Державний торговельно-економічний університет
Кафедра комп'ютерних наук та інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Система тестів для перевірки правильності виконання лабораторних робіт на Python »

Студента 4 курсу, 10 гр. групи, спеціальності 122 «Комп'ютерні науки» _____ підпис студента — Вишенський Євгеній Володимирович

Науковий керівник Кандидат педагогічних наук, доцент _____ підпис керівника — Базурін Віталій Миколайович

Гарант освітньої програми кандидат технічних наук, доцент _____ підпис керівника — Демідов Павло Георгійович

Київ 2023

Державний торговельно-економічний університет

Факультет інформаційних технологій

Кафедра комп'ютерних наук та інформаційних систем

Спеціальність 122 «Комп'ютерні науки»

Затверджую

Зав. кафедри _____Пурський О.І.

«12» грудня 2022 р.

Завдання

на випускну кваліфікаційну роботу студенту

Вишенський Євгеній Володимирович

1. Тема випускної кваліфікаційної роботи

«Система тестів для перевірки правильності виконання лабораторних робіт на мові Python»

Затверджена наказом ректора від *«09» грудня 2022 р. № 3332*

2. Строк здачі студентом закінченої роботи *30 травня 2023 року*

3. Цільова установка та вихідні дані до роботи

Мета роботи: *розробка системи тестів для перевірки правильності виконання лабораторних робіт на мові Python*

Об'єкт дослідження: *процес тестування програмного забезпечення*.

Предмет дослідження: *система тестів*.

4. Перелік графічного матеріалу _____

_____

5. Консультанти по роботі із зазначенням розділів, за якими здійснюється консультування:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання отримав
1	Базурін В.М.	21.12.2022 р	21.12.2022 р
2	Базурін В.М.	21.12.2022 р	21.12.2022 р
3	Базурін В.М.	21.12.2022 р	21.12.2022 р

6. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. Аналітичне дослідження процесу тестування програмного забезпечення

1.1. Аналіз стану проблеми

1.2. Особливості тестування програм для лабораторних робіт

1.3. Характеристика існуючих технологій тестування

РОЗДІЛ 2. Розробка моделі тестів

2.1. Загальна концепція

2.2. Моделі обчислень

2.3. Алгоритми обчислень

РОЗДІЛ 3. Система тестів для програм на мові Python

3.1. Розробка інформаційно-логічної моделі

3.2. Програмна реалізація тестів

3.4. Технологія використання системи тестів

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

7. Календарний план виконання роботи

№	Назва етапів випускної кваліфікаційної	Термін виконання

пор.	роботи	етапів роботи	
		За планом	Фактично
1	Вибір теми випускної кваліфікаційної роботи	*04.10.2022*	*04.10.2022*
2	Розробка та затвердження завдання на випускну кваліфікаційну роботу	*15.12.2022*	*15.12.2022*
3	Вступ	*03.02.2023*	*03.02.2023*
4	Розділ 1. Аналітичне дослідження процесу тестування програмного забезпечення	*28.02.2023*	*28.02.2023*
5	Розділ 2. Розробка моделі тестів	*06.04.2023*	*06.04.2023*
6	Розділ 3. Система тестів для програм на мові Python	*12.05.2023*	*12.05.2023*
	Висновки	*15.05.2023*	*15.05.2023*
	Здача кваліфікаційної роботи на кафедру науковому керівнику	*30.05.2023*	*30.05.2023*
	Попередній захист випускної кваліфікаційної роботи	*31.05.2023* - *01.06.2023*	*31.05.2023* - *01.06.2023*
	Виправлення зауважень, зовнішнє рецензування випускної кваліфікаційної роботи	*02.06.2023*	*02.06.2023*
	Представлення готової зшитої випускної кваліфікаційної роботи на кафедру	*05.06.2023*	*05.06.2023*
13	Публічний захист випускної кваліфікаційної роботи	За розкладом ЕК	

8. Дата видачі завдання *«» грудня 2022 р*.

9. Керівник випускної кваліфікаційної роботи

*Базурін В.М.*_____

(прізвище, ініціали, підпис)

10. Гарант освітньої програми

Демідов П.Г. _____

(прізвище, ініціали, підпис)

11. Завдання прийняв до виконання студент-дипломник

Вишенський Є.В._____

(прізвище, ініціали, підпис)

12. Відгук керівника випускної кваліфікаційної роботи

_____

_____

_____

_____

_____

_____

_____

_____

Керівник випускної кваліфікаційної роботи

_____ «___»_____2023 р.

(підпис, дата)

13. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента _____

(прізвище, ініціали)

може бути допущена до захисту в екзаменаційній комісії.

Гарант освітньої програми _____*Демідов П.Г.*

(підпис, прізвище, ініціали)

Завідувач кафедри _____*Пурський О.І.*

(підпис, прізвище, ініціали)

«_____»_____2023 р.

ЗМІСТ

ВСТУП

Актуальність дослідження. Сучасний світ насичений програмними продуктами, які знаходять широке застосування в різних галузях життя. Перевірка правильності функціонування цих програм є надзвичайно важливою задачею. Мова програмування Python, завдяки своїй простоті, широкій функціональності та популярності, використовується великою кількістю розробників програмного забезпечення. Однак, перевірка коректності програм на мові Python може становити складність через різноманітність сценаріїв використання та можливі помилки, що можуть з'явитися. Тому створення ефективної системи тестів для перевірки програм на мові Python є актуальною проблемою, яка вимагає ретельного дослідження.

Інформаційна база. Для дослідження були використані різноманітні джерела, включаючи наукові статті, конференційні доповіді, книги та онлайн-ресурси. Документація мови Python містить розділи, присвячені тестуванню, включаючи огляд різних модулів тестування, приклади використання та поради щодо тестування програм на Python. Багато програмістів та експертів з Python публікують статті та блоги, де діляться своїм досвідом щодо тестування програм на Python. Модуль unittest входить до стандартної бібліотеки Python і надає базову підтримку для написання тестів. Офіційна документація unittest містить інформацію про його функціональність, рекомендації щодо написання тестів і приклади використання.

Зокрема, були проаналізовані дослідження Базуріна В.М., Костюченко А.О., Креневич А.П., Мокіна Б.І., Яковенко А. тощо [1; 4-5; 7; 14]. В роботах було зазначено, що автоматизоване тестування є популярною технологією використання системи тестування, яка передбачає написання спеціальних скриптів або програм для виконання тестових сценаріїв і перевірки правильності поведінки програмного забезпечення. Системи

тестування, такі як Pytest, Unittest або Robot Framework, дозволяють розробникам створювати тести для програм і автоматично виконувати їх на різних етапах розробки. Вони дозволяють перевіряти окремі функції, модулі та інтеграцію між ними.

Мета дослідження. Метою даної кваліфікаційної роботи є розробка системи тестів для перевірки програм на мові Python. Основним завданням є створення набору тестів, що охоплюють різні аспекти програмної логіки та можливі варіанти використання, а також забезпечення автоматизації процесу тестування. Дослідження спрямоване на поліпшення процесу розробки програм на мові Python та забезпечення високої якості програмного забезпечення.

Завдання дослідження. Для досягнення даної мети, перед дослідженням були поставлені наступні завдання:

- здійснити аналіз стану проблеми;
- визначити особливості тестування програм для лабораторних робіт;
- охарактеризувати існуючі технології тестування;
- висвітлити моделі обчислень;
- визначити алгоритми обчислень;
- здійснити програмну реалізацію тестів;
- визначити технологію використання системи тестів.

Об’єкт дослідження. Об'єктом даного дослідження є система тестів для перевірки програм на мові Python. В рамках роботи буде розглянуто різні аспекти тестування, включаючи вхідні дані, вихідні результати, роботу з базами даних та взаємодію з іншими програмними модулями.

Предмет дослідження. Предметом дослідження є розробка та імплементація системи тестів, яка буде забезпечувати перевірку коректності програм на мові Python. Робота орієнтована на виявлення помилок, аналіз виключних ситуацій та впевнену роботу програмного забезпечення в різних сценаріях використання.

Методи дослідження. Для досягнення поставленої мети будуть використані наступні методи дослідження: аналіз наукової літератури та існуючих рішень у галузі тестування програм на мові Python, розробка набору тестів для різних компонентів програмної системи на мові Python, використання методів автоматичного тестування та створення тестових сценаріїв, аналіз результатів тестування та вдосконалення системи тестів.

Наукова новизна дослідження. Дослідження включає розробку нових програмних методів для тестування програм на мові Python та ивчення ефективності тестування в рамках конкретних проектів, виявлення найбільш продуктивних підходів до організації тестів.

Практичне значення. Результати цієї роботи матимуть практичне значення для розробників програмного забезпечення, які працюють з мовою Python. Система тестів, яка буде розроблена, допоможе автоматизувати процес перевірки програм, забезпечивши високу якість та надійність програмного забезпечення. Крім того, дане дослідження може послужити основою для подальшого вдосконалення системи тестування та розширення її функціональності.

Структура та обсяг випускної кваліфікаційної роботи. Випускна кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних джерел із 30 найменувань, додатків і містить 55 сторінок тексту, 7 рисунків і 3 таблиці.

РОЗДІЛ 1
АНАЛІТИЧНЕ ДОСЛІДЖЕННЯ ПРОЦЕСУ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Аналіз стану проблеми

Стан проблеми тестування програмного забезпечення продовжує еволюціонувати і змінюватися відповідно до розвитку технологій та індустрії програмного забезпечення. Ключові аспекти, які варто враховувати при аналізі стану проблеми тестування програмного забезпечення, наведені в таблиці 1.1.

Таблиця 1.1

Ключові аспекти стану проблеми [4]

№	Аспект	Характеристика
1	Автоматизація тестування	В останні роки автоматизація тестування стала основним напрямком розвитку. Інструменти автоматизації дозволяють зменшити залежність від ручного тестування, прискорити процес тестування та знизити його вартість
2	Agile та DevOps	Agile-методології розробки програмного забезпечення, такі як Scrum та Kanban, стали широко поширеними. Вони покладають акцент на ітераційний підхід до розробки, що вимагає гнучкості в процесі тестування. DevOps також став популярним, об'єднуючи розробку та експлуатацію ПЗ, що має вплив на тестування
3	Continuous Integration / Continuous Delivery (CI/CD)	CI/CD - це практика, яка дозволяє автоматично тестувати та постійно впроваджувати зміни в програмному забезпеченні. Це дозволяє розробникам швидко відстежувати проблеми та негайно вносити виправлення
4	Тестування безпеки	З огляду на зростаючу загрозу кібербезпеки, тестування безпеки стало надзвичайно важливим етапом розробки програмного забезпечення. Виявлення і усунення потенційних вразливостей та захист від атак стали пріоритетом

Продовження таблиці 1.1

№	Аспект	Характеристика
5	Тестування мобільних додатків	За останні кілька років зростає популярність мобільних додатків. Тестування таких додатків потребує специфічних підходів, включаючи тестування на різних платформах та пристроях
6	Хмарні та веб-орієнтовані додатки	Зростання використання хмарних послуг та веб-орієнтованих додатків призводить до необхідності тестування в різних середовищах та залежності від зовнішніх служб
7	Штучний інтелект та машинне навчання	Використання штучного інтелекту та машинного навчання в програмному забезпеченні відкриває нові можливості для автоматизації тестування, виявлення дефектів та аналізу даних тестування

В останні роки автоматизація тестування стала основним напрямком розвитку в індустрії програмного забезпечення. Інструменти автоматизації тестування включають фреймворки, бібліотеки та програми, які допомагають розробникам та тестувальникам автоматизувати виконання тестових сценаріїв, перевірку функціональності та вимог, тестування інтерфейсів користувача та інше. Переваги автоматизації тестування наступні:

1. Ефективність. Автоматизація тестування дозволяє виконувати тести швидше та більш ефективно, порівняно з ручним тестуванням. Інструменти автоматизації можуть виконувати тести швидше, ніж людина, і забезпечують однакові результати при кожному запуску тестів.

2. Покриття. Автоматизовані тести можуть охоплювати більш широкий спектр функціональності програмного забезпечення і виконувати складні операції, які можуть бути важкими або неможливими для ручного тестування. Це дозволяє забезпечити високу якість тестування та виявлення потенційних проблем.

3. Вартість. Хоча автоматизація тестування може вимагати певних витрат на початкове налаштування та підтримку, вона зазвичай приносить

вигоду у вигляді зниження загальних витрат на тестування. Зменшення залежності від ручного тестування дозволяє зменшити кількість ресурсів, необхідних для виконання тестів, і збільшити продуктивність команди тестувальників.

4. Повторюваність. Автоматизовані тести можуть бути легко повторюваними і запускатися безперервно під час розробки та випуску програмного забезпечення. Це дозволяє швидко виявляти ізольовані помилки та забезпечує постійну якість продукту.

5. Регресійне тестування. Одним із ключових використань автоматизації тестування є регресійне тестування, коли виконуються попередньо пройдені тести для переконання у відсутності нових помилок після внесення змін у програмне забезпечення. Автоматизоване регресійне тестування дозволяє заощадити час і зусилля, порівняно з ручним повторенням тестів.

Хоча автоматизація тестування має багато переваг, варто враховувати, що не всі види тестування можна повністю автоматизувати. Деякі тести, такі як тести на користувацький досвід або тести на емоційну реакцію користувачів, все ще краще виконувати вручну. Оптимальний підхід полягає у поєднанні автоматизованого тестування з ручним тестуванням для забезпечення найвищої якості продукту.

Agile і DevOps є двома ключовими концепціями, які впливають на стан проблеми тестування програмного забезпечення.

Agile-методології, такі як Scrum та Kanban, набули широкої популярності в розробці програмного забезпечення. Вони замінюють традиційний водопадний підхід до розробки на ітераційний підхід, де розробка відбувається шляхом невеликих ітерацій або спринтів. Це вимагає гнучкості в процесі тестування, оскільки тести повинні бути виконані для кожної ітерації, а не тільки в кінці проекту. Тестувальники працюють пліч-о-пліч з розробниками та зацікавленими сторонами, щоб забезпечити постійний контроль якості та швидке реагування на зміни.

DevOps - це філософія та практика, яка об'єднує розробку (Development) і експлуатацію (Operations) програмного забезпечення. Основна ідея полягає в тому, щоб забезпечити співпрацю та інтеграцію між розробниками та командами експлуатації, щоб швидко та ефективно розробляти, тестувати та впроваджувати програмне забезпечення. Тестування входить до цього процесу, де автоматизація тестування та інструменти для автоматизованого впровадження грають важливу роль у забезпеченні швидкості, надійності та стабільності процесу розробки та впровадження.

За допомогою Agile та DevOps підходів тестування отримує більші можливості для реагування на зміни, покращення комунікації та співпраці між командами, а також швидкого впровадження та доставки продукту. Основні принципи цих підходів, такі як ітераційність, автоматизація та колективна відповідальність, впливають на тестування та вимагають використання відповідних методів та інструментів, щоб забезпечити високу якість програмного забезпечення в умовах швидко змінюючогося середовища розробки.

Continuous Integration/Continuous Delivery (CI/CD) - це практика, що дозволяє автоматизувати процеси тестування та впровадження змін в програмне забезпечення. Основна ідея полягає в тому, щоб постійно та автоматично інтегрувати зміни, виконувати тести та доставляти програмне забезпечення в продуктивне середовище.

Continuous Integration (CI) включає автоматичну інтеграцію коду розробників в спільний репозиторій та виконання автоматизованих тестів для перевірки функціональності та стабільності. Кожного разу, коли розробник вносить зміни у код, вони інтегруються зі зв'язаним кодом команди, і виконуються тестування для перевірки, що зміни не порушують функціональність програмного забезпечення.

Continuous Delivery (CD) покликаний автоматизувати та спростити процес доставки програмного забезпечення у виробниче середовище. Це означає, що після успішної інтеграції та пройдених тестах програмне

забезпечення готове для впровадження в реальному середовищі. Використовуючи автоматизовані процеси, CD забезпечує швидку та безпечну доставку програмного забезпечення на продуктивні сервери або інші цільові платформи.

Основна перевага CI/CD полягає в тому, що він дозволяє розробникам швидко відстежувати проблеми та вносити виправлення без затримок. Автоматизовані тести виконуються після кожного внесення змін, що допомагає виявляти проблеми рано та швидко виправляти їх. Це сприяє покращенню якості програмного забезпечення, скороченню часу циклу розробки та зниженню ризику помилок під час впровадження змін.

Однак, важливо враховувати, що автоматизація тестування є ключовою складовою CI/CD, і потребує налагодження відповідних тестових наборів, інструментів автоматизації та налагодження процесу. Також потрібно враховувати, що не всі види тестування можуть бути повністю автоматизовані, і ручне тестування може бути потрібним для певних аспектів якості програмного забезпечення, таких як користувацький досвід або емоційна реакція користувачів.

Тестування безпеки стало невід'ємною складовою процесу розробки програмного забезпечення, оскільки кібербезпека стала однією з найбільших загроз для організацій та користувачів. Виявлення і усунення потенційних вразливостей та захист від атак стали пріоритетом для команд розробки та тестування.

Основні аспекти тестування безпеки включають [12]:

1. Тестування вразливостей. Цей тип тестування спрямований на виявлення вразливостей у програмному забезпеченні, таких як недостатня обробка введених даних, незахищені точки входу або недостатня автентифікація та авторизація. Тестування вразливостей може включати побудову спеціально створених тестових наборів або використання автоматизованих інструментів для виявлення вразливостей у програмному коді.

2. Тестування атак. Цей тип тестування передбачає спроби зламати систему шляхом використання різних методів атак, таких як введення шкідливого коду, SQL-ін'єкції, перехоплення сесії тощо. Це допомагає виявити потенційні проблеми безпеки та перевірити, наскільки система стійка до різних видів атак.

3. Тестування захисту. Цей тип тестування спрямований на перевірку ефективності заходів безпеки, включених в програмне забезпечення, таких як шифрування даних, контроль доступу, механізми виявлення вторгнень тощо. Це допомагає переконатися, що система здатна ефективно захищати дані та запобігати несанкціонованому доступу.

4. Тестування забезпечення персональних даних. З огляду на регуляторні вимоги та зростаючу свідомість щодо захисту персональних даних, тестування забезпечення персональних даних стає все більш важливим. Це включає перевірку дотримання політик конфіденційності, захисту особистих даних та дотримання стандартів, таких як Загальний регламент щодо захисту даних (GDPR).

Для проведення тестування безпеки використовуються спеціалізовані інструменти, такі як засоби сканування вразливостей, інструменти аналізу коду, фідери тестування на проникнення та інші. Важливо відзначити, що тестування безпеки повинно бути постійним процесом, оскільки загрози безпеки постійно змінюються, і нові вразливості можуть виникати з часом.

Тестування безпеки вимагає спеціалізованого знання та досвіду, тому команди розробки зазвичай співпрацюють з експертами з безпеки або спеціалізованими консультантами для забезпечення ефективного тестування та забезпечення високого рівня безпеки програмного забезпечення.

Тестування мобільних додатків стало важливим етапом розробки, оскільки мобільні пристрої стали невід'ємною частиною нашого повсякденного життя. Зростаюча популярність мобільних додатків вимагає специфічних підходів та інструментів для забезпечення їх якості та надійності.

Основні аспекти тестування мобільних додатків включають (рис. 1.1):

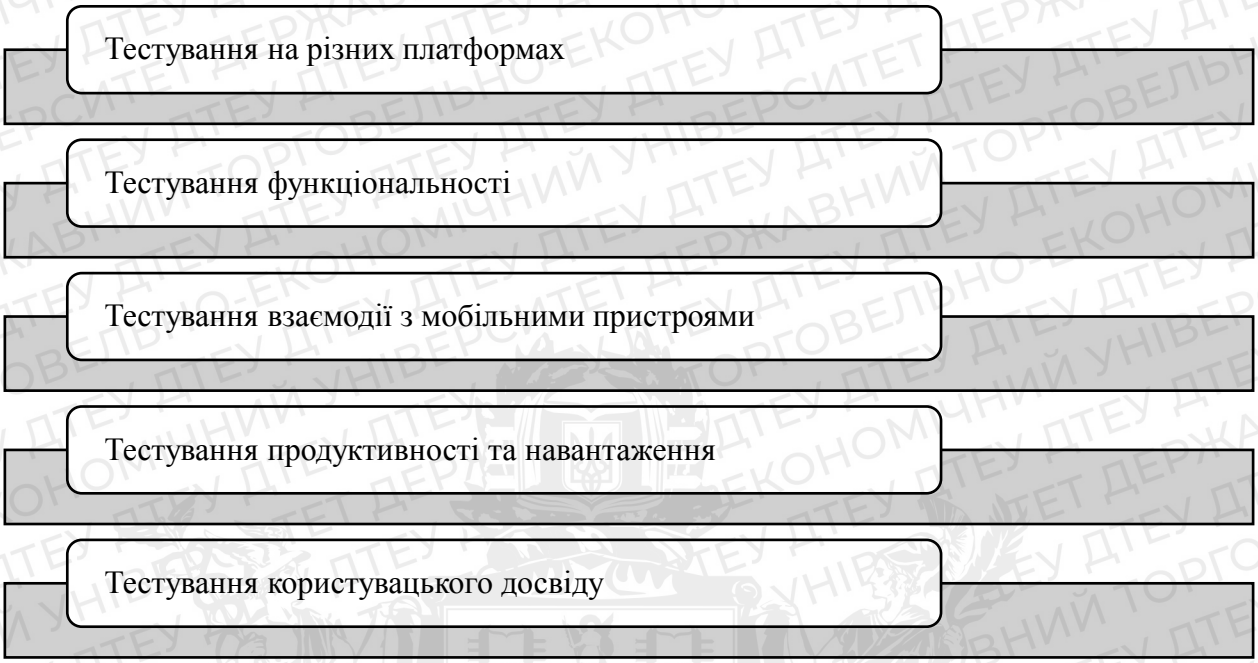


Рис. 1.1. Основні аспекти тестування мобільних додатків [8]

Мобільні додатки часто розробляються для різних платформ, таких як iOS та Android. Тестування повинно включати перевірку сумісності та коректну роботу додатку на різних операційних системах та версіях пристроїв. Це може включати тестування на різних пристроях, розмірах екранів та роздільній здатності.

Важливо перевірити, чи працюють всі функції додатка правильно. Це включає тестування різних сценаріїв використання, взаємодію з користувачем, обробку введених даних, навігацію та інші аспекти функціональності.

Мобільні додатки можуть використовувати різні функціональні можливості пристроїв, такі як камера, GPS, датчики руху тощо. Тестування повинно перевіряти, чи працюють ці можливості правильно та взаємодіють з додатком належним чином.

Мобільні додатки повинні бути ефективними та швидкими у роботі. Тестування продуктивності оцінює швидкодію, відгук та ресурсомісткість

додатка при різних умовах навантаження, таких як велика кількість користувачів або об'єм даних.

Мобільні додатки повинні мати зручний та привабливий для користувача інтерфейс. Тестування користувацького досвіду оцінює, наскільки легко користуватися додатком, чи зручна навігація, наявність зрозумілих та інтуїтивно зрозумілих елементів управління.

Для тестування мобільних додатків використовуються спеціалізовані інструменти, такі як фреймворки для автоматизованого тестування мобільних додатків (наприклад, Appium, Calabash, Xamarin.UITest) та віртуальні середовища для емуляції різних пристроїв та операційних систем.

Тестування мобільних додатків повинно бути комплексним і охоплювати різні аспекти, щоб забезпечити їх надійність, якість та задоволення користувачів.

Зростання використання хмарних послуг і веб-орієнтованих додатків вносить свої особливості до процесу тестування програмного забезпечення. Основні аспекти, які потрібно враховувати при тестуванні таких додатків, включають:

1. Сумісність з різними платформами та браузерами. Веб-орієнтовані додатки повинні працювати на різних платформах (Windows, macOS, Linux) та різних браузерах (Chrome, Firefox, Safari, Internet Explorer). Тестування повинно охоплювати перевірку сумісності додатка з різними комбінаціями платформи та браузера, а також перевірку коректності відображення та функціональності.

2. Тестування залежностей від зовнішніх служб. Веб-орієнтовані додатки можуть залежати від різних зовнішніх служб, таких як бази даних, сервіси платежів, системи аутентифікації тощо. Тестування повинно включати перевірку взаємодії з цими зовнішніми службами, а також перевірку стійкості додатка у випадку недоступності або неправильної роботи зовнішніх служб.

3. Тестування масштабованості. Хмарні та веб-орієнтовані додатки повинні бути здатні до роботи з великою кількістю користувачів та обробки великого обсягу даних. Тестування масштабованості включає оцінку продуктивності та швидкодії додатка при великому навантаженні, а також перевірку його стійкості та надійності при масштабному застосуванні.

4. Тестування безпеки. Хмарні та веб-орієнтовані додатки можуть потребувати особливої уваги до безпеки. Тестування безпеки повинно включати перевірку потенційних вразливостей, таких як кросс-сайтовий скриптинг, вразливості введення даних та злам паролів. Також варто перевірити захист даних та взаємодію зі зовнішніми службами, щоб забезпечити конфіденційність та надійність додатка.

5. Автоматизоване тестування. Завдяки розповсюдженню хмарних та веб-орієнтованих додатків, автоматизоване тестування стає все більш важливим. Використання спеціалізованих інструментів для автоматизації тестування дозволяє зменшити затрати на тестування, покращити швидкодію та надійність процесу тестування.

Тестування хмарних та веб-орієнтованих додатків повинно враховувати ці особливості та забезпечувати їхадійність, безпеку та якість для користувачів.

Загалом, стан проблеми тестування програмного забезпечення постійно змінюється, пристосовуючись до вимог ринку та нових технологій. Аналіз цих аспектів допоможе оцінити сучасний стан тестування ПЗ та спланувати ефективні підходи до його вирішення.

1.2. Особливості тестування програм для лабораторних робіт

Тестування програм для лабораторних робіт має свої особливості, оскільки вимагає перевірки наочності, точності результатів та відповідності певним науковим або технічним стандартам. особливості тестування програм для лабораторних робіт наведені в таблиці 1.2.

Таблиця 1.2

Особливості тестування програм для лабораторних робіт [1]

№	Особливість	Коротка характеристика
1	Валідація результатів	Важливо перевірити, чи програма повертає правильні результати для відомих тестових випадків. Тестові дані повинні бути обраними таким чином, щоб було відомо очікувані результати
2	Обробка помилок	При тестуванні програми для лабораторних робіт слід переконатися, що вона правильно обробляє можливі помилки. Наприклад, якщо введені дані неправильні або недоступні, програма повинна повідомляти про це користувачеві і відповідно реагувати
3	Візуалізація результатів	Часто лабораторні роботи включають велику кількість даних, які можуть бути представлені у вигляді графіків, діаграм або інших візуальних елементів. Важливо перевірити, що програма вірно відображає ці дані та графічно візуалізує їх
4	Ефективність	При обробці великих обсягів даних програма повинна працювати ефективно і швидко. Тестування на швидкодію та оптимізацію коду можуть бути важливими аспектами тестування програм для лабораторних робіт
5	Переносимість	Якщо програма призначена для використання на різних платформах або операційних системах, важливо перевірити, що вона працює на всіх цільових платформах і операційних системах, як очікується
6	Інтеграція з обладнанням	Якщо програма взаємодіє з лабораторним обладнанням, таким як датчики, пристрої вимірювання або зберігання даних, важливо переконатися, що програма правильно взаємодіє з цим обладнанням і отримує коректні дані з нього
7	Відновлення та збереження даних	При великих обсягах даних, збереження та відновлення їх може бути критично важливим. Перевірка правильності збереження та відновлення даних є важливою частиною тестування

Це лише кілька особливостей тестування програм для лабораторних робіт. Конкретні вимоги тестування будуть залежати від типу програми та призначення лабораторної роботи.

Саме тому розглянемо особливості тестування програм для лабораторних робіт, визначених у [1]. Можна виділити такі особливості тестування програм:

1. Валідація Git:

Перевірка правильності встановлення та налаштування Git. Тестування основних операцій системи контролю версій, таких як створення репозиторію, додавання, зміна та видалення файлів, коміти, злиття гілок тощо. Перевірка правильності відображення історії комітів та відновлення попередніх версій файлів.

2. Тестування середовищ розробки Python:

Перевірка правильності встановлення та налаштування IDE Visual Studio Code та IDLE. Тестування здатності редакторів до забезпечення зручного редагування коду, автодоповнення, відлагодження та керування проектами.

3. Перевірка введення і виведення даних у Python:

Тестування коректності зчитування та виведення різних типів даних: числа, рядки, списки і т.д. Перевірка обробки помилок при некоректному введенні даних.

4. Тестування функцій, математичних операторів і бібліотек:

Перевірка правильності роботи функцій та математичних операцій. Тестування взаємодії з бібліотеками math і random та перевірка коректності результатів їх використання.

5. Тестування розгалужень і циклів:

Перевірка коректності виконання умовних операторів і розгалужень в програмі. Тестування циклів і перевірка правильності обробки ітерацій.

6. Тестування роботи зі списками, рядками, масивами, кортежами, словниками та множинами:

Перевірка додавання, видалення, зміни елементів та інших операцій над структурами даних. Тестування пошуку, сортування та фільтрації даних у вказаних структурах.

7. Тестування роботи з файлами:

Перевірка коректності відкриття, читання, запису та закриття файлів. Тестування обробки різних форматів файлів, таких як текстові файли, CSV і JSON.

8. Тестування об'єктно-орієнтованого програмування:

Перевірка створення класів, створення та використання об'єктів. Тестування наслідування, поліморфізму та інших принципів ООП на мові Python.

9. Тестування збору даних з веб-документів за допомогою Python:

Перевірка коректності отримання та обробки даних з веб-сторінок або API. Тестування правильності парсингу HTML, XML або JSON даних.

10. Тестування розробки віконних додатків на мові Python:

Перевірка відображення і взаємодії з елементами інтерфейсу. Тестування реакції програми на дії користувача.

11. Тестування побудови графіків математичних функцій на мові Python:

Перевірка правильності побудови графіків за допомогою бібліотеки для візуалізації даних. Тестування точності та зручності використання графічного інтерфейсу.

12. Тестування застосування бібліотеки Pandas для обробки даних:

Перевірка правильності імпорту бібліотеки та коректності виконання основних операцій над даними, таких як фільтрація, групування, сортування тощо. Тестування ефективності обробки великих обсягів даних.

1.3. Характеристика існуючих технологій тестування

Тестування програмного забезпечення (Software Testing) включає в себе процес валідації та верифікації програм, з метою виявлення помилок, дефектів та недоліків. Існує багато різних технологій тестування програмного забезпечення, які використовуються для забезпечення якості програм та зниження ризиків.

Найбільш поширені технології тестування ПЗ наведені у таблиці:

Таблиця 1.3

Технології тестування ПЗ

№	Технологія	Опис технології
1	Модульне тестування (Unit Testing)	Це рівень тестування, в якому окремі модулі або компоненти програми перевіряються на наявність дефектів. Часто використовуються автоматизовані тестові фреймворки, які дозволяють легко створювати та виконувати модульні тести.
2	Інтеграційне тестування (Integration Testing)	Цей тип тестування виконується для перевірки взаємодії між різними модулями або компонентами програми. Його мета - виявити проблеми, які можуть виникнути при об'єднанні окремих частин програми в один функціональний блок.
3	Системне тестування (System Testing)	У цьому виді тестування програма перевіряється в цілому, як єдиний інтегрований продукт. Він спрямований на перевірку, чи відповідає програма вимогам та специфікаціям, і чи працює вона належним чином у різних сценаріях використання.
4	Автоматизоване тестування (Automated Testing)	Це використання спеціальних програмних інструментів для автоматизації виконання тестів. Воно дозволяє зменшити ручну працю та забезпечити швидке виконання тестових сценаріїв, що полегшує процес тестування.
5	Функціональне тестування (Functional Testing)	Це вид тестування, який перевіряє, чи відповідає програмне забезпечення функціональним вимогам. Це включає тестування інтерфейсу користувача, реакції програми на вхідні дані та правильність виведення результатів.
6	Навантажувальне тестування (Load Testing)	Цей тип тестування використовується для оцінки поведінки програми під навантаженням. Воно перевіряє, як програма працює за умов постійного навантаження, великої кількості користувачів або обробки великих обсягів даних.

Модульне тестування (Unit Testing) є одним з найбільш поширених видів тестування програмного забезпечення. Воно зосереджується на перевірці окремих модулів або компонентів програми на наявність дефектів або помилок.

Основна ідея модульного тестування полягає в тому, щоб кожен модуль програми тестувався окремо і незалежно від інших модулів. Таким чином, ви можете переконатися, що кожен модуль працює правильно перед їх об'єднанням в більш складну систему.

Основні особливості модульного тестування:

1. Ізоляція модулів: Кожен модуль тестується ізольовано від інших модулів та залежностей. Це означає, що для проведення модульного тестування можуть використовуватися фейкові або заглушкові об'єкти, які замінюють реальні залежності.

2. Використання автоматизованих тестових фреймворків: Для створення та виконання модульних тестів зазвичай використовуються спеціальні автоматизовані тестові фреймворки, такі як JUnit для Java, NUnit для .NET, або pytest для Python. Ці фреймворки надають зручний спосіб описати тести, виконувати їх та аналізувати результати.

3. Частота виконання: Модульні тести зазвичай виконуються дуже часто, ідеально - після кожного змінного цільового коду. Це допомагає виявляти дефекти та помилки на ранніх етапах розробки, що забезпечує їх швидке виправлення.

4. Покриття коду: Модульне тестування дозволяє досягнути високого покриття коду, оскільки кожен модуль тестується окремо. Покриття коду вимірює, яка частина програмного коду була протестована, тим самим допомагаючи виявити недостатньо протестовані частини програми.

Переваги модульного тестування включають покращення якості програмного забезпечення, зниження ризиків, полегшення розуміння коду, сприяння швидкому виявленню та виправленню дефектів.

Проте модульне тестування має певні обмеження. Воно не перевіряє взаємодію між модулями та залежностями, а також не виявляє проблеми, які можуть виникнути при інтеграції модулів. Тому додаткові рівні тестування, такі як інтеграційне тестування та системне тестування, також є необхідними для забезпечення якості програмного забезпечення в цілому.

Інтеграційне тестування (Integration Testing) є важливим етапом в процесі тестування програмного забезпечення. Його основна мета - перевірити взаємодію між різними модулями, компонентами або сервісами програми під час їх об'єднання в один функціональний блок або систему. Це допомагає виявити проблеми, які можуть виникнути через несправну взаємодію між компонентами програми.

Особливості інтеграційного тестування:

1. Способи інтеграції: Інтеграційне тестування може виконуватися поетапно, де модулі об'єднуються і тестуються послідовно, або одночасно, коли всі модулі взаємодіють одночасно для перевірки повного функціонування системи.

2. Тести на взаємодію: При інтеграційному тестуванні перевіряється, чи відбувається правильна взаємодія між компонентами програми, чи передаються дані та повідомлення вірно, і чи повертаються очікувані результати.

3. Інтеграційні залежності: Під час інтеграційного тестування перевіряються залежності між модулями або компонентами програми. Це означає, що, крім самого тестування, також важливо переконатися, що всі залежності належним чином налаштовані та виконуються.

4. Мокування та заглушки: Для успішного інтеграційного тестування можуть використовуватися техніки мокування (mocking) або створення заглушок (stubbing), щоб замінити реальні компоненти, які ще не готові або недоступні.

5. Регресійні тести: Інтеграційне тестування також може включати регресійні тести, щоб переконатися, що попередні виправлення або зміни в програмі не призвели до появи нових проблем у взаємодії між компонентами.

Переваги інтеграційного тестування включають виявлення проблем на ранніх етапах розробки, забезпечення стабільної взаємодії між компонентами, зменшення ризиків при інтеграції та поліпшення якості програмного забезпечення в цілому.

Проте інтеграційне тестування може бути складним та вимагати багато ресурсів, особливо при наявності багатьох компонентів або складних залежностей. Для успішного виконання інтеграційного тестування важливо добре спланувати тести, забезпечити належну конфігурацію середовища та мати достатню кількість тестових данних.

Системне тестування (System Testing) є одним з ключових етапів тестування програмного забезпечення і зосереджується на перевірці програми як єдиного інтегрованого продукту. Його ціль полягає у визначенні, чи відповідає програма вимогам, специфікаціям та очікуванням користувачів, і чи працює вона належним чином в різних сценаріях використання.

Особливості системного тестування:

1. Тестування функціональності: Під час системного тестування перевіряється, чи працюють всі функціональні можливості програми згідно з вимогами та специфікаціями. Це включає перевірку коректності введення та виведення даних, правильність обробки операцій, відповідність очікуваним результатам тощо.

2. Тестування сценаріїв використання: Системне тестування включає перевірку програми в різних сценаріях використання, що можуть зустрічатися у реальному житті. Це означає виконання типових дій користувача, тестування різних шляхів взаємодії з програмою, а також перевірку, як програма реагує на різні ситуації.

3. Тестування навантаження: У системному тестуванні можуть бути виконані тести навантаження, які оцінюють поведінку програми при роботі з великими обсягами даних або при великій кількості одночасних користувачів. Це допомагає виявити можливі проблеми продуктивності та масштабованості.

4. Тестування безпеки: Важливо включити тести безпеки в системне тестування, щоб перевірити, чи відповідає програма стандартам безпеки і чи захищена від вразливостей та атак.

5. Регресійні тести: Під час системного тестування також можуть проводитись регресійні тести для перевірки, чи не виникли нові проблеми після внесення змін в програму або після виправлення попередніх помилок.

6. Умовне тестування: Системне тестування може включати виконання тестів у різних умовах, таких як різні операційні системи, конфігурації апаратного забезпечення або мережеві умови, щоб переконатися, що програма працює належним чином у всіх таких ситуаціях.

Переваги системного тестування включають підтвердження відповідності програми вимогам та специфікаціям, виявлення проблем на рівні системи та переконання, що програма готова до використання користувачами. Проте системне тестування може бути часо- та ресурсозатратним, а також вимагати належного планування та підготовки тестових середовищ.

Автоматизоване тестування (Automated Testing) - це використання спеціальних програмних інструментів і технологій для автоматизації виконання тестів. Замість виконання тестів вручну, автоматизоване тестування дозволяє створювати, виконувати і аналізувати тестові сценарії автоматично.

Особливості автоматизованого тестування:

1. Скриптовані тести: У автоматизованому тестуванні тестові сценарії зазвичай створюються у вигляді скриптів або сценаріїв, що можуть бути виконані автоматично. Це можуть бути скрипти на мові програмування,

спеціальні мови скриптування для тестування (наприклад, Selenium для веб-тестування) або візуальні інструменти створення тестів.

2. Швидкість та ефективність: Автоматизоване тестування дозволяє швидко виконувати велику кількість тестів, що значно прискорює процес тестування. Також воно дозволяє виконувати тести у повторюваних умовах і швидко перевіряти, чи виконує програма очікувані результати.

3. Повторюваність тестів: Автоматизоване тестування дозволяє легко повторювати тести в будь-який момент. Це дозволяє швидко перевіряти, чи вплинули внесені зміни на раніше пройдені тести і чи виникли нові проблеми.

4. Тестування регресії: Автоматизоване тестування особливо корисне для тестування регресії, коли необхідно перевірити, чи не виникли нові проблеми після внесення змін у програму або після виправлення попередніх помилок.

5. Тестування складних сценаріїв: Автоматизоване тестування дозволяє створювати та виконувати складні тестові сценарії, які можуть включати взаємодію з багатьма компонентами або виконання послідовності дій. Воно полегшує тестування складних функціональностей та взаємодій.

Переваги автоматизованого тестування включають підвищену ефективність, швидкість та повторюваність тестів, полегшений процес регресійного тестування, зменшення ручної праці та зниження ймовірності людських помилок. Однак, автоматизоване тестування також вимагає вкладення часу і ресурсів для розробки та підтримки тестових скриптів, а також врахування його обмежень у випадках, коли тестування вимагає людського експерта або оцінки візуальних аспектів.

Функціональне тестування (Functional Testing) є одним з основних видів тестування програмного забезпечення і спрямоване на перевірку, чи відповідає програма вимогам функціональності. Його основна мета полягає в перевірці того, чи працює програмне забезпечення так, як очікується від нього.

Особливості функціонального тестування:

1. Перевірка вимог: Функціональне тестування спрямоване на перевірку, чи відповідає програмне забезпечення вимогам, які визначені у специфікації або вимогах до продукту. Тестування проводиться для кожної функції окремо та в контексті їх взаємодії.

2. Тестування інтерфейсу користувача: Функціональне тестування включає перевірку роботи інтерфейсу користувача, його взаємодії з користувачем та правильності відображення інформації на екрані. Це може включати перевірку кнопок, полів введення, меню, діалогових вікон та інших елементів інтерфейсу.

3. Тестування вхідних та вихідних даних: Під час функціонального тестування перевіряється, як програма обробляє вхідні дані і генерує відповідні результати. Тести перевіряють коректність обробки різних типів даних, включаючи валідацію, розрахунки, маніпуляції даними тощо.

4. Тестування функціональних сценаріїв: Функціональне тестування включає перевірку роботи програми у відповідності до різних сценаріїв використання. Тести виконують послідовність дій, які користувачі зазвичай виконують в програмі, і перевіряють, чи працює програма відповідно до очікувань.

5. Тестування виключних ситуацій: Функціональне тестування також включає перевірку реакції програми на виключні ситуації, такі як помилкові вхідні дані, некоректні виклики функцій, відсутність ресурсів тощо. Тести перевіряють, чи відбувається правильна обробка помилок і відновлення після них.

Переваги функціонального тестування включають виявлення проблем функціональності, перевірку відповідності вимогам та забезпечення коректної роботи програмного забезпечення. Однак, функціональне тестування не охоплює інші аспекти, такі як продуктивність, безпека та надійність, які можуть бути об'єктом інших видів тестування.

Навантажувальне тестування (Load Testing) є видом тестування програмного забезпечення, спрямованим на оцінку поведінки системи під навантаженням, коли об'єм роботи або кількість користувачів перевищує нормальні робочі навантаження. Його основна мета - визначити межі та реагування системи на інтенсивне використання та перевантаження.

Особливості навантажувального тестування:

1. Створення навантаження: У цьому виді тестування створюється штучне навантаження, що може бути викликане одночасно великою кількістю користувачів або великим обсягом даних. Це може бути здійснене за допомогою спеціальних інструментів для навантажувального тестування, які генерують симульовані запити до системи.

2. Вимірювання продуктивності: Під час навантажувального тестування збираються дані про продуктивність системи, такі як час відгуку, завантаженість ресурсів (пам'ять, процесор) та пропускна здатність. Це дозволяє оцінити, наскільки система здатна витримувати велике навантаження та якісно виконувати свої функції.

3. Ідентифікація проблем: Навантажувальне тестування допомагає виявити проблеми, які можуть виникнути під час перевантаження системи, такі як падіння продуктивності, збої, перевантаження мережі тощо. Це дозволяє розробникам інформаційних систем виявити й виправити такі проблеми перед їх виникненням у реальних умовах.

4. Масштабування системи: Навантажувальне тестування також використовується для оцінки масштабованості системи. Шляхом поступового збільшення навантаження, тести дозволяють визначити, до яких меж система здатна розширюватися та які ресурси необхідні для забезпечення стабільної роботи при збільшенні обсягів.

Переваги навантажувального тестування включають виявлення проблем продуктивності, визначення потенційних меж системи та вдосконалення її масштабованості. Це допомагає забезпечити належну

роботу програмного забезпечення під високим навантаженням та запобігти проблемам у реальних умовах експлуатації.

РОЗДІЛ 2
РОЗРОБКА МОДЕЛІ ТЕСТІВ

2.1. Загальна концепція

Розробка моделі тестів для перевірки правильності виконання лабораторних робіт на мові Python передбачає створення набору тестів, які будуть перевіряти коректність виконання завдань і відповідність отриманих результатів очікуваним.

Основна концепція полягає у створенні автоматизованого процесу перевірки:

1. Аналіз завдань: Детально проаналізуйте лабораторні роботи, зрозумійте, які результати повинні бути отримані при правильному виконанні. Розгляньте всі можливі сценарії та вхідні дані, щоб визначити, які тести потрібно створити.

2. Визначення критеріїв оцінювання: Встановіть критерії оцінювання для кожного завдання. Визначте, які результати повинні бути правильними, які формати даних повинні використовуватися і які умови повинні бути задоволені.

3. Створення тестів: Напишіть набір тестів для кожного завдання. Тести можуть бути реалізовані у вигляді окремих функцій або класів, які перевіряють вхідні дані та порівнюють отримані результати з очікуваними.

4. Автоматизація перевірки: Створіть систему, яка автоматично запускає набір тестів для кожного завдання. Використовуйте фреймворк для тестування, такий як **unittest** або **pytest**, щоб виконати тести та зібрати результати.

5. Звітність: Забезпечте зручний звіт про результати виконання тестів. Це може включати загальну кількість тестів, кількість пройдених тестів, кількість непройдених тестів та деталізовані повідомлення про помилки.

6. Постійне вдосконалення: Регулярно переглядайте тести та оновлюйте їх відповідно до змін у лабораторних завданнях або мові Python. Також розгляньте можливість додавання нових тестів для покриття додаткових сценаріїв.

Це загальна концепція розробки моделі тестів для перевірки правильності виконання лабораторних робіт на мові Python. За допомогою цієї моделі ви зможете автоматизувати процес перевірки та отримувати об'єктивну звітність щодо правильності виконання завдань.

2.2. Моделі обчислень

У розробці моделі тестів для перевірки правильності виконання лабораторних робіт на мові Python можна використовувати різні моделі обчислень. Нижче наведено кілька прикладів таких моделей:

1. Детерміновані моделі.

У детермінованих моделях тестування ви використовуєте конкретні вхідні дані та очікувані результати, щоб перевірити правильність виконання завдань. Ось кілька кроків, які можна виконати при розробці тестів на основі детермінованих моделей:

– Визначте тестові випадки: Ретельно проаналізуйте завдання і визначте конкретні випадки, які ви хочете перевірити. Виберіть різноманітні вхідні дані, включаючи граничні значення і типові сценарії.

– Розробіть очікувані результати: Для кожного тестового випадку визначте очікуваний результат або вихідні дані. Це може бути конкретне значення, список, словник або будь-який інший тип даних, залежно від природи завдання.

– Напишіть тести: Напишіть код тестів, які запускають вхідні дані через вашу функцію або програму і порівнюють отримані результати з

очікуваними. Використовуйте оператори порівняння (наприклад, ==) або спеціальні функції для порівняння даних.

– Запустіть тести: Запустіть ваш набір тестів і перевірте, чи проходять всі тести. Ви можете використовувати фреймворк для тестування, такий як **unittest** або **pytest**, для створення тестових наборів і запуску їх одним натисканням.

– Аналіз результатів: Після запуску тестів проаналізуйте результати. Виявіть, які тести пройшли успішно, а які не пройшли. Для невдалого тесту виведіть повідомлення про помилку, яке допоможе вам знайти та виправити проблему.

– Вдосконалення тестів: Регулярно переглядайте ваші тести та оновлюйте їх відповідно до змін у завданнях або програмі. Додавайте нові тести, якщо знайдете нові сценарії або вимоги до коду.

Це загальна концепція розробки тестів на основі детермінованих моделей. Вона дозволяє вам використовувати конкретні вхідні дані та очікувані результати для перевірки правильності виконання завдань.

2. Генеративні моделі.

У генеративних моделях тестування ви використовуєте генеративні алгоритми для створення випадкових вхідних даних і перевірки результатів вашої функції або програми. Ось кілька кроків, які можна виконати при розробці тестів на основі генеративних моделей:

– Визначте параметри генерації: Визначте параметри генерації випадкових вхідних даних, такі як діапазон значень, розподіл даних або шаблони, що потрібно враховувати. Наприклад, якщо ви тестуєте функцію для обчислення суми двох чисел, параметром може бути діапазон цих чисел або їх розподіл.

– Генеруйте тестові набори: Використовуйте генераційні алгоритми, щоб створити велику кількість випадкових тестових наборів з різними

вхідними даними. Застосовуйте вашу функцію або програму до кожного тестового набору і зберігайте отримані результати.

– Порівнюйте результати: Порівняйте отримані результати з очікуваними результатами, використовуючи заздалегідь визначені вимоги до правильності виконання завдань. Ви можете використовувати оператори порівняння або спеціальні функції для порівняння даних.

– Аналізуйте результати: Проаналізуйте результати тестів. Виявіть, які тести пройшли успішно, а які не пройшли. Для невдалого тесту виведіть повідомлення про помилку, яке допоможе вам знайти та виправити проблему.

– Вдосконалюйте тестування: Регулярно переглядайте тестові набори та оновлюйте їх відповідно до змін у завданнях або програмі. Додавайте нові тестові набори з різними сценаріями, щоб покрити якомога більше можливих випадків.

Генеративні моделі тестування дозволяють створювати широкий спектр вхідних даних і перевіряти, чи правильно працює ваша функція або програма в різних сценаріях. Вони особливо корисні, коли важко вручну визначити всі можливі тестові випадки.

3. Властивісні моделі.

– Властивісні моделі тестування зосереджені на перевірці певних властивостей або характеристик, які повинні бути виконані функцією або програмою. Ось кілька кроків, які можна виконати при розробці тестів на основі властивісних моделей:

– Визначте властивості: Визначте властивості, які повинна мати ваша функція або програма. Наприклад, це може бути правильний тип поверненого значення, правильна обробка виключень, коректність стану після виконання функції і т.д.

– Розробіть тести: Напишіть тести, які перевіряють властивості вашої функції або програми. Наприклад, для перевірки типу поверненого значення

ви можете написати тест, який викликає функцію і перевіряє тип поверненого значення за допомогою вбудованих функцій Python, таких як isinstance().

– Запустіть тести: Запустіть ваші тести і перевірте, чи властивості виконуються для вашої функції або програми. Якщо властивість не виконується, виведіть повідомлення про помилку, яке допоможе вам знайти та виправити проблему.

– Аналізуйте результати: Проаналізуйте результати тестів і визначте, які властивості були порушені. Це допоможе вам знайти потенційні проблеми у вашій функції або програмі і виправити їх.

– Вдосконалюйте тестування: Регулярно переглядайте ваші тести і оновлюйте їх відповідно до змін у властивостях або характеристиках, які повинні бути виконані вашою функцією або програмою. Додавайте нові тести, якщо знайдете нові властивості, які потрібно перевірити.

4. Моделі залежності.

Моделі залежності в тестуванні дозволяють перевірити правильність взаємодії між різними частинами програми. Основна ідея полягає в тому, що ви перевіряєте, чи взаємодіють різні компоненти програми відповідно до очікувань.

2.3. Алгоритми обчислень

Система тестування для перевірки програм на мові Python може включати в себе різні алгоритми обчислень. Основна мета такої системи полягає в перевірці правильності виконання програми та її відповідності до очікувань.

Ось кілька алгоритмів обчислень, які можна використовувати в системі тестування:

1. Детерміновані алгоритми: Використовуйте детерміновані алгоритми, які передбачають використання конкретних вхідних даних і порівняння фактичних результатів з очікуваними. Наприклад, ви можете створити тест, який передає певні вхідні дані до функції і перевіряє, чи повертає вона правильний результат.

2. Генеративні алгоритми: Використовуйте генеративні алгоритми для створення випадкових вхідних даних та перевірки результатів. Наприклад, ви можете створити тест, який генерує випадкові числа і передає їх до функції, перевіряючи правильність результату.

3. Алгоритми залежності: Використовуйте алгоритми залежності для перевірки взаємодії різних частин програми. Наприклад, ви можете створити тест, який перевіряє, чи викликаються потрібні функції з правильними аргументами або чи відбувається взаємодія між класами.

4. Алгоритми властивостей: Використовуйте алгоритми властивостей для перевірки характеристик програми. Наприклад, ви можете перевірити, чи виконуються певні властивості, такі як правильний тип поверненого значення або правильна обробка виключень.

5. Алгоритми покриття коду: Використовуйте алгоритми покриття коду для перевірки, чи були виконані всі шляхи в програмі. Наприклад, ви можете перевірити, чи виконується кожен рядок коду, чи викликаються всі функції та методи.

Ці алгоритми можна поєднувати та комбінувати для створення різних видів тестів та покриття різних аспектів програми. Важливо враховувати специфіку вашого проекту та поставити конкретні цілі для системи тестування.

РОЗДІЛ 3
СИСТЕМА ТЕСТІВ ДЛЯ ПРОГРАМ НА МОВІ PYTHON

3.1. Розробка інформаційно-логічної моделі

У ході даного дослідження була розроблена автоматизована система, яка підключатиме файл, створений студентами, в якості модуля, і перевірятиме, чи відповідатимуть результати заданим значенням. На основі лабораторних робіт, розроблених В.М. Базуріним [1], була розроблена така автоматизована система для лабораторних робіт № 3, 8 та 9.

Розглянемо інформаційно-логічні моделі розроблених АС.

АС для лабораторної роботи № 3.

Ця програма складається з двох функцій: **find_divisors** та **remove_extremes**.

Функція **find_divisors** отримує на вхід ціле число **n** і повертає масив додатніх дільників числа **n**. Вона проходить по всім числам від 1 до **n** і перевіряє, чи ділиться **n** націло на кожне з цих чисел. Якщо так, то воно додається до масиву **divisors**. На виході функція повертає масив **divisors.**

Функція **remove_extremes** отримує на вхід масив **arr** і видаляє з нього найбільший і найменший елементи. Спочатку функція перевіряє, чи масив має довжину менше або дорівнює 2. Якщо так, то повертається пустий масив. Якщо ж довжина масиву більше 2, то знаходяться найменший і найбільший елементи за допомогою функцій **min** і **max**. Потім ці елементи видаляються з масиву за допомогою методів **remove**. На виході функція повертає змінений масив **arr**.

У основній частині програми спочатку користувач вводить ціле число **N**. Потім викликається функція **find_divisors** з аргументом **N**, щоб знайти додатні дільники числа **N**. Отриманий масив передається функції **remove_extremes**, яка видаляє найбільший і найменший елементи. На виході

отримується змінений масив, який виводиться на екран за допомогою функції **print**.

АС для лабораторної роботи № 8.

Ця програма складається з однієї функції **convert_string_to_tuple** і основної частини програми.

Функція **convert_string_to_tuple** отримує на вхід рядок **string**, який містить дійсні числа, розділені пробілами. Спочатку функція використовує метод **split**, щоб розділити рядок на окремі числа. Результатом є список **numbers**, де кожен елемент - це окреме число у вигляді рядка.

Далі функція проходить по кожному елементу списку **numbers** за допомогою генератора списку. Кожний елемент перетворюється на дійсне число за допомогою функції **float**. Отриманий список дійсних чисел має назву **float_numbers**.

Нарешті, функція створює кортеж **tuple_numbers**, використовуючи функцію **tuple** і передає у неї список **float_numbers**. На виході функція повертає кортеж **tuple_numbers**.

У основній частині програми спочатку користувач вводить рядок дійсних чисел, розділених пробілами, за допомогою функції **input**. Введений рядок передається функції **convert_string_to_tuple**, щоб перетворити його у кортеж чисел.

Отриманий кортеж зберігається у змінній **result_tuple**. На екран виводиться повідомлення, що містить результат - кортеж чисел, за допомогою функції **print**.

АС для лабораторної роботи № 9.

Ця програма складається з однієї функції **calculate_triangle_area** і основної частини програми.

Функція **calculate_triangle_area** отримує на вхід довжини катетів прямокутного трикутника **a** і **b**. Вона обчислює площу прямокутного трикутника за формулою **0.5 * a * b** і зберігає результат у змінній **area**. На виході функція повертає значення площі **area**.

У основній частині програми спочатку користувач вводить довжини катетів прямокутного трикутника за допомогою функції **input**. Введені значення перетворюються на дійсні числа за допомогою функції **float** і зберігаються у змінних **a** і **b**.

Потім викликається функція **calculate_triangle_area** з аргументами **a** і **b**, щоб обчислити площу прямокутного трикутника. Отримане значення площі зберігається у змінній **area**.

На екран виводиться повідомлення, що містить результат - площу прямокутного трикутника, за допомогою функції **print**. Значення **area** підставляється у повідомлення за допомогою оператора кома.

3.2. Програмна реалізація тестів

Розглянемо програмну реалізації розроблених АС.

АС для лабораторної роботи № 3.

Лабораторна робота №3. Середовища розробки IDLE і Visual Studio Code. Введення і виведення даних у Python

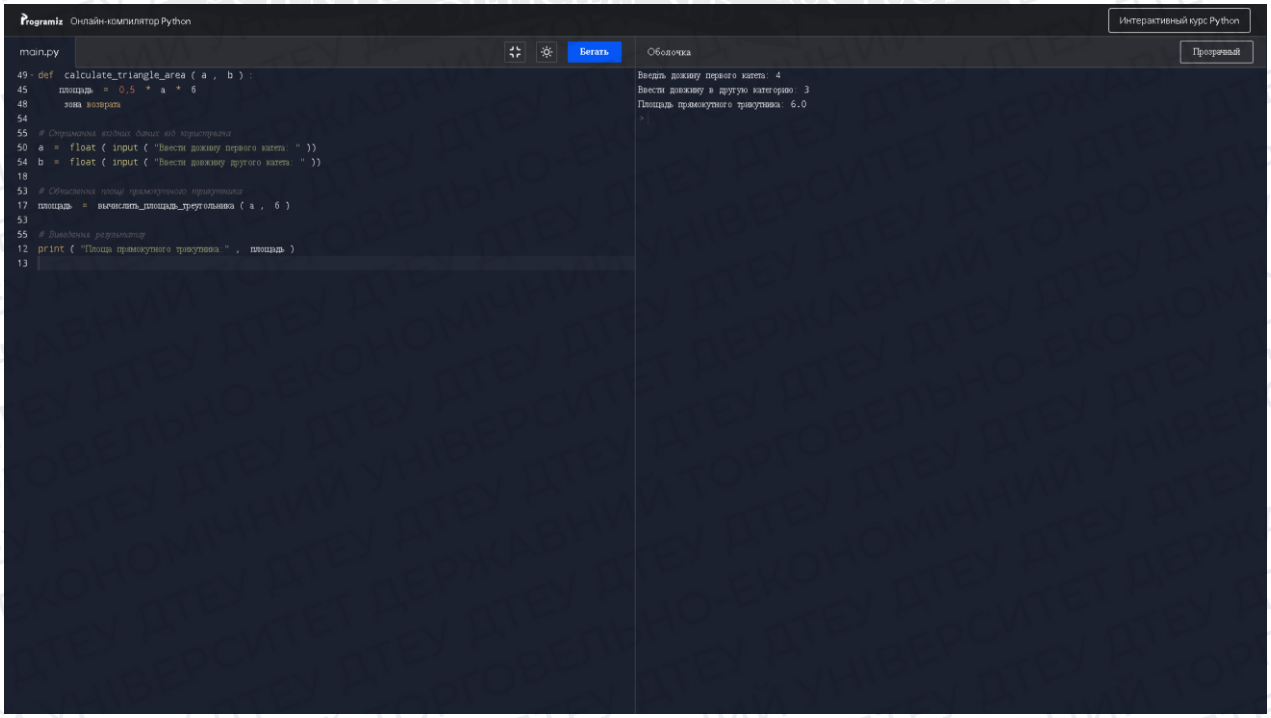


Рис. 3.1. Програмна реалізація розробленої АС для ЛР № 3

Варіант 1

Написати програму, яка обчислюватиме площу прямокутного трикутника. Вхідні дані – довжини катетів.

```
def calculate_triangle_area(a, b):
    area = 0.5 * a * b
    return area
# Отримання вхідних даних від користувача
a = float(input("Введіть довжину першого катета: "))
b = float(input("Введіть довжину другого катета: "))
# Обчислення площі прямокутного трикутника
area = calculate_triangle_area(a, b)
# Виведення результату
print("Площа прямокутного трикутника:", area)
```

АС для лабораторної роботи № 8.

Користувач задає ціле число N. Програма повинна заповнити масив додатними дільниками числа N і видалити з масиву найбільший і найменший елементи.

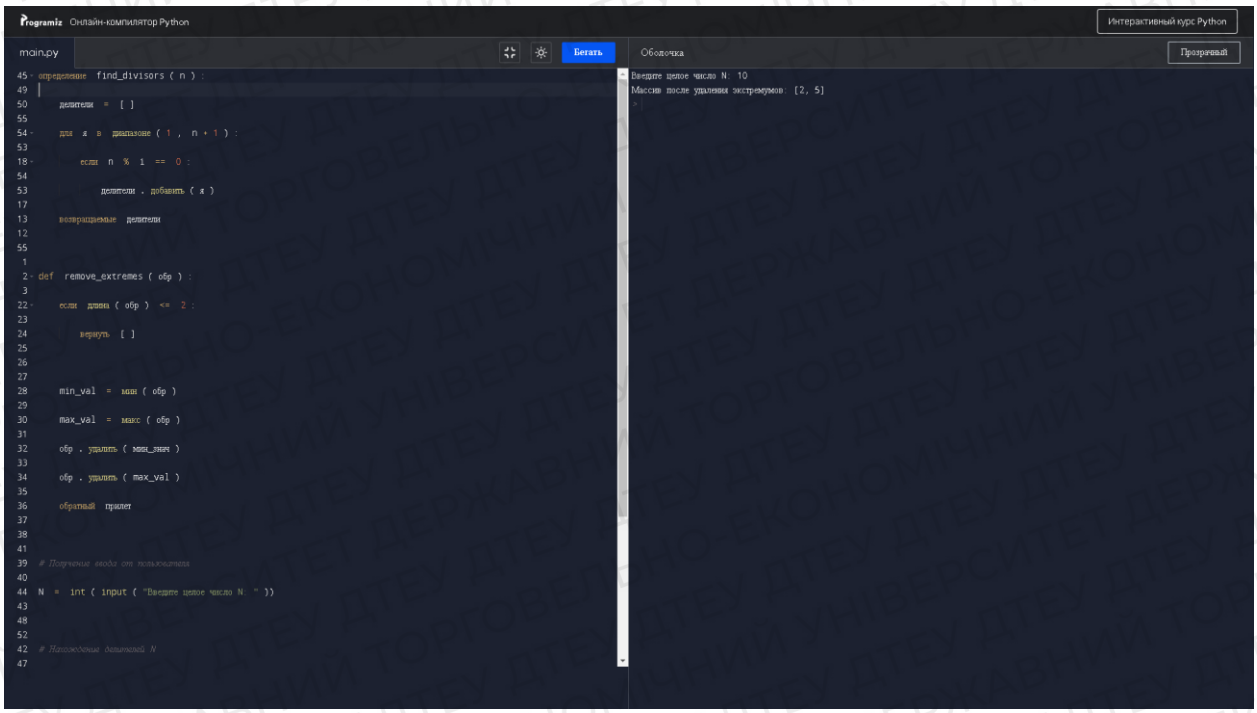


Рис. 3.2. Програмна реалізація розробленої АС для ЛР № 9

```
Варіант 1
def find_divisors(n):
    divisors = []
    for i in range(1, n+1):
        if n % i == 0:
            divisors.append(i)
    return divisors
def remove_extremes(arr):
    if len(arr) <= 2:
        return []
    min_val = min(arr)
    max_val = max(arr)
    arr.remove(min_val)
    arr.remove(max_val)
    return arr
# Getting input from the user
N = int(input("Enter an integer N: "))
# Finding the divisors of N
divisors = find_divisors(N)
# Removing the largest and smallest elements
result = remove_extremes(divisors)
# Printing the resulting array
print("Array after removing extremes:", result)
```

АС для лабораторної роботи № 9.

Варіант 1

1. Написати програму, яка перетворює рядок дійсних чисел, записаних через пробіл, у кортеж.

```
def convert_string_to_tuple(string):
```

```
    numbers = string.split()
    float_numbers = [float(num) for num in numbers]
    tuple_numbers = tuple(float_numbers)
    return tuple_numbers
```

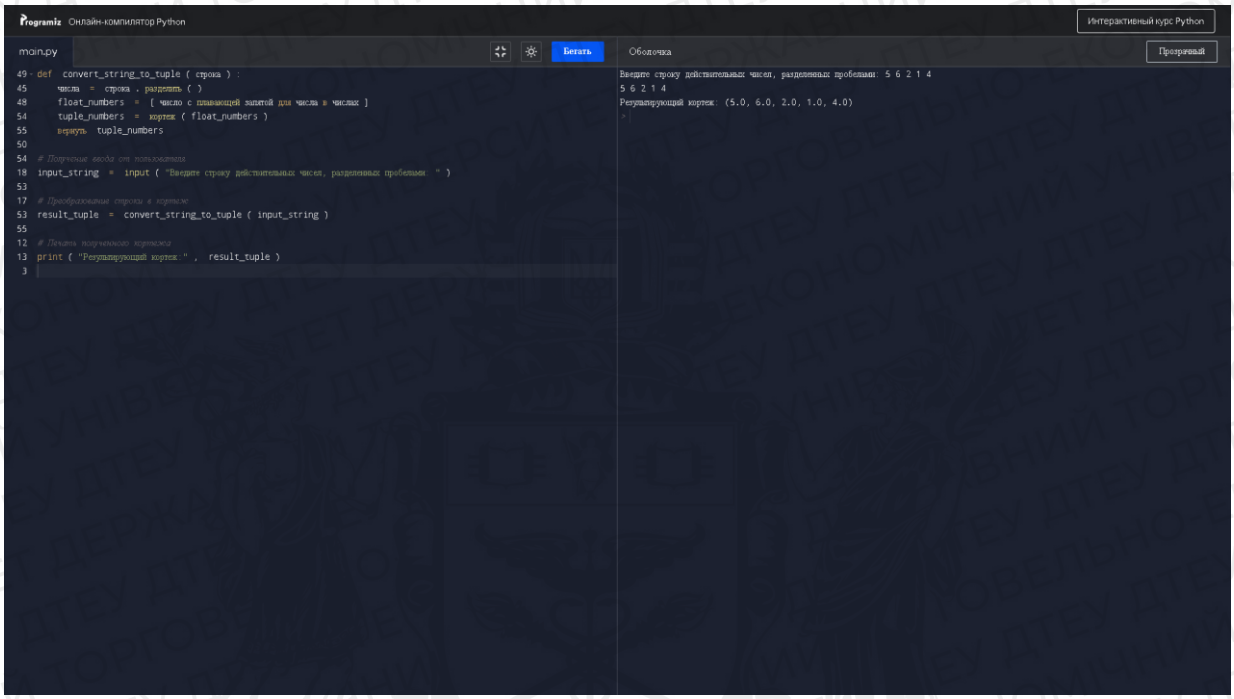


Рис. 3.3. Програмна реалізація розробленої АС для ЛР № 9

```
# Getting input from the user
input_string = input("Enter a string of real numbers separated by spaces: ")
 # Converting the string to a tuple
result_tuple = convert_string_to_tuple(input_string)
 # Printing the resulting tuple
print("Resulting tuple:", result_tuple)
```

3.3. Технологія використання системи тестів

Однією з популярних технологій використання системи тестування є автоматизоване тестування. Автоматизоване тестування передбачає написання спеціальних скриптів або програм, які виконують тестові сценарії та перевіряють, чи поводиться програмне забезпечення вірно.

Системи тестування, такі як Pytest, Unittest або Robot Framework, дозволяють розробникам створювати тести для своїх програм і автоматично виконувати їх на різних етапах розробки. Тести можуть перевіряти правильність роботи окремих функцій або модулів, а також інтеграцію між ними.

При використанні системи тестування розробники можуть описувати очікувані результати для кожного тесту, що дозволяє автоматично перевірити, чи відповідає фактичний результат очікуваному. Це зменшує кількість ручного тестування і дозволяє швидше виявляти помилки та проблеми.

Також, системи тестування забезпечують можливість створення набору тестів, що можуть бути виконані автоматично при кожному зміні коду. Це допомагає виявляти проблеми з регресією - коли зміни в програмі призводять до неправильної роботи раніше працюючих функцій.

Загальна ідея використання системи тестування полягає в тому, щоб мати автоматизовані тести, які можуть бути запущені в будь-який момент, щоб перевірити, чи працює програма вірно, та сповіщати розробників про виявлені проблеми. Це допомагає підтримувати якість програмного забезпечення, полегшує процес розробки і забезпечує більш надійну роботу програми.

Розглянемо технологію використання системи тестів:

АС для лабораторної роботи № 3.

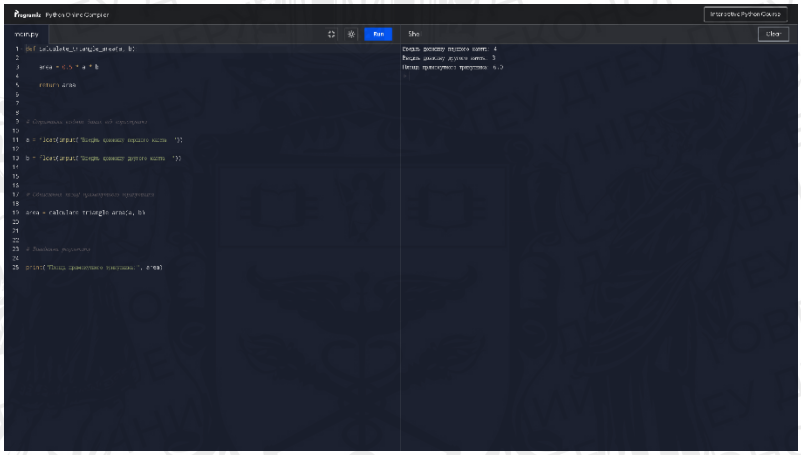
Ця програма може бути протестована за допомогою системи тестування, щоб перевірити правильність роботи функції **calculate_triangle_area** та коректність вводу та виводу даних.

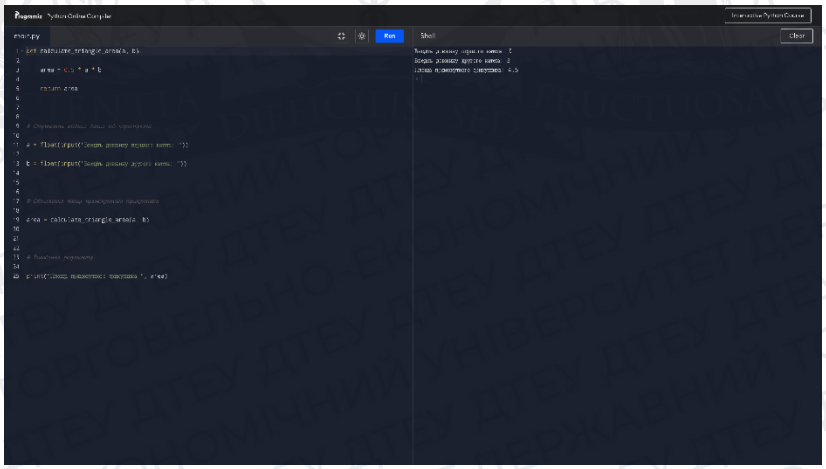
Можна написати такі тести:

1. Тест **test_calculate_triangle_area** для функції **calculate_triangle_area**:
   - Вхідні дані: довжини катетів a і b.
   - Очікуваний результат: обчислена площа прямокутного трикутника за формулою 0.5 * a * b.

- Перевірка, чи повертає функція **calculate_triangle_area** очікуваний результат для різних вхідних значень a і b.

2. Тест **test_input_output** для перевірки вводу та виводу даних:
   - Вхідні дані: довжини катетів a і b (симулюється введенням у програму).
   - Очікуваний результат: вивід обчисленої площі прямокутного трикутника.
   - Перевірка, чи відбувається правильний ввід та вивід даних.





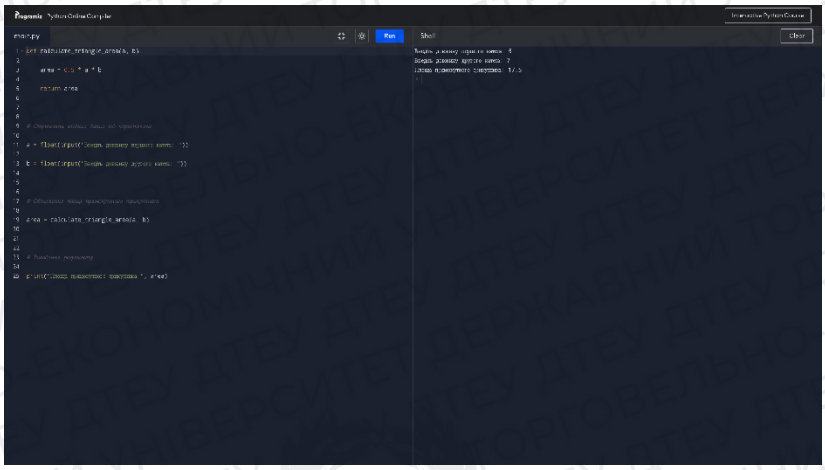


Рис. 3.4. Технологія використання системи тестів для ЛР № 3

Кожен тест може бути описаний у вигляді окремої функції, де перевіряються очікувані результати за допомогою оператора **assert**. Потім ці тести можуть бути запущені автоматично за допомогою системи тестування, наприклад, Pytest, для перевірки, чи працюють функції правильно.

Таким чином, система тестування допомагає забезпечити коректність роботи програми, перевіряє правильність обчислення площі прямокутного трикутника та забезпечує надійність вводу та виводу даних.

АС для лабораторної роботи № 8.

Ця програма використовує систему тестування для перевірки правильності роботи функцій **find_divisors** і **remove_extremes**, а також для перевірки коректності вводу та виводу даних.

У системі тестування можуть бути написані наступні тести:

1. Тест **test_find_divisors** для функції **find_divisors**:
   - Вхідні дані: ціле число N.
   - Очікуваний результат: масив додатніх дільників числа N.
   - Перевірка, чи повертає функція **find_divisors** очікуваний результат для різних вхідних значень N.
2. Тест **test_remove_extremes** для функції **remove_extremes**:
   - Вхідні дані: масив цілих чисел.

- Очікуваний результат: масив без найменшого і найбільшого елементів.
- Перевірка, чи повертає функція **remove_extremes** очікуваний результат для різних вхідних масивів.

3. Тест **test_input_output** для перевірки вводу та виводу даних:
   - Вхідні дані: ціле число N (симулюється введенням у програму).
   - Очікуваний результат: вивід масиву після видалення найменшого і найбільшого елементів.
   - Перевірка, чи відбувається правильний ввід та вивід даних.

Кожен тест може бути описаний у вигляді окремої функції, де перевіряються очікувані результати за допомогою оператора **assert**. Потім ці тести можуть бути запущені автоматично за допомогою системи тестування, наприклад, Pytest, для перевірки, чи працюють функції правильно.

Рис. 3.5. Технологія використання системи тестів для ЛР № 8

Таким чином, система тестування допомагає забезпечити коректність роботи програми, перевіряє правильність виконання функцій та забезпечує надійність вводу та виводу даних.

АС для лабораторної роботи № 9.

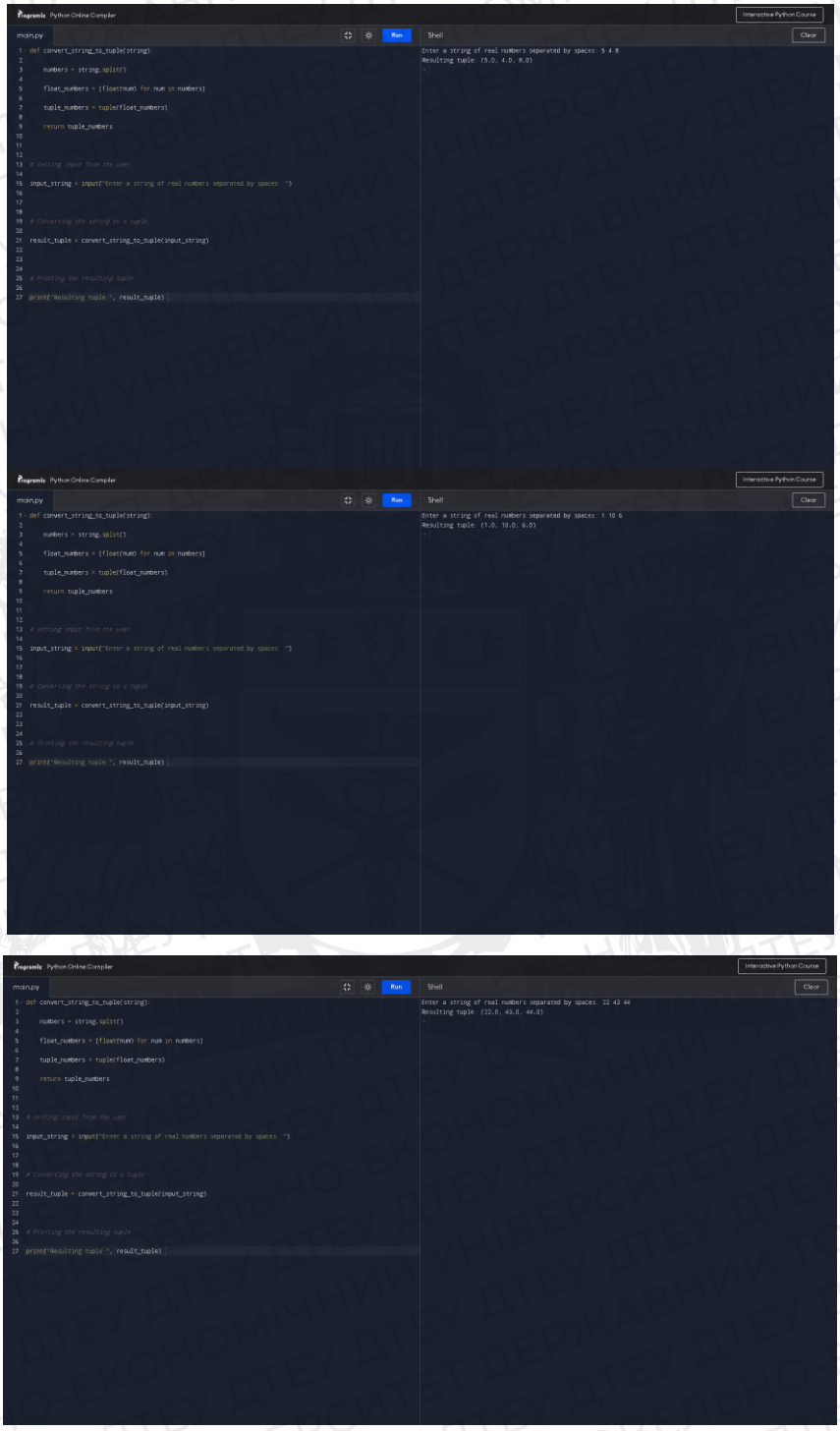


Рис. 3.6. Технологія використання системи тестів для ЛР № 9

Ця програма може бути протестована за допомогою системи тестування, щоб перевірити правильність роботи функції **convert_string_to_tuple** та коректність вводу та виводу даних.

Можна написати такі тести:

1. Тест **test_convert_string_to_tuple** для функції **convert_string_to_tuple**:
   - Вхідні дані: рядок дійсних чисел, записаних через пробіл.
   - Очікуваний результат: кортеж чисел, отриманий з рядка.
   - Перевірка, чи повертає функція **convert_string_to_tuple** очікуваний результат для різних вхідних рядків.
2. Тест **test_input_output** для перевірки вводу та виводу даних:
   - Вхідні дані: рядок дійсних чисел, записаних через пробіл (симулюється введенням у програму).
   - Очікуваний результат: вивід отриманого кортежу чисел.
   - Перевірка, чи відбувається правильний ввід та вивід даних.

Кожен тест може бути описаний у вигляді окремої функції, де перевіряються очікувані результати за допомогою оператора **assert**. Потім ці тести можуть бути запущені автоматично за допомогою системи тестування, наприклад, Pytest, для перевірки, чи працюють функції правильно.

Таким чином, система тестування допомагає забезпечити коректність роботи програми, перевіряє правильність перетворення рядка в кортеж та забезпечує надійність вводу та виводу даних.

ВИСНОВКИ

В результаті проведеного дослідження були здійснені наступні висновки:

1. Автоматизація тестування стала основним напрямком розвитку в останні роки. Вона дозволяє зменшити залежність від ручного тестування, прискорити процес тестування та знизити його вартість. Agile-методології розробки програмного забезпечення, такі як Scrum та Kanban, домінують у сучасній розробці. Вони вимагають гнучкості в процесі тестування, а DevOps об'єднує розробку та експлуатацію ПЗ, що має вплив на тестування. Continuous Integration/Continuous Delivery (CI/CD) дозволяє автоматично тестувати та постійно впроваджувати зміни в програмному забезпеченні. Це покращує швидкість виявлення проблем та негайну їх виправлення. Зростаюча загроза кібербезпеки робить тестування безпеки надзвичайно важливим етапом розробки ПЗ. Виявлення потенційних вразливостей та захист від атак стали пріоритетом. Мобільні додатки здобувають популярність, що вимагає специфічних підходів до тестування, включаючи тестування на різних платформах та пристроях. Хмарні та веб-орієнтовані додатки також потребують спеціалізованого підходу до тестування, зокрема у вимогах до сумісності, тестуванні залежностей від зовнішніх служб, масштабованості та безпеки. Усі ці тенденції впливають на розвиток тестування програмного забезпечення, роблять його більш автоматизованим, гнучким та спрямованим на забезпечення безпеки, якості та задоволення користувачів.

2. Тестування програм для лабораторних робіт з дисципліни "Вступ до комп'ютерних наук" включає перевірку різних аспектів роботи програм, пов'язаних з використанням системи контролю версій Git, мовою Python та пов'язаними з нею технологіями. Основні аспекти тестування включають перевірку правильності встановлення та налаштування Git, коректність введення і виведення даних у Python, роботу з функціями, математичними

операторами та бібліотеками, розгалуження та цикли, роботу зі списками, рядками, масивами, кортежами, словниками та множинами, обробку файлів, використання бібліотеки Pandas для обробки даних, об'єктно-орієнтоване програмування, збір даних з веб-документів, розробку віконних додатків та побудову графіків математичних функцій. Важливо перевірити, що програма правильно виконує всі вказані завдання, зберігає та відновлює дані, взаємодіє з обладнанням (у випадку потреби) та демонструє правильну логіку та функціональність згідно вимог лабораторних робіт. Конкретні вимоги та тестові сценарії можуть варіюватися залежно від кожної лабораторної роботи та тематики, але загальна мета полягає в перевірці правильності та функціональності програм у контексті вступу до комп'ютерних наук.

3. Тестування програмного забезпечення має різні типи, які виконуються на різних рівнях та спрямовані на перевірку різних аспектів функціональності, продуктивності та стійкості системи. Модульне тестування дозволяє перевірити окремі модулі або компоненти програми на наявність дефектів. Інтеграційне тестування перевіряє взаємодію між різними модулями або компонентами програми. Системне тестування виконується для перевірки програми в цілому, з фокусом на відповідність вимогам та специфікаціям. Автоматизоване тестування дозволяє зменшити ручну працю та прискорити процес виконання тестів. Функціональне тестування перевіряє, чи відповідає програмне забезпечення функціональним вимогам та правильно виконує свої функції. Навантажувальне тестування оцінює поведінку системи під великим навантаженням та допомагає виявити проблеми продуктивності та масштабованості. Використання різних типів тестування у комплексі допомагає забезпечити високу якість програмного забезпечення, виявити та виправити проблеми ранньою стадією розробки та забезпечити задоволення вимог користувачів. Кожен тип тестування має свої особливості та спрямований на перевірку конкретних аспектів програми, що робить їх важливими компонентами процесу розробки програмного забезпечення.

4. Розробка моделей тестів для перевірки правильності виконання лабораторних робіт на мові Python є важливим етапом процесу тестування. Ці моделі допомагають визначити критерії правильності виконання завдань і забезпечити об'єктивну перевірку програм. Детерміновані моделі тестів передбачають використання відомих вхідних даних та очікуваних результатів для порівняння з фактичними результатами виконання програми. Генеративні моделі тестів створюють випадкові вхідні дані і перевіряють результати програми для різних сценаріїв.

Розробка моделей тестів на основі цих підходів дозволяє створити комплексну систему тестування, яка перевіряє різні аспекти правильності виконання програми на мові Python. Комбінація різних моделей дозволяє досягти широкого покриття тестування і забезпечити надійність та якість програмного продукту.

5. Для конкретних лабораторних робіт (№ 3, 8 та 9) були розроблені відповідні тести, що перевірятимуть правильність роботи функцій, коректність вводу та виводу даних. Тести описані у вигляді окремих функцій з використанням оператора assert, і їх можна автоматично запускати за допомогою системи тестування, такої як Pytest, Unittest або Robot Framework. Для лабораторної роботи №3, 8 та 9, що включать функції для обробки даних, були створені тести, що перевіряють правильність обробки вхідних даних та очікуваний результат. Наприклад, для функції, що виконує певні обчислення, був написаний тест, який перевіряє, чи повертає функція правильний результат при певних вхідних даних. Ці тести можна запустити автоматично за допомогою системи тестування. Наприклад, у випадку використання Pytest, необхідно створити окремий файл з тестами, де розмістити всі тести для лабораторної роботи №3, та запустити його за допомогою команди pytest в командному рядку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Базурін В.М. Методичні рекомендації до лабораторних занять з дисципліни «Вступ до комп’ютерних наук» / В.М. Базурін // Київ, Державний торговельно-економічний університет. – 2022. – 152 с.

2. Інтегроване середовище розробки Eclipse [Електронний ресурс] // Офіційний сайт Eclipse Foundation – Режим доступу: https://www.eclipse.org

3. Інформатика та програмування // Київський національний університет імені Тараса Шевченка. [Електронний ресурс]. – Режим доступу: http://www.matfiz.univ.kiev.ua/userfiles/files/Pres30.pdf

4. Костюченко А.О. Основи програмування мовою Python: навчальний посібник / А.О. Костюченко // Ч.: ФОП Баликіна С.М., 2020 – 180 с.

5. Креневич А.П. Python у прикладах і задачах. Частина 1. Структурне програмування / А.П. Креневич // Навчальний посібник із дисципліни "Інформатика та програмування". – К.: ВПЦ "Київський Університет", 2017 – 206 с.

6. Мова програмування Python для інженерів і науковців [Електронний ресурс]. – Режим доступу: https://vkopey.github.io/Python-for-engineers-and-scientists/

7. Мокін, Б. І. М Навчальний посібник для опанування студентами способів розв’язання задач з функціонального аналізу мовою Python. Частина 1 / Б. І. Мокін, В. Б. Мокін, О. Б. Мокін. – Вінниця : ВНТУ, 2022. – 124 с.

8. Основи програмування. Python. Частина 1 [Електронний ресурс]: підручник для студ. спеціальності 122 "Комп’ютерні науки", спеціалізації "Інформаційні технології в біології та медицині" / А. В. Яковенко ; КПІ ім. Ігоря Сікорського. – Електронні текстові данні (1 файл: 1,59 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 195 с.

9. Пакет регресійних тестів для Python. [Електронний ресурс]. – Режим доступу: https://docs.python.org/uk/3/library/test.html

10.Практикум з програмування мовою Python. [Електронний ресурс]. – Режим доступу: https://pythonexercises.rozh2sch.org.ua/

11.Програмування на мові Python (3.x). Початковий курс. [Електронний ресурс]. – Режим доступу: https://sites.google.com/site/pythonukr/

12.Програмування числових методів мовою Python : підруч. / А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий ; за ред. А. В. Анісімова. – К. : Видавничо-поліграфічний центр "Київський університет", 2014. – 640 с.

13.Процеси та системи підтримки якості програмних систем. [Електронний ресурс]. – Режим доступу: https://dspace.uzhnu.edu.ua/jspui/bitstream/lib/16456/1/Процеси%20та%20системи%20підтримки%20якості%20програмних%20систем.pdf

14.Яковенко А. Основи програмування: методичні вказівки до виконання комп'ютерних практикумів з дисципліни "Основи програмування". Основи програмування мовою Python / А. В. Яковенко. − Київ : НТУУ "КПІ ім. І. Сікорського", 2017. − 87 с.

15.An all-in-one test automation solution [Електронний ресурс] – Режим доступу до ресурсу: https://www.katalon.com/

16.Automation Testing Tutorial: What is Automated Testing. [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/automation-testing.html

17.Burns D. Selenium 2 Testing Tools: Beginner's Guide / Burns D. – Birmingham: Packt Publishing, 2012. – 437 с.

18.Complete Guide to Test Automation: Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects [Електронний ресурс] – Режим доступу до ресурсу: http://tisten.ir/wp-content/uploads/2019/01/Complete-Guide-to-TestAutomation-Techniques-Practices-and-Patterns-for-Building-andMaintaining-Effective-Software-Projects-Apress-2018-Arnon-Axelrod.pdf

19.Driver requirements Maven [Електронний ресурс] – Режим доступу до ресурсу: https://www.selenium.dev/documentation/en/webdriver/driver_requirements/

20.Eclipse IDE for Java Developers, Eclipse Packages [Electronic source]. – Access mode: https://www.eclipse.org/downloads/packages/release/oxygen/3a/eclipse-ide-javadevelopers

21.Installing Selenium libraries [Електронний ресурс] – Режим доступу до ресурсу: https://www.selenium.dev/documentation/en/selenium_installation/installing_ selenium_libraries/

22.IntelliJ IDEA [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/ru-ru/idea/

23.Oracle Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.oracle.com/java/

24.Python 3.8.0 documentation. [Електронний ресурс]. – Режим доступу: https://docs.python.org/3

25.Selenium Webdriver [Електронний ресурс] – Режим доступу до ресурсу: https://drive.google.com/file/d/0B7TBmsv_w76nQ1FfQVdyWWRXYk0/view

26.Selenium with Java : Getting Started to Run Automated Tests [Електронний ресурс] – Режим доступу до ресурсу: https://www.browserstack.com/guide/selenium-with-java-for-automated-test

27.Strategies for Testing Web Applications from the Client Side [Електронний ресурс] – Режим доступу до ресурсу: https://cs.fit.edu/media/TechnicalReports/cs-2003-11.pdf

28.Top 10 Automation Testing Tools in 2021 [Електронний ресурс] – Режим доступу до ресурсу: https://www.netsolutions.com/insights/top-10-automation-testing-tools/

29.Welcome to Apache Maven [Електронний ресурс] – Режим доступу до ресурсу: https://maven.apache.org/

30.What is automation testing? [Electronic source]. – Access mode: https://www.globalapptesting.com/blog/what-is-automation-testing