МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА»
Кафедра кібербезпеки та математичного моделювання

ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт
для здобувачів
першого (бакалаврського) рівня вищої освіти
освітньо-професійної програми «Кібербезпека»
спеціальності 125 Кібербезпека та захист інформації

Обговорено і рекомендовано
на засіданні кафедри
Кібербезпеки та математичного
моделювання
Протокол №2
від 13 лютого 2024 р.

Чернігів 2024

Технології програмування. Методичні вказівки до виконання лабораторних робіт для здобувачів першого (бакалаврського) рівня вищої освіти освітньо-професійної програми «Кібербезпека» спеціальності 125 Кібербезпека та захист інформації. – Чернігів: НУ «Чернігівська політехніка», 2024 – 43 с.

Укладачі: ДЮБА ІГОР МИКОЛАЙОВИЧ, викладач кафедри кібербезпеки та математичного моделювання;
СЕМЕНДЯЙ СЕРГІЙ МАТВІЙОВИЧ, старший викладач кафедри кібербезпеки та математичного моделювання
ГРЕБЕННИК АЛЛА ГРИГОРІВНА, старший викладач кафедри кібербезпеки та математичного моделювання;

Відповідальний за випуск – ТКАЧ ЮЛІЯ МИКОЛАЇВНА,
завідувач кафедри кібербезпеки та математичного моделювання, доктор педагогічних наук, професор

Рецензент – ПЕТРЕНКО ТАРАС АНАТОЛІЙОВИЧ,
доцент кафедри кібербезпеки та математичного моделювання, кандидат технічних наук

ЗМІСТ

1 Вступ

Курс лабораторних робіт має на меті сформувати розуміння сучасних парадигм програмування під час програмної реалізації завдань. В процесі виконання здобувачі вищої освіти ознайомляться з принципами набуття теоретичних знань в галузі нової мови програмування, застосування теоретичних знаннь на практиці, проведення поверхневого аналізу завдання та розбивку на під задачі, принципами вибору між існуючими варіантами для вирішення завдання, перевірки результатів роботи.

Мова програмування Python вибрана для цього курсу завдяки своїй легкості використання та швидкому розгортанню. Python – це універсальна мова, що широко використовується в усьому світі для самих різних цілей. Серед переваг мови Python можна виділити переносимість написаних програм, на комп'ютери різної архітектури та з різними операційними системами, лаконічність запису алгоритмів, можливість отримати ефективний код програм за швидкістю виконання. Теми включають базові поняття мови, алгоритмічні структури, функціональне програмування, робота з винятковими ситуаціями, використання бібліотек, написання тестів власного коду, роботу з файлами та базові поняття об'єктно орієнтованого програмування.

Серед головних цілей цього курсу отримання практичного досвіду здобувачів вищої освіти в опанування нової мови програмування, виділення типових підходів для опанування нової мови програмування та використання досвіду для вивчення нових мов програмування в майбутньому.

Важливим аспектом курсу є набуття практичного досвіду в читанні чужого коду, що є основною роботу в промисловому програмуванні. Читання чужого коду дозволяє розвивати абстрактне мислення, оскільки під час ознайомлення формується уявлення про те, що саме виконує та чи інша програмна реалізація. Читання чужого коду дозволяє виділити певні ідеї для використання у власних програмних реалізаціях. При влаштування на роботу молодим спеціалістам в першу чергу пропонується ознайомитись з існуючим кодом для подальшої модифікації чи виправлення помилок.

2 Лабораторна робота №1: Списки. Словники. Кортежі

Мета роботи: Використовуючи теоретичне підґрунтя про складні структури даних Списки, Словники Кортежі, та використовуючи існуючий код, доробити програму додавши функціонал, що буде вказано в завданні до лабораторної роботи.

2.1 Теоретичні відомості

2.1.1 Списки

Масив – набір фіксованої кількості елементів, що розміщені в пам'яті комп'ютера безпосередньо один за одним, а доступ до них здійснюється за індексом (номер даного елементу в масиві).

В Python для реалізації масиву використовуються списки. Список – тип даних, що представляє собою послідовність певних значень, що можуть повторюватись. Але на відміну від масиву – кількість елементів у списку може бути довільною.

Списки – структура даних, що може містити елементи різних типів, що перераховані через кому та заключені в квадратні дужки.

Списки служать для того, щоб зберігати об'єкти в певному порядку, особливо якщо порядок або вміст можуть змінюватися. Можна змінювати список, додати в нього нові елементи, а також видалити або перезаписати існуючі. Можна змінити кількість елементів у списку, а також самі елементи. Одне і те ж значення може зустрічатися в списку кілька разів.

Приклад визначення списку:

```
list_num = ["1", "2", "3"]
print(list_num)
list_str = ["aa", "bb", "cc"]
print(list_str)
```

Результат

```
['1', '2', '3']
['aa', 'bb', 'cc']
```

Крім того, за допомогою функції list() можна створити порожній список

Список містить різні дані, звертатися до яких можна через ім'я списку та вказавши зміщення необхідного елементу

```
students = ["Ihor", "Dima", "Serge"]
print(students)
print(students[0])
```

Результат

```
['Ihor', 'Dima', 'Serge']
Ihor
```

Використовуючи методи списку можна виконувати необхідні операції. Для додавання елементів в кінець списку – використовують метод **append()**. Можна об'єднати один список з іншим за допомогою методу **extend()**. Функція **append()** додає елементи тільки в кінець списку. Коли потрібно додати елемент в задану позицію, використовується функція **insert().**

За допомогою функції **pop()** можна отримати елемент зі списку і в той же час видалити його. Якщо викликати функцію **pop()** і вказати зсув, вона поверне елемент, що знаходиться в заданій позиції. Якщо аргумент не вказано – буде використано значення -1. Так, виклик pop(0) поверне головний (початковий) елемент списку, а виклик pop() або pop(-1) – кінцевий елемент.

Для проходження по списку використовуються цикли

```
students = ["Ihor", "Dima", "Serge"]
for name in students:
    print(name)
```

Результат

```
Ihor
Dima
Serge
```

2.1.2 Словники

Словник дуже схожий на список, але порядок елементів в ньому не має значення, і вони вибираються не за допомогою зміщення. Замість цього для кожного значення вказується пов'язаний з ним унікальний ключ. Таким ключем може бути об'єкт одного з незмінних типів: рядок, булева змінна, ціле число, число з плаваючою точкою, кортеж і іншими об'єктами. Елементи словника можуть містити об'єкти довільного типу даних і мати необмежений рівень вкладеності. Елементи в словниках розташовуються в довільному порядку.

Словники можна змінювати – це означає, що можна додати, видалити і змінити їх елементи, які мають вигляд "ключ – значення"

Щоб створити словник, необхідно заключити в фігурні дужки ({}) розділені комами пари ключ: значення.

```
animals = {
    "dog" : 4,
    "cat" : 4,
    "goose": 2
}

print(animals)
```

Результат

```
{'dog': 4, 'cat': 4, 'goose': 2}
```

Можна використовувати функцію **dict**(), щоб створити порожній словник, якщо не вказати параметри функції

Звернення до елементів словника здійснюється за допомогою квадратних дужок, в яких вказується ключ.

```
animals = {
    "dog" : 4,
    "cat" : 4,
    "goose": 2
}
print(animals["dog"])
```

Результат

```
4
```

Щоб дізнатися, чи міститься в словнику якийсь ключ, використовується ключове слово **in**. Якщо ключ знайдений, то повертається значення True, в іншому випадку – **False**.

```
animals = {
    "dog" : 4,
    "cat" : 4,
    "goose": 2
}

print("cat" in animals)
```

Оскільки словники відносяться до змінюваних типів даних, то можна додати або змінити елемент по ключу. Додати елемент в словник досить легко. Потрібно просто звернутися до елементу по його ключу і привласнити йому значення. Якщо ключ вже існує в словнику, наявне значення буде замінено новим. Якщо ключ новий, він і вказане значення будуть додані в словник.

Для словників розроблено набір методів. **update()** – додає елементи в словник. Метод змінює поточний словник і нічого не повертає.

Видалити елемент зі словника можна за допомогою інструкції **del**.

```
dict_2 = {"a": 1, "b": 2}
print(dict_2)
del dict_2 ["b"] # Видаляємо елемент з ключем "b"
print(dict_2)
```

Результат:

```
{'a': 1, 'b': 2}
{'a': 1}
```

Щоб видалити всі ключі і значення зі словника, слід використовувати функцію **clear()** або просто привласнити порожній словник заданому імені.

Скориставшись функцією **keys()** можна отримати всі ключі словника. Щоб отримати всі значення словника, використовується функція **values().** Щоб отримати всі пари "ключ – значення" із словника, використовується функція **items**().

```
testDisct = {"a": 1, "b": 2}
print(testDisct.keys())
print(testDisct.values())
print(testDisct.items())
```

Результат:

```
dict_keys(['a', 'b'])
dict_values([1, 2])
dict_items([('a', 1), ('b', 2)])
```

2.1.3 Кортежі

Кортежі, як і списки, є послідовностями довільних елементів. На відміну від списків кортежі незмінні.

Всі операції над списками, що не змінюють список (додавання, множення на число, функції index() і count() і деякі інші операції) можна застосовувати до кортежів. Можна також по-різному змінювати елементи місцями і так далі.

Щоб створити порожній кортеж використовується оператор ().

```
xy = (12, 21)
print(xy)
```

Результат

```
(12, 21)
```

Функція перетворення tuple() створює кортежі з інших об’єктів

```
students = ['Alex', 'Helen', 'Olga']
print(students)
tuple_students = tuple(students)
print(tuple_students)
```

Результат

```
['Alex', 'Helen', 'Olga']
('Alex', 'Helen', 'Olga')
```

2.2 Початковий текст програми

Для виконання лабораторної роботи надається початковий текст програми який необхідно скопіювати та зберегти у файл **lab_01.py.** Виконання всіх завдань має виконуватись з використанням наведеного прикладу

```
## List [Item1, Item2, Item3]
## Item {"name":"Jon", "phone":"0631234567"}

# already sorted list
list = [
    {"name":"Bob", "phone":"0631234567"},
    {"name":"Emma", "phone":"0631234567"},
    {"name":"Jon",  "phone":"0631234567"},
    {"name":"Zak",  "phone":"0631234567"}
]

def printAllList():
    for elem in list:
        strForPrint = "Student name is " + elem["name"] + ",  Phone is
" + elem["phone"]
        print(strForPrint)
    return

def addNewElement():
    name = input("Pease enter student name: ")
    phone = input("Please enter student phone: ")
    newItem = {"name": name, "phone": phone}
    # find insert position
    insertPosition = 0
    for item in list:
        if name > item["name"]:
            insertPosition += 1
        else:
            break
    list.insert(insertPosition, newItem)
    print("New element has been added")
    return

def deleteElement():
    name = input("Please enter name to be delated: ")
    deletePosition = -1
    for item in list:
        if name == item["name"]:
            deletePosition = list.index(item)
            break
    if deletePosition == -1:
        print("Element was not found")
    else:
        print("Dele position " + str(deletePosition))
        # list.pop(deletePosition)
        del list[deletePosition]
    return
```

```
def updateElement():
    name = input("Please enter name to be updated: ")
    # implementation required

def main():
    while True:
        chouse = input("Please specify the action [ C create, U update,
D delete, P print,  X exit ] ")
        match chouse:
            case "C" | "c":
                print("New element will be created:")
                addNewElement()
                printAllList()
            case "U" | "u":
                print("Existing element will be updated")
            case "D" | "d":
                print("Element will be deleted")
                deleteElement()
            case "P" | "p":
                print("List will be printed")
                printAllList()
            case "X" | "x":
                print("Exit()")
                break
            case _:
                print("Wrong chouse")


main()
```

2.3 Завдання до лабораторної роботи

Реалізувати **відсортований** телефонний довідник студентів групи.

Для виконання задання надано частину готового функціоналу, яка розміщена в пункті 1.2. Наведений приклад необхідно зберегти на локольному комп'ютері під назвою **lab_01.py.**

Частина готового функціоналу реалізує безкінечний цикл запитів до користувача. Типи запитів: додати нового студента, змінити данні про існуючого студента, видалити запис, роздрукувати всю таблицю та вихід із програми. Розроблено функціонал додавання нового запису та видалення існуючого. Всі дії відбуваються з **відсортованим** списком студентів. Перед виконанням роботи слід ознайомитись з існуючим функціоналом.

В процесі виконання лабораторної роботи необхідно виконати наступні доопрацювання:

1. Розширити відомості про студента до 4х полів. На даний час використовується лише два поля (name та phone).
2. Переробити існуючий функціонал враховуючи розширення відомості про студента до 4х полів.

3. Реалізувати з нуля функціонал зміни інформації про студента враховуючи той факт, що вже існує реалізація додавання нового запису та видалення існуючого. **При зміні інформації про студента список має залишатись відсортованим**.

Текст програми разом зі звітом розмістити в каталозі lab_01. Каталог lab_01 розмістити в каталозі, що використовується для виконання практичних завдань по кожній лекції та має назву **ТР-КВ-22[1 or 2]-Name-Surname**.

3 Лабораторна робота №2: Робота з файлами. Юніт тести

Мета роботи: Використовуючи теоретичне підґрунтя про роботу з файлами та тестування коду у мові Python розширити програму телефонного довідника студентів додавши функціонал, що буде вказано в завданні до лабораторної роботи.

3.1 Теоретичні відомості

3.1.1 Параметри командного рядка

Одним із механізмом визначення параметрів необхідних для виконання програми – використання аргументів командного рядка. Для можливості використання аргументів командного необхідно підключити модуль **argv**

Розглянемо приклад:

```
from sys import argv

print(f"Script name: {argv[0]}")
print(f"Input parameter: {argv[1]}")
```

В самому початку підключаються модуль **argv**, зо забезпечує можливість використання параметрів командного рядка. Все, що було вказано користувачем під час запуску програми зберігається в список доступ до елементів якого здійснюється використовуючи індекси.

Запустивши програму на виконання наступним чином:

```
python lab2.py lab2.csv
```

Отримаємо результат:

```
Script name: lab2.py
Input parameter: lab2.csv
```

Під нульовим індексом завжди зберігається ім’я програми, яка запускається. Починаючи з індексу один розміщуються параметри які буди вказані під час запуску програми на виконання.

3.1.2 Робота з файлами

Для роботи з файлами існують набір стандартних функцій, тож для відкриття, закриття, читання та запису інформації достатньо використовувати готовий функціонал.

Для того, щоб відкрити файл необхідно виконати функцію open() (https://docs.python.org/3/library/functions.html#open). Функція повертає посилання на файл, яке можна використовувати для наступних маніпуляцій. При закінченні використання файлу, його необхідно закрити.

```
file_name = argv[1]
file = open(file_name, "r")

file.close()
```

Проте, використовуючи ключове слово **with** можна не використовувати **close()** в кінці використання файлу.

Після відкриття файлу існує можливість пройти всі рядки, що в ньому збережені. Розглянемо приклад:

```
file_name = argv[1]
with open(file_name, "r") as file:
    for line in file:
        print(f"line from file: {line}")
```

Отримаємо результат

```
line from file: Name,Phone
line from file: Bob,1112233
line from file: Dilan,2223344
line from file: Zak,3334455
```

Для запису даних в файл використовується функція **write()**

```
with open(file_name, "a") as file:
    file.write("New student name,1231213")
```

Для роботи з файлами формату **CSV** розроблено однойменний модуль, який одразу може сформувати словники з вхідних даних, що збережені в форматі CSV (https://docs.python.org/3/library/csv.html).

Функція **DictReader**() забезпечу читання файлу у вигляді словників.

Розглянемо приклад:

```
file_name = argv[1]

students = []

with open(file_name) as file:
    reader = csv.DictReader(file)
    for row in reader:
        students.append({"Name":row["Name"], "Phone":row["Phone"]})

print(students)
```

Отримаємо результат

```
[{'Name': 'Bob', 'Phone': '1112233'}, {'Name': 'Dilan', 'Phone': '2223344'}, {'Name': 'Zak', 'Phone': '3334455'}]
```

Для запису даних в CSV файл необхідно виконати дещо більше дій: відкрити файл для запису, створити обект класу **DictWriter**(), записати верхній рівень з назвою стовбців та безпосередньо записати данні в файл.

```
import csv
with open("lab2_out.csv", "w", newline='') as csvfile:
    fieldnames = ["Name", "Age"]
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerow({'Name': 'Ihor', 'Age': '37'})
```

3.1.3 Юніт тести

Процес виконання завдання може включати розбиття великої задачі на маленькі під задачі, при цьому результатом виконання маленької частина має бути певна кількість функцій, сумарний результат виконання яких задовольняє критеріям виділеної під задачі. Для того, щоб впевнитись, що написаний код виконує саме те, що вказано в завданні, розробник формує набір тестів яким піддається написаний в рамках виконання завдання код. **Тестування коду – написання коду для тестування коду.** При цьому поняття Юніт тест означає тестування окремого функціоналу, наприклад однієї функції (юніту).

Для мові Python існує стороння бібліотека **pytest**, яка реалізує механізм написання тестів. Детальний опис бібліотеки доступний за посиланням https://docs.pytest.org/en/stable/

Для початку використання бібліотеки необхідно виконати інсталяцію

```
pip install pytest
```

Розглянемо в якості прикладу завдання на написання програми калькулятор, що запитує у користувача два параметри над якими необхідно виконати дії. В даному прикладі операції додавання та множення винесені в окремі функції. Саме ці функціями будуть виступати окремими юнітами для тестування.

```
def add(a, b):
    return a + b

def mul(a, b):
    return a * b

def main():
    a = int(input("What is a: "))
```

```
    b = int(input("What is b: "))
    print(add(a, b))
    print(mul(a, b))

if __name__ == "__main__":
    main()
```

Перед початком написання тестів необхідно відмітити існування ключового слова **assert**, що надає можливість перевірити на правдивість вказану умову. Також необхідно вказати, згідно конвенції назва юніт тесту має починатися зі слова **test_**

Наведемо приклад файлу з тестами:

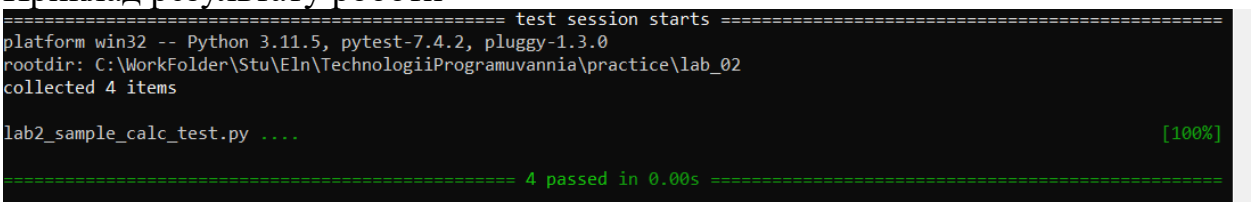
```
from lab2_sample_calc import add
from lab2_sample_calc import mul

def test_add():
    assert add(2, 2) == 4
    assert add(3, 2) == 5
    assert add(4, 2) == 6

def test_mul_positive_both():
    assert mul(2, 2) == 4
    assert mul(3, 2) == 6
    assert mul(4, 2) == 8

def test_mul_positive_and_negative():
    assert mul(2, -2) == -4
    assert mul(-3, 2) == -6
    assert mul(4, -2) == -8

def test_mul_negative_both():
    assert mul(-2, -2) == 4
    assert mul(-3, -2) == 6
    assert mul(-4, -2) == 8
```

З самого початку необхідно підключити файл і функції які будуть піддаватись тестуванню. Далі відбувається тестування Юніту додавання та юніту множення. Зверніть увагу, що для юніту множення розроблено три незалежних тести.

Для запуску тестів використовується наступна команда:

```
pytest lab2_sample_calc_test.py
```

Приклад результату роботи



3.2 Завдання до лабораторної роботи

Перед виконанням роботи слід ознайомитись з прикладами.

Розширити функціональність телефонного довідника студентів групи, що був розроблений у Лабораторній роботі №1 наступними можливостями:

1. Завантаження початкових данні для довідника з csv файлу. Ім’я файлу для завантаження даних повинно передаватись в якості параметрів командного рядку.
2. Зберігання довідника студентів у форматі CSV перед виходом із програми.
3. Покриття основного функціоналу програми (додавання нового запису, видалення існуючого запису, зміна інформації, завантаження з файлу та зберігання в файл) **Юніт тестами**.

Текст програми разом зі звітом розмістити в директорії lab_02. Директорію lab_02 розмістити в директорії, що використовується для виконання практичних завдань по кожній лекції та має назву **TP-KB-22[1 or 2]-Name-Surname**.

4 Лабораторна робота №3: ООП

Мета роботи: Використовуючи теоретичне підґрунтя про ООП у мові Python переробити програму телефонного довідника студентів використовуючи принципи ООП для формування відомостей про студентів.

4.1 Теоретичні відомості

4.1.1 ООП та мова Python

Python має безліч вбудованих типів, наприклад, int, str і так далі, які ми можемо використовувати у програмі. Але Python також дозволяє визначати власні типи за допомогою класів. Клас є деякою сутністю. Конкретним здійсненням класу є об'єкт.

Можна ще провести таку аналогію. У нас у всіх є деяке уявлення про людину, яка має ім'я, вік, якісь інші характеристики Людина може виконувати деякі дії - ходити, бігати, думати і т.д. Тобто це уявлення, яке включає набір характеристик та дій, можна назвати класом. Конкретне втілення цього шаблону може відрізнятися, наприклад, одні мають одне ім'я, інші - інше ім'я. І реально існуюча людина представлятиме об'єкт цього класу.

Клас визначається за допомогою ключового слова:

```
class class_name:
    attributes
    methods
```

Усередині класу визначаються його атрибути, які зберігають різні характеристики класу, та методи – функції класу.

Приклад найпростішого класу:

```
class Person:
    pass
```

У разі визначено клас Person, який умовно представляє людини. В даному випадку в класі не визначається жодних методів чи атрибутів. Однак оскільки в ньому має бути щось визначено, то як замінник функціоналу класу застосовується оператор **pass**. Цей оператор застосовується, коли синтаксично необхідно визначити певний код, проте ми не хочемо його, і замість конкретного коду вставляємо оператор **pass**.

Після створення класу, можна визначити об'єкти цього класу. Наприклад:

```
class Person:
    pass

tom = Person()
bob = Person()
```

Після визначення класу Person створюються два об'єкти класу Person – tom і bob. Для створення об'єкта застосовується спеціальна функція – конструктор, яка називається як ім'я класу і яка повертає об'єкт класу. Тобто у цьому випадку виклик Person() представляє виклик конструктора. Кожен клас за замовчуванням має конструктор без параметрів.

4.1.2 Методи класу

Методи класу фактично представляють функції, які визначені всередині класу і які визначають його поведінку. Наприклад, визначимо клас Person з одним методом:

```
class Person:
     def say_hello(self):
        print("Hello")

tom = Person()
tom.say_hello()
```

Тут визначено метод say_hello(), який умовно виконує вітання – виводить рядок на консоль. При визначенні методів будь-якого класу слід враховувати, що всі вони повинні приймати як перший параметр посилання на поточний об'єкт, який відповідно до умов називається self. Через це посилання всередині класу ми можемо звернутися до функціональності об'єкта. Але при самому виклик методу цей параметр не враховується.

Використовуючи ім'я об'єкта, ми можемо звернутися до його способів. Для звернення до методів застосовується нотація точки – після імені об'єкта ставиться точка і після неї йде виклик методу. Наприклад, звернення до методу say_hello() для виведення привітання на консоль:

```
tom.say_hello()
```

У результаті ця програма виведе на консоль рядок "Hello".

Якщо метод повинен приймати інші параметри, вони визначаються після параметра self, і за виклику подібного методу їм необхідно передати значення:

```
class Person:
    def say(self, message):
        print(message)


tom = Person()
tom.say("Hello, World!")
```

Тут визначено метод say(). Він приймає два параметри: self і message. І другим параметром - message при виклику методу необхідно передати значення.

4.1.3 self

Через ключове слово self можна звертатися всередині класу до функціональності поточного об'єкта. Наприклад, визначимо два методи у класі Person:

```
class Person:

    def say(self, message):
        print(message)

    def say_hello(self):
        self.say("Hello, world")


tom = Person()
tom.say_hello()
```

Тут в одному методі - say_hello() викликається інший метод - say(). Оскільки метод say() приймає крім self ще параметри (параметр message), то за виклику методу цього параметра передається значення.

4.1.4 Конструктори

Для створення класу об'єкта використовується конструктор. Так, вище коли ми створювали об'єкти класу Person, ми використовували за замовчуванням конструктор, який не приймає параметрів і який неявно мають всі класи:

```
tom = Person()
```

Однак ми можемо явно визначити в класах конструктор за допомогою спеціального методу, який називається **__init__()** (по два прочерки з кожної сторони). Наприклад, змінимо клас Person, додавши до нього конструктор:

```
class Person:
    # конструктор
    def __init__(self):
        print("Person creating")

    def say_hello(self):
        print("Hello")
```

```
tom = Person()
tom.say_hello()
```

Отже, тут у коді класу Person визначено конструктор та метод say_hello(). Як перший параметр конструктор, як і методи, також приймає посилання на поточний об'єкт - self. Зазвичай конструктори застосовуються визначення дій, які мають здійснюватися під час створення об'єкта.

Тепер під час створення об'єкта буде здійснено виклик конструктора **__init__()** з класу Person, який виведе на консоль рядок " Person creating ".

4.1.5 Атрибути об'єкту

Атрибути зберігають стан об'єкта. Для визначення та встановлення атрибутів усередині класу можна використовувати слово self. Наприклад, визначимо наступний клас Person:

```
class Person:
    def __init__(self, name):
        self.name = name
        self.age = 1

tom = Person("Tom")

print(tom.name)
print(tom.age)
# зміна значення
tom.age = 37
print(tom.age)
```

Тепер конструктор класу Person приймає ще один параметр – name. Через цей параметр в конструктор буде передаватися ім'я людини, що створюється. Усередині конструктора встановлюються два атрибути - name і age (умовно ім'я та вік людини).

Якщо ми визначили у класі конструктор __init__, ми вже не зможемо викликати конструктор за замовчуванням. Тепер нам треба викликати наш явним чином оподаткований конструктор __init__, який необхідно передати значення для параметра name:

Далі на ім'я об'єкта ми можемо звертатися до атрибутів об'єкта - отримувати та змінювати їх значення:

```
print(tom.name)
tom.age = 37
```

Для звернення до атрибутів об'єкта всередині класу у його методах також застосовується слово self.

4.1.6 Створення об'єктів

Кількість об'єктів, що може бути створена – необмежена.

```
class Person:

    def __init__(self, name):
        self.name = name
        self.age = 1

    def display_info(self):
        print(f"Name: {self.name}  Age: {self.age}")


tom = Person("Tom")
tom.age = 37
tom.display_info()      # Name: Tom  Age: 37

bob = Person("Bob")
bob.age = 41
bob.display_info()      # Name: Bob  Age: 41
```

Тут створюються два об'єкти класу Person: tom та bob. Вони відповідають визначенню класу Person, мають однаковий набір атрибутів та методів, проте їхній стан відрізнятиметься. При виконанні програми Python динамічно визначатиме self - він представляє об'єкт, у якого викликається метод.

4.2 Завдання до лабораторної роботи

Переробити функціональність телефонного довідника студентів групи, що був розроблений у Лабораторній роботі №2 використовуючи принципи ООП:

1. Розробити клас Студент групи з відповідними атрибутами.
2. Розробити клас Список групи, має містити не словники, як виконано в лабораторній роботі №2, а об’єкти класу Студент групи; додавання нового запису, видаленні існуючого чи зміна даних має бути виконана через методи класу Список групи.
3. Розробити клас для роботи з файлами для зчитування початкової інформації про список групи та збереження інформації по завершенню програми.
4. Список студентів має містити не словники, як виконано в лабораторній роботі №2, а об’єкти класу Студент групи.
5. Описання всіх класів мають міститися в окремих файлах, що мають відповідні імена(наприклад Studen, StudentList, Utils).
6. Основний функціонал програми має бути покритий Юніт тестами.

Текст програми разом зі звітом розмістити в директорії lab_03. Директорію lab_03 розмістити в директорії, що використовується для виконання практичних завдань по кожній лекції та має назву **TP-KB-22[1 or 2]-Name-Surname**.

5 Лабораторна робота №4: Зворотний польський запис

Мета роботи: Використовуючи теоретичне підґрунтя про зворотний польський запис розробити програму на вхід якої подається математичний вираз, що має довільний набір операндів, операторів та дужок, на виході програма обчислює результат математичного виразу.

5.1 Теоретичні відомості.

5.1.1 Зворотний польський запис.

Звичною формою запису виразів є інфіксна, коли знак бінарної операції записують між позначеннями операндів цієї операції, наприклад, a + b. Розглянемо запис знаків операцій після позначень операндів, тобто постфіксний запис, наприклад, a b +. Такий запис має також назву зворотного польського, бо його запропонував польський логік Ян Лукасевич. Далі словосполучення: «зворотний польський запис» позначатимемо ЗПЗ. Позначення для функції традиційно записують перед аргументами. Природно такий запис назвати префіксним. При описі ЗПЗ переважно обмежуються перетворенням інфіксного запису у ЗПЗ.

Зворотний польський запис має чудові властивості, які перетворюють її на ідеальну проміжну ланку при трансляції коду програми.

Обчислення виразу, записаного в зворотному польському записі, можна проводити шляхом однократного перегляду ЗПЗ.

Зворотний польський запис виразу з арифметичними діями та піднесенням до степеня можна отримати, дотримуючись алгоритму, запропонованого Дейкстрою. Алгоритм отримав назву «сортувальна станція», за подібність його операцій із тим, що відбувається на залізничних сортувальних станціях. Як і алгоритм обчислення ЗПЗ, алгоритм сортувальної станції ґрунтується на стеку. У перетворенні беруть участь дві текстові змінні: вхідний і вихідний рядки. У процесі перетворення використовується стек, що зберігає ще не додані до вихідного рядка операції. Перетворювальна програма читає вхідний рядок послідовно символ за символом (символ – це не обов'язково буква), виконує на кожному кроці деякі дії залежно від того, який символ було прочитано.

5.1.2 Алгоритм.

1. У випадку, коли є символи для обробки необхідно зчитати символ.
2. Якщо символ є числом або додаємо його до вихідного рядка.
3. Якщо символ є дужкою, поміщаємо його в стек.
4. Якщо символ є дужкою, що закривається то доки верхнім елементом стека не стане відкриваюча дужка, виштовхуємо елементи зі стека у

вихідний рядок. При цьому дужка, що відкриває, видаляється зі стека, але у вихідний рядок не додається. Якщо стек закінчився раніше, ніж ми зустріли дужку, це означає, що у виразі або неправильно поставлений роздільник, або не узгоджені дужки.

5. Якщо символ є бінарною операцією та операція на вершині стеку має більший або такий самий пріоритет, то необхідно “виштовхнути” верхній елемент до вихідного рядка. Помістити операцію в стек.
6. Коли вхідний рядок закінчився, виштовхуємо всі символи зі стека у вихідний рядок

5.1.3 Пріоритетність операцій.

Найвищий – вираз в дужках.
Високий – піднесення до степеня.
Середній – множення або ділення.
Низький – додавання або віднімання.

5.1.4 Приклад.

```
Вхід: 3 + 4 * 2 / (1 - 5) ^ 2

Читаємо «3»
  Додаємо «3» до вихідного рядка
   Вихід: 3

Читаємо «+»
  Кладемо «+» у стек
   Вихід: 3
   Стек: +

Читаємо «4»
  Додамо «4» до вихідного рядка
   Вихід: 3 4
   Стек: +

Читаємо «*»
  Кладемо «*» у стек
   Вихід: 3 4
   Стек: + *

Читаємо «2»
  Додамо «2» до вихідного рядка
   Вихід: 3 4 2
   Стек: + *

Читаємо «/»
  Виштовхуємо «*» зі стека у вихідний рядок, кладемо «/» у стек
   Вихід: 3 4 2 *
   Стек: +/

Читаємо «(»
```

```
  Кладемо «(» у стек
   Вихід: 3 4 2 *
   Стек: + / (

Читаємо «1»
  Додамо «1» до вихідного рядка
   Вихід: 3 4 2 * 1
   Стек: + / (

Читаємо «–»
  Кладемо «-» у стек
   Вихід: 3 4 2 * 1
   Стек: + / ( –


Читаємо «5»
  Додамо «5» до вихідного рядка
   Вихід: 3 4 2 * 1 5
   Стек: + / (-

Читаємо «)»
  Виштовхуємо «–» зі стека у вихідний рядок, виштовхуємо «(»
   Вихід: 3 4 2 * 1 5 –
   Стек: +/

Читаємо «^»
  Кладемо «^» у стек
   Вихід: 3 4 2 * 1 5 –
   Стек: +/^

Читаємо «2»
  Додамо «2» до вихідного рядка
   Вихід: 3 4 2 * 1 5 – 2
   Стек: +/^

Кінець виразу
  Виштовхуємо всі елементи зі стека в рядок


Вихід: 3 4 2 * 1 5 – 2 ^ / +
```

5.1.5 Обчислення виразу

Використовуючи алгоритм ЗПЗ математичний вираз 3 + 4 * 2 / (1 - 5) ^ 2 був записаний у вигляді 3 4 2 * 1 5 – 2 ^ / +

Обчислення проводиться зліва направо. Якщо в запису зустрічається число, то число поміщається в стек. Якщо в запису зустрічається оператор, то він застосовується до двох верхніх елементів стеку які виштовхуються із стеку, а результат виконання поміщається в стек.

Запис інтерпретується як зазначено у наведеній нижче таблиці (зазначено стан стека після виконання операції, вершина стека виділена червоним кольором)

Символ	Дія	Стек
3	помістити в стек	3
4	помістити в стек	3 4
2	помістити в стек	3 4 2
*	множення	3 8
1	помістити в стек	3 8 1
5	помістити в стек	3 8 1 5
-	віднімання	3 8 -4
2	помістити в стек	3 8 -4 2
^	піднесення до степеню	3 8 16
/	ділення	3 0.5
+	додавання	3.5

Результат **3.5**, в кінці обчислень знаходиться на вершині стека.

5.2 Завдання до лабораторної роботи

Використовуючи теоретичне відомості розробити програму яка на вхід отримує математичний вираз з довільною кількістю операндів, операторів та дужок. В першу чергу сформувати послідовність символів у ЗПН. На другому етапі виконання лабораторної роботи вирахувати результат послідовності, що була сформована, використовуючи алгоритм запису математичного виразу у ЗПН.

Текст програми разом зі звітом розмістити в директорії lab_04. Директорію lab_04 розмістити в директорії, що використовується для виконання практичних завдань по кожній лекції та має назву **ТР-КВ-22[1 or 2]-Name-Surname**.

6 Додаток 1. Перші кроки з git

Розроблені вказівки ставлять за мету надати первинне уявлення про можливості використання системи контролю версій **git**.

Вказівки розбити на три основні секції: встановлення програмного забезпечення, що дозволяє використовувати систему контроля версій **git**; створення та використання репозиторію на сервері **github**; практичні приклади використання під час розробки **власного** програмного забезпечення.

6.1 Встановлення програмного забезпечення (git клієнта)

1) Скачайте з офіційного сайту https://git-scm.com/ інсталятор.

Рисунок 1 – Приклад інтерфейсу

2) Запустіть інсталятор на виконання.
3) Використовуйте всі опції за замовченням окрім останнього пункту. В останньому пункті виберіть None.

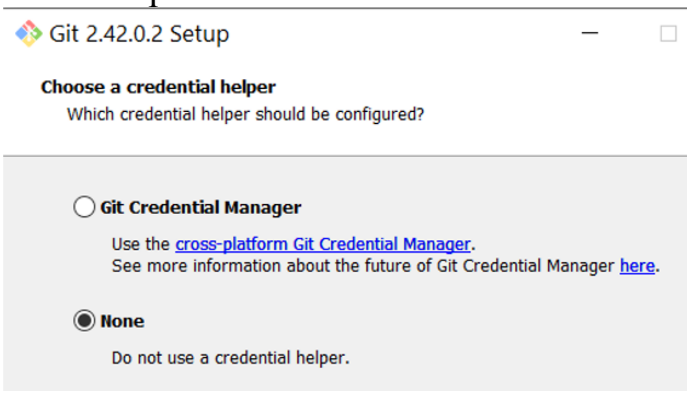


Рисунок 2 – Приклад інтерфейсу

4) Програмне забезпечення буде встановлено.

5) Натисніть правою кнопкою миші в будь якій частині екрану та вибуріть наступний пункт.

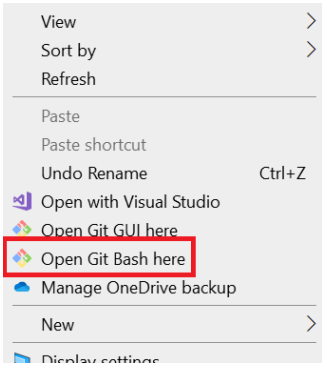


Рисунок 3 – Приклад інтерфейсу

6) Відкриється термінал для роботи. Далі в документі, коли буде зустрічатись фраза «відкрийте гіт термінал» означитиме виконання пункту 5 у вказаній директорії.

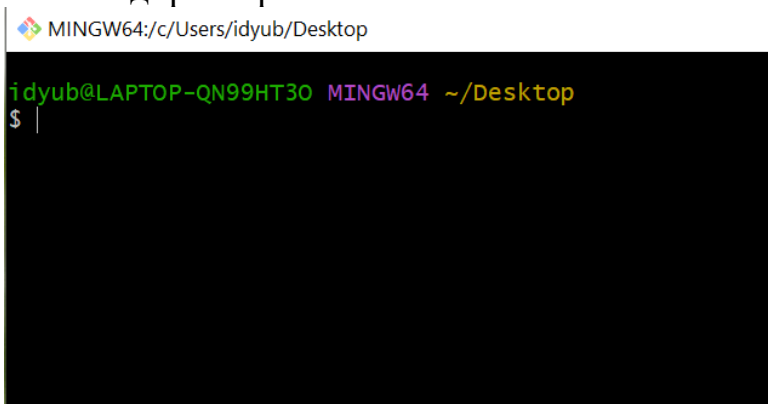


Рисунок 4 – Приклад інтерфейсу

7) Створіть пару SSH ключів виконавши команду

```
ssh-keygen -t ed25519 -C your_email@example.com
```

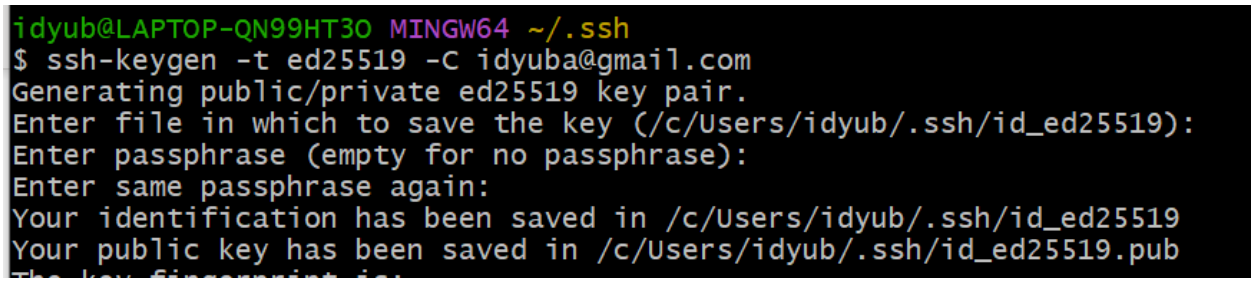


Рисунок 5 – Приклад інтерфейсу

8) **Зверніть увагу** на те куди були збережені ключі, вони нам знадобляться для конфігурації **github** акаунта.

6.2 Сервер контролю версій github

1) Створіть аккаунт на https://github.com/ та залогіньтесь.
2) Перейдіть в конфігурацію користувача:

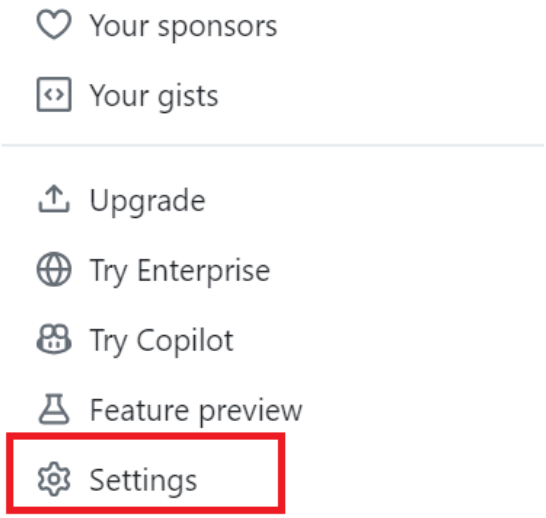


Рисунок 6 – Приклад інтерфейсу

3) Перейдіть в розділ SSH and GPG keys

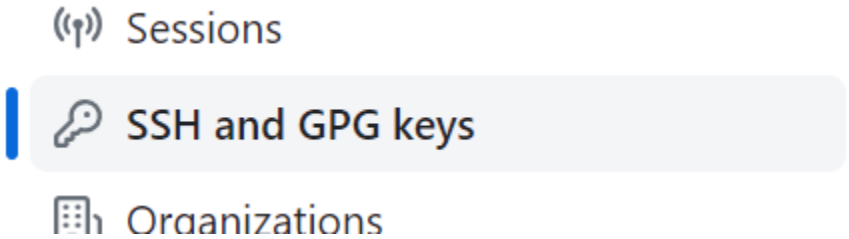


Рисунок 7 – Приклад інтерфейсу

4) Натисніть кнопку Додати новий ключ

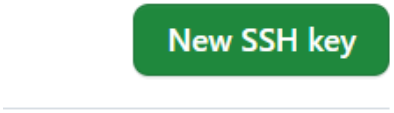


Рисунок 8 – Приклад інтерфейсу

5) Вкажіть назву ключа та вставте вміст згенерованого файлу id_ed25519.pub, що був згенерований в пункті 7 попереднього розділу.
6) Перейдіть на головну сторінку **github**.
7) Натисніть кнопку створення нового репозиторію.

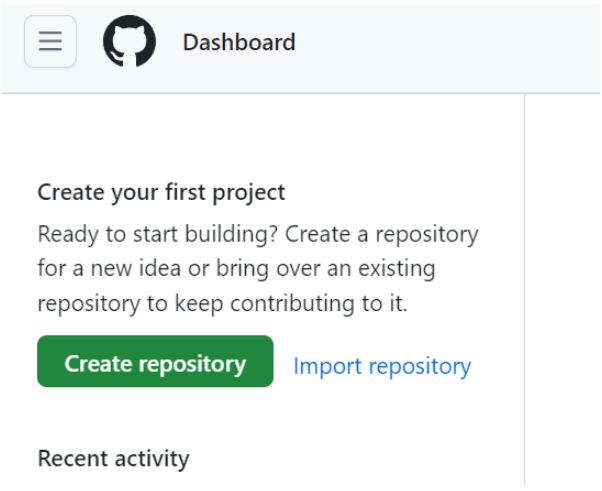


Рисунок 9 – Приклад інтерфейсу

8) Створіть новий репозиторій. Вкажіть ім’я. Виберіть Public як специфікатор доступу. Відмітьте пункт додавання README файлу. Натисніть кнопку

Рисунок 10 – Приклад інтерфейсу

9) Після створення репозиторія відбудеться перенаправлення на цей репозиторій

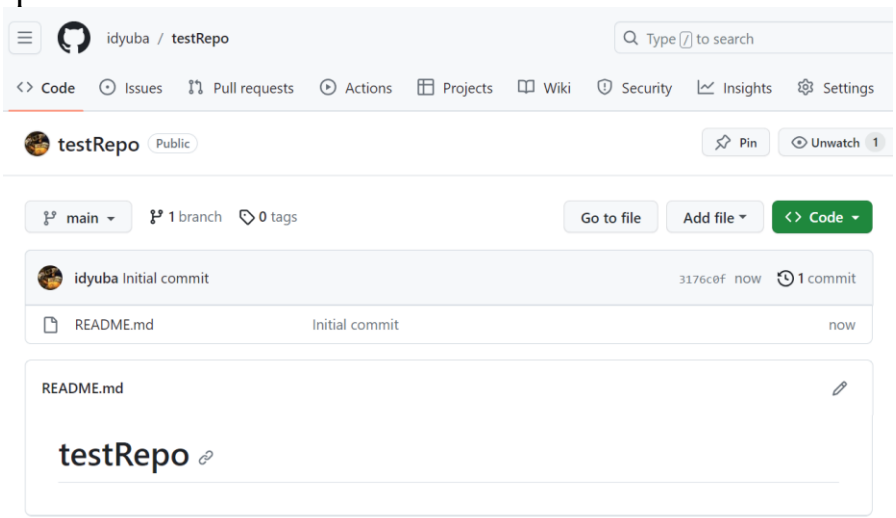


Рисунок 11 - Приклад інтерфейсу

6.3 Перші кроки з використання git

1) На локальному комп’ютері створіть директорії та перейдіть в неї

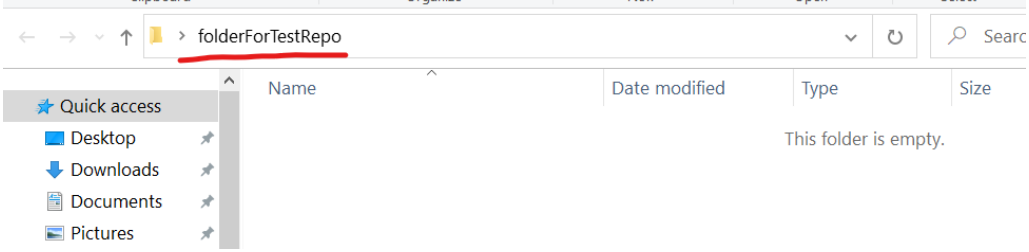


Рисунок 12 – Приклад інтерфейсу

2) Запустіть git термінал через контекстне меню в цій директорії. В пункті **6** частини про встановлення програмного забезпечення детально вказано як запустити git термінал.

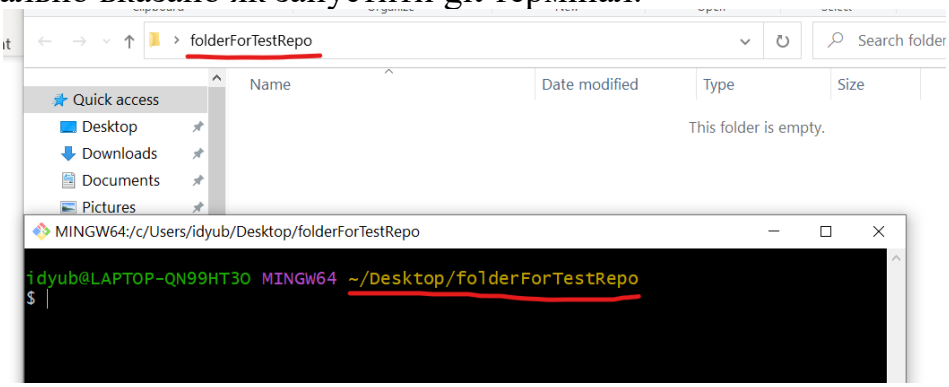


Рисунок 13 – Приклад інтерфейсу

3) На веб інтерфесі github відкрийте створений репозиторій та натисніть кнопку

Рисунок 14 - Приклад інтерфейсу

4) У вікні, що відкриється виберіть SSH та натисніть кнопку копіювання посилання.

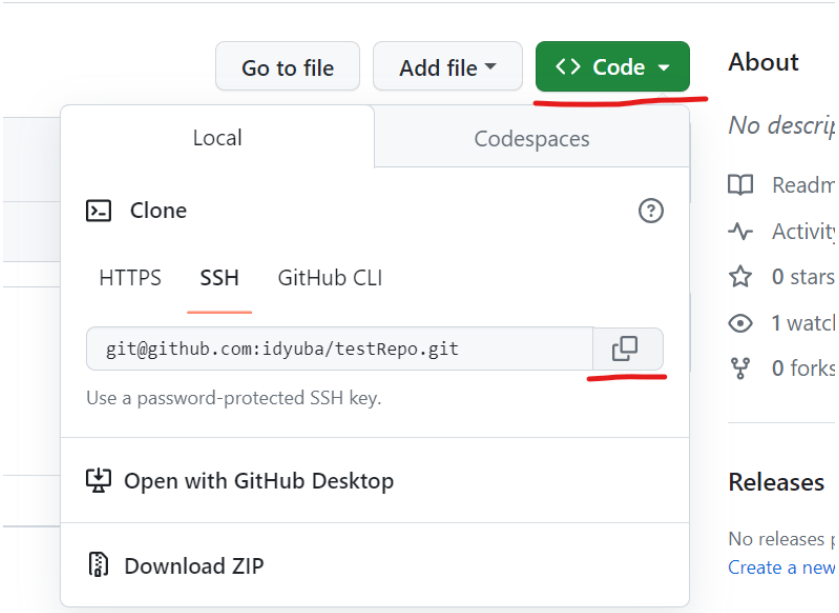


Рисунок 15 – Приклад інтерфейсу

5) В буфері обміну буде збережено наступну інформацію.

```
git@github.com:idyuba/testRepo.git
```

6) Поверніться в git термінал та виконайте команду **клонування репозиторію**. Необхідна адреса на даний час збережена в буфері обміну.

```
git clone git@github.com:idyuba/testRepo.git
```

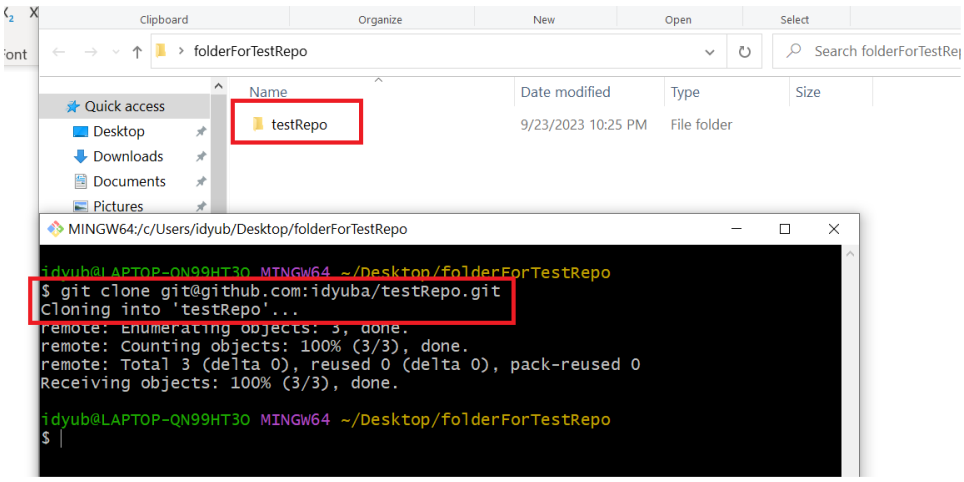


Рисунок 16 - Приклад інтерфейсу

7) Після успішного клонування з сервера github з’явиться директорія в якій буде розміщена локальна копія репозиторія, що був створений на github.
8) Перейдіть в склоновону директорію як через провідник так і через гіт термінал

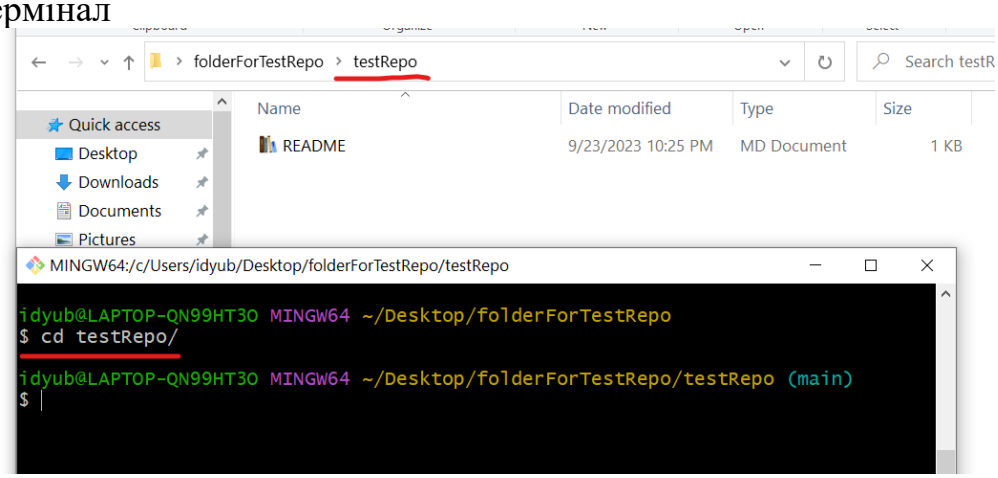


Рисунок 17 – Приклад інтерфейсу

9) **Увага!** Всі git команди про які буде йти мова в наступному розділі мають виконувати тільки в директорії, що була створена під час клонування.

6.4 Початок роботи

1) Створіть новий файл index.html.

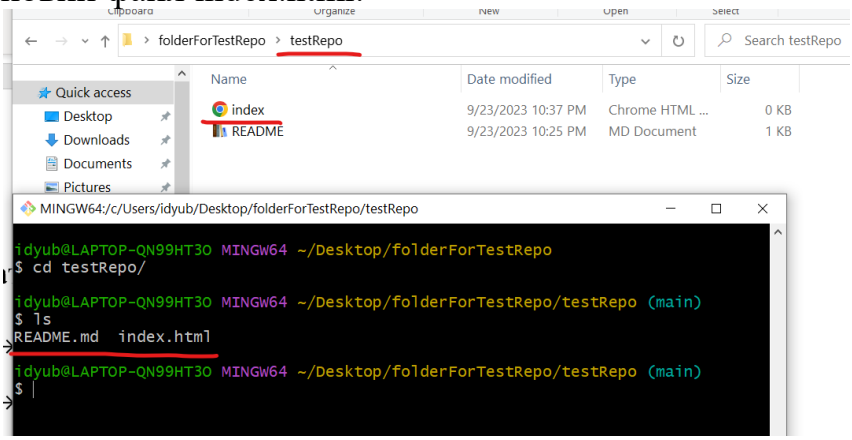


Рисунок 18 – Приклад інтерфейсу

2) Для першої спроби роботи з github цього достатньо.
3) У терміналі виконайте команду git status

```
git status
```

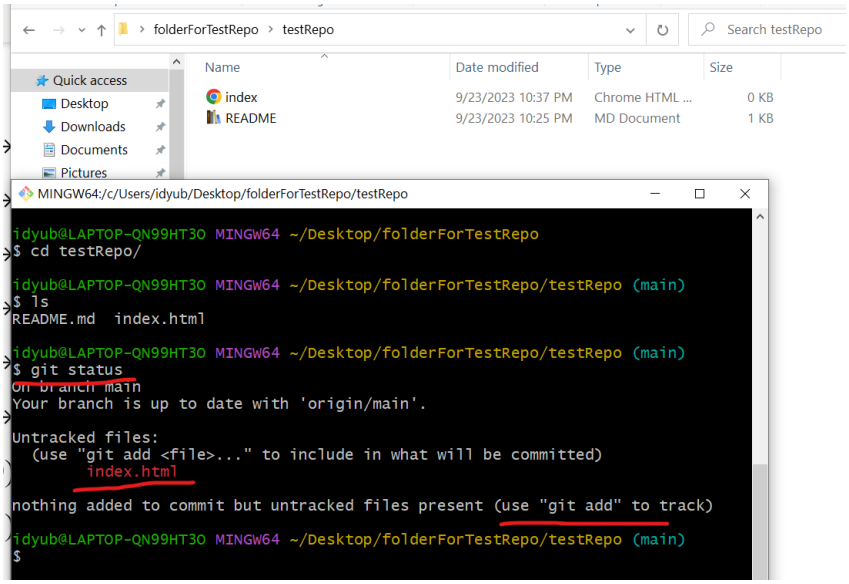


Рисунок 19 – Приклад інтерфейсу

4) Як видно з малюнку вище створений файл index.html відмічений таким чином, що файл присутній проте система не знає для чого він потрібен. Також пропонується команда git add для того, щоб відмітити файл.

5) Виконайте команду

```
git add index.html
```

Та знову

```
git status
```

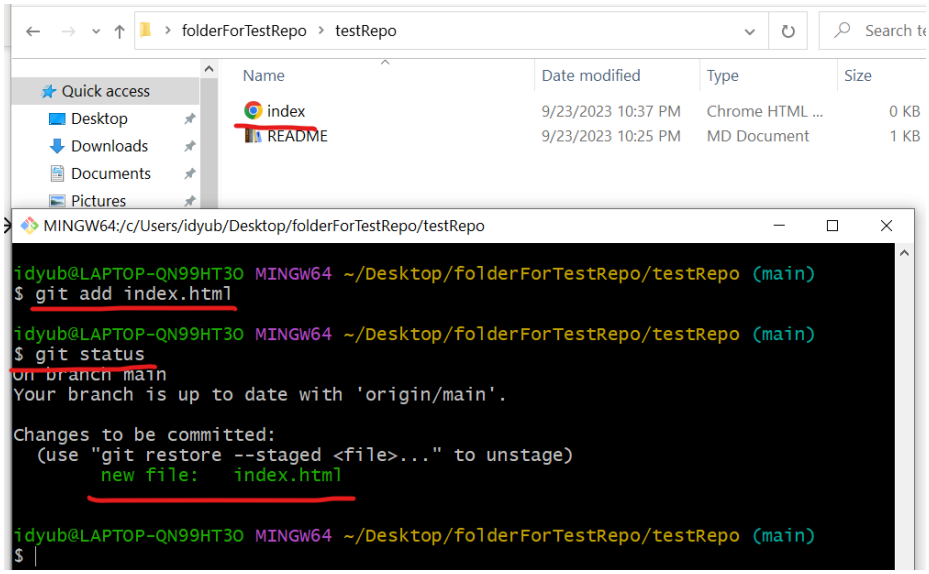


Рисунок 20 – Приклад інтерфейсу

6) Тепер git знає, що файл буде використовуватись. Зверніть увагу що після виконання команди git add файл index.html став відмічений як новий. Це цілком зрозуміло, бо файл був створений в попередніх пунктах на локальному комп'ютері, та сервер про цей файл нічого поки не знає.
7) Після того, як файл був відмічений, необхідно сформувати комміт команду зі зрозумілим повідомленням.

```
git commit -m "add index.html"
```

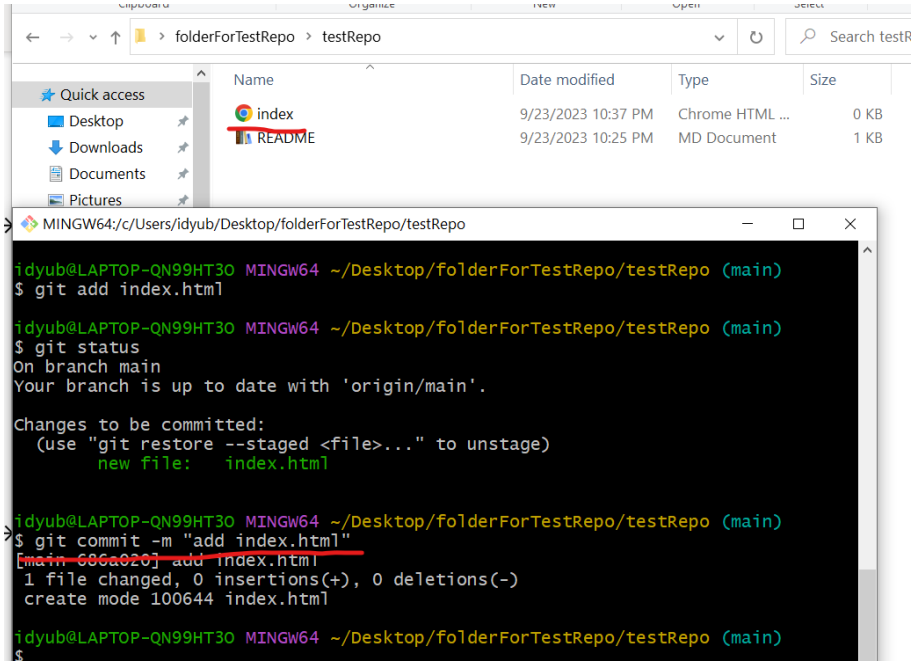


Рисунок 21 - Приклад інтерфейсу

8) Створений комміт необхідно відправити на github сервер.

```
git push
```

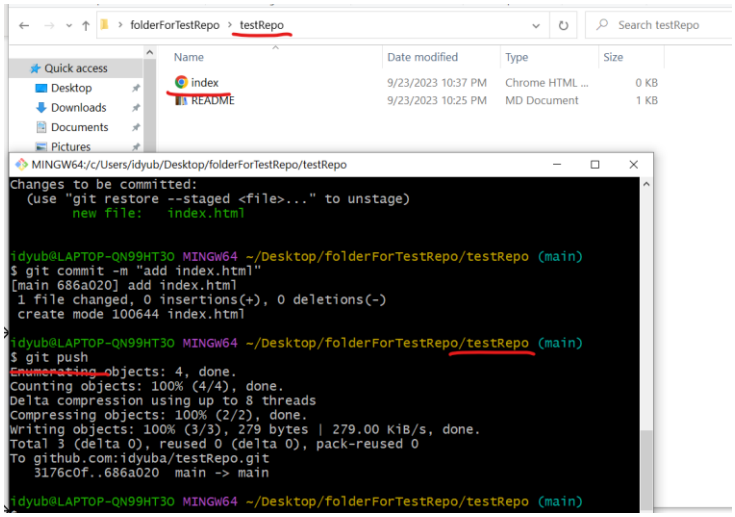


Рисунок 22 – Приклад інтерфейсу

9) Перейдіть на веб інтерфейс github та переконайтесь, що файл index.html присутній на сервері.

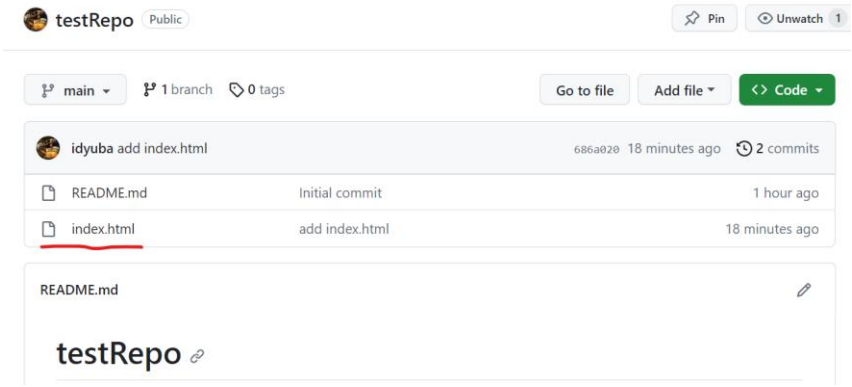


Рисунок 23 – Приклад інтерфейсу

10) Створіть ще два файли main.cpp та backend.py та виконайте повторно всі дії для розміщення цих файлів на github.

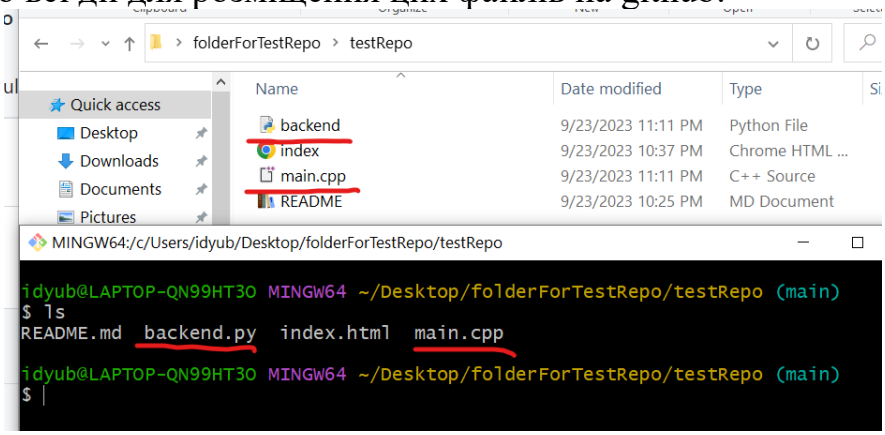


Рисунок 23 – Приклад інтерфейсу

```
git add main.cpp backend.py
git status
git commit -m "add main.cpp and backend.py"
git push
```

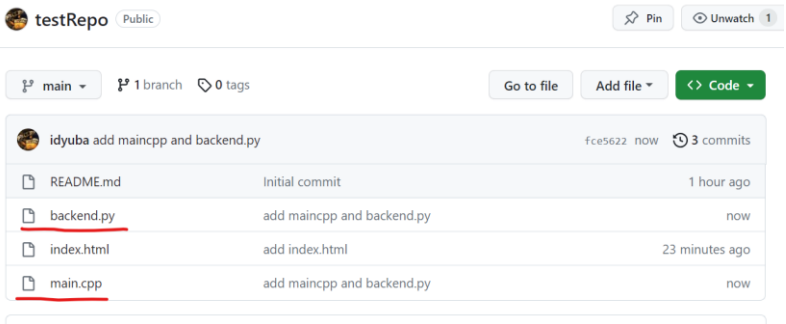


Рисунок 24 – Приклад інтерфейсу

11) Перегляньте історію коммітів на веб інтерфейсі

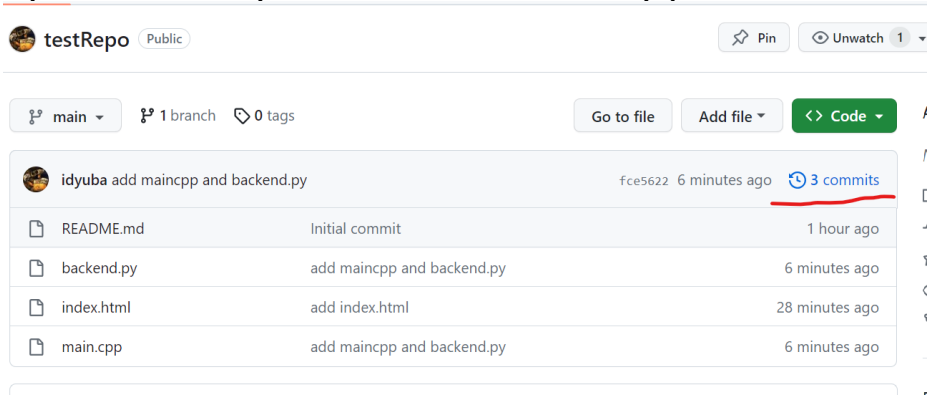


Рисунок 25 – Приклад інтерфейсу

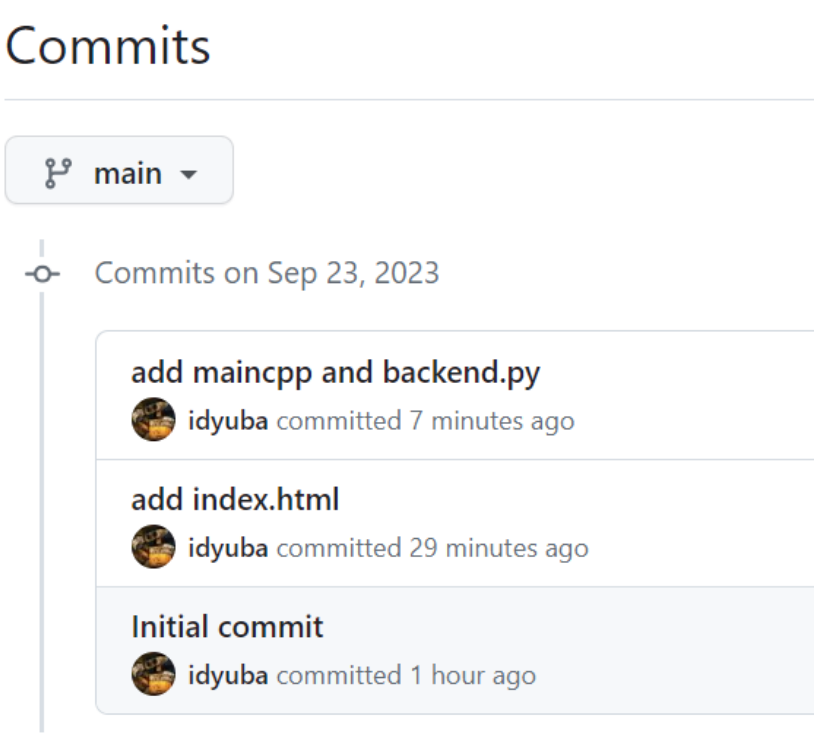


Рисунок 26 – Приклад інтерфейсу

12) Кожен комміт зберігає всю інформацію про зміни, що були внесені.

6.5 Збереження змін під час роботи

1) Відкрийте файл backend.py та додайте декілька рядків коду. Збережіть файл.

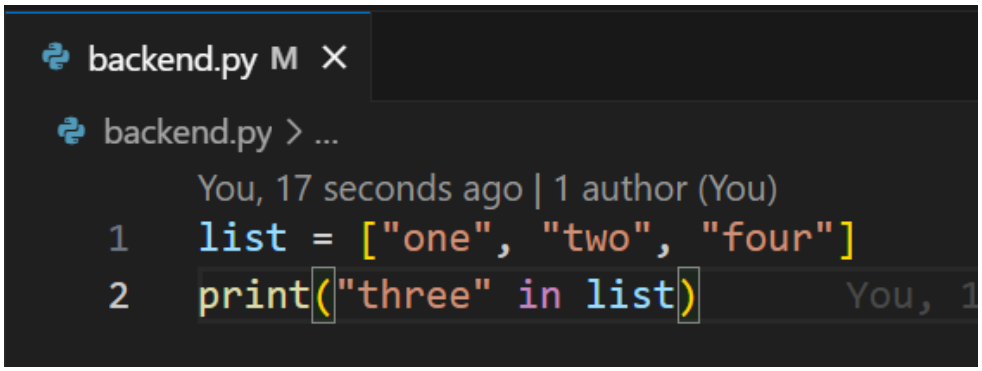


Рисунок 27 – Приклад інтерфейсу

2) Перейдіть в git термінал та виконайте команду

```
git status
```

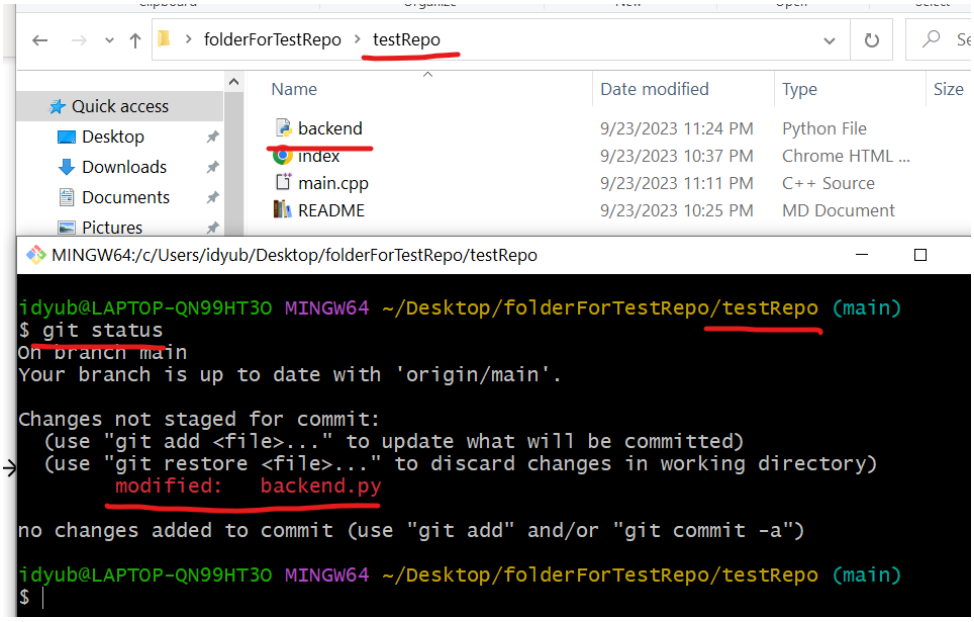


Рисунок 28 – Приклад інтерфейсу

3) Файл backend.py цього разу відмічений як змінений, а не як невідомий файл. Git пропонує дві опції: зберегти зміни використовуючи git add чи відмінити зміни використовуючи git restore.

4) Виконайте наступні команди для того, щоб зберігти зміни, та переконатись, що зміни збережені.

```
git add backend.py
git status
```

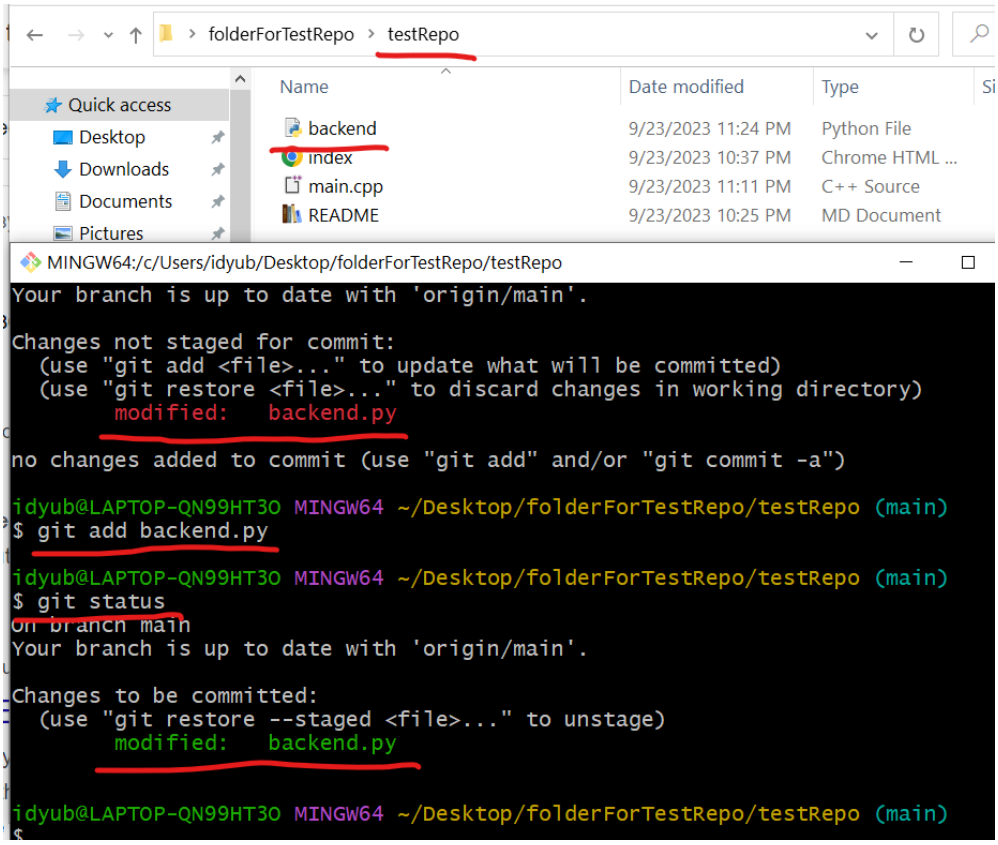


Рисунок 30 – Приклад інтерфейсу

5) Сформуйте комміт з відповідним повідомленням та відправте зміни на git сервер.

```
git commit -m "update backed.py"
git push
```

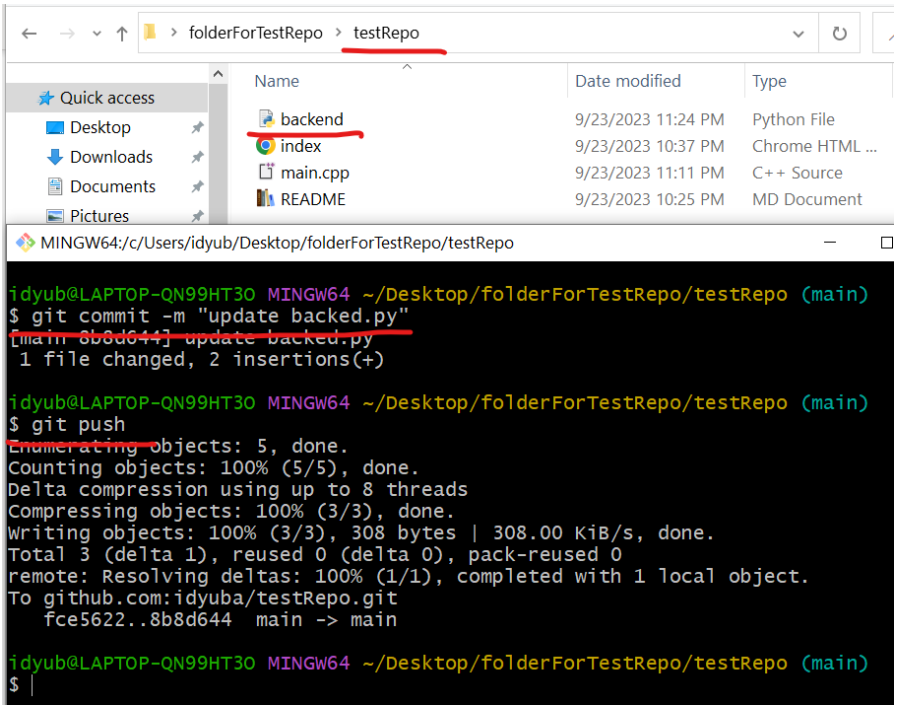


Рисунок 31 – Приклад інтерфейсу

6) Перейдіть на веб інтерфейс та відкрийте файл backed.py. Зміни, що були виконані на локальному комп'ютері відображаються на сервері.

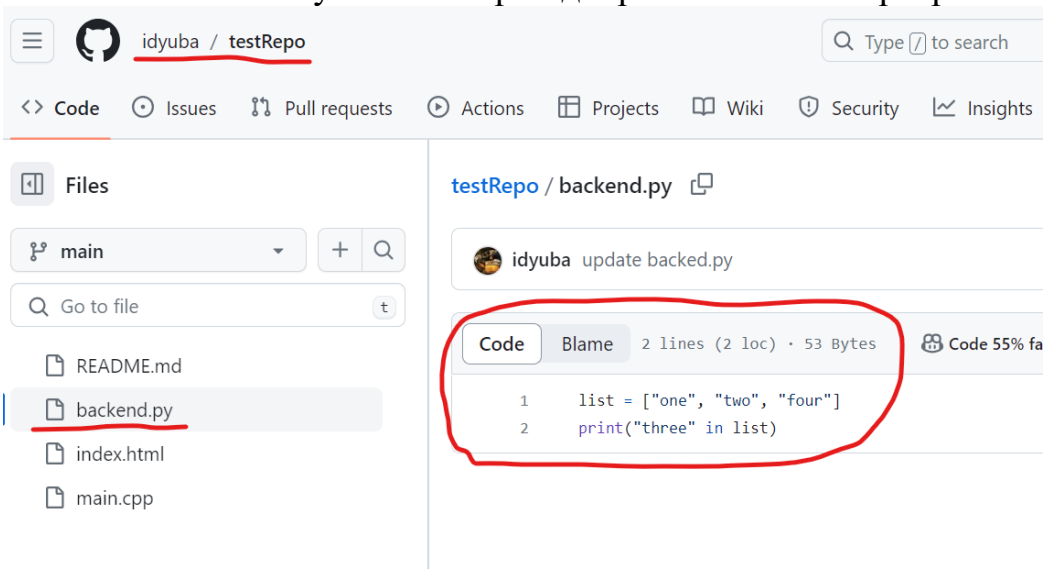


Рисунок 32 - Приклад інтерфейсу

7) Перейдіть в список коммітів на веб інтерфейсі та відкрийте комміт в якому були внесені зміни до файлу backend.py. Переконайтесь що і повідомлення і зміни відповідають тим, що вносились на локальному комп’ютері.

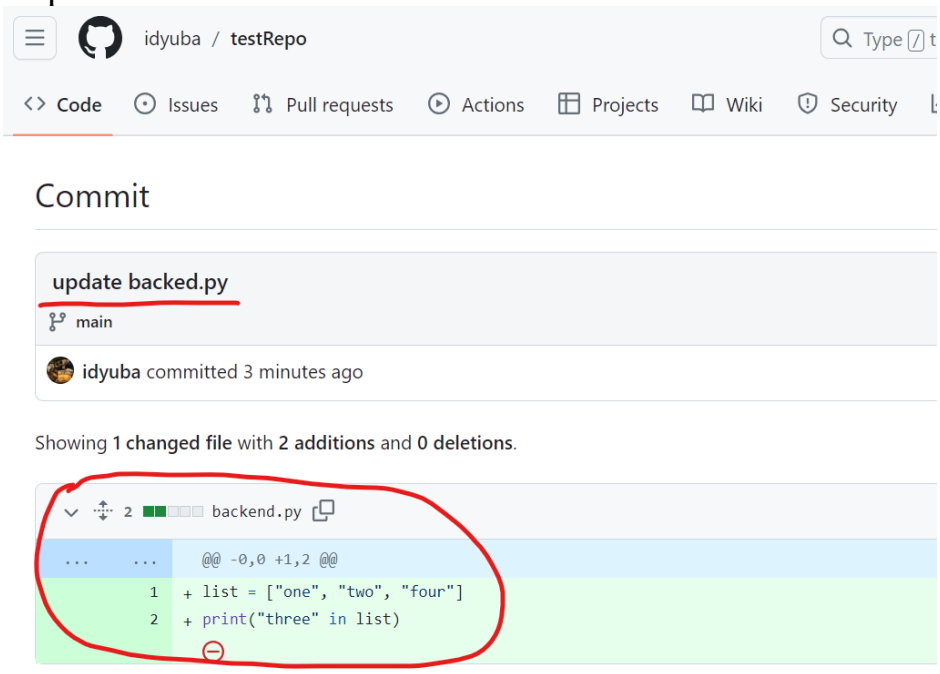


Рисунок 33 – Приклад інтерфейсу

7 Рекомендована література

1. C Programming Language. Prentice Hall, 1988. 274 p.
2. Python Basics: A Practical Introduction to Python 3 / D. Bader et al. Real Python (realpython.com), 2021. 635 p.
3. Ramalho L. Fluent Python: Clear, Concise, and Effective Programming / ed. by M. Blanchette, R. Roumeliotis. O'Reilly Media, 2015. 792 p.
4. Lutz M. Learning Python. 5th ed. 2013. 1540 p.
5. Barry P. Head First Python: A Brain-Friendly Guide. O'Reilly Media, 2016. 624 p.
6. Modern Operating Systems: Forth Edition. Pearson, 2015.
7. Matthes E. Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming. No Starch Press, 2019. 544 p.
8. Automate the Boring Stuff with Python: Practical Programming for Total Beginners. San Francisco, USA : No Starch Press, 2015. 504 p.
9. Think Python: How to Think Like a Computer Scientist / ed. by M. Blanchette. 2nd ed. Sebastopol, California, United States of America : O'Reilly Media, 2015. 222 p.
10. Slatkin B. Effective Python: 90 Specific Ways to Write Better Python. Addison-Wesley Professional, 2019. 480 p.
11. Gorelick M., Ozsvald I. High Performance Python: Practical Performant Programming for Humans. O'Reilly Media, Incorporated, 2020. 450 p.