МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

ПРОГРАМУВАННЯ ІНТЕРНЕТ

Методичні рекомендації до виконання лабораторних робіт

для здобувачів вищої освіти спеціальності 121 "Інженерія програмного забезпечення" освітньої програми "Інженерія програмного забезпечення" першого (бакалаврського) рівня

Харків
ХНЕУ ім. С. Кузнеця
2025

УДК 004.42(072.034)
П78

Укладач Ю. Е. Парфьонов

Затверджено на засіданні кафедри інформаційних систем.
Протокол № 8 від 19.12.2024 р.

Самостійне електронне текстове мережеве видання

Програмування Інтернет [Електронний ресурс] : методичні
П78 рекомендації до виконання лабораторних робіт для здобувачів вищої освіти спеціальності 121 "Інженерія програмного забезпечення" освітньої програми "Інженерія програмного забезпечення" першого (бакалаврського) рівня / уклад. Ю. Е. Парфьонов. – Харків : ХНЕУ ім. С. Кузнеця, 2025. – 55 с.

Наведено методичні рекомендації, що сприяють поглибленню й закріпленню знань із теоретичних питань інтернет-програмування та набуттю практичних навичок щодо розроблення серверної частини вебзастосунків.

Рекомендовано для здобувачів вищої освіти спеціальності 121 "Інженерія програмного забезпечення" освітньої програми "Інженерія програмного забезпечення" першого (бакалаврського) рівня.

УДК 004.42(072.034)

Вступ

Відомо, що використання потужних комп'ютеризованих засобів неможливе без програмного забезпечення. Галузь розроблення програмного забезпечення набуває все більшого значення, оскільки складність і функціональні можливості комп'ютерної техніки, які швидко зростають, вимагають більш досконалих програмних засобів для задоволення потреб користувачів.

Сьогодні найбільш поширеними програмними системами є системи, розроблені з використанням інтернет-технологій. Це потребує від фахівців з інженерії програмного забезпечення чіткого уявлення про загальні концепції інтернет-програмування та використання сучасних засобів розроблення серверної частини вебзастосунків.

Здебільшого їх розробляють із застосуванням певного вебфреймворку, що спрощує процес розроблення застосунків за рахунок автоматизації і виключає написання рутинного коду.

Одним із популярних інструментів для швидкого розроблення серверної частини вебзастосунків мовою програмування Python є фреймворк Django. Він надає повний стек інструментів для створення вебзастосунків, зокрема вбудований ORM-фреймворк, та містить десятки додаткових функцій, які можна використовувати для виконання поширених завдань веброзроблення.

Django піклується про автентифікацію користувачів, адміністрування вмісту, карти сайтів, RSS-канали та багато інших завдань – прямо з коробки.

Базовою мовою програмування для виконання лабораторних робіт є Python, що використовують багато компаній-розробників програмного забезпечення.

Запропонований курс лабораторних робіт спрямовано на формування стійких практичних навичок щодо розроблення серверної частини вебзастосунків.

Для розроблення програм здобувачам вищої освіти рекомендовано використовувати інтегровані середовища розроблення, такі як PyCharm і Visual Studio Code.

Результати навчання та компетентності, які формує навчальна дисципліна, визначено в табл. 1.

Таблиця 1

Результати навчання та компетентності, які формує навчальна дисципліна

Результати навчання	Компетентності, якими має оволодіти здобувач вищої освіти
РН5	СК14
РН7	ЗК2, ЗК5, СК10, СК13
РН8	ЗК5, ЗК6
РН14	СК4
РН15	ЗК2, СК10, СК13
РН16	ЗК7
РН17	ЗК2, ЗК7, СК10, СК12, СК13

Примітка.

РН5. Знати і застосовувати відповідні математичні поняття, методи доменного, системного і об'єктно-орієнтованого аналізу та математичного моделювання для розробки програмного забезпечення.

РН7. Знати і застосовувати на практиці фундаментальні концепції, парадигми і основні принципи функціонування мовних, інструментальних і обчислювальних засобів інженерії програмного забезпечення.

РН8. Вміти розробляти людино-машинний інтерфейс.

РН14. Застосовувати на практиці інструментальні програмні засоби доменного аналізу, проектування, тестування, візуалізації, вимірювань та документування програмного забезпечення.

РН15. Мотивовано обирати мови програмування та технології розробки для розв'язання завдань створення і супроводження програмного забезпечення.

РН16. Мати навички командної розробки, погодження, оформлення і випуску всіх видів програмної документації.

РН17. Вміти застосовувати методи компонентної розробки програмного забезпечення.

ЗК02. Здатність застосовувати знання у практичних ситуаціях.

ЗК05. Здатність вчитися і оволодівати сучасними знаннями.

ЗК06. Здатність до пошуку, оброблення та аналізу інформації з різних джерел.

ЗК07. Здатність працювати в команді.

СК04. Здатність формулювати та забезпечувати вимоги щодо якості програмного забезпечення у відповідності з вимогами замовника, технічним завданням та стандартами.

СК10. Здатність накопичувати, обробляти та систематизувати професійні знання щодо створення і супроводження програмного забезпечення та визнання важливості навчання протягом всього життя.

СК12. Здатність здійснювати процес інтеграції системи, застосовувати стандарти і процедури управління змінами для підтримки цілісності, загальної функціональності і надійності програмного забезпечення.

СК13. Здатність обґрунтовано обирати та освоювати інструментарій з розробки та супроводження програмного забезпечення.

СК14. Здатність до алгоритмічного та логічного мислення.

Загальні рекомендації до виконання лабораторних робіт

Необхідним елементом успішного засвоєння матеріалу навчальної дисципліни "Програмування Інтернет" є самостійна робота здобувачів вищої освіти з технічною літературою. Тому в процесі виконання лабораторних робіт, разом із використанням теоретичних відомостей щодо тематики кожної роботи, наведених у цих методичних рекомендаціях, також доцільно переглянути певні джерела зі списку рекомендованої літератури.

Звіт з лабораторної роботи має містити:

1. Титульний аркуш (відомості про назву навчальної дисципліни; тему лабораторної роботи; прізвище та ініціали здобувача вищої освіти, курс, спеціальність, шифр навчальної групи; прізвище, ініціали та посаду викладача тощо).

2. Опис виконаних завдань:

а) умова завдання;

б) опис архітектури програми (структура класів, зв'язки між ними, алгоритми тощо);

в) приклади результатів роботи програми на тестових вихідних даних.

3. Висновки з лабораторної роботи з урахуванням усіх виконаних завдань: аналіз результатів, отриманих за кожним завданням; ступінь відповідності розроблених програм, формулюванню завдання; інша інформація.

Шаблон звіту наведено в додатку А.

Оцінювання результатів виконання лабораторної роботи

Результати виконання лабораторної роботи оцінюють за такими параметрами:

1. Якість розробленого програмного продукту.

Оцінюють за ступенем його оригінальності, відповідності вимогам певного завдання роботи та якості розробленого програмного коду.

2. Виконання календарного плану роботи відповідно до технологічної карти дисципліни.

3. Якість презентації результатів роботи під час її захисту.

4. Якість виконання звіту.

Оцінюють за ступенем оригінальності його контенту та відповідності вимогам до змісту й оформлення, викладеним у цих методичних рекомендаціях.

Установлюють таки рівні оцінювання результатів роботи (відповідно до технологічної карти навчальної дисципліни):

Рівень 1. До 70 % від максимальної оцінки.

Рівень 2. До 80 % від максимальної оцінки.

Рівень 3. До 90 % від максимальної оцінки.

Рівень 4. До 100 % від максимальної оцінки.

Змістовий модуль 1. Використання засобів інтернет-програмування

Теми 1, 2. Вступ до дисципліни. Використання інтернет-протоколів у Python мовою Python

Лабораторна робота 1. Використання інтернет-протоколів у Python

Мета лабораторної роботи:

1. Набуття практичних навичок використання певних інтернет-протоколів у програмах мовою Python.
2. Удосконалення навичок роботи з рекомендованими інтегрованими середовищами програмування.

Перед виконанням лабораторної роботи здобувач вищої освіти має знати основи використання інтернет-протоколів SMTP, POP3, IMAP тощо в Python.

Після виконання лабораторної роботи здобувач вищої освіти має вміти використовувати бібліотеки Python для роботи з електронною поштою.

Теоретична частина

Розподілені програмні системи (РПС) – це системи, що складаються з кількох автономних комп'ютерів або пристроїв, які обмінюються інформацією через мережу.

Компоненти РПС може бути розташовано на різних фізичних локаціях, але вони взаємодіють через стандартизовані інтерфейси, забезпечуючи таким чином синхронну або асинхронну взаємодію. РПС широко використовують у хмарних обчисленнях, корпоративних мережах, розподілених базах даних і в багатьох інших сферах.

Програмні компоненти РПС містять різноманітні модулі, що виконують конкретні функції в межах розподіленої системи. До основних компонентів належать *клієнтські компоненти* – програми, які ініціюють запити до серверів або інших компонентів системи; *серверні компоненти* – програми, що обробляють запити від клієнтів і надають відповідні послуги; *мережеві компоненти* – протоколи, що використовують для передачі даних між компонентами системи.

Архітектура РПС визначає організацію та структуру компонентів і способи їх взаємодії. Вона може бути різною залежно від конкретних вимог та застосувань, наприклад, клієнт-серверна архітектура, багатошарова архітектура, Peer-to-peer (P2P) тощо.

Для спрощення взаємодії між компонентами розподіленої системи використовують так зване проміжне програмне забезпечення (ППЗ). Воно може містити різноманітні сервіси, такі як управління сесіями, безпека, а також оброблення запитів.

Головними типами ППЗ є: *комунікаційне* – забезпечує комунікацію між компонентами через мережу; *транзакційне* – підтримує виконання транзакцій у розподілених базах даних; *оброблення повідомлень* – відповідає за асинхронний обмін даними між компонентами.

Потужним засобом для розроблення розподілених програмних систем є мова програмування Python. Зокрема, вона дозволяє розробляти клієнт-серверні застосунки, що підтримують популярні протоколи: HTTP, FTP, SMTP та ін. Це можливо завдяки численним бібліотекам, таким як socket, requests, Beautiful Soup, asyncio, Twisted, selenium тощо.

Основними інтернет-протоколами для обміну електронною поштою є SMTP, POP3 та IMAP. SMTP (Simple Mail Transfer Protocol) використовують для відправлення повідомлень, тоді як POP3 (Post Office Protocol) та IMAP (Internet Message Access Protocol) призначено для отримання листів із поштових серверів. У сучасних поштових клієнтах SMTP використовують для надсилання пошти, а IMAP або POP3 – для доступу до повідомлень, що зберігаються на сервері.

У Python для роботи із цими протоколами використовують вбудовані бібліотеки, такі як smtplib, poplib і imaplib.

Далі наведено короткий огляд кожного з протоколів та приклади їх використання, які демонструють базові операції з кожним протоколом і можуть бути розширені відповідно до потреб користувача.

SMTP (Simple Mail Transfer Protocol)

SMTP використовують для надсилання електронних листів. Цей протокол також широко застосовують для автоматизації поштових розсилок у маркетингових або корпоративних застосунках. Багато сервісів для масових розсилок (як-от SendGrid або Mailgun) працюють через SMTP.

У Python для роботи із SMTP доступна бібліотека smtplib.

Основні кроки роботи з протоколом SMTP:

1. Підключення до SMTP-сервера.
2. Автентифікація (за потреби).
3. Надсилання повідомлення.

Приклад коду:

```
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

# Налаштування SMTP-сервера
smtp_server = "smtp.gmail.com"
smtp_port = 587
username = "your_email@gmail.com"
password = "your_password"

# Створення повідомлення
msg = MIMEMultipart()
msg['From'] = username
msg['To'] = "recipient@example.com"
msg['Subject'] = "Тема листа"
msg.attach(MIMEText("Текст повідомлення", "plain"))

try:
    # Підключення до сервера
    with smtplib.SMTP(smtp_server, smtp_port) as server:
        server.starttls()  # Захищене з'єднання
        server.login(username, password)
        server.send_message(msg)
        print("Лист успішно відправлено!")
```

```
except Exception as e:
    print(f"Помилка: {e}")
```

POP3 (Post Office Protocol, версія 3)

Протокол POP3 використовують для отримання електронних листів. У Python для цього можна використовувати бібліотеку poplib.

Основні кроки роботи:

1. Підключення до POP3-сервера.
2. Автентифікація.
3. Завантаження повідомлень.

Приклад коду:

```
import poplib
from email.parser import BytesParser

# Налаштування POP3-сервера
pop3_server = "pop.gmail.com"
username = "your_email@gmail.com"
password = "your_password"

try:
    # Підключення до сервера
    with poplib.POP3_SSL(pop3_server) as server:
        server.user(username)
        server.pass_(password)

        # Отримання списку листів
        messages = server.list()[1]
        print(f"Знайдено {len(messages)} листів")

        # Завантаження першого листа
        response, lines, octets = server.retr(1)
        message_content = b"\n".join(lines)
        message = BytesParser().parsebytes(message_content)
        print(f"Тема: {message['subject']}")
except Exception as e:
    print(f"Помилка: {e}")
```

IMAP (Internet Message Access Protocol)

IMAP забезпечує більш гнучкий доступ до поштових скриньок порівняно з POP3. Йому зазвичай віддають перевагу перед POP3 для отримання пошти, оскільки IMAP дозволяє зберігати копії всіх листів на сервері,

що зручно для архівування або перегляду історії листування на різних пристроях.

Для роботи з протоколом IMAP у Python використовують бібліотеку imaplib.

Основні кроки роботи:

1. Підключення до IMAP-сервера.
2. Автентифікація.
3. Виконання команд (наприклад, пошук або завантаження листів).

Приклад коду:

```
import imaplib
import email

# Налаштування IMAP-сервера
imap_server = "imap.gmail.com"
username = "your_email@gmail.com"
password = "your_password"

try:
    # Підключення до сервера
    with imaplib.IMAP4_SSL(imap_server) as server:
        server.login(username, password)
        server.select("inbox")  # Вибір папки "Вхідні"

        # Пошук непрочитаних листів
        status, messages = server.search(None, "UNSEEN")
        if messages[0]:
            for num in messages[0].split():
                status, data = server.fetch(num, "(RFC822)")
                raw_email = data[0][1]
                msg = email.message_from_bytes(raw_email)
                print(f"Тема: {msg['subject']}")
        else:
            print("Немає непрочитаних листів.")
except Exception as e:
    print(f"Помилка: {e}")
```

Отже, Python надає зручні інструменти для роботи з основними інтернет-протоколами для електронної пошти. Використання бібліотек smtplib, poplib та imaplib дозволяє інтегрувати функції надсилання і отримання електронних листів у сучасні застосунки.

Типові проблеми, які можуть виникнути при використанні цих протоколів, – це помилки автентифікації, помилки із SSL-сертифікатами, а також обмеження на кількість листів, що можна надіслати через SMTP-сервери.

Для розв'язання цих проблем важливо забезпечити правильні налаштування на сервері та використовувати відповідні методи оброблення помилок у коді.

Практична частина

Завдання. Додайте до консольної програми мовою Python, призначеної для автоматизації предметної області відповідно до варіанта завдання, нову функціональність для моніторингу дій користувача та сповіщення "адміністратора" про його результати.

Функціональні вимоги:

1. Циклічне введення з консолі початкових значень певних полів та обчислення розрахункових даних кожного запису відомості; виведення на консоль початкових і розрахункових даних усіх записів відомості у формі таблиці з горизонтальними і вертикальними лініями сітки (числові значення слід виводити з певною кількістю знаків після коми з використанням засобів форматного виведення).
2. Відстеження всіх можливих варіантів використання програми користувачем.
3. Додавання запису про певну дію до лог-файлу одразу після її виконання (назва, дата й час виконання) з використанням модуля logging.
4. Після вибору користувачем варіанта використання "Завершення програми" відправляти лог-файл на email "адміністратора" як вкладення до електронного листа.
5. Електронний лист має містити адреси відправника та отримувача, тему, контент (звичайний текст і HTML). Для надсилання електронної пошти використовувати модулі ssl, smtplib та пакет email.

Нефункціональні вимоги:

Вихідний код програми має складатися з визначень функцій та їх викликів (або з визначень класів та викликів їхніх методів) і міститися в декількох файлах, розподілених по пакетах відповідно до функціонального призначення (рівні 2 – 4).

1. Наявність текстового меню для вибору не менш ніж трьох можливих варіантів дій користувача під час роботи з програмою (рівні 1 – 4), яке реалізовано з використанням рекурсивних викликів відповідної функції (рівень 4).

2. Програму потрібно виконувати у віртуальному середовищі, створеному з використанням утиліти venv, pipenv тощо (рівні 3, 4).

Варіант 1

Відомість нарахування зарплати співробітникам підприємства

Прізвище	Зарплата, грн	Утримано, грн	Видано, грн
F	Z	P	S = Z − P
…	…	…	…

Варіант 2

Відомість витрати палива на автобазах міста

Автобаза	Витрачено палива, кг	Кількість автомашин, шт.	Середня витрата палива, кг
A	T	K	C = T / K
…	…	…	…

Варіант 3

Відомість використання машинного часу на обчислювальному центрі

Кафедра	Використання машинного часу, год		Відхилення від плану	
	за планом	фактично	у годинах	у %
K	P	F	O1 = P − F	O2 = O1 × 100 / P
…	…	…	…	…

Варіант 4

Відомість споживання електроенергії на заводах міста

Завод	Споживання електроенергії, кВт / год		Відхилення від плану	
	за планом	фактично	у кВт / год	у %
Z	P	F	$O_1 = P - F$	$O_2 = O_1 \times 100 / P$
…	…	…	…	…

Варіант 5

Відомість руху матеріалів на складі підприємства за звітний період

Склад	Рух матеріалів за період, грн			Залишок на кінець періоду
	залишок на початок періоду	отримано	видано	
C	O_c	P	V	$R = O_c + P - V$
…	…	…	…	…

Варіант 6

Відомість прибутку підприємства за звітний період за видами продукції

Продукція	Кількість, шт.	Оптова ціна, грн	Собівартість, грн	Прибуток, грн
Pr	K	Z	C	P = K(Z − C)
…	…	…	…	…

Варіант 7

Відомість відвідування занять студентами

Прізвище	Пропущена, годин		Пропуски	
	усього	виправдано	у годинах	у %
F	V	O	P1 = V − O	P2 = P1 × 100 / V
…	…	…	…	…

Варіант 8

Відомість обсягу поставок продукції

Продукція	Шифр	Обсяг поставки, шт.	Оптова ціна, грн	Обсяг поставки, грн
P	H	V	Z	O = V × Z
…	…	…	…	…

Варіант 9

Відомість розрахунку середньої вартості перевезення авіапасажирів

Тип літака	Рейс	Витрати на рейс, грн	Кількість пасажирів	Середня вартість перевезення, грн
T	R	Z	K	S = Z / K
…	…	…	…	…

Варіант 10

Відомість обліку часу роботи верстатів підприємства

Тип верстата	Час роботи, год		Відхилення від плану	
	за планом	фактично	у годинах	у %
Z	P	F	O1 = P − F	O2 = O1 × 100 / P
…	…	…	…	…

Варіант 11

Відомість випуску деталей робочими цеху

Прізвище	Кількість деталей, шт.		Брак	
	виготовлено	прийнято	шт.	%
Z	P	F	O1 = P − F	O2 = O1 × 100 / P
…	…	…	…	…

Варіант 12

Відомість наявності та руху основних фондів підприємства

Назва фонду	Наявність на початок року, шт.	Надійшло, шт.	Вибуло, шт.	Наявність на кінець року, шт.
F	N1	P	V	N2 = N1 + P − V
…	…	…	…	…

Варіант 13

Відомість обліку запчастин на складі підприємства

Марка автомобіля	Назва запчастини	Кількість одиниць товару, шт.	Ціна без ПДВ, грн	Ціна з ПДВ, грн
M	N	Q	P	AVP = P + 0,2 × P
…	…	…	…	…

Варіант 14

Відомість обліку успішності здобувача вищої освіти

Прізвище	Ім'я	Оцінка з першої дисципліни	Оцінка з другої дисципліни	Середня оцінка
LN	FN	M1	M2	AM = (M1 + M2) / 2
…	…	…	…	…

Варіант 15

Відомість обліку оплати за послуги мобільного зв'язку

Прізвище	Нараховано в середині мережі	Нараховано в інших мережах	Нараховано в роумінгу	Нараховано всього з ПДВ
LN	IH	IO	F	T = (IH + IO + F) × 1,2
…	…	…	…	…

Контрольні запитання

1. Що таке SMTP? Для яких завдань його використовують?
2. Яке призначення функції starttls() у бібліотеці smtplib? Чому її важливо викликати перед автентифікацією?
3. Як у Python можна створити багаточастинне повідомлення для надсилання через SMTP?
4. Що станеться, якщо під час відправлення листа в SMTP-з'єднанні виникне помилка?
5. Яка основна відмінність між POP3 і IMAP у способі доступу до листів?
6. Яку функцію використовують для завантаження вмісту конкретного повідомлення за протоколом POP3?
7. Які переваги IMAP порівняно з POP3?
8. Що означають команди SELECT, SEARCH і FETCH у бібліотеці imaplib?
9. Як отримати список непрочитаних повідомлень за допомогою IMAP у Python?

Тема 3. Використання Web Scraping у Python

Лабораторна робота 2. Використання Web Scraping у Python

Мета лабораторної роботи:

1. Набуття практичних навичок використання Python-бібліотек для Web Scraping.
2. Удосконалення навичок роботи з рекомендованими інтегрованими середовищами програмування.

Перед виконанням лабораторної роботи здобувач вищої освіти має знати основи використання Python-бібліотек для Web Scraping.

Після виконання лабораторної роботи здобувач вищої освіти має вміти самостійно розробляти застосунки для Web Scraping.

Теоретична частина

Web Scraping (вебскрапінг) – це процес автоматичного збирання даних із вебсторінок.

Python є однією із найпопулярніших мов для вебскрапінгу завдяки своїм простим і потужним бібліотекам, таким як *BeautifulSoup*, *requests* і *selenium*.

Етапи вебскрапінгу:

1. Вибір джерела даних.

Виберіть вебсторінку, що містить потрібну інформацію. Переконайтеся, що скрапінг цього ресурсу не порушує правил його використання. Для аналізу структури сторінки використовуйте інструменти розробника, що надає певний веббраузер.

2. Отримання HTML-коду сторінки.

Завантажте вміст сторінки за допомогою HTTP-запиту, використовуючи бібліотеку *requests*. Перевірте статус відповіді (код 200 означає успішне завантаження).

3. Аналіз HTML-коду.

Визначте потрібні HTML-елементи, наприклад, заголовки, списки, таблиці або посилання. Для цього можна використовувати CSS-селектори або XPATH. Бібліотека *BeautifulSoup* спрощує роботу з HTML-структурою.

4. Витягування даних.

Виконайте парсинг HTML, знаходячи конкретні елементи за допомогою методів find або select. Для отримання JavaScript-контенту динамічних вебсторінок доцільно скористатися бібліотекою *Selenium*.

5. Збереження даних.

Зберігайте зібрані дані у форматах JSON, CSV або базах даних для подальшого аналізу чи використання. Цей процес можна автоматизувати для регулярного збирання інформації.

Основні інструменти вебскрапінгу:

1. *Requests.*

Requests – це бібліотека Python для відправлення HTTP-запитів. Вона підтримує GET, POST та інші методи запитів. Ця бібліотека проста у використанні, а також дозволяє додавати до запитів заголовки кукі або інші параметри.

Приклад коду:

```
import requests

url = "https://example.com"
response = requests.get(url)
if response.status_code == 200:
   print("Сторінку успішно завантажено!")
   print(response.text)  # Виведення HTML-коду на консоль
else:
   print(f"Помилка: {response.status_code}")
```

2. *BeautifulSoup.*

BeautifulSoup дозволяє зручно аналізувати і парсити HTML або XML документи. Ця бібліотека ефективна для статичних сторінок і пропонує багато методів для витягу даних, наприклад:

find – знаходить перший збіг за тегом чи атрибутами;

find_all – знаходить усі збіги;

select – знаходить елементи за CSS-селекторами.

Також вона підтримує навігацію за допомогою атрибутів: .parent, .children, .next_sibling, .previous_sibling та інших.

Приклад коду:

```
from bs4 import BeautifulSoup

html = """<html><body><h1>Hello, world!</h1></body></html>"""
soup = BeautifulSoup(html, "html.parser")

# Знаходимо елемент <h1>
h1 = soup.find("h1")
paragraph = soup.find("p", class_="intro")
print(f"Заголовок: {header.text}")
print(f"Абзац: {paragraph.text}")
```

3. *Selenium.*

Selenium дозволяє автоматизувати роботу браузера й отримувати динамічний контент. Це корисно для сайтів, які завантажують дані через JavaScript.

Можливості Selenium:

використання різних браузерів (Chrome, Firefox, Edge);

симуляція дій користувача (натискання кнопок, уведення тексту);

робота з динамічними елементами за допомогою wait.

Приклад скрапінгу динамічної сторінки:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

# Налаштування драйвера
options = webdriver.ChromeOptions()
options.add_argument("--headless")  # Запуск без інтерфейсу

# Відкриття сторінки
driver = webdriver.Chrome(options=options)
driver.get("https://example.com")

try:
    # Очікування завантаження елемента
    element = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.ID, "dynamic-content"))
    )
    print(f"Динамічний контент: {element.text}")
finally:
    driver.quit()
```

Selenium дозволяє виконувати рендеринг динамічних веб-сторінок та взаємодіяти з елементами сторінки, як-от кнопки, випадаючі списки тощо. Але, швидкість її роботи значно нижча, ніж у бібліотек для статичних сторінок. Також вона потребує додаткових ресурсів для роботи браузера.

Отримані дані може бути збережено у форматах JSON, CSV або в базі даних.

Приклад збереження в CSV:

```
import csv

data = [
   {"Name": "Alice", "Age": 25},
   {"Name": "Bob", "Age": 30},
]
with open("data.csv", "w", newline="") as file:
   writer = csv.DictWriter(file, fieldnames=["Name", "Age"])
   writer.writeheader()
   writer.writerows(data)

print("Дані збережено в data.csv")
```

Web Scraping у Python – це ефективний спосіб автоматизованого збирання даних із вебсторінок. Інструменти, такі як *requests*, *Beautiful Soup* та *selenium*, забезпечують гнучкість і функціональність для реалізації проєктів різної складності. Головне – дотримуватись етики та законодавства під час роботи з вебресурсами.

Юридичні аспекти вебскрапінгу залежать від законодавства конкретної країни та умов використання вебсайтів. У багатьох юрисдикціях вебскрапінг може порушувати авторські права або умови користування, якщо дані копіюють без дозволу. Крім того, доступ до інформації шляхом обхідних методів (наприклад, ігнорування файлу robots.txt) може бути кваліфіковано як несанкціонований доступ до комп'ютерних систем, що регулюється кримінальним законодавством. Щоб уникнути юридичних ризиків, важливо отримати дозвіл власника сайту або працювати виключно з відкритими даними, які надають для загального користування.

Практична частина

Завдання. Розробіть консольну програму з використанням Python-бібліотек, таких як requests, Beautiful Soup, selenium тощо, яка витягує всю контактну інформацію компанії (години роботи, номери телефонів, url-адресу вебсайту тощо) з корпоративного вебсайту та відображує її.

Інтерфейс користувача має містити текстові повідомлення, що стосуються введення та виведення основних і допоміжних даних, їх валідації тощо.

Варіанти цільових вебсайтів:

1) keramasvit.com.ua;
2) rozetka.com.ua;
3) elmir.ua;
4) santehzavod.com;
5) aquakharkov.com.ua;
6) aqua.tt.ua;
7) keramis.com.ua;
8) allo.ua;
9) santehnika.kh.ua;
10) eurosant.kiev.ua;
11) dnipro-m.ua;
12) chipchip.ua;
13) epicentrk.ua;
14) atlant-shop.com.ua;
15) будмаркет.com.

Контрольні запитання

1. Як створити об'єкт BeautifulSoup для аналізу HTML-коду?
2. Як знайти всі теги певного типу (наприклад, <a>) на вебсторінці?
3. У чому різниця між методами .find() і .find_all() у BeautifulSoup?
4. Як у BeautifulSoup отримати текст із HTML-тегу та доступ до його атрибутів, наприклад, href?
5. Як обмежити пошук тегів за допомогою CSS-селекторів у Beautiful Soup?
6. Як виконати HTTP-запит на отримання вебсторінки за допомогою бібліотеки requests?
7. Що таке статус-код HTTP-відповіді? Як перевірити успішність запиту за допомогою requests?
8. Як передати додаткові параметри (наприклад, заголовки або дані форми) під час виконання запитів із requests?
9. Як ініціалізувати вебдрайвер Selenium для автоматизації браузера?
10. Як дочекатися завантаження певного елементу на сторінці перед його обробленням за допомогою Selenium?

Змістовий модуль 2. Основи розроблення серверної частини вебзастосунків на базі фреймворку Django

Тема 4. Основи фреймворку Django

Лабораторна робота 3. Основи Django

Мета лабораторної роботи:

1. Набуття практичних навичок розроблення вебзастосунків мовою Python.
2. Набуття практичних навичок використання фреймворку Django.
3. Удосконалення навичок роботи з рекомендованими інтегрованими середовищами програмування.

Перед виконанням лабораторної роботи здобувач вищої освіти має вивчити основи використання фреймворку Django.

Після виконання лабораторної роботи здобувач вищої освіти має вміти самостійно розробляти прості вебзастосунки з використанням фреймворку Django.

Теоретична частина

Django – це потужний серверний вебфреймворк на Python, який дозволяє швидко розробляти вебзастосунки.

Він використовує архітектуру MVT (Model-View-Template) (рис. 1).

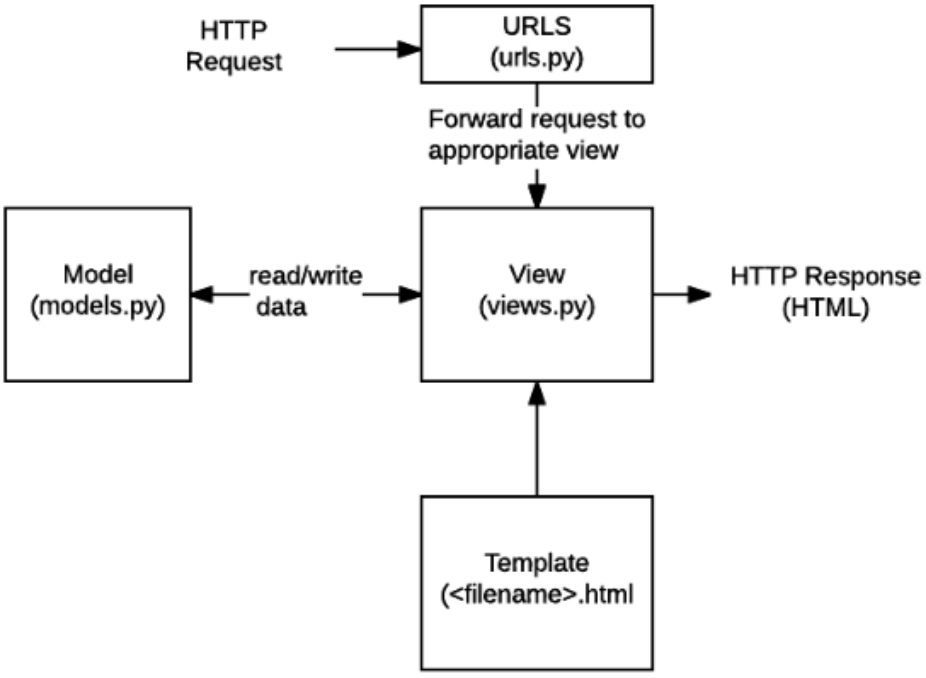

Рис. 1. **Архітектура MVT**

Основними компонентами Django є моделі (Model), види (View), шаблони (Template) та URL-диспетчер. Далі подано опис кожного із цих компонентів із прикладами коду.

Моделі відповідають за роботу з базою даних. Вони визначають структуру даних та надають інтерфейс для взаємодії з ними.

У найпростішому Django-застосунку може не бути жодної моделі, якщо не передбачено взаємодію з базою даних.

За замовчуванням Django використовує СКБД SQLite, але в секції DATABASES файлу settings.py її можна змінити на MySQL, MariaDB, PostgreSQL або Oracle, які офіційно підтримує фреймворк, чи на Microsoft SQL Server та деякі інші СКБД.

Приклад секції DATABASES файлу settings.py:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mydatabase',
        'USER': 'myuser',
        'PASSWORD': 'mypassword',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

Приклад моделі Post:

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title
```

Після створення моделей для фіксації змін у схемі бази даних потрібно застосувати міграції:

```
python manage.py makemigrations
python manage.py migrate
```

У результаті виконання цих команд для моделі Post у певній базі даних за замовчуванням буде створено таблицю app_name.Post, де app_name – назва застосунку.

Види – це функції або класи, які обробляють HTTP-запити та повертають HTTP-відповіді.

Приклад виду home на базі функції:

```
from django.shortcuts import render

def home(request):
    context = {
        'title': 'Welcome to my blog!',
        'content': 'This is the home page.',
    }
    return render(request, 'home.html', context)
```

Шаблони використовують для створення HTML-сторінок із динамічними даними.

Приклад простого шаблону home.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>My Blog</title>
</head>
<body>
    <h1>{{ title }}</h1>
    <p>{{ content }}</p>
</body>
</html>
```

URL-диспетчер визначає, яка функція обробляє запити до певного URL, наприклад:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
]
```

Також Django може автоматично створити адмін-панель для управління даними моделей. У найпростішому випадку для цього потрібно зареєструвати модель, таку як Post:

```
from django.contrib import admin
from .models import Post

admin.site.register(Post)
```

Крім того, цей фреймворк дозволяє обробляти CSS, JavaScript і зображення. Треба створити теку static у вашому застосунку, додати туди статичні файли, наприклад, styles.css, і "підключити" файли в певному шаблоні:

```
{% load static %}
<link rel="stylesheet" href="{% static 'styles.css' %}">
```

Практична частина

Завдання. Розробіть вебзастосунок на базі фреймворку Django, який надає функціональність "калькулятора" для виконання за запитом користувача будь-якої з чотирьох арифметичних операцій ("+", "−", "×", "/") із двома дійсними числами та додаткової операції, що визначено варіантом завдання (див. далі).

Загальні вимоги до клієнтської частини (front end):

1. Наявність не менш ніж однієї вебсторінки, контент якої відповідає постановці завдання.
2. Стилізація вебсторінок із використанням сучасних версій CSS (рівень 3) або фреймворку Bootstrap (рівень 4).
3. Перевірка даних, уведених користувачем, щодо наявності та відповідності формату дійсного числа (рівні 2 – 4), а також належності до діапазону відповідно до варіанта завдання (рівні 3, 4), із видачею, за потреби, попереджувального повідомлення.
4. Після вибору користувачем будь-якої операції введені дані (у разі їх припустимості) буде відправлено на сервер, який має повертати результат виконання вибраної операції.
5. Результат виконання вибраної операції слід виводити у вебінтерфейсі користувача з трьома знаками після коми *(використання для цього функції математичного округлення заборонено)*.

Загальні вимоги до серверної частини (back end):

Приймає дані від клієнтської частини, передає їх певному FBV (Function-based View) для оброблення, повертає результати обчислень клієнтській частині.

Обчислювальний блок "калькулятора" має бути реалізовано в окремому сервісному пакеті, який містить модуль Python для виконання операцій "калькулятора" (рівні 3, 4).

Номер варіанта	**Додаткова операція калькулятора**
1	$a^{1/2}$, де a > 0
2	Перетворення аргументу a (кілометр→миля), де 1 ≤ a ≤ 1 000
3	exp(a), де −5 ≤ a ≤ 5
4	Перетворення аргументу a (Цельсій→Фаренгейт), де −273 ≤ a ≤ 273
5	lg (a), де a > 0
6	Перетворення аргументу a (байт→кілобайт), де 1 ≤ a ≤ 10^9
7	sin(a), де 0° ≤ a ≤ 360°
8	cos(a), де 0° ≤ a ≤ 360°
9	tg(a), де −90° < a < 90°
10	Перетворення аргументу a (гривня→євро), де 1 ≤ a ≤ 100 000
11	ctg(a), де −180° < × < 180°
12	a^b, де a > 0, b > 0
13	ln(a), де a > 0
14	Перетворення аргументу a (радіани→градуси), де 0 < a < 6,28
15	Перетворення аргументу a (градуси→радіани), де 0° ≤ a ≤ 360°

Примітка. Слід урахувати, що стандартні функції модуля math для обчислення тригонометричних функцій очікують параметр у радіанах, а дані, які вводить користувач, мають бути в градусах.

Контрольні запитання

1. Що таке Django? Які основні завдання виконує цей фреймворк?
2. Для чого використовують моделі в Django?
3. Які типи полів найчастіше використовують у моделях Django?
4. Що таке міграції в Django? Які команди використовують для їх створення та застосування?
5. Як налаштовують маршрути (URL) у Django?
6. Чим відрізняється функціональний вигляд від класового вигляду в Django? У яких випадках доцільно використовувати кожен із них?
7. Що таке шаблони в Django? Як у них передавати динамічні дані з виду?
8. Як підключити CSS-стилі та JavaScript у шаблоні Django?
9. Як зареєструвати модель в адмін-панелі Django?
10. Як змінити стандартну базу даних SQLite на PostgreSQL або іншу базу в Django?

Теми 5 – 7. Моделі Django. Види та шаблони в Django. Форми

Лабораторна робота 4. Використання баз даних у вебзастосунках на базі фреймворку Django

Мета лабораторної роботи:

1. Набуття практичних навичок використання Django Database ORM.
2. Удосконалення навичок роботи з рекомендованими інтегрованими середовищами програмування.

Перед виконанням лабораторної роботи здобувач вищої освіти має вивчити основні поняття щодо роботи з базою даних із вебзастосунку Django.

Після виконання лабораторної роботи здобувач вищої освіти має вміти самостійно розробляти прості вебзастосунки з використанням Django Database ORM.

Теоретична частина

Django дозволяє швидко створювати вебзастосунки з інтеграцією баз даних.

Однією з ключових особливостей цього фреймворку є шар Django ORM (Object-Relational Mapping), який забезпечує роботу з базами даних через моделі.

Модель у Django – це Python-клас, який описує структуру таблиці в базі даних. Атрибути класу відповідають стовпцям таблиці, а кожен об'єкт класу – рядку.

Приклад моделі Book:

```
from django.db import models
class Book(models.Model):
    title = models.CharField(max_length=255)
    author = models.CharField(max_length=100)
    publication_date = models.DateField()
    price = models.DecimalField(max_digits=10, decimal_places=2)

    def __str__(self):
        return self.title
```

Для синхронізації моделей із базою даних Django використовує систему міграцій. Міграції створюються автоматично на основі змін у моделях.

Django ORM дозволяє працювати з базою даних за допомогою Python-коду, що робить запити більш читабельними та безпечними.

Створення нового об'єкта Book та відповідного запису в таблиці бази даних:

```
book = Book.objects.create(
    title="Python for Beginners",
    author="John Doe",
    publication_date="2024-01-01",
    price=29.99
)
```

Отримання всіх записів таблиці бази даних:

```
# Отримати всі книги
books = Book.objects.all()
# Фільтрація за автором
books_by_author = Book.objects.filter(author="John Doe")
# Отримати одну книгу
book = Book.objects.get(id=1)
```

Оновлення певного запису таблиці бази даних:

```
book = Book.objects.get(id=1)
book.price = 34.99
book.save()
```

Видалення певного запису таблиці бази даних:

```
book = Book.objects.get(id=1)
book.delete()
```

Django підтримує різні типи зв'язків між таблицями: "один до одного" через поле типу OneToOneField, "багато до одного" через поле типу ForeignKey та "багато до багатьох" через поле типу ManyToManyField.

Приклад зв'язку "багато до одного" між моделями Author і Book:

```
class Author(models.Model):
    name = models.CharField(max_length=100)
class Book(models.Model):
    title = models.CharField(max_length=255)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
```

У цьому прикладі кожна книга має автора, а видалення автора призведе до видалення пов'язаних із ним книг.

Практична частина

Завдання 1. Використовуючи фреймворк Django, розробіть "скелет" вебзастосунку, призначеного для виконання CRUD-операцій із даними предметної області, що зберігаються в реляційній базі даних SQLite (за замовчуванням). Конкретну предметну область вибирає здобувач вищої освіти відповідно до власних уподобань і погоджує її з викладачем. Дані предметної області мають складатися не менш ніж із двох сутностей, між якими є зв'язок "багато до одного". Для роботи із цими даними треба використовувати Django Database ORM.

Додайте до "скелета" необхідні моделі даних і зв'язки між ними згідно з предметною областю та згенеруйте таблиці бази даних.

Виконайте CRUD-операції з даними предметної області з використанням Django Database ORM та відповідного Python API:

одну операцію кожного типу з окремими моделями (рівні 1, 2);

дві операції кожного типу, зокрема з використанням зв'язку "один до багатьох" (рівні 3, 4).

Код команд Python API підготуйте заздалегідь і продемонструйте викладачу результати їх виконання в Python-консолі.

Завдання 2 (Ця та наступна лабораторні роботи – по 50 % функціональності в кожній).

Увага! Розроблення застосунку рекомендовано виконувати з використанням системи контролю версій Git і репозиторію GitHub, GitLab тощо.

На базі "скелета", створеного в першому завданні, розробіть повноцінний вебзастосунок для виконання CRUD-операцій із даними предметної області, що зберігаються в базі даних.

Його слід побудувати відповідно до архітектурного шаблону Model-View-Template. Застосунок має містити декілька вебсторінок із контентом згідно з предметною областю (текст, зображення тощо), види (Function-based Views), класи-моделі (Models) та інші необхідні компоненти.

Дані предметної області мають складатися не менш ніж із двох сутностей, між якими є зв'язок "багато до одного". Для роботи з цими даними треба використовувати Django Database ORM та підхід "Code First".

Загальні вимоги до клієнтської частини (front end)

На вебсторінках застосунку слід використовувати Django Template Language, стилі CSS (рівень 2), сучасну версію CSS-фреймворку Bootstrap (рівні 3, 4).

Загальні вимоги до серверної частини (back end)

Приймає дані від клієнтської частини, передає їх певному FBV (Function-based View) для оброблення, повертає результати оброблення даних клієнтській частині. Бізнес-логіку застосунку потрібно реалізовувати в окремому сервісному пакеті, який містить не менш ніж один модуль Python (рівні 3, 4).

Логіка функціонування вебзастосунку

Користувач завантажує головну вебсторінку. Із цієї сторінки він може перейти до елементів застосунку, що реалізують варіанти його використання.

Перший варіант використання – заповнення деякої форми з подальшим унесенням інформації до бази даних:

уведені дані перевіряються в клієнтській частині застосунку щодо їх наявності (рівні 2 – 4), відповідності формату та належності до діапазону відповідно до варіанта завдання (рівні 3, 4);

якщо дані коректні, вони передаються на сервер та записуються в базу даних; у разі некоректності введених даних, передача їх на сервер не відбувається, а видається попереджувальне повідомлення з описом проблеми та шляхів її усунення (рівні 2 – 4);

користувач може натиснути кнопку очищення введених даних форми (у цьому разі очищаються всі поля форми і передача даних на сервер не відбувається).

Другий варіант використання – формування списку "об'єктів" предметної області та відправлення його на клієнтський комп'ютер. У результаті цього всі дані, збережені раніше, завантажуються з бази даних та відображуються в табличній формі на відповідній вебсторінці.

Третій варіант використання (рівні 3, 4) – видалення будь-якого з "об'єктів" предметної області з бази даних за запитом користувача.

Четвертий варіант використання (рівні 3, 4) – оновлення будь-якого з "об'єктів" предметної області у базі даних за запитом користувача.

П'ятий варіант використання (рівні 4) – перегляд детальної інформації щодо будь-якого з "об'єктів" предметної області за запитом користувача.

Контрольні запитання

1. Що таке модель у Django? Яку роль вона відіграє в роботі з базами даних?
2. Що таке міграція в Django? Які основні файли міграції генеруються?
3. Яку команда використовують для створення міграцій у Django? Що відбувається під час її виконання?
4. Як задати зв'язок між моделями типу "один до багатьох" у Django?
5. Чим відрізняються методи get(), filter() і all() у Django ORM?
6. Як можна створити новий запис у таблиці бази даних за допомогою Django ORM?

7. Що таке аргумент on_delete поля типу ForeignKey? Якими можуть бути його можливі значення?

8. Які типи полів підтримуються в моделях Django?

9. Як оновити наявний запис у базі даних за допомогою Django ORM?

Тема 8. Види, що базуються на класах у вебзастосунках Django

Лабораторна робота 5. Використання видів на базі класів у вебзастосунках Django

Мета лабораторної роботи:

1. Набуття практичних навичок використання Class-based Views.

2. Удосконалення навичок роботи з рекомендованими інтегрованими середовищами програмування.

Перед виконанням лабораторної роботи здобувач вищої освіти має вивчити основні поняття щодо Class-based Views.

Після виконання лабораторної роботи здобувач вищої освіти має вміти самостійно розробляти прості вебзастосунки з використанням Class-based Views.

Теоретична частина

Class-based Views (CBV) у Django дозволяють використовувати об'єктно орієнтований підхід, що сприяє повторному використанню коду та зменшує його дублювання. Основна ідея полягає в тому, щоб створювати класи, які реалізують певну поведінку для оброблення запитів.

Переваги CBV:

1. Модульність.

Класова структура дозволяє легко розділяти логіку, створюючи батьківські класи для загальних функцій.

2. Повторне використання коду.

Django надає багато бібліотечних класів (наприклад, TemplateView, ListView, DetailView), які спрощують написання коду.

3. Розширюваність.

CBV дозволяють розширювати поведінку через перевизначення методів або наслідування.

Кожен CBV є підкласом класу View з модуля django.views. Основні HTTP-методи (GET, POST, PUT тощо) реалізуються у формі методів класу.

Приклад простого CBV MyView для оброблення HTTP-запитів типів GET і POST та його підключення до маршруту my-view/ в urls.py:

```
from django.http import HttpResponse
from django.views import View

class MyView(View):
    def get(self, request):
        return HttpResponse('Hello, GET request!')

    def post(self, request):
        return HttpResponse('Hello, POST request!')

from django.urls import path

urlpatterns = [
    path('my-view/', MyView.as_view(), name='my_view'),
]
```

CBV, аналогічні наведеному прикладу, можна розглядати як батьківські види, які можна використовувати самостійно або успадковувати від них. Вони дозволяють реалізувати будь-яку логіку оброблення інформації.

Крім того, у Django є досить багато так званих узагальнених CBV. Їх побудовано на основі цих базових видів і призначено для застосування в поширених варіантах використання, таких як відображення деталей об'єкта. Вони беруть певні загальні ідіоми та шаблони, що трапляються під час розроблення видів, і абстрагують їх для можливості швидкого розроблення коду.

Розглянемо деякі узагальнені CBV, що часто використовують.

Найпростішим із них є TemplateView. Його використовують для відображення статичних вебсторінок або вебсторінок із мінімальною обчислювальною логікою, наприклад:

```
from django.views.generic import TemplateView

class AboutPageView(TemplateView):
    template_name = 'about.html'
```

Підключення AboutPageView у *файлі urls.py*:

```
urlpatterns = [
    path('about/', AboutPageView.as_view(), name='about'),
]
```

Узагальнені CBV дуже корисні для репрезентації вмісту бази даних. Оскільки це дуже поширене завдання, Django постачають із кількома вбудованими узагальненими CBV, такими як ListView та DetailView для перегляду даних і CreateView, UpdateView, DeleteView для їх редагування.

Є така модель даних, що зберігаються в певній базі:

```
class Article(models.Model):
    pub_date = models.DateField()
    headline = models.CharField(max_length=200)
    content = models.TextField()
    reporter = models.CharField(max_length=50)
```

Клас ListView використовують для відображення списків об'єктів із бази даних, наприклад:

```
from django.views.generic import ListView
from .models import Article

class ArticleListView(ListView):
    model = Article
    template_name = 'article_list.html'
```

Клас DetailView забезпечує відображення даних окремого об'єкта:

```
from django.views.generic import DetailView
from .models import Article

class ArticleDetailView(DetailView):
    model = Article
    template_name = 'article_detail.html'
```

CreateView – це вид, який відображає форму для створення об'єкта, повторне відображення форми з помилками перевірки (якщо такі є) і збереження об'єкта:

```
class ArticleCreateView(CreateView):
    model = Article
```

UpdateView – вид, який відображає форму для редагування наявного об'єкта, повторне відображення форми з помилками перевірки (якщо такі є) і збереження змін до об'єкта. Тут використовують форму, автоматично створену з класу моделі об'єкта (якщо клас форми не вказано вручну):

```
class ArticleUpdateView(CreateView):
   model = Article
```

DeleteView – це вид для відображення сторінки підтвердження та видалення наявного об'єкта. Цей об'єкт буде видалено, лише якщо методом запиту є POST. Якщо цей вид отримано через GET, він відобразить сторінку підтвердження, яка має містити форму, що надсилає POST на ту саму URL-адресу:

```
class ArticleDeleteView(CreateView):
   model = Article
```

CBV легко налаштовують за допомогою перевизначення методів. Наприклад, для фільтрації об'єктів у ListView можна перевизначити метод get_queryset:

```
class PublishedArticleListView(ListView):
   model = Article
   template_name = 'article_list.html'

   def get_queryset(self):
      return Article.objects.filter(status='published')
```

Практична частина

Завдання. Продовжіть *розроблення повноцінного* вебзастосунку для виконання CRUD-операцій з даними предметної області, що *зберігаються в базі даних (див. завдання 2 попередньої лабораторної роботи).*

Зокрема, додайте в цей вебзастосунок один або декілька власних Class-based Views (видів на базі класів).

Ці види повинні бути безпосередніми нащадками класу django.views.generic.base.View та мати функціональність, аналогічну наявним Function-

based Views (видам на базі функцій), розробленим для цього застосунку раніше.

Продемонструйте використання в доопрацьованому застосунку видів на базі класів порівняно з видами на базі функцій.

Контрольні запитання

1. Що таке CBV (Class-Based Views) у Django? Яка їх головна перевага порівняно з FBV (Function-Based Views)?
2. Який базовий клас у загальному випадку потрібно успадкувати для створення CBV?
3. Як підключити CBV до маршруту в urls.py?
4. Які переваги надає використання узагальнених CBV під час розроблення масштабованих вебзастосунків?
5. Які стандартні узагальнені CBV надає Django?
6. Для чого використовують клас TemplateView?
7. Як працює клас ListView? Яке його основне призначення?
8. Що таке DetailView? Як він дозволяє працювати з окремими об'єктами?
9. Як можна змінити поведінку CBV, наприклад, фільтрувати об'єкти в ListView?
10. Як застосувати декоратор, такий як login_required, до CBV?

Тема 9. Використання сайту адміністратора

Лабораторна робота 6. Використання сайту адміністратора Django

Мета лабораторної роботи:

1. Набуття практичних навичок використання сайту адміністратора.
2. Удосконалення навичок роботи з рекомендованими інтегрованими середовищами програмування.

Перед виконанням лабораторної роботи здобувач вищої освіти має вивчити основні поняття щодо сайту адміністратора Django.

Після виконання лабораторної роботи здобувач вищої освіти має вміти використовувати та налаштовувати сайт адміністратора Django.

Теоретична частина

Сайт адміністратора Django (Django Admin) – це потужний інструмент для управління даними вебзастосунку через вебінтерфейс. Він надає можливість швидко виконувати CRUD-операції (створення, читання, оновлення та видалення даних), налаштовувати інтерфейс для зручності адміністраторів і виконувати спеціалізовані завдання без додаткового кодування.

Він доступний за замовчуванням у Django-проєктах. Щоб переконатися, що це дійсно так, доцільно подивитися на вміст секції INSTALLED_APPS у файлі settings.py. Тут мають бути такі застосунки:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
]
```

Далі потрібно створити суперкористувача, що має всі дозволи для його адміністрування:

```
python manage.py createsuperuser
```

Щоб увійти до сайту адміністратора на локальному комп'ютері, треба запустити сервер і перейти за адресою http://127.0.0.1:8000/admin.

Для відображення певної моделі вебзастосунку в інтерфейсі адміністратора її слід заздалегідь зареєструвати у файлі admin.py.

Приклад реєстрації моделі Article:

```
from django.contrib import admin
from .models import Article

@admin.register(Article)
class Article Admin(admin.ModelAdmin):
    pass
```

Клас ArticleAdmin (ім'я може бути іншим) репрезентує модель Article у користувацькому інтерфейсі сайту адміністратора.

У наведеному прикладі цей нащадок класу ModelAdmin поки що не визначає жодних елементів. У результаті буде надано стандартний інтерфейс адміністратора.

Якщо це відповідає вимогам, то взагалі не потрібно визначати похідний від ModelAdmin клас – можна просто зареєструвати клас моделі. Тоді попередній приклад можна спростити так:

```
from django.contrib import admin
from .models import Article

admin.site.register(Article)
```

Для налаштування адміністративного інтерфейсу в класі ModelAdmin є багато атрибутів та методів.

Базові налаштування

1. Фільтрація та пошук.

Можливість додавати фільтри та пошукові поля дозволяє швидко знаходити потрібні записи. За фільтрацію та пошук відповідають атрибути list_filter та search_fields відповідно.

Наприклад, фільтрація за полями pub_date та reporter моделі Article, а також пошук за полем headline можуть бути такими:

```
list_filter = ['pub_date', 'reporter']
search_fields = ['headline']
```

2. Налаштування списку елементів.

Атрибут list_display визначає, які поля відображатимуться в списку записів. Це дозволяє легко переглядати важливу інформацію без необхідності відкривати кожен запис окремо.

Приклад:

```
list_display = ('headline', 'content')
```

3. Інлайн-редагування.

Для пов'язаних моделей можна використовувати інлайн-редагування. Це дозволяє редагувати певну модель безпосередньо на сторінці її батьківської моделі.

Припустімо, що є такі моделі, між якими є зв'язок "багато до одного":

```
from django.db import models

class Author(models.Model):
   name = models.CharField(max_length=100)

class Book(models.Model):
   author = models.ForeignKey(Author, on_delete=models.CASCADE)
   title = models.CharField(max_length=100)
```

Тоді для можливості редагування книг, написаних автором, на сторінці автора треба використати атрибут inlines:

```
from django.contrib import admin

class BookInline(admin.TabularInline):
   model = Book
class AuthorAdmin(admin.ModelAdmin):
   inlines = [BookInline, ]
```

Для налаштування елементів адміністративного інтерфейсу, значення яких можуть змінюватися залежно від певних умов, використовують методи класу ModelAdmin, такі як get_list_filter, get_search_fields, get_list_display, get_inlines тощо.

Адміністративні дії

Основний робочий процес використання сайту адміністратора Django полягає в тому, щоб вибрати певний об'єкт, а потім змінити його.

Здебільшого це працює добре. Однак, якщо потрібно внести однакові зміни до багатьох об'єктів одночасно, такий рутинний процес може бути досить виснажливим.

У цих випадках сайту адміністратора Django дозволяє реєструвати "адміністративні дії" – функції, які викликають для декількох об'єктів, вибраних на сторінці списку змін.

Якщо подивитися на будь-який список змін сайту адміністратора, можна побачити цю функціональність: за замовчування Django має адміністративну дію "видалити вибрані об'єкти", доступну для всіх моделей.

Django дозволяє зареєструвати власні адміністративні дії для користувацьких моделей за допомогою атрибуту actions:

```
from django.contrib import admin
from myapp.models import Article

@admin.action(description="Mark selected stories as published")
def make_published(modeladmin, request, queryset):
   queryset.update(status="p")

@admin.register(Article)
class ArticleAdmin(admin.ModelAdmin):
     actions = [make_published]
```

Тепер адміністратор може вибрати кілька записів та застосувати адміністративну дію "Mark selected stories as published".

Також сайт адміністратора Django може бути значно розширений для підтримання специфічних потреб, а саме: створення користувацьких вебсторінок в адміністративному інтерфейсі, підключення віджетів JavaScript (наприклад, для вибору дати), використання сторонніх пакетів для надання додаткових можливостей тощо.

Практична частина

Завдання. Створіть суперкористувача для доступу на сайт адміністратора вебзастосунку з попередньої лабораторної роботи та зареєструйте моделі цього застосунку для використання їх сумісно із сайтом адміністратора.

Виконайте програмне налаштування вебінтерфейсу сайту адміністратора для забезпечення:

відображення власного заголовка вебсторінок замість "Django administration";

можливості фільтрації даних, що відображуються в списку, за певними критеріями;

можливості пошуку даних, що відображуються в списку;

можливості редагування пов'язаних записів на сторінці батьківського запису (наприклад, редагування записів про книги під час створення запису про їхніх авторів) – рівні 3 та 4;

можливості виконання операцій над декількома записами зі списку одночасно із застосуванням користувацьких адміністративних дій – рівень 4.

Продемонструйте викладачу використання сайту адміністратора з виконаними налаштуваннями.

Контрольні запитання

1. Що таке сайт адміністратора Django? Для чого його використовують?
2. Які застосунки мають бути в секції INSTALLED_APPS файлу settings.py, щоб активувати сайт адміністратора Django?
3. Як створити суперкористувача в Django?
4. Як зареєструвати модель на сайті адміністратора?
5. Яке призначення атрибуту list_display у класі ModelAdmin?
6. Які можливості надає атрибут search_fields класу ModelAdmin? Як його налаштовують?
7. Як додати інлайн-редагування для пов'язаних моделей у Django Admin?
8. Як створити власну адміністративну дію і додати її до сайту адміністратора Django?

Теми 10, 11. Сесії. Авторизація й автентифікація

Лабораторна робота 7. Використання сесій, автентифікації й авторизації в Django

Мета лабораторної роботи:

1. Набуття практичних навичок використання автентифікації й авторизації в Django.
2. Удосконалення навичок роботи з рекомендованими інтегрованими середовищами програмування.

Перед виконанням лабораторної роботи здобувач вищої освіти має вивчити головні поняття щодо автентифікації та авторизації.

Після виконання лабораторної роботи здобувач вищої освіти має вміти самостійно розробляти прості вебзастосунки з використанням автентифікації й авторизації.

Теоретична частина

Одними з ключових елементів Django є сесії, автентифікація й авторизація. Вони дозволяють додавати в застосунок функціональність для управління користувачами, забезпечення доступу до захищених ресурсів та зберігання тимчасових даних для кожного користувача.

Сесії дозволяють зберігати тимчасові дані для кожного користувача на стороні сервера, із прив'язкою до унікального ідентифікатора сесії.

Спочатку треба переконатися, що у файлі settings.py увімкнуто механізм сесій:

```
INSTALLED_APPS = [
...
'django.contrib.sessions',
]

MIDDLEWARE = [
...
'django.middleware.session.SessionMiddleware',
]
```

За замовчуванням для збереження сесій Django використовує базу даних. За потреби можна змінити бекенд у налаштуваннях, наприклад:

```
# За замовчуванням
SESSION_ENGINE = 'django.contrib.sessions.backends.db'

# З використанням кешу
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
```

Якщо SessionMiddleware активовано, тоді кожен об'єкт HttpRequest – перший аргумент будь-якої функції виду Django – матиме атрибут session, який є об'єктом, схожим на словник. Він доступний для зчитування або запису в будь-який момент. Також цей атрибут можна редагувати кілька разів.

Приклад використання сесій:

```
from django.http import HttpResponse

def set_session(request):
```

```
    request.session['username'] = 'JohnDoe'
    return HttpResponse("Сесію встановлено!")

def get_session(request):
    username = request.session.get('username', 'Гість')
    return HttpResponse(f"Привіт, {username}!")

def delete_session(request):
    if 'username' in request.session:
    del request.session['username']
    return HttpResponse("Сесію видалено!")
```

У наведеному прикладі види set_session, get_session та delete_ session відповідно встановлюють значення атрибута session, зчитують і видаляють його.

Автентифікація – це процес перевірки "справжності" особи користувача. Для цього за замовчуванням використовують його логін та пароль.

Авторизація – це процес перевірки прав доступу користувача до певних ресурсів.

Фреймворк Django забезпечує як автентифікацію, так і авторизацію разом, що зазвичай називають системою автентифікації, оскільки ці функції дещо пов'язані.

Ядром системи автентифікації є об'єкти класу User. Зазвичай вони представляють людей, які взаємодіють із вебзастосунком. Їх використовують для обмеження доступу, реєстрації профілів користувачів, пов'язування вмісту з творцями тощо.

Первинні атрибути користувача за замовчуванням – це username, password, email, first_name та last_name.

Django надає готову систему автентифікації, реалізовану як системний застосунок. Тому він має бути в секції INSTALLED_APPS файлу settings.py:

```
INSTALLED_APPS = [
...
'django.contrib.auth',
]
```

Приклади реєстрації нового користувача, входу його в систему та виходу з неї наведено далі.

Реєстрація нового користувача:

```
from django.contrib.auth.models import User
from django.http import HttpResponse

def register_user(request):
    user = User.objects.create_user(username='johndoe',
                                    password='securepassword')
    user.email = 'johndoe@example.com'user.save()
    return HttpResponse("Користувача зареєстровано!")
```

Приклад входу користувача в систему:

```
from django.contrib.auth import authenticate, login, logout
from django.http import HttpResponse

def user_login(request):
    user = authenticate(request, username='johndoe', password='securepassword')
    if user is not None:
        login(request, user)
        return HttpResponse("Вхід успішний!")
    return HttpResponse("Неправильний логін або пароль!")
```

Приклад виходу користувача із системи:

```
def user_logout(request):
    logout(request)
    return HttpResponse("Вихід виконано!")
```

Django містить вбудовану систему дозволів. Вона забезпечує призначення дозволів певним користувачам і групам користувачів.

Кожна модель у Django має чотири базові дозволи: add_<modelname>, change_<modelname>, delete_<modelname>, view_<modelname>.

Приклад перевірки дозволів:

```
from django.contrib.auth.decorators import permission_required
from django.http import HttpResponse

@permission_required('app_name.add_modelname')
def add_view(request):
return HttpResponse("У вас є дозвіл додавати об'єкти!")
```

Групи – це узагальнений спосіб класифікації користувачів, щоб до них можна було застосувати дозволи або інші мітки. Користувач може належати до будь-якої кількості груп.

Користувач у групі автоматично отримує дозволи, надані цій групі. Наприклад, якщо певна група має дозвіл can_edit_home_page, будь-який користувач у цій групі матиме цей дозвіл.

Також групи є зручним способом класифікувати користувачів, щоб надати їм певну мітку або розширити функціональність. Наприклад, можна створити групу "Адміністратори" та розробити код, який міг би надати їм доступ до частини сайту, призначеної лише для адміністраторів, або надсилати їм повідомлення електронної пошти.

Приклад створення групи:

```
from django.contrib.auth.models import Group

# Створення групи
managers = Group.objects.create(name='Managers')

# Додавання користувача до групи
user = User.objects.get(username='johndoe')
managers.user_set.add(user)
```

У видах можна перевіряти, чи належить користувач до певної групи:

```
from django.contrib.auth.decorators import login_required
from django.http import HttpResponse

@login_required
def restricted_view(request):
if request.user.groups.filter(name='Managers').exists():
return HttpResponse("Ви менеджер!")
return HttpResponse("Доступ заборонено!")
```

У Django також можна створювати дозволи програмним чином. *Наприклад, дозволу can_publish для моделі BlogPost у застосунку myapp:*

```
from myapp.models import BlogPost
from django.contrib.auth.models import Permission
from django.contrib.contenttypes.models import ContentType
content_type = ContentType.objects.get_for_model(BlogPost)
```

```
permission = Permission.objects.create(
    codename="can_publish",
    name="Can Publish Posts",
    content_type=content_type,
)
```

Практична частина

Завдання. Додайте до вебзастосунку Django з попередньої лабораторної роботи нову функціональність, пов'язану з використанням сесій, автентифікацією й авторизацією.

Підсистема автентифікації й авторизації застосунку має:

забезпечувати можливість самостійної реєстрації нових користувачів;

дозволяти доступ до застосунку тільки зареєстрованим користувачам;

використовувати ролі "адміністратор", "менеджер" та "звичайний користувач" (рівні 3, 4);

дозволяти доступ до варіантів використання застосунку, пов'язаних із додаванням, редагуванням та видаленням даних, тільки зареєстрованим користувачам із роллю "менеджер" (рівні 3, 4);

забезпечувати можливість редагування даних звичайних користувачів та призначення їм певних ролей тільки користувачем із роллю "адміністратор" (рівні 3, 4);

із використанням сесій відстежувати час входу користувачів із роллю "звичайний користувач" і "менеджер" у систему та тривалість їхнього перебування в системі, а також зберігати ці дані за кожним користувачем у файлі або базі даних (рівень 4);

забезпечувати можливість самостійного скидання пароля для вже зареєстрованих користувачів шляхом відправлення відповідного посилання на email користувача (рівень 4).

Контрольні запитання

1. Що таке сесії в Django? Для чого їх використовують?
2. Які бекенди для збереження сесій підтримує Django?
3. Наведіть приклад функції, яка записує дані в сесію та читає їх.

4. Які компоненти потрібно додати до секцій INSTALLED_APPS та MIDDLEWARE у settings.py, щоб увімкнути сесії в Django?

5. Що таке автентифікація в Django? Які готові методи вона надає для роботи з користувачами?

6. Чим відрізняються функції authenticate() та login() у Django?

7. Як створити нового користувача в Django із застосуванням моделі User?

8. Що таке дозволи в Django? Які базові дозволи створюються автоматично для кожної моделі?

9. Як використовують декоратор @permission_required для перевірки дозволів?

10. Як створити групу користувачів у Django, призначити їй дозволи та додати користувача до групи?

Тема 12. Інтернаціоналізація та локалізація

Лабораторна робота 8. Застосування інтернаціоналізації та локалізації в Django

Мета лабораторної роботи:

1. Набуття практичних навичок використання інтернаціоналізації та локалізації в Django.

2. Удосконалення навичок роботи з рекомендованими інтегрованими середовищами програмування.

Перед виконанням лабораторної роботи здобувач вищої освіти має вивчити основні поняття щодо інтернаціоналізації та локалізації.

Після виконання лабораторної роботи здобувач вищої освіти має вміти самостійно розробляти прості вебзастосунки з використанням інтернаціоналізації та локалізації.

Теоретична частина

Інтернаціоналізація (i18n) та локалізація (i10n) дозволяють вебзастосункам підтримувати кілька мов і культур. Django має потужний вбудований інструментарій для їх реалізації, що значно спрощує адаптацію додатків для користувачів із різних країн.

Інтернаціоналізація – це процес підготовки застосунку до підтримання кількох мов та культур, наприклад, переклад тексту, форматування дати, часу, чисел тощо.

Локалізація полягає в адаптації застосунку до конкретної мови та культури шляхом використання локальних ресурсів.

Django реалізує інтернаціоналізацію та локалізацію через модулі перекладу для текстових рядків, автоматичне форматування дат, чисел тощо відповідно до локалі, а також – через мовні файли (спеціальні файли .po та .mo, які містять переклади).

Щоб налаштувати інтернаціоналізацію та локалізацію, треба виконати такі кроки:

1. Перевірити, що такі параметри у файлі settings.py мають значення True:

```
USE_I18N = True
USE_L10N = True
USE_TZ = True
```

2. Визначити підтримувані мови, наприклад, англійську, українську та французьку:

```
LANGUAGES = [
   ('en', 'English'),
   ('uk', 'Ukrainian'),
   ('fr', 'French'),
]
```

3. Позначити текст, який потребує перекладу, за допомогою функції перекладу gettext або gettext_lazy (загальноприйнято імпортувати це як коротший псевдонім "_"):

```
from django.utils.translation import gettext as _

# Позначення тексту для перекладу
welcome_message = _('Welcome to our site!')
```

4. Розробити контент мовних файлів.

Спочатку потрібно згенерувати мовні файли шляхом виконання команди makemessages для кожної мови, наприклад, для підтримання української мови:

```
python manage.py makemessages -l uk
```

Це створить файл locale/uk/LC_MESSAGES/django.po.

Далі потрібно виконати редагування мовного файлу, додавши всі переклади із python-файлів:

```
msgid "Welcome to our site!"
msgstr "Ласкаво просимо на наш сайт!"
```

Для перекладу тексту в шаблонах використовують тег {% translate %}, а для перекладу блоків тексту – тег {% blocktranslate %}:

```
<h1>{% translate "Welcome to our site!" %}</h1>

{% blocktranslate %}
   This site is available in {{ language_count }} languages.
{% endblocktranslate %}
```

Потім виконують компіляцію мовного файлу (команда compilemessages):

```
python manage.py compilemessages
```

Після компіляції в каталозі locale/uk/LC_MESSAGES буде згенеровано файл django.mo.

Практична частина

Завдання. Додайте до вебзастосунку Django з попередньої лабораторної роботи нову функціональність, пов'язану з використанням інтернаціоналізації та локалізації.

Модифікований застосунок має забезпечувати можливість вибору мови інтерфейсу користувача з трьох мов (українська – за замовчуванням, англійська та мова за бажанням розробника). Поточну мову може змінювати користувач під час виконання застосунку.

Інтернаціоналізацію застосунку має бути виконано у вихідному коді моделей (Models) та видів (Views), а також у шаблонах (Templates).

Контрольні запитання

1. Що таке інтернаціоналізація (i18n) та локалізація (l10n)? Чим вони відрізняються?

2. Які основні налаштування у файлі settings.py відповідають за інтернаціоналізацію та локалізацію?

3. Що таке мовні файли .po і .mo? Яка їх роль у локалізації?

4. Як за допомогою Django створити мовні файли для конкретної мови? Яку команду використовують для цього?

5. Як відредагувати текст у мовних файлах і зберегти зміни для подальшого використання в застосунку?

6. Як використовують функцію gettext у Python-коді для забезпечення перекладу тексту?

7. Як можна змінити мову на сайті за допомогою LocaleMiddleware?

8. Що таке тег {% trans %} у шаблонах Django? Як його використовують?

9. Як налаштувати список підтримуваних мов у Django? Де це визначено?

10. Які переваги автоматичного форматування дат і чисел у Django? Як локалізація на це впливає?

Рекомендована література

Основна

1. Feldroy D. Two Scoops of Django 3.X: Best Practices for Django / D. Feldroy. – Los Angeles : Two Scoops Press, 2020. – 477 p.

2. Mele A. Django by Example / A. Mele. – Birmingham : Packt Publishing Limited, 2020. – 568 p.

3. Vincent W. Django for Beginners: Build websites with Python and Django / W. Vincent. – Manchester : WelcomeToCode, 2023. – 349 p.

Інформаційні ресурси

4. Парфьонов Ю. Е. Вибір служби вебхостингу для застосунків на базі фреймворку Django / Ю. Е. Парфьонов, О. Г. Колгатін // Вісник Харківського національного автомобільно-дорожнього університету : зб. наук. пр. – Харків : ХНАДУ, 2022. – Вип. 96. – С. 66–70. – Режим доступу : http://repository.hneu.edu.ua/handle/123456789/27384.

5. Парфьонов Ю. Е. Питання міграції схеми бази даних під час супроводу вебзастосунків на базі фреймворку Django [Електронний ресурс] / Ю. Е. Парфьонов // Системи обробки інформації. – Харків : ХУПС ім. Івана Кожедуба, 2024. – Вип. 2 (177). – С. 63–67. – Режим доступу : https://journal-hnups.com.ua/index.php/soi/article/view/1658.

6. Програмування Інтернет [Електронний ресурс] // Персональні навчальні системи ХНЕУ ім. С. Кузнеця. – Режим доступу : https://pns.hneu.edu.ua/course/view.php?id=4650.

7. Django Admin Cookbook [Electronic resource]. – Access mode : https://books.agiliq.com/projects/django-admin-cookbook/en/latest/.

8. Django documentation [Electronic resource]. – Access mode : https://docs.djangoproject.com/en/5.1/.

9. Django forum [Electronic resource]. – Access mode : https://forum.djangoproject.com/.

10. DjangoSites: Shiny websites, powered by Django [Electronic resource]. – Access mode : https://djangosites.org/.

11. Django Tutorial [Electronic resource]. – Access mode : https://www.w3schools.com/django/.

12. Django Tutorial: The Local Library website [Electronic resource]. – Access mode : https://developer.mozilla.org/en/docs/Learn/Server-side/Django/Tutorial_local_library_website.

13. Interactive Python tutorial [Electronic resource]. – Access mode : https://www.learnpython.org/.

14. The Python Standard Library [Electronic resource]. – Access mode : https://docs.python.org/3/library/index.html.

Додатки

Додаток А

Шаблон звіту про лабораторну роботу

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ СЕМЕНА КУЗНЕЦЯ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ

"Програмування Інтернет"

Звіт

про лабораторну роботу № 1
на тему "Основи Django"

Виконав:
студент 3-го курсу
групи 6.04.122.010.23.1
ННІ ІТ Петренко І. С.

Перевірив:
доцент кафедри ІС
канд. техн. наук, ст. наук. співробітник
Парфьонов Ю. Е.

Харків – 202_

Завдання 1

Умова завдання

Розробити консольний Python-застосунок для обчислення суми двох дійсних чисел, що вводить користувач.

Програма має складатися з двох класів: MainClass і Calculator.

Клас MainClass містить точку входу в програму. Метод main має зчитувати з консолі два дійсних числа, обчислювати їх суму за допомогою класу Calculator та виводити її на консоль з двома десятковими знаками.

Клас Calculator визначає арифметичні операції. Його статичний метод calculateSum має виконувати додавання двох дійсних чисел і повертати їх суму.

Опис архітектури програми

У цьому підрозділі звіту має бути:

для об'єктно орієнтованих програмних систем – UML-діаграма класів (Class Diagram), що реалізують основну бізнес-логіку програмної системи, моделі даних тощо, та її короткий опис;

для інших програмних систем – UML-діаграма діяльності (Activity Diagram), яка відбиває основну бізнес-логіку програмної системи, та її короткий опис.

Приклади результатів виконання програми

Випадок А (рис. 1). Сума чисел 2,3 та 1,7823 дорівнює 4,0823. Згодом вона виводиться з двома знаками після коми.

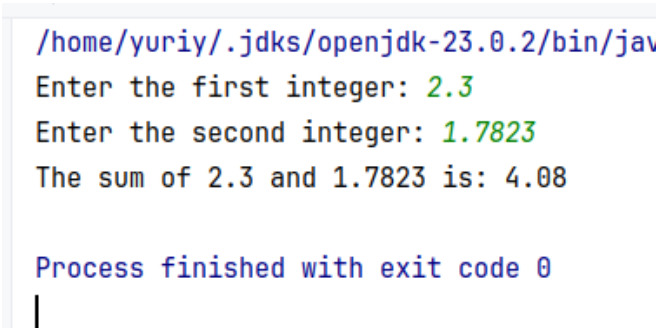

Рис. 1. **Скриншот випадку А**

Випадок Б (рис. 2). Сума чисел 10,345 та −0,1234 дорівнює 10,221. Згодом вона виводиться з двома знаками після коми.

```
/home/yuriy/.jdks/openjdk-23.0.2/bin/java
Enter the first integer: 10.345
Enter the second integer: -0.1234
The sum of 10.345 and -0.1234 is: 10.22

Process finished with exit code 0
```

Рис. 2. **Скриншот випадку Б**

Висновки

Під час виконання лабораторної роботи № 1 набуто практичних навичок створення програм із використанням головних елементів фреймворку Django, а також інтегрованого середовища розроблення PyCharm.

Прототип вихідного коду було згенеровано за допомогою ChatGPT, а потім доопрацьовано мною особисто.

Розроблений консольний застосунок для обчислення суми двох дійсних чисел, наданих користувачем, загалом відповідає поставленим вимогам.

Усі питання, пов'язані з розробленням програми, були успішно вирішені.

Зміст

НАВЧАЛЬНЕ ВИДАННЯ

ПРОГРАМУВАННЯ ІНТЕРНЕТ

Методичні рекомендації до виконання лабораторних робіт

для здобувачів вищої освіти спеціальності 121 "Інженерія програмного забезпечення" освітньої програми "Інженерія програмного забезпечення" першого (бакалаврського) рівня

Самостійне електронне текстове мережеве видання

Укладач **Парфьонов** Юрій Едуардович

Відповідальний за видання *Д. О. Бондаренко*

Редактор *Н. Г. Войчук*

Коректор *Н. В. Завгородня*

План 2025 р. Поз. № 113 ЕВ. Обсяг 55 с.

Видавець і виготовлювач – ХНЕУ ім. С. Кузнеця, 61165, м. Харків, просп. Науки, 9-А

Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру ***ДК № 4853 від 20.02.2015 р.***