Лабораторна Робота

Розробка програм з використанням процедур і функцій.

**Мета лабораторної роботи:**

Метою цієї лабораторної роботи є вивчення та практичне використання концепцій процедурного програмування в мові програмування Python. Студенти отримають навички розробки програм, які використовують процедури і функції для структуризації та полегшення обслуговування коду.

**Теоретичні відомості:**

**Інструкція `def`** використовується для створення об'єкта функції та його пов'язання з ім'ям. Загальний формат інструкції виглядає наступним чином:

def ім'я(arg1, arg2,... , argN):

# блок інструкцій (тіло функції)

<statements>

Інструкція `def` має рядок заголовка та наступний за ним блок інструкцій, який зазвичай відокремлюється відступами. Блок інструкцій представляє собою тіло функції - код, який виконується при кожному виклику функції. У рядку заголовка вказується ім'я функції та список аргументів у круглих дужках.

Часто тіло функції містить інструкцію `return`:

def ім'я(arg1, arg2,... , argN):

# тіло функції

...

return значення

**Інструкцію `return`** можна розмістити в будь-якому місці тіла функції. Вона завершує виконання функції та повертає результат програмі, яка її викликала. `return` містить об'єктний вираз, який представляє собою результат функції. Ця інструкція є необов'язковою - якщо вона відсутня, виконання функції завершується, коли потік управління доходить до кінця тіла функції. Функція без інструкції `return` повертає об'єкт `None`, проте це значення зазвичай ігнорується.

Визначення функції відбувається під час виконання програми, і тому імена функцій не мають особливого значення. Основною роллю визначення функції є створення об'єкта функції та його пов'язання з ім'ям. Функції є звичайними об'єктами, які зберігаються в пам'яті під час виконання програми. Крім підтримки виклику, функції можуть мати атрибути для зберігання інформації для подальшого використання.

def func():

pass

othername = func

othername()

value = 'some value'

def func():

pass

func()

func.attr = value

print(func.\_\_dict\_\_)

**Щодо параметрів за замовчуванням**, їх можна використовувати для встановлення значень, які будуть використовуватися, якщо вони не передаються явно при виклику функції. Наприклад, у функції `ask\_ok` параметри `retries` та `complaint` мають значення за замовчуванням.

from pip.\_vendor.distlib.compat import raw\_input

def ask\_ok(prompt, retries=4, complaint='Yes or no, please!'):

"""Функція ask\_ok із параметрами за замовчуванням"""

while retries > 0:

ok = raw\_input(prompt)

if ok.lower() in ('т', 'так', 'yes'):

return True

if ok.lower() in ('н', 'ні', 'no', 'nope'):

return False

retries -= 1

print(complaint)

raise IOError('Помилка')

result = ask\_ok("Прошу ввести:", 2, "Так чи ні, будь ласка:")

**Документування функцій**:

Добрим стилем є документування кожної функції. Для цього важливо включити короткий опис функції в наступному рядку, що йде одразу після заголовка. Цей опис слід укладати у потрійні апострофи або лапки. Для отримання інформації про функцію можна використовувати інструкцію `print(ім'я\_функції.\_\_doc\_\_)`.

**Передавання у функцію змінної кількості аргументів:**

Часто використовується передавання змінного числа аргументів у функцію. У Python це можна здійснити за допомогою символу `\*` перед списком аргументів змінної довжини. Перед списком аргументів може бути один або кілька обов'язкових аргументів.

def fprintf(message, \*args):

print(message, '{}'.format(args))

fprintf('Аргументи', 1)

fprintf('Аргументи', 1, 'sun')

fprintf('Аргументи', 1, [1, 2, 3])

**Функції lambda:**

Lambda-вирази у Python дозволяють створювати об'єкти функцій у формі виразів. Вони використовуються для створення малих функцій або для відкладення виконання фрагментів коду. Вони є анонімними функціями, створеними за допомогою ключового слова `lambda`.

Приклад використання lambda-виразу

f = lambda x, y, z: x + y + z

print(f(2, 3, 4)) #

Lambda-вирази дозволяють створювати короткі функції на льоту, що може бути корисно в певних ситуаціях.

# Створення списку з lambda-виразами

L = [lambda x: x \*\* 2, lambda x: x \*\* 3, lambda x: x \*\* 4]

for f in L:

print(f(2)) # Виведе 4, 8, 16

print(L[0](3)) # Виведе 9

**Завдання для лабораторної роботи:**

### **Варіант 1:**

* **Додати можливість зберігання додаткової інформації про студента:**
  + Запитати в користувача про факультет чи курс.
  + Додати цю інформацію до даних про кожного студента.
* **Вивести інформацію про студентів на конкретному курсі:**
  + Створити функцію, яка приймає номер курсу і виводить імена, прізвища та рейтинг студентів, які вивчають на цьому курсі.
* **Виправлення даних про студента:**
  + Реалізувати можливість зміни даних про студента (наприклад, рейтинг чи факультет).

### **Варіант 2:**

* **Зберігання дати народження:**
  + Розширити можливості додавання студентів, щоб зберігати їхню дату народження.
  + Вивести студентів, які святкують день народження в певний період.
* **Сортування студентів за рейтингом:**
  + Створити функцію для сортування студентів за їхнім рейтингом в спадаючому порядку.
* **Обчислення віку студента:**
  + Додати функцію для обчислення віку студента на основі його дати народження.

### **Варіант 3:**

* **Робота з файлами:**
  + Зберігати та завантажувати дані про студентів у текстовий файл.
  + Створити функції для читання та запису даних у файл.
* **Пошук студента за ім'ям чи прізвищем:**
  + Реалізувати функцію для пошуку та виведення інформації про студента за його ім'ям чи прізвищем.
* **Статистика за рейтингом:**
  + Розрахувати середній рейтинг та вивести студента з найвищим та найнижчим рейтингом.

### **Варіант 4:**

* **Використання lambda-функцій:**
  + Замість звичайних функцій використовуйте lambda-функції для обчислень.
* **Групування студентів за факультетами:**
  + Створити функцію, яка групує студентів за факультетами та виводить кількість студентів на кожному факультеті.
* **Створення генератора випадкових даних:**
  + Створити генератор, який автоматично створює дані про студентів з випадковими іменами, прізвищами та рейтингами.