

МІНІСТУЕРСТВО ОСВІТИ І НАУКИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

ЗВІТ

з виконання лабораторної роботи №2

з дисципліни «Функціональне програмування»

за темою «Функції вищого порядку. Знайомство з можливостями функціонального
програмування мови Python»

Виконав:

Ковалик Вадим Валерійович

Перевірив:

Міхаль Олег Пилипович

Харків 2024

2.1 Мета роботи

Знайомство з прийомами функціонального програмування щодо застосування функцій вищого порядку на прикладі мовних можливостей мови Python.

2.2 Порядок виконання роботи

2.2.1 Завдання:

На даній лабораторній роботі необхідно ознайомитися з прийомами функціонального програмування та використання функцій вищого порядку на прикладі мови Python. Передбачається вивчення інтерфейсу Python, засвоєння основ синтаксису, зокрема типів даних, базових операцій, роботи зі списками та функціями. Також потрібно освоїти методи діалогового налагодження елементів програми.

2.2.2 Хід роботи

2.2.2.1 Функціональне програмування в Python

Функціональне програмування в Python базується на використанні функцій вищого порядку, рекурсії, обробці списків та лінійних обчислень. Функції `map`, `filter` та `reduce` спрощують роботу з колекціями, а генератори забезпечують відкладене обчислення, підвищуючи ефективність при роботі з великими даними. Python дозволяє гармонійно поєднувати функціональний підхід з іншими парадигмами, що робить код більш виразним і гнучким.

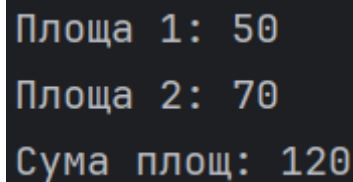
2.2.2.2 Визначення та використання функцій

У Python функції визначаються через оператор `def` або лямбда-вирази. Аргументи можуть мати значення за замовчуванням, які вказуються після обов'язкових. Під час виклику позиційні аргументи задаються за порядком, а іменовані – у будь-якому. Функція повертає одне значення, яке може бути кортежем для передачі кількох значень одночасно. Лямбда-вирази зручні для коротких функцій, а функції є об'єктами першого класу, тобто можуть використовуватися як інші типи даних.

Лістинг 2.1 – Використання функцій

```
def calculate_area(length, width=5):  
    return length * width  
areal = calculate_area(10)
```

```
area2 = calculate_area(10, 7)
result = (lambda x, y: x + y)(area1, area2)
print("Площа 1:", area1)
print("Площа 2:", area2)
print("Сума площ:", result)
```



```
Площа 1: 50
Площа 2: 70
Сума площ: 120
```


Рисунок 2.1 – Результат використання функцій

2.2.2.3 Списочні вирази

Списочні вирази – це потужний інструмент Python, що дозволяє створювати списки на основі існуючих послідовностей, використовуючи компактний і зрозумілий синтаксис. Замість традиційних циклів `for` можна створити новий список, застосувавши вираз до кожного елемента початкової послідовності. Списочні вирази дозволяються включати умови, відфільтровуючи елементи за певними критеріями, що робить їх особливо зручними для створення списків на основі правил або фільтрацій.

Лістинг 2.2 – Використання списочних виразів

```
cubes = [x**3 for x in range(1, 11) if x % 3 == 0]
print(cubes)
```



```
[27, 216, 729]
```

Рисунок 2.2 – Результат використання списочних виразів

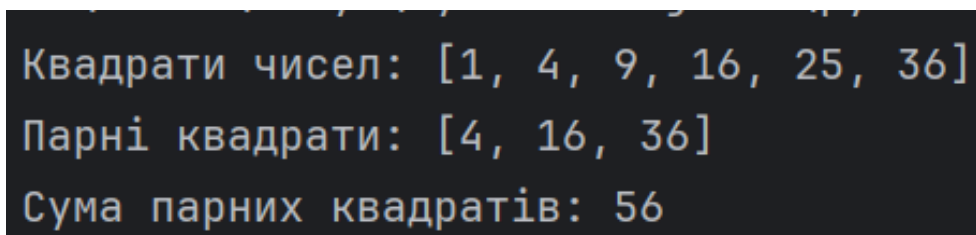
2.2.2.4 Вбудовані функції вищих порядків

Вбудовані функції вищих порядків у Python, такі як `map()`, `filter()`, `reduce()` та `apply()`, дозволяють обробляти послідовності з використанням інших функцій як аргументів. `Map()` застосовує задану функцію до кожного елемента послідовності, повертаючи нову послідовність результатів. `Filter()` створює послідовність із

елементів, які відповідають певній умові. `Reduce()` виконує послідовні обчислення на елементах списку, акумулюючи результат, а `apply()` застосовує функцію до списку позиційних і словника іменованих аргументів. Ці функції дозволяють елегантно вирішувати задачі обробки та трансформації даних.

Лістинг 2.3 – Використання вбудованих функцій

```
from functools import reduce
numbers = [1, 2, 3, 4, 5, 6]
squared_numbers = list(map(lambda x: x**2, numbers))
even_numbers = list(filter(lambda x: x % 2 == 0, squared_numbers))
sum_even_squares = reduce(lambda x, y: x + y, even_numbers)
print("Квадрати чисел:", squared_numbers)
print("Парні квадрати:", even_numbers)
print("Сума парних квадратів:", sum_even_squares)
```



```
Квадрати чисел: [1, 4, 9, 16, 25, 36]
Парні квадрати: [4, 16, 36]
Сума парних квадратів: 56
```

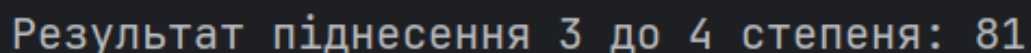
Рисунок 2.3 – Результат виконання вбудованих функцій

2.2.2.5 Замикання

Замикання в Python – це функції, що визначені всередині інших функцій і зберігають доступ до змінних із зовнішньої області видимості навіть після завершення роботи зовнішньої функції. Це дозволяє створювати функції, які «пам'ятають» певні значення, використовуючи їх у подальших обчисленнях.

Лістинг 2.4 – Будова замикань в Python

```
def power(base):
    def exponent(exp):
        return base ** exp
    return exponent
power_of_3 = power(3)
print("Результат піднесення 3 до 4 степеня:", power_of_3(4))
```



```
Результат піднесення 3 до 4 степеня: 81
```

Рисунок 2.4 – Вигляд результату замикання

2.2.2.6 Ітератори

Ітератори забезпечують ефективний спосіб обробки послідовностей, дозволяючи перебирати елементи без необхідності завантажувати їх у пам'ять. Модуль `itertools` надає додаткові інструменти для роботи з ітераторами, що підтримують складні обчислення та трансформації послідовностей у функціональному стилі. Серед можливостей — об'єднання, групування, створення безперервних циклів, побудова комбінацій, а також фільтрація та сортування.

Лістинг 2.5 – Складена програма, яка використовує ітератори

```
from itertools import chain, groupby
from math import cos
merged_sequence = list(chain("ABC", "DEF"))
print("Об'єднана послідовність:", merged_sequence)
values = [cos(x * 0.4) for x in range(10)]
grouped_values = [list(group) for key, group in groupby(values, lambda x: x > 0)]
print("Груповані значення:", grouped_values)
```

```
Об'єднана послідовність: ['A', 'B', 'C', 'D', 'E', 'F']
Груповані значення: [[1.0, 0.9210609940028851, 0.6967067093471654, 0.3623577544766734], [-0.02919
```

Рисунок 2.5 – Використання ітераторів

2.2.2.7 Модуль `functools`

Модуль `functools` надає додаткові функціональні можливості, зокрема часткове застосування функцій. Це дозволяє створити нову функцію, зафіксувавши деякі аргументи вихідної функції, що особливо корисно для часто повторюваних операцій або складних функцій із фіксованими параметрами.

Лістинг 2.6 – Приклад часткового застосування функції

```
from functools import partial
def multiply(a, b):
    return a * b
multiply_by_5 = partial(multiply, 5)
print("Часткове застосування функції(5*3): ", multiply_by_5(3))
print("Часткове застосування функції(5*10): ", multiply_by_5(10))
```

```
Часткове застосування функції(5*3): 15
Часткове застосування функції(5*10): 50
```

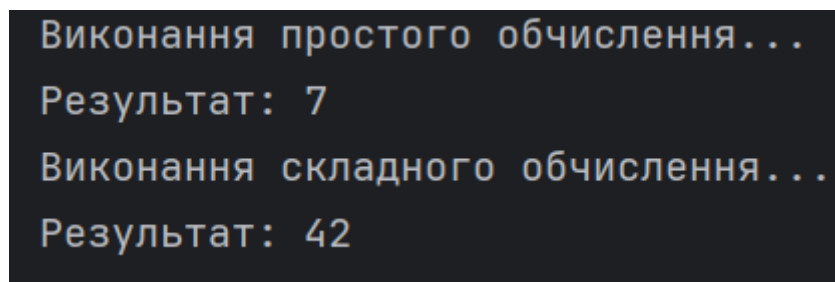
Рисунок 2.6 – Результат використання

2.2.2.8 Лінійні обчислення

Python підтримує лінійні обчислення, дозволяючи відкладати виконання виразів до моменту, коли їхній результат потрібен. Це досягається за допомогою логічних операторів `or` і `and`, лямбда-виразів, генераторів та генераторних виразів, а також умовних `if`-виразів, які обчислюють лише потрібний операнд.

Лістинг 2.7 – Приклад лінійного обчислення в `if`-виразі

```
def expensive_calculation():
    print("Виконання складного обчислення...")
    return 42
def simple_calculation():
    print("Виконання простого обчислення...")
    return 7
result = simple_calculation() if True else expensive_calculation()
print("Результат:", result)
result = simple_calculation() if False else expensive_calculation()
print("Результат:", result)
```



```
Виконання простого обчислення...
Результат: 7
Виконання складного обчислення...
Результат: 42
```

Рисунок 2.7 – Результат виконання лінійного обчислення

2.2.2.9 Функтори

Функтори в Python - це об'єкти, які можуть бути викликані як функції, оскільки вони перевантажують оператор виклику `()` за допомогою методу `__call__`. Вони аналогічні функціям, але мають особливості, які дають змогу використовувати їх як колбеки і замінювати деякі патерни замикань. Функтори можуть зберігати стан і надають усі можливості об'єктно-орієнтованого програмування, що робить їх корисними у функціональному програмуванні.

Функтори також підтримують ледачі обчислення, комбінуючи наявні функції для створення нових, при цьому результати обчислюються тільки в міру необхідності. Це дає змогу оптимізувати виконання програми та уникнути дорогих обчислень, якщо вони не потрібні.

Лістинг 2.8 – Приклад використання функтаторів

```
class Add:
    def __init__(self, value):
        self.value = value

    def __call__(self, x):
        return self.value + x

add_five = Add(5)
result = add_five(10)
print(result)
```




Рисунок 2.8 – Результат виконання коду

2.3 Висновок

Функціональне програмування в Python є потужним інструментом для обробки даних і написання елегантного коду. Завдяки функціям вищого порядку, лямбда-виразам, списковим включенням та модулям, як-от `functools` і `itertools`, програмісти можуть реалізовувати складні алгоритми просто і зрозуміло. Функтори, які є об'єктами, що викликаються як функції, додають гнучкість, сприяючи розробці ефективних і масштабованих рішень.