**Звіт про виконання практичних завдань до лекцій з курсу Технології програмування на мові Python**

Звіт до Теми №1

Функції та змінні

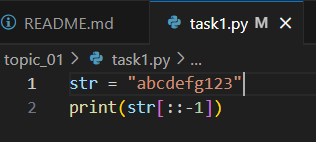
Під час виконання практичного завдання до Теми №1 було надано варіанти рішення до наступних задач:

**Перетворення рядка**

Необхідно рядок, що має вигляд "abcdefg123" перетворити наступним чином "321gfedcba", вважаючи сталою довжину рядку в 10 символів.

Хід виконання завдання:

Використовую змінну де записую наш текст, далі використовую функцію ::-1 в команді print яка виведе текст на екран в зворотному порядку





Текст програми:

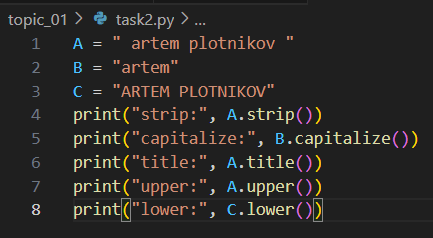
|  |
| --- |
| str = "abcdefg123"  print(str[::-1]) |

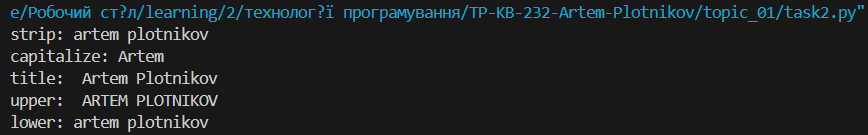
**Тестування функцій**

Необхідно виконати тестування функцій, що працюють з рядками: strip(), capitalize(), title(), upper(), lower().

Хід виконання завдання:

Я написав кілька змінних зі своїм ім'ям, там де вони доречно будуть працювати з певними функціями, використав функції та показав їх у терміналі





Текст програми:

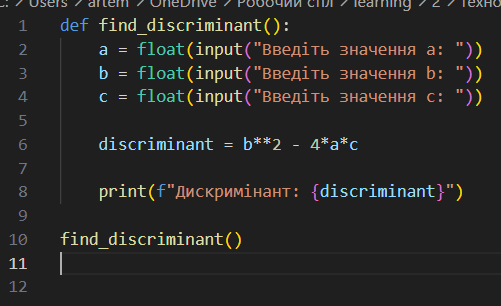
|  |
| --- |
| A = " artem plotnikov "  print(A.strip())  B = "artem"  print(B.capitalize())  print(A.title())  print(A.upper())  C = "ARTEM PLOTNIKOV"  print(C.lower()) |

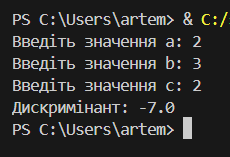
**Функція пошуку дискримінанта**

Необхідно написати функцію пошуку дискримінанту квадратного рівняння

Хід виконання завдання:

Спершу створюю змінні для вводу даних коефіцієнтів a, b, c, потім пишу функцію безпосередньо формули дискримінанту, вивожу значення в термінал та звертаюся в кінці до функції

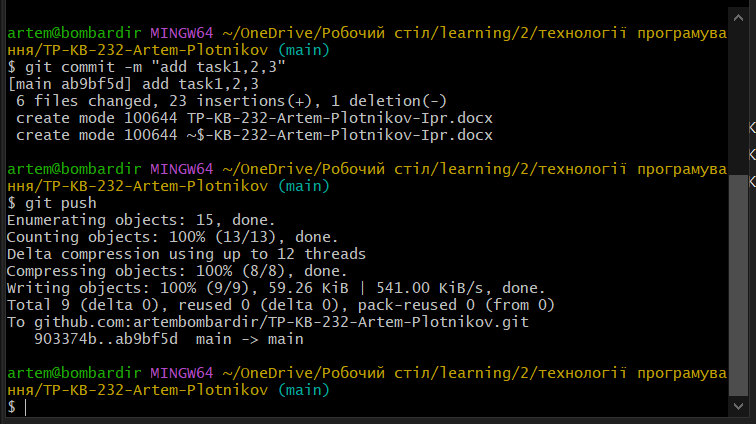




Текст програми:

|  |
| --- |
| def find\_discriminant():  a = float(input("Введіть значення a: "))  b = float(input("Введіть значення b: "))  c = float(input("Введіть значення c: "))    discriminant = b\*\*2 - 4\*a\*c    print(f"Дискримінант: {discriminant}")  find\_discriminant() |

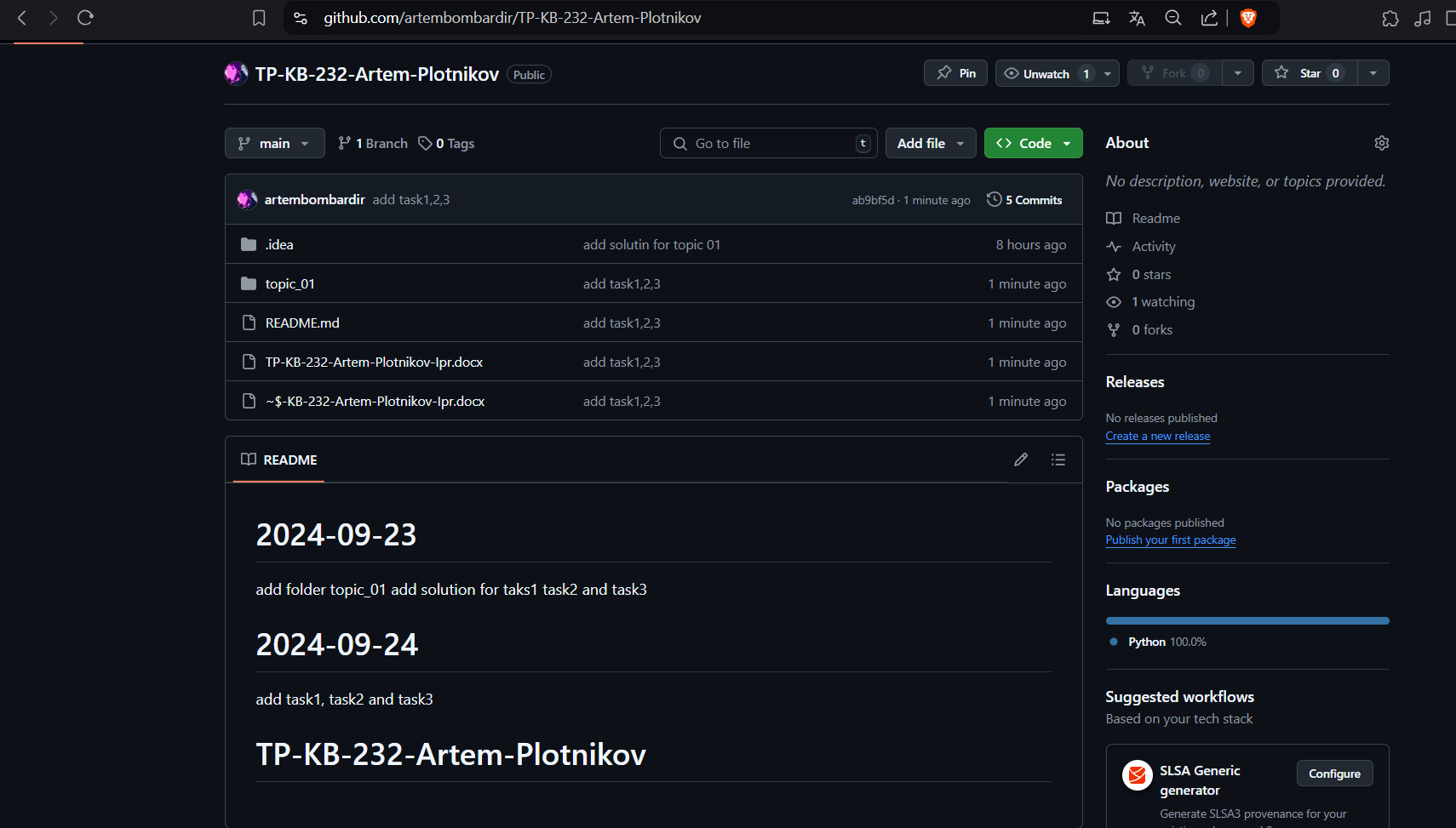
Зберігаю, роблю змінні в файлі read me та відсилаю на гітхаб



Посилання на github:

https://github.com/artembombardir/TP-KB-232-Artem-Plotnikov

Знімок екрану з посилання на github:



Звіт до Теми №2

Умовний перехід

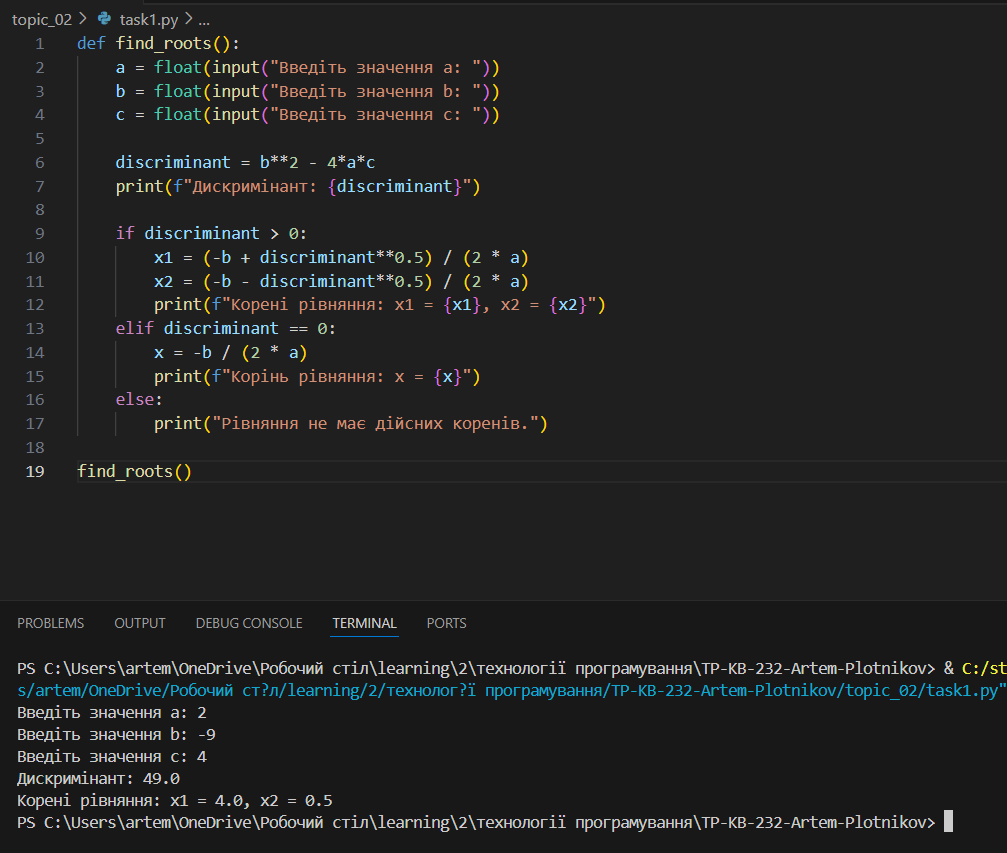
Під час виконання практичного завдання до Теми №2 було надано варіанти рішення до наступних задач:

**Пошук коренів**

Необхідно написати функцію пошуку коренів квадратного рівняння використовуючи функцію розрахунку дискримінанту з попередньої теми та умовні переходи.

Хід виконання завдання:

Використовую наступні конструкції: if D > 0, print два корені рівняння, elif D == 0 print один корінь, else print дійсних коренів не існує



Текст програми:

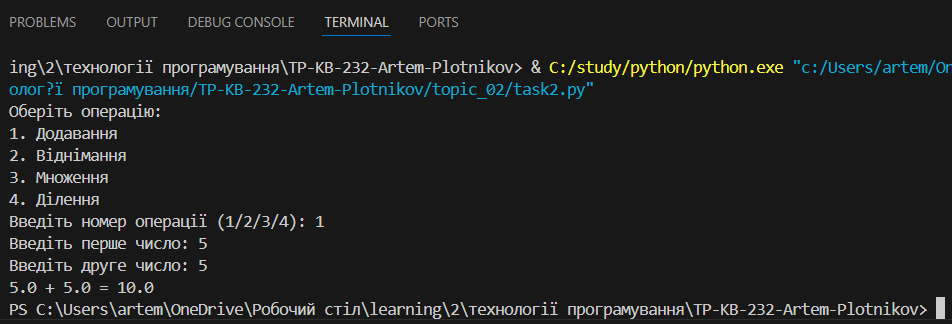
|  |
| --- |
| def find\_roots():  a = float(input("Введіть значення a: "))  b = float(input("Введіть значення b: "))  c = float(input("Введіть значення c: "))    discriminant = b\*\*2 - 4\*a\*c  print(f"Дискримінант: {discriminant}")    if discriminant > 0:  x1 = (-b + discriminant\*\*0.5) / (2 \* a)  x2 = (-b - discriminant\*\*0.5) / (2 \* a)  print(f"Корені рівняння: x1 = {x1}, x2 = {x2}")  elif discriminant == 0:  x = -b / (2 \* a)  print(f"Корінь рівняння: x = {x}")  else:  print("Рівняння не має дійсних коренів.")  find\_roots() |

**Програма калькулятор**

Необхідно написати програму калькулятор використовуючи **if else** конструкцію. Кожна операція має бути виконана в окремій функції.

Хід виконання завдання:

Додав окремі функції для додавання, віднімання, множення, ділення(використовую if - else, спираючись на те що ділення на 0 неможливе), після чого використав if else для вибору користувачем його завдання для калькулятора



Текст програми:

|  |
| --- |
| def add(x, y):  return x + y  def subtract(x, y):  return x - y  def multiply(x, y):  return x \* y  def divide(x, y):  if y != 0:  return x / y  else:  return "Помилка: Ділення на нуль неможливе!"  def calculator():  print("Оберіть операцію:")  print("1. Додавання")  print("2. Віднімання")  print("3. Множення")  print("4. Ділення")    choice = input("Введіть номер операції (1/2/3/4): ")    num1 = float(input("Введіть перше число: "))  num2 = float(input("Введіть друге число: "))    if choice == '1':  print(f"{num1} + {num2} = {add(num1, num2)}")  elif choice == '2':  print(f"{num1} - {num2} = {subtract(num1, num2)}")  elif choice == '3':  print(f"{num1} \* {num2} = {multiply(num1, num2)}")  elif choice == '4':  print(f"{num1} / {num2} = {divide(num1, num2)}")  else:  print("Невірний вибір операції. Спробуйте ще раз.")  calculator() |

**Програма калькулятор 2**

Необхідно написати програму калькулятор використовуючи **match** конструкцію. Кожна операція має бути виконана в окремій функції.

Хід виконання завдання:

Все теж саме що і в минулому завданні, тільки замість if else використовую match case

Текст програми:

|  |
| --- |
| match choice:  case '1':  print(f"{num1} + {num2} = {add(num1, num2)}")  case '2':  print(f"{num1} - {num2} = {subtract(num1, num2)}")  case '3':  print(f"{num1} \* {num2} = {multiply(num1, num2)}")  case '4':  print(f"{num1} / {num2} = {divide(num1, num2)}")  case \_:  print("Невірний вибір операції. Спробуйте ще раз.") |

Звіт до Теми №3

Цикли

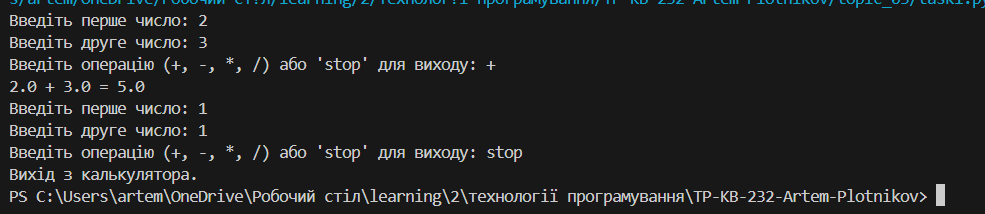
Під час виконання практичного завдання до Теми №3 було надано варіанти рішення до наступних задач:

**Калькулятор з постійними запитами на введення нових даних**

Необхідно написати програму калькулятор з постійними запитами на введення нових даних та операцій. За основу взяти програму калькулятор з попередньої теми. Реалізувати механізм завершення програми після отримання відповідної команди.

Хід виконання завдання:

Використовую цикл while до калькулятора, допоки користувач не введе слово stop у виборі операції



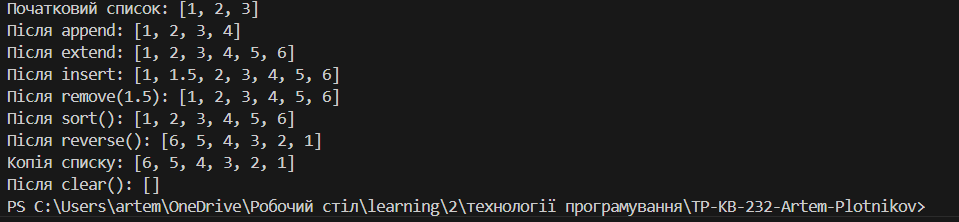
Текст програми:

|  |
| --- |
| def add(x, y):  return x + y  def subtract(x, y):  return x - y  def multiply(x, y):  return x \* y  def divide(x, y):  if y != 0:  return x / y  else:  return "Помилка: ділення на нуль!"  def calculator():  while calculator:  num1 = float(input("Введіть перше число: "))  num2 = float(input("Введіть друге число: "))  operation = input("Введіть операцію (+, -, \*, /) або 'stop' для виходу: ")  if operation == 'stop':  print("Вихід з калькулятора.")  break    match operation:  case '+':  print(f"{num1} + {num2} = {add(num1, num2)}")  case '-':  print(f"{num1} - {num2} = {subtract(num1, num2)}")  case '\*':  print(f"{num1} \* {num2} = {multiply(num1, num2)}")  case '/':  print(f"{num1} / {num2} = {divide(num1, num2)}")  case \_:  print("Невірна операція.")  calculator() |

**Тестування функцій**

Необхідно написати програму тестування функцій списків таких як: extend(), append(), insert(id, val), remove(val), clear(), sort(), reverse(), copy()

У функції test\_list\_functions() було продемонстровано використання методів списків: створено список lst = [1, 2, 3], додано елемент 4 через append, розширено список [5, 6] методом extend, вставлено значення 1.5 на позицію 1 через insert, видалено 1.5 методом remove, виконано сортування sort, реверс reverse, створено копію списку методом copy, після чого список очищено методом clear. Функція виводить результати змін у списку на кожному етапі.



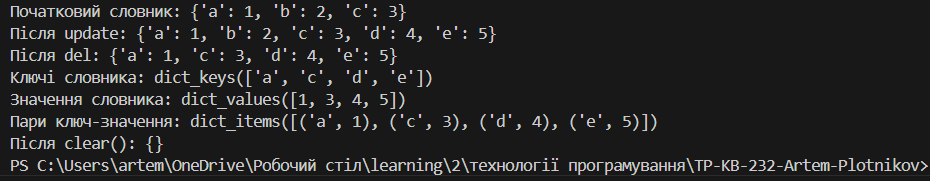
Текст програми:

|  |
| --- |
| def test\_list\_functions():      lst = [1, 2, 3]      print("Початковий список:", lst)      lst.append(4)      print("Після append:", lst)      lst.extend([5, 6])      print("Після extend:", lst)      lst.insert(1, 1.5)      print("Після insert:", lst)      lst.remove(1.5)      print("Після remove(1.5):", lst)      lst.sort()      print("Після sort():", lst)      lst.reverse()      print("Після reverse():", lst)      lst\_copy = lst.copy()      print("Копія списку:", lst\_copy)      lst.clear()      print("Після clear():", lst)  test\_list\_functions() |

**Тестування словників**

Необхідно написати програму тестування функцій словників таких як: update(), del(), clear(), keys(), values(), items()

У функції test\_dict\_functions() продемонстровано використання методів словників: створено словник d = {'a': 1, 'b': 2, 'c': 3}, додано елементи через update({'d': 4, 'e': 5}), видалено ключ 'b' через del, отримано ключі методом keys(), значення методом values(), пари ключ-значення методом items(), а також очищено словник через clear(). Проміжні результати змін у словнику виводяться у консоль.



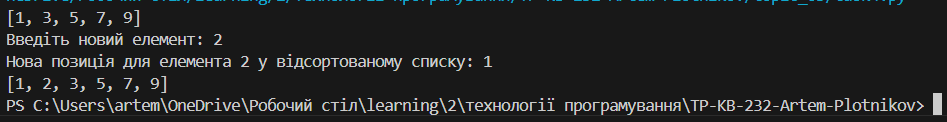
Текст програми:

|  |
| --- |
| def test\_dict\_functions():  d = {'a': 1, 'b': 2, 'c': 3}  print("Початковий словник:", d)  d.update({'d': 4, 'e': 5})  print("Після update:", d)  del d['b']  print("Після del:", d)  print("Ключі словника:", d.keys())  print("Значення словника:", d.values())  print("Пари ключ-значення:", d.items())  d.clear()  print("Після clear():", d)  test\_dict\_functions() |

Пошук вставки нового елементу

Необхідно маючи відсортований список, написати функцію пошуку позиції для вставки нового елементу в список.

Створено функцію find\_position, яка проходить відсортований список і визначає позицію для вставки нового елементу, після чого цей елемент додається у список методом insert, зберігаючи порядок сортування; програма обробляє некоректне введення і виводить оновлений список.



Текст програми:

|  |
| --- |
| def find\_position(sorted\_list, new\_element):  position = 0  for elem in sorted\_list:  if new\_element > elem:  position += 1  return position  sorted\_list = [1, 3, 5, 7, 9]  print(sorted\_list)  try:  new\_element = int(input("Введіть новий елемент: "))  except ValueError:  print("Будь ласка, введіть коректне число.")  else:  position = find\_position(sorted\_list, new\_element)  print(f"Нова позиція для елемента {new\_element} у відсортованому списку: {position}")  sorted\_list.insert(position, new\_element)  print(sorted\_list) |

Звіт до Теми №4

Виняткові ситуації

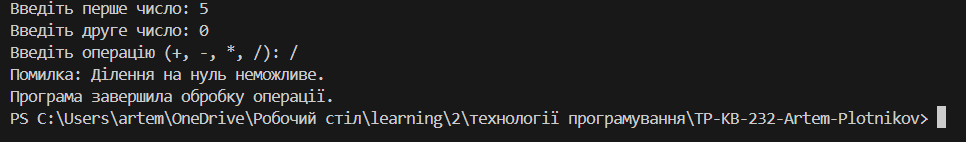
Під час виконання практичного завдання до Теми №4 було надано варіанти рішення до наступних задач:

**Попередні умови**: реалізована програма калькулятор, що використовує метод нескінченного введення даних для обробки. Всі дії (додавання, віднімання, множення, ділення) реалізовані як окремі функції та використовуються у відповідних місцях.

1. Розширити програму калькулятор функцією запитів даних для виконання операцій від користувача, що обробляє виняткові ситуації.
2. Розширити функцію ділення обробкою виняткової ситуації ділення но нуль
3. Ознайомитись зі списком виняткових ситуацій за посиланням <https://docs.python.org/3/library/exceptions.html>

Хід виконання завдання:

Реалізовано функцію calculator(), яка обробляє введення чисел, вибір операції та виконує арифметичні дії; додано обробку винятків: ValueError для некоректного введення, ZeroDivisionError для ділення на нуль і загальний Exception для інших помилок; результат операції виводиться у разі відсутності помилок, а завершення роботи програми супроводжується повідомленням у блоці finally.



Текст програми:

|  |
| --- |
| def calculator():  try:  # Введення чисел користувачем  num1 = float(input("Введіть перше число: "))  num2 = float(input("Введіть друге число: "))  operation = input("Введіть операцію (+, -, \*, /): ")  # Виконання операції  if operation == '+':  result = num1 + num2  elif operation == '-':  result = num1 - num2  elif operation == '\*':  result = num1 \* num2  elif operation == '/':  if num2 == 0:  raise ZeroDivisionError("Ділення на нуль неможливе.")  result = num1 / num2  else:  # Невідома операція  raise ValueError("Невідома операція")  except ValueError as ve:  print(f"Помилка введення: {ve}. Будь ласка, введіть коректні дані.")  except ZeroDivisionError as zde:  print(f"Помилка: {zde}")  except Exception as e:  print(f"Сталася непередбачена помилка: {e}")  else:  # Виконується, якщо не сталося жодної помилки  print(f"Результат операції: {result}")  finally:  print("Програма завершила обробку операції.")  # Викликаємо функцію  calculator() |

Ознайомився з винятковими ситуаціями, описаними в документації Python за посиланням https://docs.python.org/3/library/exceptions.html. Серед основних виключень, які можуть виникати під час виконання програм, вивчив ValueError, що виникає при некоректному введенні даних, ZeroDivisionError, пов'язаний із забороною ділення на нуль, а також Exception, який використовується для обробки загальних помилок. Ці знання дали змогу забезпечити стабільну роботу калькулятора, додаючи обробку помилок і забезп­ечуючи дружній інтерфейс для користувача.

Звіт до Теми №5

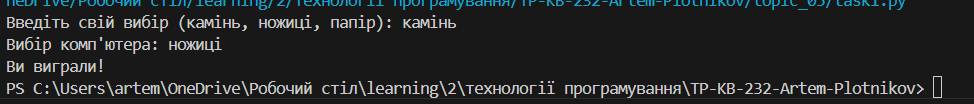
Бібліотеки

Під час виконання практичного завдання до Теми №5 було надано варіанти рішення до наступних задач:

1. Гра з комп’ютером: камінь, ножиці, папір. Програма виконує запит від користувача на введення одного із значень ["stone", "scissor", "paper"]. Наступним кроком, використовуючи модуль random, програма у випадковому порядку вибирає одне із значень ["stone", "scissor", "paper"]. В залежності від умови, що камінь перемагає ножиці, ножиці перемагають папір, а папір перемагає камінь визначити переможця.

Хід завдання:

Реалізовано функцію rock\_paper\_scissors, яка приймає вибір користувача та перевіряє його на коректність; вибір комп’ютера генерується випадково за допомогою модуля random; на основі правил гри ("камінь перемагає ножиці", "ножиці перемагають папір", "папір перемагає камінь") визначається переможець або оголошується нічия.



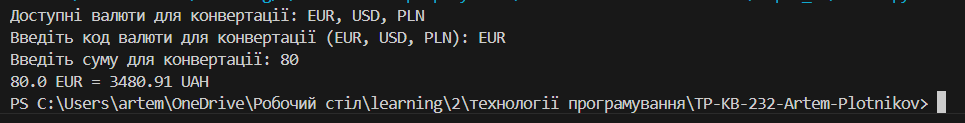
Текст програми:

|  |
| --- |
| import random  def rock\_paper\_scissors():  choices = ["камінь", "ножиці", "папір"]  user\_choice = input("Введіть свій вибір (камінь, ножиці, папір): ").strip().lower()    if user\_choice not in choices:  print("Невірний вибір! Будь ласка, введіть одне із значень: камінь, ножиці, папір.")  return # Завершуємо функцію, якщо введення некоректне  computer\_choice = random.choice(choices)  print(f"Вибір комп'ютера: {computer\_choice}")  if user\_choice == computer\_choice:  print("Нічия!")  elif (user\_choice == "камінь" and computer\_choice == "ножиці") or \  (user\_choice == "ножиці" and computer\_choice == "папір") or \  (user\_choice == "папір" and computer\_choice == "камінь"):  print("Ви виграли!")  else:  print("Комп'ютер виграв!")  # Запускаємо гру  rock\_paper\_scissors() |

1. Програма конвертування іноземної валюти в українську гривню. Для отримання актуальних курсів валют необхідно використовувати API НБУ та модуль, що надає можливість виконувати запити до сторонніх сервісів requests. Достатня умова роботи – можливість конвертації для трьох іноземних валют EUR, USD, PLN. Користувачу надається можливість введення кількості та типу валюти, результат роботи програми – конвертоване значення в українських гривнях.

Хід завдання:

Програма реалізує дві основні функції: get\_exchange\_rate, яка отримує актуальний курс валюти за допомогою запиту до API НБУ, і currency\_converter, яка дозволяє користувачу вибрати валюту та суму для конвертації. Користувач вводить код валюти та суму, після чого програма здійснює конвертацію в гривні та виводить результат. Всі винятки, пов’язані з запитами до API, обробляються коректно, і програма виводить повідомлення у разі помилок чи неправильного введення.



Текст програми:

|  |
| --- |
| import requests  def get\_exchange\_rate(currency\_code):  url = "https://bank.gov.ua/NBUStatService/v1/statdirectory/exchange?json"  try:  response = requests.get(url)  response.raise\_for\_status() # Перевірка на помилки запиту  rates = response.json()  for rate in rates:  if rate['cc'] == currency\_code:  return rate['rate']  print(f"Курс для валюти {currency\_code} не знайдено.")  except requests.exceptions.RequestException as e:  print("Помилка при з'єднанні з API НБУ:", e)  return None  def currency\_converter():  """Функція для конвертації валюти в гривні"""  print("Доступні валюти для конвертації: EUR, USD, PLN")  currency = input("Введіть код валюти для конвертації (EUR, USD, PLN): ").strip().upper()  amount = float(input("Введіть суму для конвертації: "))  if currency not in ['EUR', 'USD', 'PLN']:  print("Неправильний код валюти. Доступні валюти: EUR, USD, PLN.")  return  rate = get\_exchange\_rate(currency)  if rate is not None:  converted\_amount = amount \* rate  print(f"{amount} {currency} = {converted\_amount:.2f} UAH")  # Запускаємо конвертацію  currency\_converter() |

1. Використання модулів для програми калькулятор. Функції додавання, віднімання, множення та ділення перенести в файл functions.py. Функції запиту на введення даних для операцій та самих операцій перемістити в файл operations.py. Програму калькулятор реалізувати в файлі calc.py, до якого підключають файл functions.py та operations.py.

Хід виконання завдання:

* functions.py містить математичні функції для додавання, віднімання, множення та ділення. Для операції ділення додано обробку виняткової ситуації для випадку ділення на нуль.
* operations.py забезпечує запит чисел у користувача через функцію get\_numbers() та дозволяє вибирати математичну операцію через функцію select\_operation(), яка повертає відповідну функцію для обраної операції.
* calc.py є головним файлом програми. В ньому здійснюється виклик функцій з модулів operations та functions для виконання обраної операції, обробки результатів та повторення операцій за бажанням користувача.

Текст програм:

calc.py:

|  |
| --- |
| # calc.py  from operations import get\_numbers, select\_operation  def main():  print("Вітаємо у програмі 'Калькулятор'")  while True:  operation = select\_operation() # Вибір операції  a, b = get\_numbers() # Введення чисел для операції  try:  result = operation(a, b) # Виконання обраної операції  print(f"Результат: {result}")  except ValueError as e:  print(e)  next\_calculation = input("Бажаєте виконати іншу операцію? (так/ні): ")  if next\_calculation.lower() != 'так':  print("Дякуємо за використання калькулятора!")  break  if \_\_name\_\_ == "\_\_main\_\_":  main() |

|  |
| --- |
| # functions.py  def add(a, b):  return a + b  def subtract(a, b):  return a - b  def multiply(a, b):  return a \* b  def divide(a, b):  if b == 0:  raise ValueError("На нуль ділити не можна!")  return a / b |

|  |
| --- |
| # operations.py  from functions import add, subtract, multiply, divide  def get\_numbers():  """Запитує у користувача два числа."""  try:  a = float(input("Введіть перше число: "))  b = float(input("Введіть друге число: "))  return a, b  except ValueError:  print("Помилка: введіть коректне число.")  return get\_numbers()  def select\_operation():  """Запитує у користувача вибір операції та повертає відповідну функцію."""  print("Оберіть операцію:")  print("1. Додавання")  print("2. Віднімання")  print("3. Множення")  print("4. Ділення")  choice = input("Ваш вибір (1/2/3/4): ")  if choice == '1':  return add  elif choice == '2':  return subtract  elif choice == '3':  return multiply  elif choice == '4':  return divide  else:  print("Неправильний вибір. Спробуйте ще раз.")  return select\_operation() |

Звіт до Теми №6

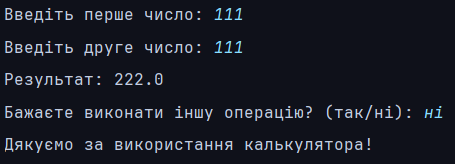
Робота з файлами

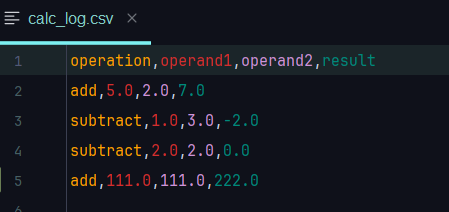
Під час виконання практичного завдання до Теми №6 було надано варіанти рішення до наступних задач:

1. Розробити механізм логування всіх дій, що виконує програма. Забезпечити зберігання інформації про введені данні, виконану операцію та результат виконання операції над даними.

Хід роботи:

У файлі functions.py додано механізм логування всіх дій калькулятора. Для цього була створена функція log\_to\_csv, яка зберігає інформацію про операцію, операнди та результат виконання в CSV-файл. Логування здійснюється кожного разу після виконання операції. Файл логів зберігає таку інформацію: тип операції, перше число, друге число та результат. Якщо файл порожній, до нього додаються заголовки.





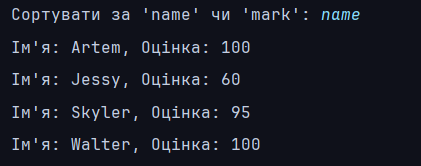
Текст програми:

|  |
| --- |
| import csv  LOG\_FILE = 'calc\_log.csv'  # functions.py  def add(a, b):  return a + b  def subtract(a, b):  return a - b  def multiply(a, b):  return a \* b  def divide(a, b):  if b == 0:  raise ValueError("На нуль ділити не можна!")  return a / b  def log\_to\_csv(operation\_name, a, b, result):  fieldnames = ['operation', 'operand1', 'operand2', 'result']  with open(LOG\_FILE, mode='a', newline='', encoding='utf-8') as file:  writer = csv.DictWriter(file, fieldnames=fieldnames)  if file.tell() == 0: # Якщо файл порожній, додаємо заголовки  writer.writeheader()  writer.writerow({  'operation': operation\_name,  'operand1': a,  'operand2': b,  'result': result  }) |

1. Маючи не відсортований список, елементами якого є словники з двома параметрами (ім’я та оцінка) виконати сортування списку, використовуючи стандартну функцію sorted(). Другим параметром для функції sorted() має бути lambda функція, що повертає ім’я або оцінку із елемента словника.

Хід роботи:

Для сортування списку студентів, що містить словники з ім'ям та оцінкою, використано функцію sorted() з параметром key, де в якості функції передано lambda вираз. Користувач має можливість вибрати, чи потрібно сортувати за ім'ям, чи за оцінкою. Після цього відсортований список виводиться на екран у відповідному форматі.



Текст програми:

|  |
| --- |
| import csv  names = []  with open("students.txt", "r") as file:  reader = csv.reader(file)  for row in reader:  names.append({'name': row[0], 'mark': int(row[1])})  key = input("Сортувати за 'name' чи 'mark': ").strip()  if key == 'name':  sorted\_names = sorted(names, key=lambda x: x['name'])  elif key == 'mark':  sorted\_names = sorted(names, key=lambda x: x['mark'])  else:  print("Неправильний вибір!")  sorted\_names = []  for name in sorted\_names:  print(f"Ім'я: {name['name']}, Оцінка: {name['mark']}") |

Звіт до Теми №7

Об'єктно-орієнтоване програмування

Під час виконання практичного завдання до Теми №7 було надано варіанти рішення до наступних задач:

1. Ознайомитись з документацією що описує можливості використання класів у мові Python <https://docs.python.org/3/tutorial/classes.html>

Я ознайомився з документацією Python, що описує використання класів. У цій документації надано основи роботи з класами, включаючи створення класу, ініціалізацію об'єктів за допомогою методу \_\_init\_\_(self), а також приклади використання різних методів класів. Важливою частиною документації є опис магічних методів, таких як \_\_str\_\_(self) для представлення об'єкта як рядка, та інших методів для реалізації стандартних операцій з об'єктами класів.

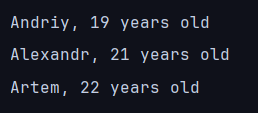
1. Ознайомитись з існуючими за замовченням методами класу по типу \_\_init\_\_(self) \_\_str\_\_(self)\_\_ та надати приклади використання.

знайомився з існуючими за замовченням методами класу, такими як \_\_init\_\_(self) і \_\_str\_\_(self).

* Метод \_\_init\_\_(self) є конструктором класу і викликається при створенні нового об'єкта. Він ініціалізує атрибути об'єкта.
* Метод \_\_str\_\_(self) використовується для повернення рядкового представлення об'єкта. Це дозволяє зробити об'єкти класів більш зручними для виведення на екран чи для логування.

1. Розробити клас **Student** атрибутами якого э два параметра **name** та **age**. Створити список елементами якого є об'єкти класу **Student**. Написати цикл який виводить на екран елементи списку у відсортованому порядку. Для сортування використати стандартну функцію **sorted**. Функція **sorted** має використовувати **lambda** функцію для визначення ключа сортування.

Я розробив клас Student, атрибутами якого є name (ім'я) та age (вік). Створив список об'єктів класу Student і за допомогою функції sorted() із lambda функцією для визначення ключа сортування, відсортував список за віком.



Текст програми:

|  |
| --- |
| class Student:  def \_\_init\_\_(self, name, age):  self.name = name  self.age = age  def \_\_str\_\_(self):  return f"{self.name}, {self.age} years old"  # Список студентів  students = [  Student("Artem", 22),  Student("Andriy", 19),  Student("Alexandr", 21)  ]  # Сортування студентів за віком  sorted\_students = sorted(students, key=lambda student: student.age)  # Виведення відсортованих студентів  for student in sorted\_students:  print(student) |

1. Використовуючи принципи ООП переписати програму Калькулятор. Завдання має бути виконано використовуючи модульний підхід.

Переписав програму калькулятора, використовуючи принципи об'єктно-орієнтованого програмування (ООП). Для цього я створив класи для кожної операції та загальний клас калькулятора. Програма виконана в модульному підході, що дозволяє зручно організувати код та розширювати функціональність.

Текст програми:

|  |
| --- |
| # operations.py  class Add:  def execute(self, a, b):  return a + b  class Subtract:  def execute(self, a, b):  return a - b  class Multiply:  def execute(self, a, b):  return a \* b  class Divide:  def execute(self, a, b):  if b == 0:  raise ValueError("На нуль ділити не можна!")  return a / b  # calc.py  from operations import Add, Subtract, Multiply, Divide  class Calculator:  def \_\_init\_\_(self):  self.operations = {  "1": Add(),  "2": Subtract(),  "3": Multiply(),  "4": Divide()  }  def get\_numbers(self):  try:  a = float(input("Введіть перше число: "))  b = float(input("Введіть друге число: "))  return a, b  except ValueError:  print("Помилка: введіть коректне число.")  return self.get\_numbers()  def select\_operation(self):  print("Оберіть операцію:")  print("1. Додавання")  print("2. Віднімання")  print("3. Множення")  print("4. Ділення")  choice = input("Ваш вибір (1/2/3/4): ")  return self.operations.get(choice)  def run(self):  while True:  operation = self.select\_operation()  if operation is None:  print("Неправильний вибір.")  continue  a, b = self.get\_numbers()  try:  result = operation.execute(a, b)  print(f"Результат: {result}")  except ValueError as e:  print(e)  next\_calculation = input("Бажаєте виконати іншу операцію? (так/ні): ")  if next\_calculation.lower() != 'так':  print("Дякуємо за використання калькулятора!")  break  if \_\_name\_\_ == "\_\_main\_\_":  calculator = Calculator()  calculator.run() |