
```

Рисунок 7.10. Код знаходження всіх співпадінь

Після цього ми отримуємо список зі всіма кафедрами, але за кодуванням у html коді всі апострофи замінені на код “'”, і нам потрібно замінити їх назад за допомогою функції заміни символів replace().

```
departments = []
for department in regex_result:
    # заміна символів
    department = department.replace('&#039;', "`")
    departments.append(department)
```

Рисунок 7.11. Код заміни символів у списку

Після цього виводимо результат у консоль за допомогою print().

```python
from requests_html import HTMLSession
import re

# створення сесії
session = HTMLSession()

# get
resp = session.get('http://www.dut.edu.ua/ua/8-struktura-universitetu-pro-universitet')

# перевірка статусу запиту
if resp.status_code == 200:
    print("get html page: success")

# отримання коду сторінки
result = resp.html.html

# запис сторінки до файлу
with open("result.html", "w", encoding="utf-8") as f:
    f.write(result)

# https://regex101.com/r/O0eTKP/1/
regex = "<a href='\/ua\/[0-9]{1,4}-kafedra-.*'>(?P<departments>.*)</a>"
regex_result = re.findall(regex, result)

# заміна символів у списку
departments = []
for department in regex_result:
    # заміна символів
    department = department.replace('&#039;', '`')
    departments.append(department)

# вивід результату
print("Кафедри Державного університету телекомунікацій:")
print(departments)
```

Рисунок 7.12. Повний код програми

```
get html page: success
Кафедри Державного університету телекомунікацій:
['Кафедра Англійської мови', 'Кафедра Безпеки життєдіяльності та фізичного виховання', 'Кафедра Вищої математики, математичного моделювання та фізики', 'Кафедра Військової підготовки', 'Кафедра Документознавства та інформаційної діяльності', 'Кафедра Економіки', 'Кафедра Інженерії програмного забезпечення', 'Кафедра Інженерії програмного забезпечення автоматизованих систем', 'Кафедра Інформаційної та кібернетичної безпеки', 'Кафедра Комп`ютерних наук', 'Кафедра Комп`ютерної інженерії', 'Кафедра Маркетингу', 'Кафедра Менеджменту', 'Кафедра Мобільних та відеоінформаційних технологій', 'Кафедра Підприємництва, торгівлі та біржової діяльності', 'Кафедра Публічного управління та адміністрування', 'Кафедра Систем інформаційного та кібернетичного захисту', 'Кафедра Системного аналізу', 'Кафедра Соціології', 'Кафедра Телекомунікаційних систем та мереж', 'Кафедра Української мови', 'Кафедра Управління інформаційною та кібернетичною безпекою', 'Кафедра Штучного інтелекту']
[Finished in 2.1s]
```

Рисунок 7.13. Вивід результату програми

Рендер JavaScript коду

На сторінках деяка інформація може бути додана за допомогою JavaScript коду, для того щоб її зібрати потрібно щоб цей код був виконаний.

Для прикладу напишемо сторінку, яка буде виводити дату коли сторінка була відкрита

```
<html>
    <head>
        <meta charset="utf-8">
        <title>Just a JavaScript sample</title>
    </head>
    <body>

        <h1>Just a JavaScript sample</h1>

        <script type="text/javascript">
        document.write("<h2>Date</h2>");
        document.write("<p>" + Date() + "</p>");
        </script>

    </body>
</html>
```

Рисунок 7.14. Вивід дати JavaScript

Якщо ми намагатимемось отримати сторінку таким чином, як ми це робили раніше, то в результаті ми отримаємо лише її вихідний код.

```
from requests_html import HTMLSession

session = HTMLSession()
resp = session.get('http://[redacted]/vlad_workspace/test.html')


result = resp.html.html
with open("render.html", "w", encoding="utf-8") as f:
    f.write(result)
```

Рисунок 7.15. Код отримання сторінки

Ми отримали лише код сторінки, а нам потрібно отримати дату, тож нам потрібно перед зберіганням виконати JavaScript код який є на сторінці.

render.html - Notepad

File Edit Format View Help

```
<html>

        <head>

                <meta charset="utf-8">

                <title>Just a JavaScript sample</title>

        </head>

        <body>


                <h1>Just a JavaScript sample</h1>


                <script type="text/javascript">

                document.write("<h2>Date</h2>");

                document.write("<p>" + Date() + "</p>");

                </script>


        </body>

</html>
```

Рисунок 7.16. Вихідний код сторінки

Перед отриманням html коду, нам потрібно виконати JavaScript код, для цього існує функція **html.render(),** яка виконує код на сторінці за допомогою chromium.

```
1  from requests_html import HTMLSession
2
3  session = HTMLSession()
4  resp = session.get('http://[redacted]/vlad_workspace/test.html')
5
6  result = resp.html.render()
7  result = resp.html.html
8  with open("render.html", "w", encoding="utf-8") as f:
9      f.write(result)
```

Рисунок 7.17. Код рендеру JavaScript коду

Після чого наша сторінка буде містити інформацію яка буде відображена JavaScript кодом

render.html - Notepad

File Edit Format View Help

```
<html><head>
                <meta charset="utf-8">
                <title>Just a JavaScript sample</title>
        </head>
        <body>

                <h1>Just a JavaScript sample</h1>

                <script type="text/javascript">
                document.write("<h2>Date</h2>");
                document.write("<p>" + Date() + "</p>");
                </script><h2>Date</h2><p>Sat Jan 08 2022 17:14:56 GMT+0200 (Eastern European Standard Time)</p>


   </body></html>
```

Рисунок 7.18. Код виконаної сторінки

У якості готового прикладу можна використати скрипт за посиланням: https://avi.im/stuff/js-or-no-js.html який буде видавати “No javascript support” або “Yay! Supports javascript” залежно від рендеру.

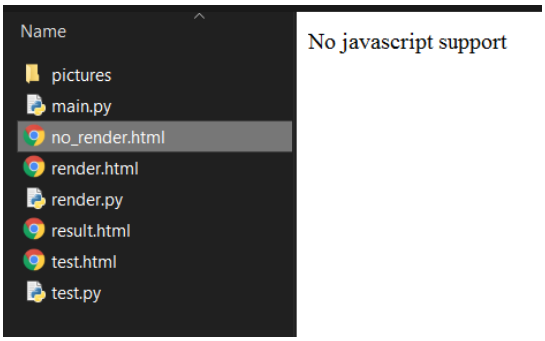


Рисунок 7.19. Сторінка без рендеру

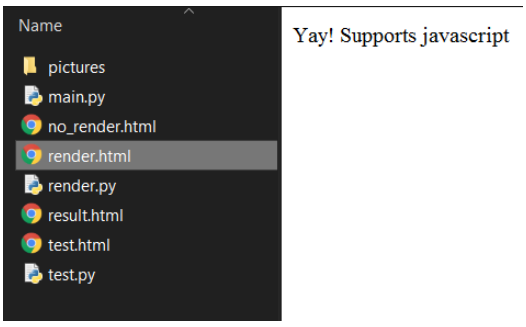


Рисунок 7.20. Сторінка з рендером

Приклад: Додавання HTTP-заголовків та проксі до HTML сесії

Заголовки запиту - повний або відносний URI ресурсу, з якого клієнт зробив поточний запит. Додаючи до HTML сесії HTTP-заголовки ми можемо представлятися для сторінки звичайним користувачем, або ж навпаки вказувати що ми являємося ботом.

Щоб додати заголовки, записуємо їх у форматі словника: ключ – значення, та додаємо за допомогою команди **headers.update(headers)**

Проксі-сервер - проміжний сервер (комплекс програм) в комп'ютерних мережах, що виконує роль посередника між користувачем і цільовим сервером (при цьому про посередництво можуть знати, так і не знати обидві сторони), що дозволяє клієнтам як виконувати непрямі запити (беручи і передаючи їх через проксі-сервер) до інших мережевих служб, так і отримувати відповіді. Додаючи до HTML сесії підключення до проксі сервера, ми можемо отримувати доступ до тих ресурсів, які нам були не доступні, наприклад коли сайт або ресурс приймає лише користувачів з певної країни.

Щоб додати заголовки, записуємо їх у форматі словника: ключ – значення, та додаємо за допомогою команди **proxies.update(proxies)**

```
4   # створення сесії
5   session = HTMLSession()
6
7   # Http заголовки
8   headers = {
9       'accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
            webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9',
10      'accept-language': 'en-US,en;q=0.9',
11      'sec-fetch-dest': 'document',
12      'sec-fetch-mode': 'navigate',
13      'sec-fetch-site': 'none',
14      'sec-fetch-user': '?1',
15      'upgrade-insecure-requests': '1',
16      'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
            Gecko) Chrome/[redacted] Safari/537.36'
17  }
18
19  # Проксі
20  proxies = {
21      'http': f'http://[redacted]',
22      'https': f'http://[redacted]'
23  }
24
25  # Додавання заголовків та проксі до html сесії
26  session.headers.update(headers)
27  session.proxies.update(proxies)
28
29  # get
30  resp = session.get('http://www.dut.edu.ua/ua/8-struktura-universitetu-pro-universitet')
```

Рисунок 7.21. Додавання HTTP заголовків та проксі до HTML сесії

Практичне завдання:

1. Повторити хід роботи.
2. Отримати результат.

Практична робота №8

Розробка: студент гр.КНД-33 Гавор А.С.

Тема. Розробка дескстоп-програми с використанням Qt designer, бібліотеки Pyqt6 та Python.

Мета роботи: Ознайомитись з використанням Python для взаємодії з популярною бібліотекою Pyqt6 та зручним IDE Qt designer для швидкого створення інтерфейсу програми с використанням API до зовнішніх джерел.

Зміст.

1. Виконання роботи.
2. Отримання результату.

Хід роботи.

Python має ефективні структури даних високого рівня та простий, але ефективний підхід до об'єктно-орієнтованого програмування. Елегантний синтаксис Python, динамічна обробка типів, а також те, що вона є інтерпретованою мовою, робить її ідеальною для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ.

Qt Designer є кросплатформним компонувальником макетів та форм графічного інтерфейсу користувача. Він дозволяє швидко спроектувати віджети та діалоги, використовуючи екранні форми з тих самих віджетів, які будуть використовуватися в додатку. Форми, створені з Qt Designer, є повністю функціональними, також можуть бути переглянуті як в реального часі.

PyQt є вільним програмним забезпеченням і розповсюджується на умовах ліцензії GNU GPL v3 для некомерційного використання.

У цій роботі ми розглянемо основи використання PyQt у зв'язці з Qt Designer. Крок за кроком ми створимо простий Python GUI додаток, який відображатиме вміст обраної директорії.

Створення інтерфейсу.

Запускаємо програму “QT Designer”

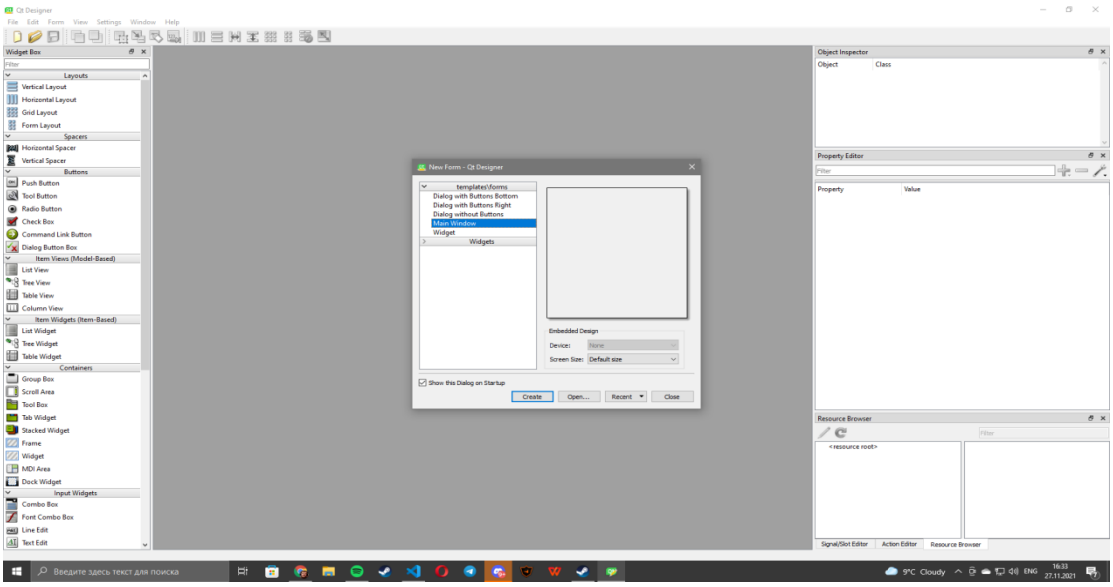


Рисунок 8.1

Створюємо середовище вибравши пункт ”Main Windows” та натиснувши кнопку “Create.

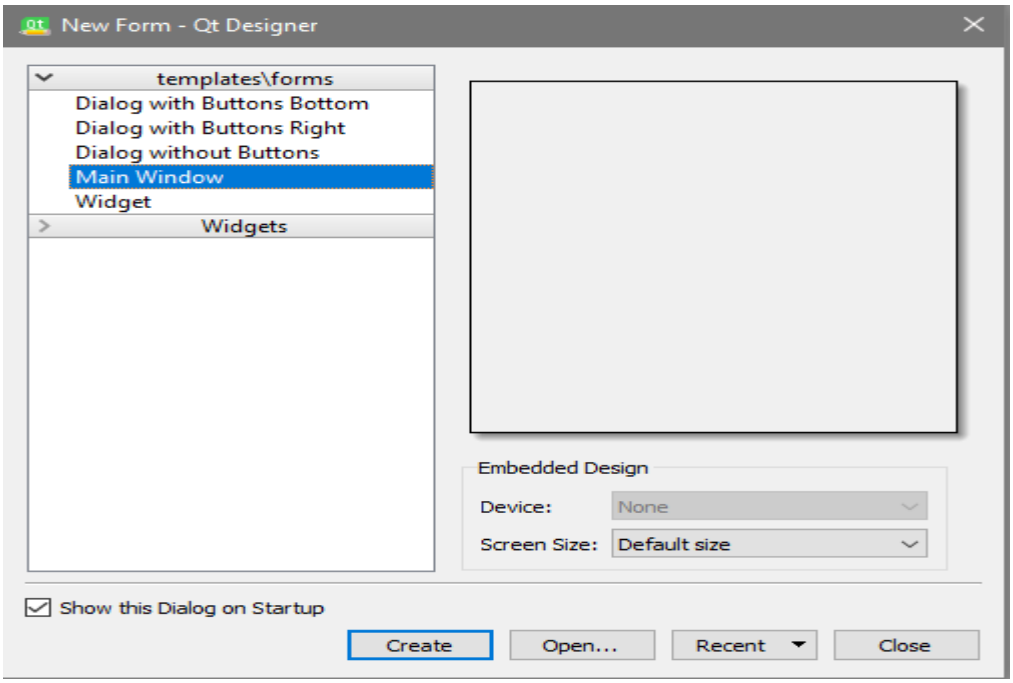


Рисунок 8.2

З’являється форма, яку ви можете редагувати.

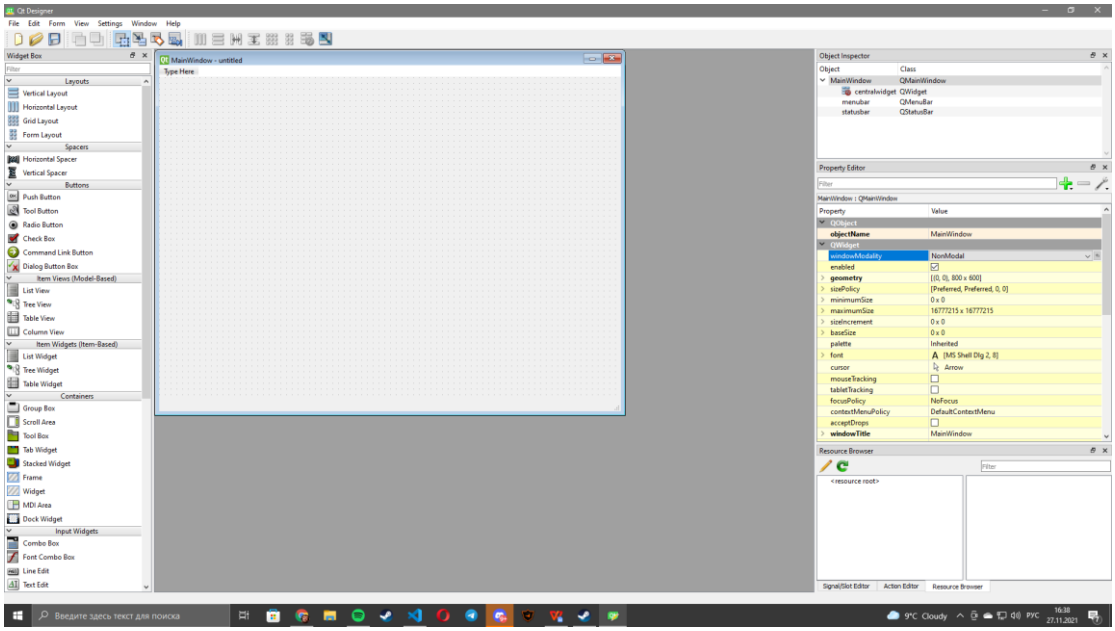


Рисунок 8.3

Змінюємо параметри екрану в параметрах елемента “Property Editor”. Змінюємо поле “objectName” на “form”, яке відповідає назві класу для зручного виклику через код.

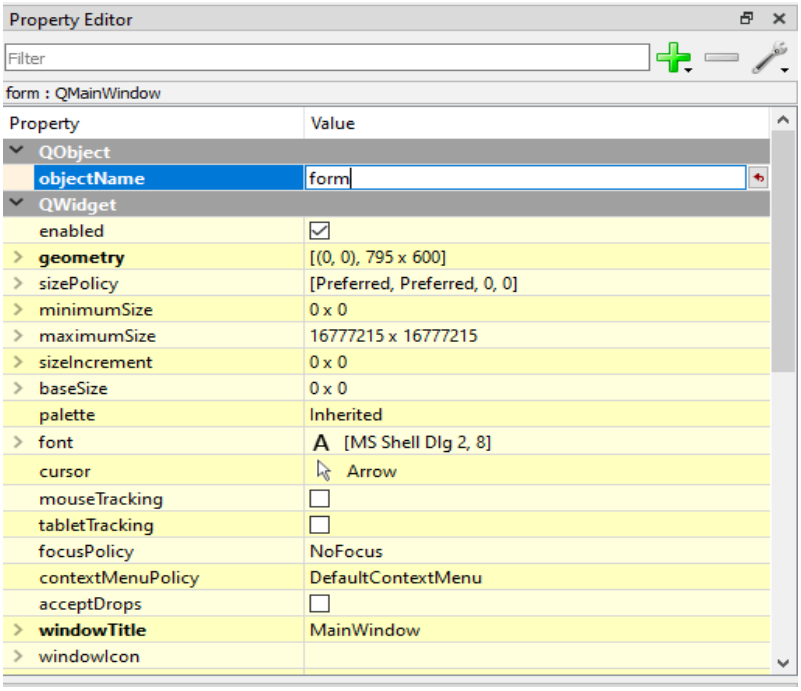


Рисунок 8.4

Змінюємо параметр головної форми в пунктах “Width” та “Height” які еквівалентні ширині та довжині форми.

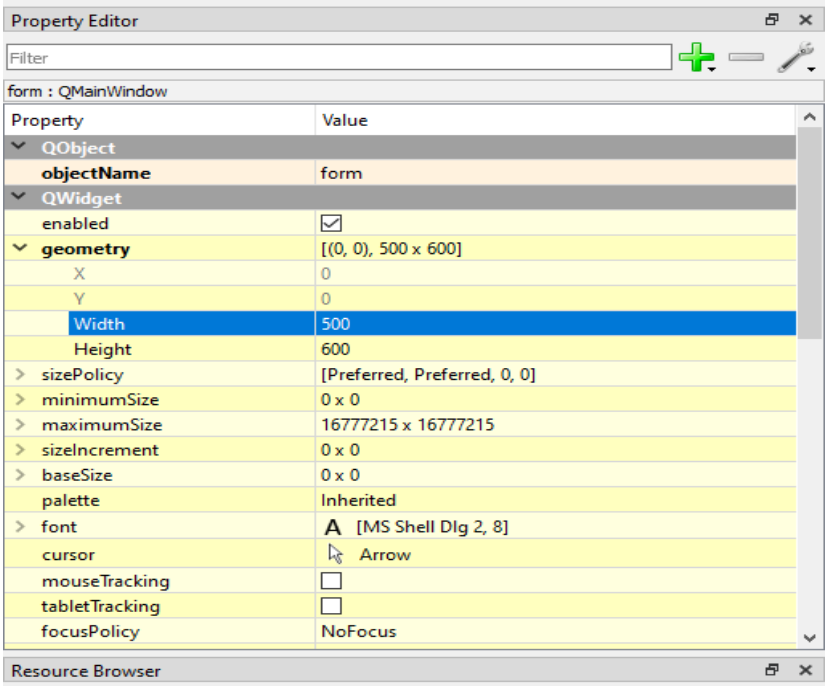


Рисунок 8.5

Змінюємо назву в панелі параметрів в полі “Windows Title”, надаємо цьому полю назву “Погода”.

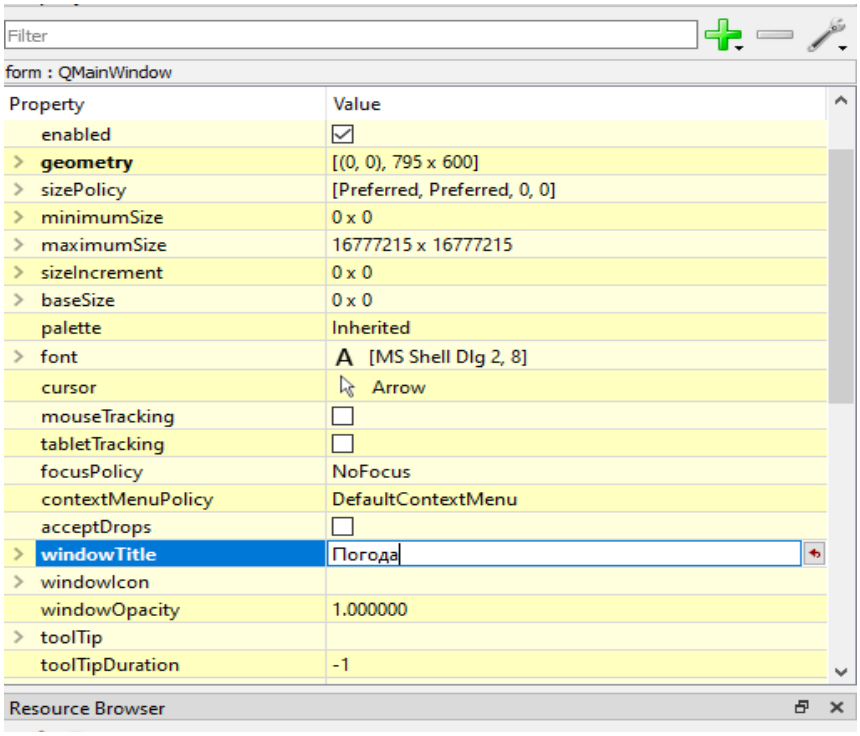


Рисунок 8.6

Таким чином ми прописали назву програми для користувача.

Рисунок 8.7

Видаляємо зайві елементи на формі в параметрах проекту “Object inspector” натиснувши праву кнопку миші та вибравши пункт “Remove Menu Bar”.

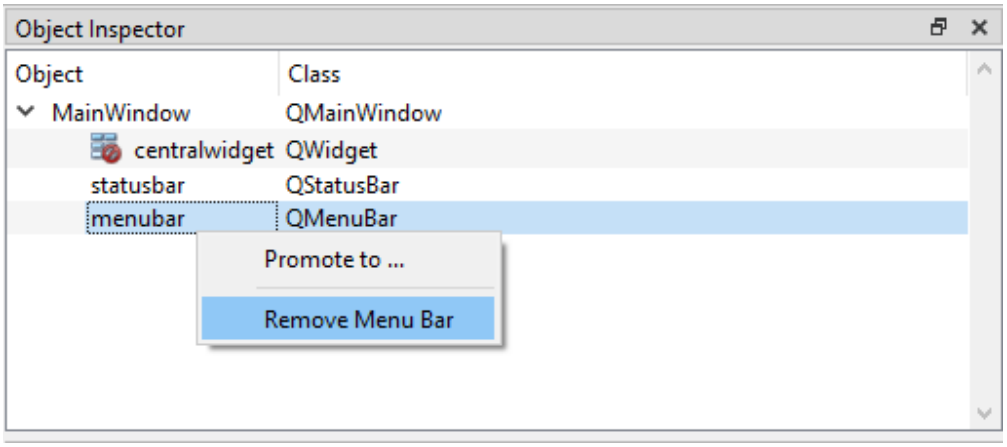


Рисунок 8.8

Розберемо такий параметр як ”Widget Box” він має багато різних заготовлених елементів які ми будемо використовувати. Також він має пошукову систему яка допомагає зручно шукати елементи.

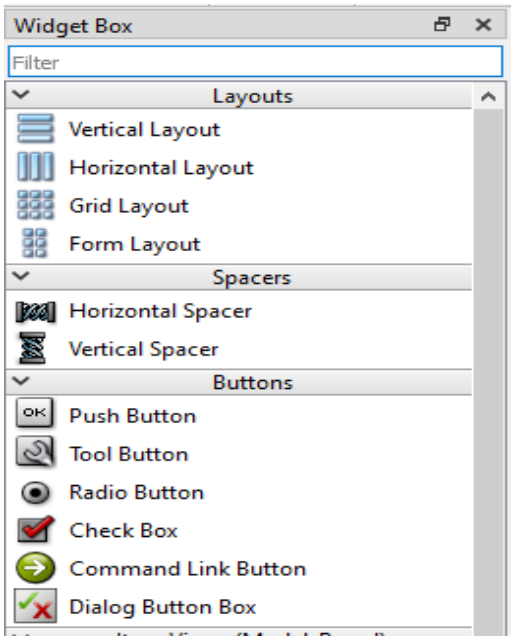


Рисунок 8.9

Шукаємо в параметрі ”Widget Box” елемент “Text Edit”.

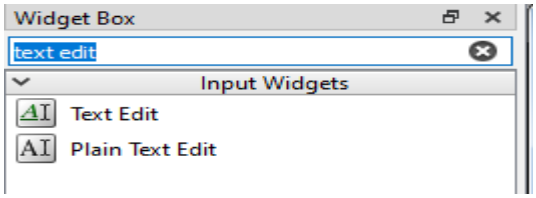


Рисунок 8.10

Вибираємо елемент та просто перетягуємо його на форму.

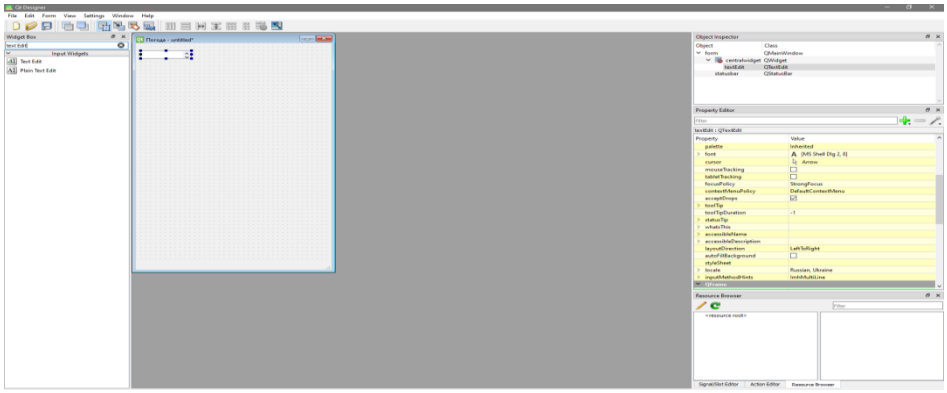


Рисунок 8.11

Змінюємо параметр головної форми в пунктах “Width” та “Height” які еквівалентні ширині та висоті елемента “Text Edit”.

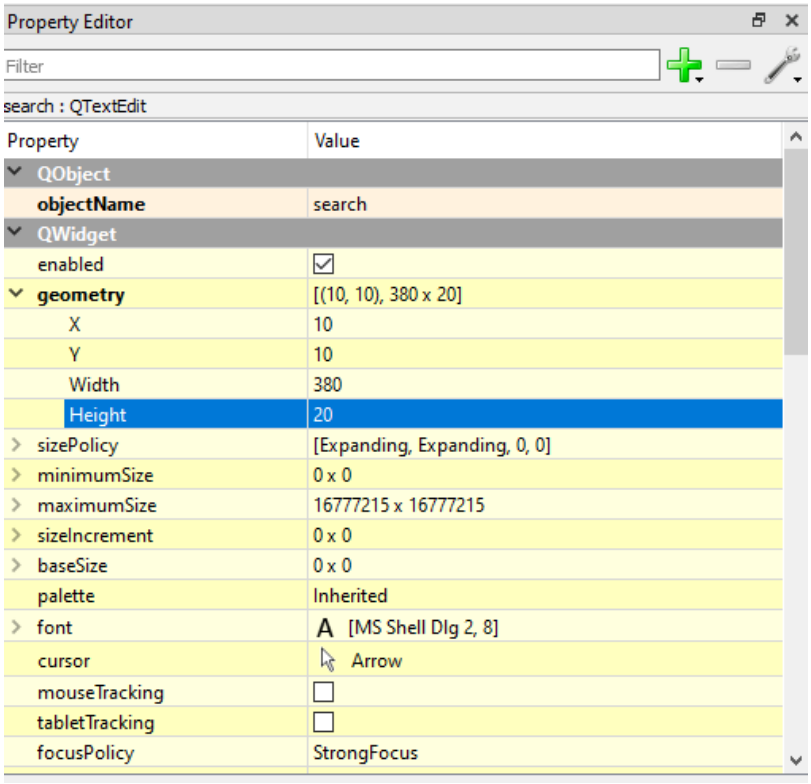


Рисунок 8.12

Вимикаємо можливість скролл-бару для елементу “TexEdit”.

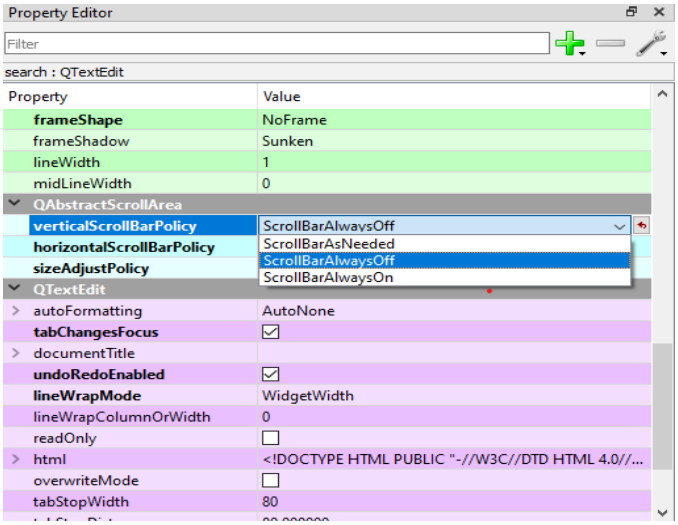


Рисунок 8.13

Додамо на головну форму елемент “label”.

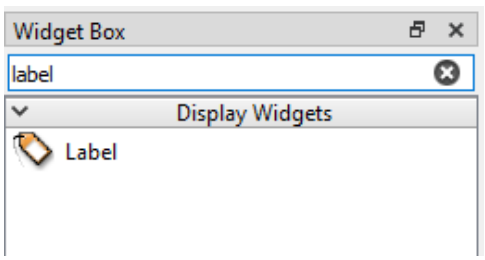


Рисунок 8.14

Змінено параметр елемента “label”, надавши параметру ”objectName” назву класу “label_1”.

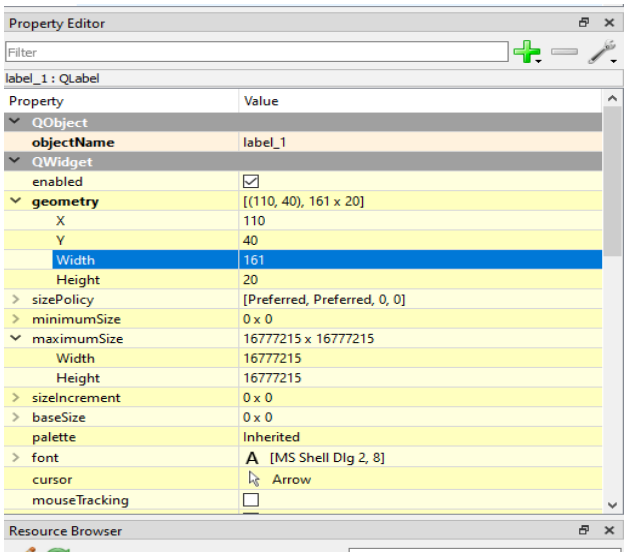


Рисунок 8.15

Надаємо текст для елементів “label_1” по “label_6”.

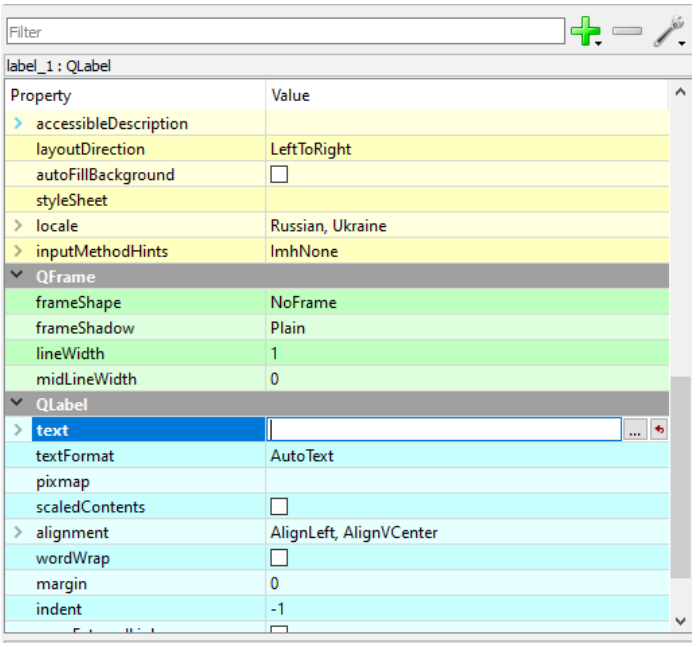


Рисунок 8.16

Наприклад від 1 до 6.

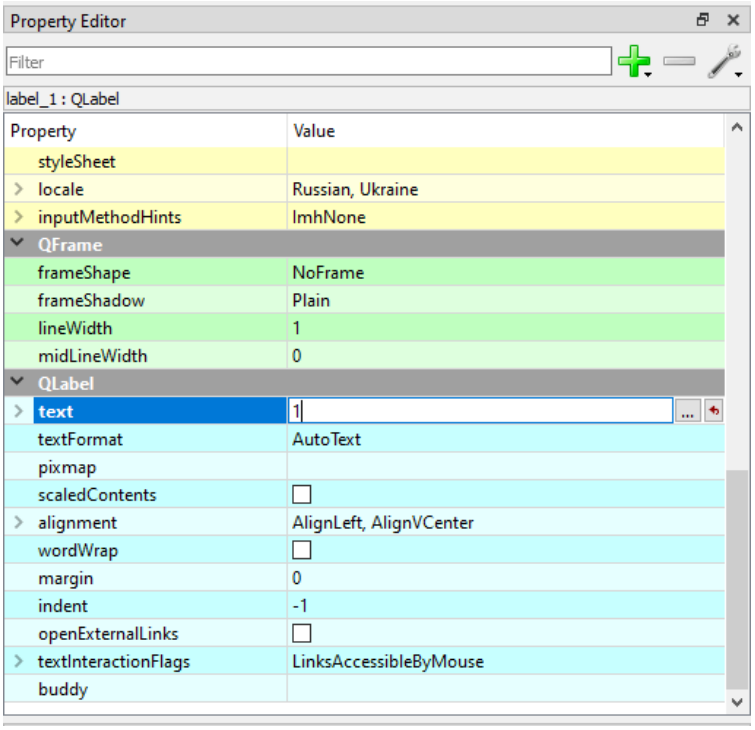


Рисунок 8.17

Додаємо ще один елемент, який буде виводити для нас картинку. Оскільки qt не має унікального елемента для цього, ми будем використовувати “label”. Назвемо його “img” та надаємо параметр тексту “img”.

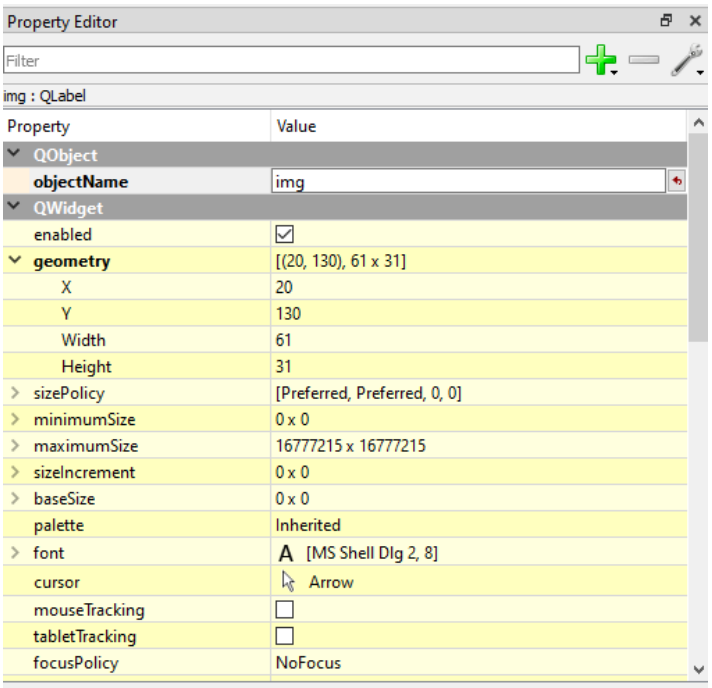


Рисунок 8.18

Додамо ще один елемент “Push Button”.

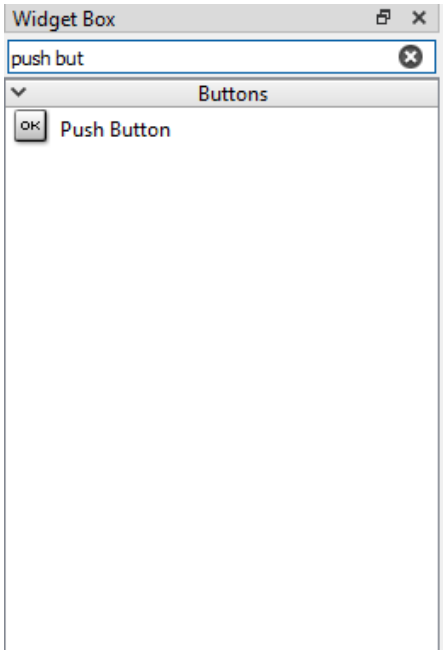


Рисунок 8.19

Дамо елементу “Push Button” назву класу “btn_search”.

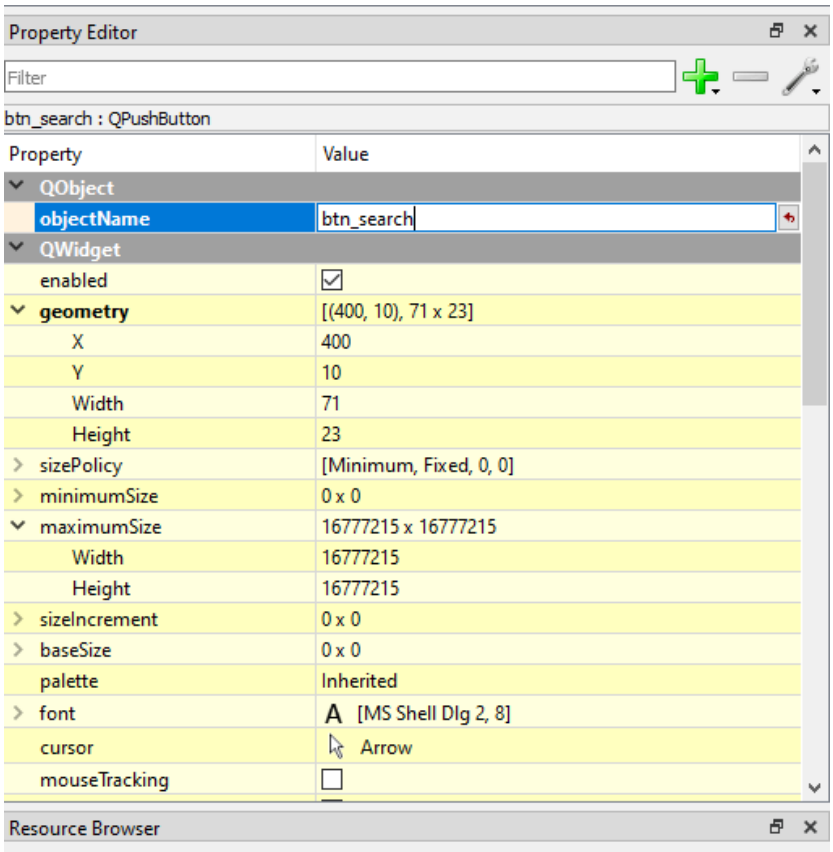


Рисунок 8.20

Текс “Пошук”.

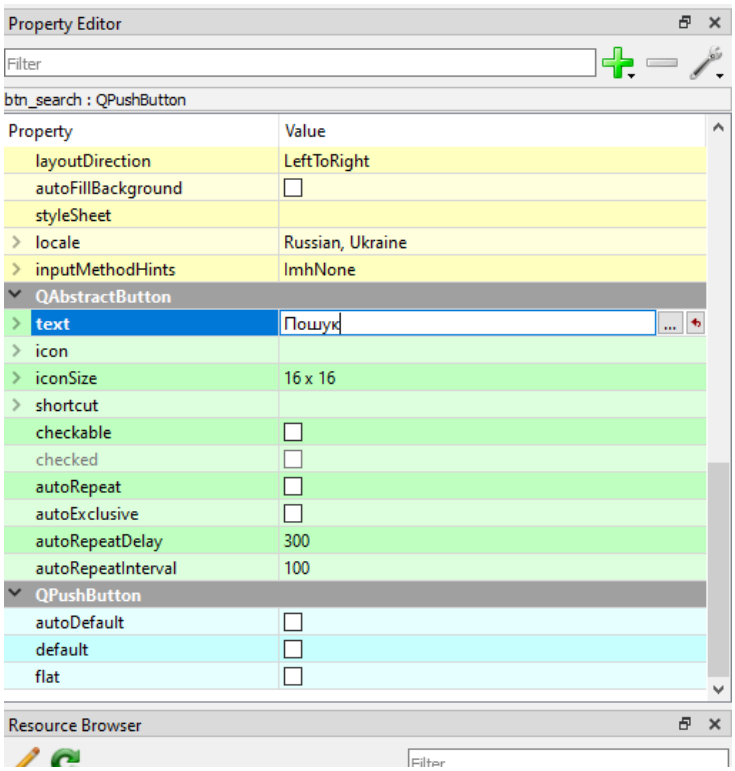


Рисунок 8.21

У вас повинні бути всі ці елементи в “Object inspector”.

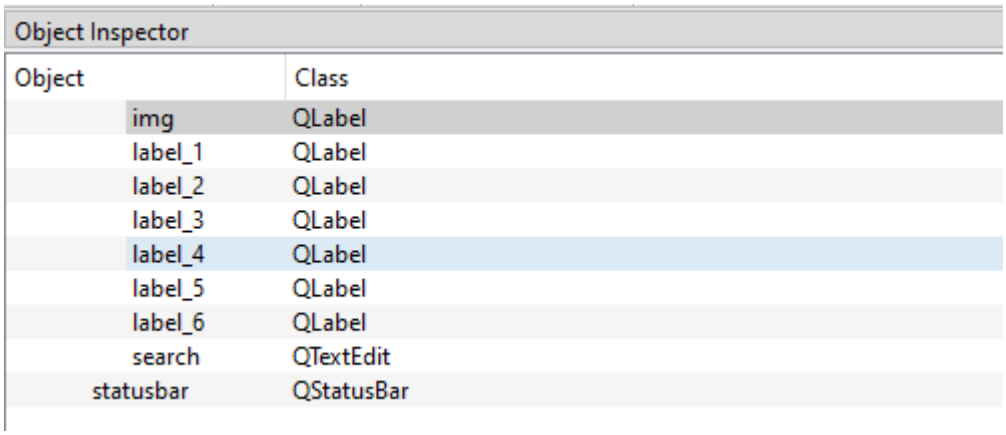


Рисунок 8.22

Змінюємо фоновий колір, натиснувши правою клавішою миші та вибравши пункт “Change StyleSheet”.

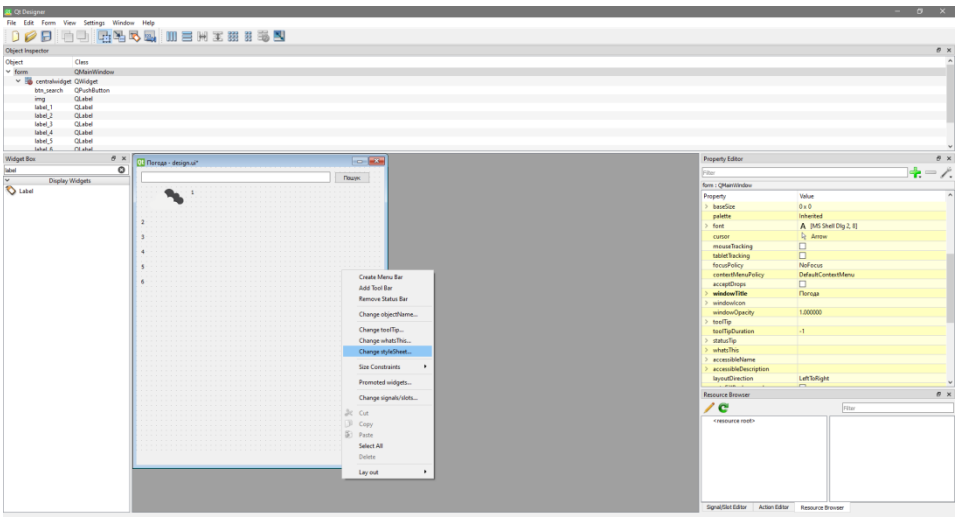


Рисунок 8.23

Впишіть в пусте поле “background-color: rgb(215, 205, 255)” та натисніть кнопку “Apply”.

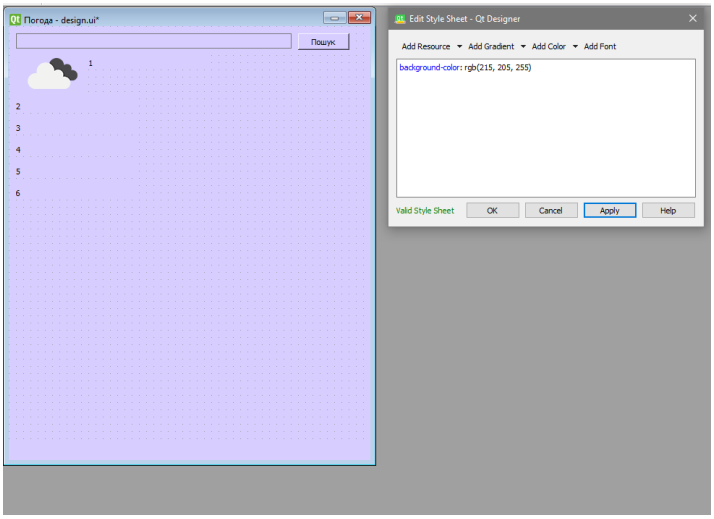


Рисунок 8.24

Розташуйте всі елементи таким чином як зображено на рис.8.25.

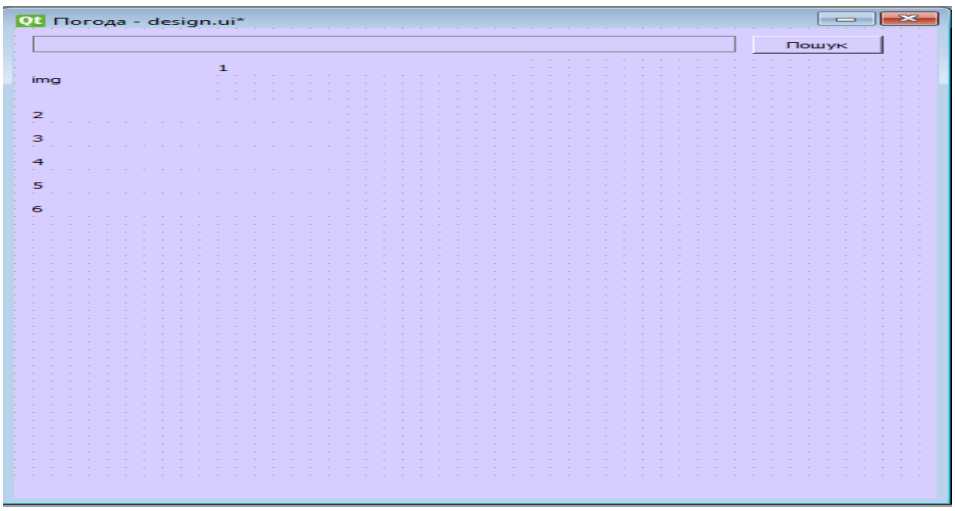


Рисунок 8.25

Зберігаємо проект с назвою “design.ui” в папці.

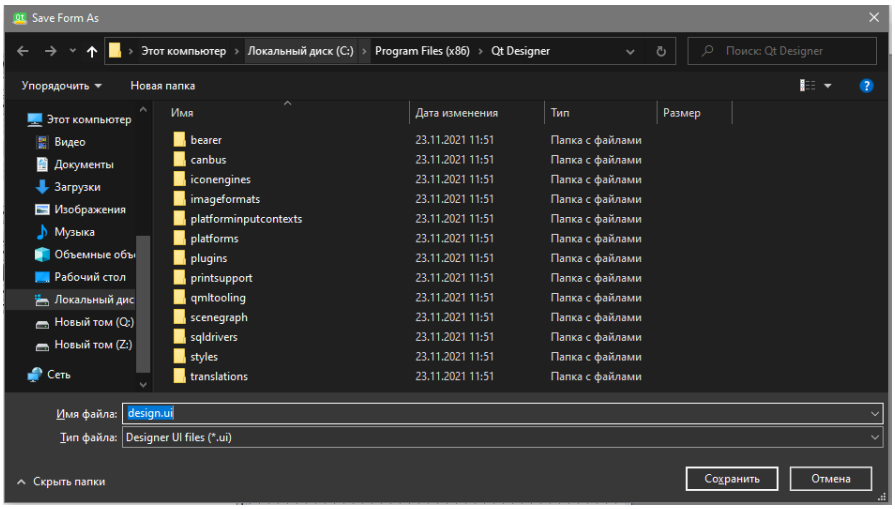


Рисунок 8.26

Встановлюємо бібліотеки qt на комп’ютер за допомогою “cmd” від імені адміністратора та встановлюємо сам pyqt6. Прописуємо pip install pyqt6.

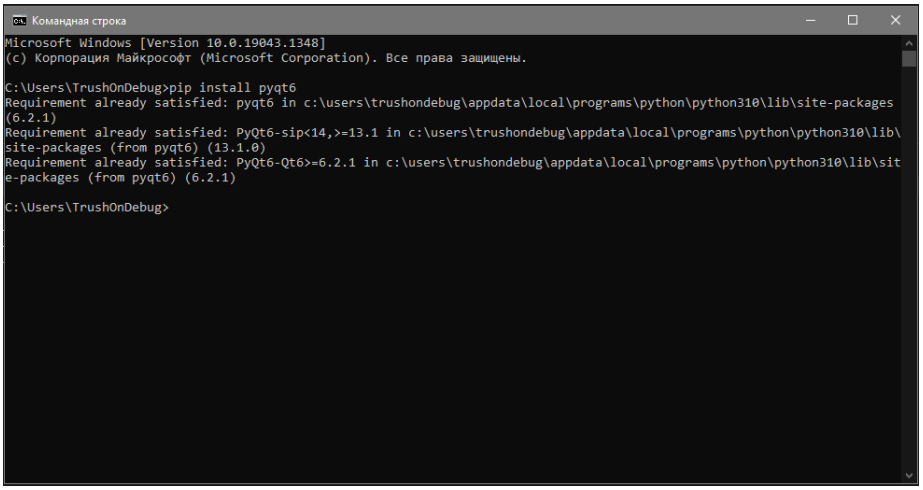


Рисунок 8.27

Встановлюємо інструменти pyqt6: вписуємо pip install pyqt6 -tools.

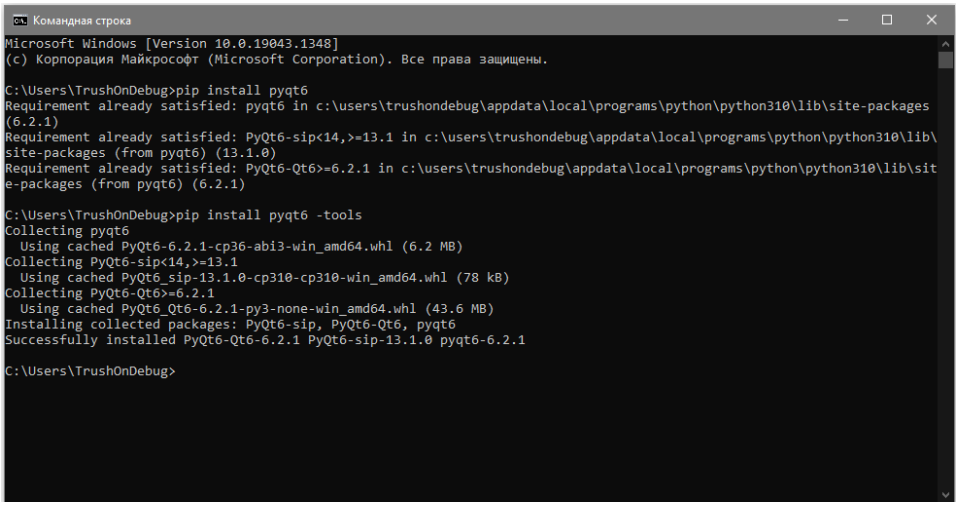


Рисунок 8.28

Функціональна реалізація інтерфейсу

Переходимо до написання коду - безпосередньо реалізації функцій елементів, які ми раніше додали на форму. В цьому проекті ми будемо використовувати API ”openweathermap”, яке надає дані про погоду та API “ip-api”, яке надає нам дані про наше місцезнаходження.

Підключаємо модулі вбудовані в сам Python:

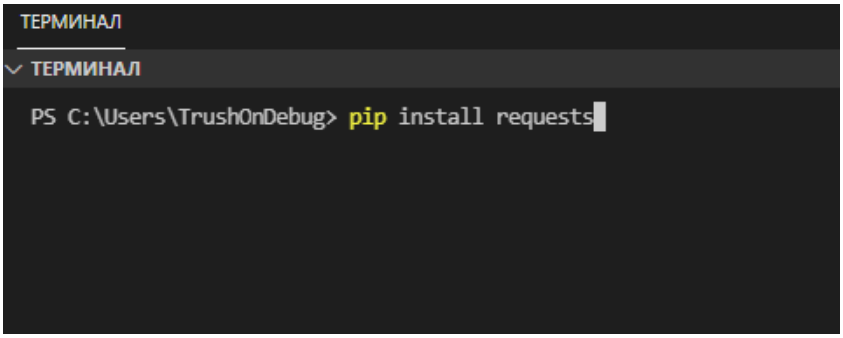
Модуль **sys** забезпечує доступ до деяких змінних та функцій, що взаємодіють з інтерпретатором Python.

Модуль **os** надає безліч функцій для роботи з операційною системою, причому їхня поведінка, як правило, не залежить від ОС, тому програми залишаються переносними.

http - це пакет, у якому зібрано кілька модулів для роботи з протоколом передачі гіпертексту. **http.client** - клієнт протоколу HTTP низького рівня.

```
import sys
import os
import http.client
```

Встановлюємо модуль request, який дає нам можливість робити API запити “Post” та ”Get”.




```
import requests
```

Підключаємо вже встановлений на комп’ютер бібліотеку “Pyqt6”.

```
from PyQt6.QtWidgets import QApplication, QWidget,QMessageBox
from PyQt6 import uic
from PyQt6.QtGui import QPixmap
```

Для початку створюємо клас, який буде опрацьовувати наш IP, та вказувати на наше місце розташування завдяки API-запиту до сайту “ip-api”. Створюємо два методи.

```
class iPhandler():
    def __init__(self):
        self.takeIP()
        self.cityIp()
```

Дізнаємось наш IP завдяки внутрішньому модулю Python http та відокремлюємо всі зайві елементи ip.

```
    def takeIP(self):
        conn = http.client.HTTPConnection("ifconfig.me")
        conn.request("GET", "/ip")
        ip=str(conn.getresponse().read())
        ip = ip.split("'")
        self.ip=ip[1]
```

```
b'185.102.186.47' >> тільки отримали
['b', '185.102.186.47', ''] в масиві
185.102.186.47  яким нам потрібний
```

Сформуємо пост-запит та отримуємо відповідь. Переформатовуємо її в формат json та виводимо в термінал.

```
def cityIp(self):
        response = requests.post('http://ip-api.com/json/'+self.ip)
        data=response.json()
        print(data)
        return data['city']
```

Тепер ми маємо щось подібне на масив. Це подібно до масиву, тільки елементи мають назви індексів. Наприклад, щоб отримати назву моєї країни мені потрібно викликати сам об’єкт та вписати ім’я елементу “country”:

data[‘country’]>>'Ukraine'

data[‘'city']>>'Kyiv'

{'status': 'success', 'country': 'Ukraine', 'countryCode': 'UA', 'region': '30', 'regionName': 'Kyiv City', 'city': 'Kyiv', 'zip': '03087', 'lat': 50.458, 'lon': 30.5303, 'timezone': 'Europe/Kiev', 'isp': 'O-net LLC', 'org': 'LLC O', 'as': 'AS200582 LLC O-NET', 'query': '185.102.186.47'}}

Створюємо клас, який буде наслідувати модуль “QWidget” з бібліотеки “PyQt6”

```
class App(QWidget):
    def __init__(self,strr):
        super () .__init__ ()
        self.str=strr
        self.way()
        self.start()
        self.response()
        self.btnb_search()
```

Отримуємо повний шлях до проекту завдяки модулю os, який допоможе нам позбавитись невдалого запуску програми, бо не в усіх користувачів повний шлях може складатися з латинської абетки.

```
def way(self):
    self.dirname=str(os.path.dirname(os.path.abspath(__file__)))
```

Мій повний шлях до проекту.

```
c:\Users\TrushOnDebug\Desktop\weather1
```

Запускаємо наш інтерфейс, підтягуючи елементи раніше збереженого “design.ui” для подальшого звернення до нього. Метод show() виводить всі елементи на екран.

```
def start(self):
    self.ui = uic.loadUi(r""+self.dirname+"\design.ui")
    self.ui.show()
```

Створюємо метод “response”, який буде опрацьовувати API-запит та оновлювати інтерфейс програми. Відразу забороняємо користувачу створювати порожній запит по містам.

```
def response(self):
    if str(self.ui.search.toPlainText())!='':
        self.str=self.ui.search.toPlainText()
```

Задаємо параметри запиту до сайту “openweathermap” та отримуємо відповідь в формат json

```
self.params = (
        ('q', self.str),
        ('units', 'metric'),
        ('lang','ua'),
        ('appid', '1d4c7545a1ded0a858a7015500fcd2f6'),
        )
```

```
        response =
requests.post('https://api.openweathermap.org/data/2.5/weather',params=self.params)
        data=response.json()
```

Отримуємо відповідь, де можемо за іменем індексу звернутися до змінної. Наприклад:

data["name"]>>Kyiv.

data['main']['temp']>> 0.91.

{'coord': {'lon': 30.5167, 'lat': 50.4333}, 'weather': [{'id': 500, 'main': 'Rain', 'description': 'легкий дощ', 'icon': '10n'}, {'id': 701, 'main': 'Mist', 'description': 'димка', 'icon': '50n'}], 'base': 'stations', 'main': {'temp': 0.91, 'feels_like': 0.91, 'temp_min': -0.19, 'temp_max': 1.51, 'pressure': 1010, 'humidity': 96}, 'visibility': 3400, 'wind': {'speed': 0.45, 'deg': 356, 'gust': 0.45}, 'rain': {'1h': 0.32}, 'clouds': {'all': 90}, 'dt': 1638803374, 'sys': {'type': 2, 'id': 2003742, 'country': 'UA', 'sunrise': 1638769382, 'sunset': 1638798915}, 'timezone': 7200, 'id': 703448, 'name': 'Kyiv', 'cod': 200}

Отримуємо код від сайту що все добре. Робимо умову: якщо код=200 тоді продовжуємо працювати.

```
 errorReq=data['cod']
        if str(errorReq) == '200':
```

Якщо код 404 або інший, ми сповіщаємо користувача про помилку, яка може бути з його провини, або з провини інтернет провайдера.

```
elif str(errorReq) == '404':
            QMessageBox.about(self, "Error", "Перевірте правильність ведення столиці в пошуку")
        else:
            QMessageBox.about(self, "Error_connection_"+str(errorReq), "Проблема с підключеням до серверу код помилки "+str(errorReq))
```

В середині умови ми опрацьовуємо код, який зберігає картинку в файлі проекту погодних умов.

```
if str(errorReq) == '200':

```

```
icon=data["weather"][0]['icon']
a=requests.get('https://openweathermap.org/img/wn/'+icon+'@2x.png').
content

with open(""+self.dirname+'\weather_icons.png','wb') as target:
        target.write(a)
```

Потім ми ініціалізуємо всі параметри в змінних.

```
city=data["name"]
self.temp=data['main']['temp']
country=data["sys"]['country']
description=data["weather"][0]['description']
wind=data["wind"]["speed"]
clouds=data['clouds']['all']
visibility=data['visibility']
self.int_r
temp=self.int_r()
```

Звертаємось до елементів на “design.ui”, яким ми давали назву класу в “qt design”.

Визиваємо метод “setText”, який вписує текст в елемент “label”, а також “setPixmap”, який виводить картинку в “img” .

```
self.ui.label_1.setText(str(country+" "+city))
            self.ui.label_2.setText("Поточна температура: "+str(temp)+" "+u'\u2103')
            self.ui.label_3.setText("Пог. явища: "+str(description))
            self.ui.label_4.setText("Швидкість вітру: "+str(wind)+" М/с")
            self.ui.label_5.setText("Хмарність: "+str(clouds)+"%")
            self.ui.label_6.setText("Видимість: "+str(visibility)+" м")
            pixmap = QPixmap(self.dirname+'\weather_icons.png')
            self.ui.img.setPixmap(pixmap)
        elif str(errorReq) == '404':
```

```
            QMessageBox.about(self, "Error", "Перевірте правильність ведення столиці в пошуку")
        else:
            QMessageBox.about(self, "Error_connection_"+str(errorReq), "Проблема с підключеням до серверу код помилки "+str(errorReq))
```

Створюємо метод “btnb_search”, який буде запускати метод “response”, елемент “push button” з назвою класу “btn_search”, щоб виводити інформацію про погоду яку задав користувач.

```
def btnb_search(self):
        self.ui.btn_search.clicked.connect(self.response)
```

Створюємо головний метод, який буде керувати всіма класами програми.

```
if __name__== '__main__':
    app = QApplication(sys.argv)
    iphandler=iPhandler()
```

Передаємо в клас “App” місто яке ми отримали в класі “iPhandler”. Та запускаємо програму через системний модуль запуску exec.

```
ex = App(str(iphandler.cityIp()))
    sys.exit(app.exec())
```

Результат

Програма відразу визначає наше місто по IP та опрацьовує наші дані.

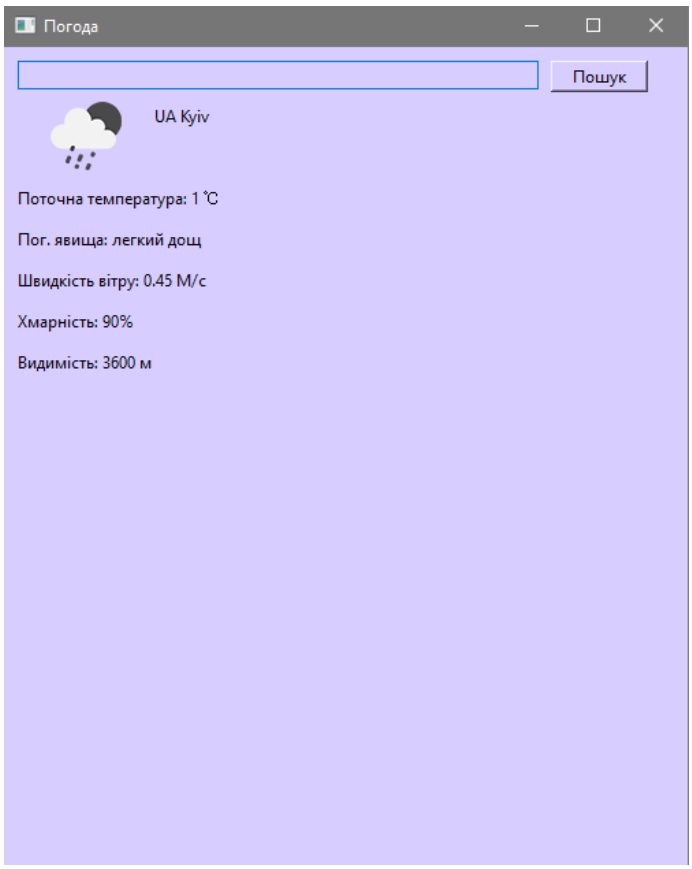


Рисунок 8.30

Перевірили пошук: програма спрацьовує вірно.

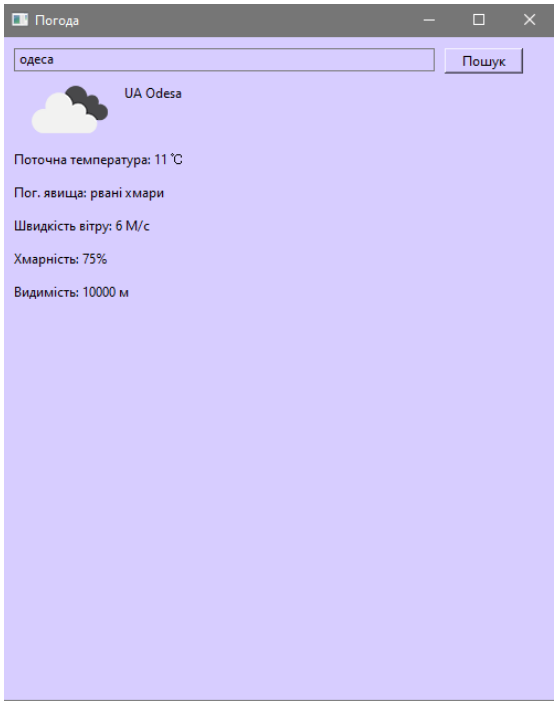


Рисунок 8.31

Код програми

```
import sys
import os
import requests
import http.client

from PyQt6.QtWidgets import QApplication, QWidget,QMessageBox
from PyQt6 import uic
from PyQt6.QtGui import QPixmap


class App(QWidget):
    def __init__(self,strr):
        super () .__init__ ()
        self.str=strr
        self.way()
        self.start()
        self.response()
        self.btnb_search()


    def way(self):
        self.dirname=str(os.path.dirname(os.path.abspath(__file__)))
        print(self.dirname)
```

```
    def start(self):
        self.ui = uic.loadUi(r""+self.dirname+"\design.ui")
        self.ui.show()
    def int_r(self):
        num = int(self.temp + (0.5 if self.temp > 0 else -0.5))
```

```
        return num
    def response(self):
        if str(self.ui.search.toPlainText())!='':
            self.str=self.ui.search.toPlainText()
        self.params = (
            ('q', self.str),
            ('units', 'metric'),
            ('lang','ua'),
            ('appid', '1d4c7545a1ded0a858a7015500fcd2f6'),
            )
        response =
requests.post('https://api.openweathermap.org/data/2.5/weather',params=self.params)
        data=response.json()
        errorReq=data['cod']
        if str(errorReq) == '200':

            icon=data["weather"][0]['icon']
            a = requests.get('https://openweathermap.org/img/wn/'+icon+'@2x.png').content
            with open(""+self.dirname+'\weather_icons.png','wb') as target:
                target.write(a)

            city=data["name"]
            self.temp=data['main']['temp']
            country=data["sys"]['country']
            description=data["weather"][0]['description']
            wind=data["wind"]["speed"]
            clouds=data['clouds']['all']
            visibility=data['visibility']
            self.int_r
            temp=self.int_r()
```

```
            self.ui.label_1.setText(str(country+" "+city))
            self.ui.label_2.setText("Поточна температура: "+str(temp)+" "+u'\u2103')
            self.ui.label_3.setText("Пог. явища: "+str(description))
            self.ui.label_4.setText("Швидкість вітру: "+str(wind)+" М/с")
            self.ui.label_5.setText("Хмарність: "+str(clouds)+"%")
            self.ui.label_6.setText("Видимість: "+str(visibility)+" м")
            pixmap = QPixmap(self.dirname+'\weather_icons.png')
            self.ui.img.setPixmap(pixmap)
        elif str(errorReq) == '404':
            QMessageBox.about(self, "Error", "Перевірте правильність ведення столиці в
пошуку")
        else:
            QMessageBox.about(self, "Error_connection_"+str(errorReq), "Проблема с
підключеням до серверу код помилки "+str(errorReq))
```

```
    def btnb_search(self):
        self.ui.btn_search.clicked.connect(self.response)



class iPhandler():
    def __init__(self):
        self.takeIP()
        self.cityIp()
    def takeIP(self):
        conn = http.client.HTTPConnection("ifconfig.me")
        conn.request("GET", "/ip")
        ip=str(conn.getresponse().read())
        print(ip,">> тільки отримали")
        ip = ip.split("'")
```

```
        print(ip,"в масиві")
        self.ip=ip[1]
        print(self.ip," яким нам потрібний")

    def cityIp(self):
        response = requests.post('http://ip-api.com/json/'+self.ip)
        data=response.json()
        print(data)
        return data['city']
if __name__== '__main__':
    app = QApplication(sys.argv)
    iphandler=iPhandler()
    ex = App(str(iphandler.cityIp()))
    sys.exit(app.exec())
```

Практичне завдання

1) Змінити колір бекграунда головної форми.

2) Вивести в інтерфейс повну інформацію про місто.

3) Вивести в інтерфейс всі зміні про погоду.

Література

1. Mark Lutz Learning Python Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2019, P-832.
2. О. Васильєв Програмування мовою Python К.: Видавництво «Навчальна книга - Богдан», 2019, С-504.
3. http://sandbox.scilink.ru/data/CIG/Python/part2.pdf
4. https://pythonworld.ru/novosti-mira-python/scientific-graphics-in-python.html
5. https://nbviewer.org/github/whitehorn/Scientific_graphics_in_python/blob/master/P3%20Chapter%2012%20Legends.ipynb
6. https://python-scripts.com/numpy#examples-numpy
7. https://numpy.org/doc/stable/reference/ufuncs.html#:~:text=A%20universal%20function%20(or%20ufunc,and%20several%20other%20standard%20features.