МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

ІНСТИТУТ КОМП’ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ



Лабораторна робота №2

з дисципліни “Спеціалізовані мови програмування”

на тему

“Основи побудови об’єктно-орієнтованих

додатків на Python”

Виконав:

студент групи ІТ-31

Андрій КОЗУБЕНКО

Прийняв:

к.т.н.,

доц. кафедри ІСМ

Сергій ЩЕРБАК

Львів-2023

**Мета роботи:** Розробка консольного калькулятора в об’єктно орієнтованому стилі з використанням класів.

**План роботи**

Завдання 1: Створення класу Calculator

Завдання 2: Ініціалізація калькулятора

Завдання 3: Введення користувача

Завдання 4: Перевірка оператора

Завдання 5: Обчислення

Завдання 6: Обробка помилок

Завдання 7: Повторення обчислень

Завдання 8: Десяткові числа

Завдання 9: Додаткові операції

Завдання 10: Інтерфейс, зрозумілий для користувача

**Виконання роботи**

**Код програми:**

|  |
| --- |
| import os  import json  import uuid  class Calculator:      def \_\_init\_\_(self, logs\_path = 'logs.json'):          self.logs\_path = logs\_path          self.initialize\_logs\_file(self.logs\_path)          self.logs = self.get\_logs(self.logs\_path)      def add(self, augend, addend):          return augend + addend      def subtract(self, minuend, subtrahend):          return minuend - subtrahend      def multiply(self, multiplier, multiplicand):          return multiplier \* multiplicand      def divide(self, numerator, denominator):          if denominator != 0:              return numerator / denominator          else:              print('Division by zero is prohibited')      def power(self, base, exponent):          return pow(base, exponent)      def root(self, radicand, degree):          return radicand\*\*(1/degree)      def modulo(self, dividend, divisor):          return dividend % divisor      def get\_operator(self, valid\_operators = {'+', '-', '\*', '/'}):          operator = input('Select one operator ("+", or "-", or "\*", or "/", or "^", or "rt", or "%"): ')          if operator in valid\_operators:              return operator          else:              print('Invalid operator. Please try again')              return self.get\_operator(valid\_operators)      def get\_operand(self):          option = ''          operand\_retrieved = False          while operand\_retrieved == False:              option = input('\nSelect one option:\n[1] Enter a numeric value\n[2] Use the result from logs instead\n')              match option:                  case '1':                      try:                          return float(input('Enter a numeric value: '))                      except:                          print('Entered value is not numeric', end='')                          continue                  case '2':                      id = input('Enter the ID of the log: ')                      desired\_log = None                      for log in self.logs:                          if log['id'].startswith(id):                              desired\_log = log                              break                      if desired\_log == None:                          print('Log with such ID was not found', end='')                          continue                      else:                          print('Value retrieved:', desired\_log['result'])                          return desired\_log['result']                    case '\_':                      print('Wrong option. Please try again')      def get\_operands(self, operator, operators\_and\_operand\_names):          print('\nInput ', operators\_and\_operand\_names[operator][0], '(', operators\_and\_operand\_names[operator][2], ')')          first\_operand = self.get\_operand()          print('\nInput ', operators\_and\_operand\_names[operator][1], '(', operators\_and\_operand\_names[operator][2], ')')          second\_operand = self.get\_operand()          return [first\_operand, second\_operand]      def perform\_operation(self, operator, operands):          return self.operators\_and\_functions[operator](self, operands[0], operands[1])      def determine\_continuation(self):          decision = input('Continue? Y/N: ')          if decision.lower() == 'y' or decision.lower() == 'yes':              return True          return False      def initialize\_logs\_file(self, logs\_path = 'logs.json'):          if os.path.isfile(logs\_path) == False:              logs\_file = open(logs\_path, 'w')              logs\_file.write('[]')              logs\_file.close()      def get\_logs(self, logs\_path = 'logs.json'):          logs\_file = open(logs\_path)          logs\_string = logs\_file.read()          logs\_file.close()          if len(logs\_string) != 0:              return json.loads(logs\_string)          else:              return []      def append\_logs(self, operands, operator, result, logs):          logs.append({"id": str(uuid.uuid4()), "operands": operands, "operator": operator, "result": result})      def save\_logs\_file(self):          logs\_file = open(self.logs\_path, 'w')          logs\_file.write(json.dumps(self.logs))          logs\_file.close()      operators\_and\_functions = {          '+': add,          '-': subtract,          '\*': multiply,          '/': divide,          '^': power,          'rt': root,          '%': modulo      }      operators\_and\_operand\_names = {          '+':  ['augend', 'addend', 'augend + addend'],          '-':  ['minuend', 'subtrahend', 'minuend - subtrahend'],          '\*':  ['multiplier', 'multiplicand', 'multiplier \* multiplicand'],          '/':  ['numerator', 'denominator', 'numerator / denominator'],          '^':  ['base', 'exponent', 'base ^ exponent'],          'rt': ['radicand', 'degree', 'radicand ^ (1 / degree)'],          '%':  ['dividend', 'divisor', 'dividend mod divisor']      }      def perform\_calculation(self):          continue\_calculation = True          while continue\_calculation == True:              operator = self.get\_operator(list(self.operators\_and\_functions.keys()))              operands = self.get\_operands(operator, self.operators\_and\_operand\_names)              result = self.perform\_operation(operator, operands)              print('Result: ', result)              self.append\_logs(operands, operator, result, self.logs)              continue\_calculation = self.determine\_continuation()      def display\_logs(self):          print('\n\nLogs:')          for log in self.logs:              print('id:', log['id'])              print('expression:', log['operands'][0], log['operator'], log['operands'][1], '=', log['result'], end='\n\n')      def load\_logs(self):          path = input('Specify path to logs file: ')          if os.path.isdir(path) == True:              print('Specified file is a directory. Try again')              self.load\_logs()          if os.path.isfile(path) == True:              self.logs = self.get\_logs(path)              self.logs\_path = path              print('Successfully retrieved the logs')          else:              print('Specified file does not exist')  def main():      option = ''      calculator = Calculator()      while option != '0':          option = input('\nSelect one option:\n[1] - Perform calculation\n[2] - Display logs\n[3] - Load logs\n\n[0] - Exit\n')          match option:              case '1':                  calculator.perform\_calculation()              case '2':                  calculator.display\_logs()              case '3':                  calculator.load\_logs()              case '0':                  calculator.save\_logs\_file()  main() |

**Посилання на GitHub:** <https://github.com/Famezzs/uni-spec-lang-labs.git>

**Висновок:** Виконавши ці завдання, я перетворив консольний калькулятор у об'єктно-орієнтований калькулятор, використовуючи класи в Python. Цей проект допоміг мені вивчити концепції об'єктно-орієнтованого програмування та організацію, зберігаючи функціональність і інтерфейс користувача калькулятора.