Practice Session 8

Зміст

📚 Вступ до функцій:

На сьогоднішній практиці ми попрактикуємося працювати з функціями - одним з найважливіших інструментів для організації та повторного використання коду в Python.

Функції дозволяють нам розбивати складні задачі на менші, більш керовані частини, що робить наш код чистішим і легшим для розуміння.

Вони також дозволяють нам уникнути повторення коду, що є однією з основних принципів програмування - Don't Repeat Yourself.

📌 План заняття:

- 💪🏻 Warm Up
- 🛠️ Функції, як інструмент для організації коду (створення, рітьорн, аргументи та параметри)
  - 👩‍💻 Live Coding 1: Робот кур'єр
- 🔃 Рекурсивні функції
  - 👩‍💻 Live Coding 2: Факторіали та рекурсії

- ❔ Анонімні функції та λ-вирази
  - 👨‍💻 Live Coding 3: Обробка даних - від класичних функцій до λ-виразів
- ⚙️ Завдання для самостійного опрацювання

📝 Functions Cheat Sheet

🛠️ 1. Анатомія та базові концепти

Створення: Використовуємо ключове слово def, ім'я (зрозуміле, що робить функція) та дужки.

Параметри vs Аргументи:

- Параметр — це змінна-заглушка в оголошенні функції (напр., `def get_area(radius):`). Це порожня коробка, яка чекає на дані.
- Аргумент — це реальне значення, яке ми передаємо під час виклику (напр., `get_area(5)`). Це те, що ми в цю коробку кладемо.

Return:

- Функція має віддавати результат своєї роботи через return, а не виводити його через `print()`.
- Раннє повернення (Early Return): Команда `return` миттєво зупиняє роботу всієї функції. Це ідеально підходить для "відсіювання" невалідних даних на самому початку (наприклад, `if price < 0: return "Помилка"`).

Scope (Область видимості):

- Глобальні змінні/константи: Створюються поза функціями. Їх бачать усі (добре підходить для загальних налаштувань, напр., PI = 3.14).
- Локальні змінні: Створюються всередині функції. Вони народжуються під час виклику і назавжди зникають після return. Спроба викликати таку змінну ззовні призведе до помилки.

🔄 2. Рекурсивні функції

Рекурсія — це підхід, при якому функція викликає саму себе. Щоб вона працювала і не "поклала" програму помилкою переповнення пам'яті (Stack Overflow), вона обов'язково має складатися з двох частин:

- База рекурсії: Умова виходу. Ситуація, при якій функція перестає викликати себе і просто повертає конкретне значення (наприклад, факторіал нуля дорівнює 1: `if n == 0: return 1`).
- Рекурсивний крок: Виклик цією ж функцією самої себе, але зі зміненими аргументами, які з кожним кроком наближають нас до бази (наприклад, `return n * factorial(n - 1)`).

🥷 3. Анонімні функції (Lambda-вирази)

Це міні-функції, які пишуться в один рядок і не мають імені. Їх використовують як "одноразовий" інструмент, коли писати повноцінну def немає сенсу.

- Синтаксис: `lambda arguments: expression`
- Головне застосування: Передача лямбди як правила в інші функції, такі як `sort()`, `map()`, `filter()`.

```
# Приклад: звичайне сортування вишикує слова за алфавітом.
# Але ми можемо передати лямбду як ключ (key), щоб відсортувати їх за довжиною:

words = ["яблуко", "кіт", "університет", "код"]
words.sort(key=lambda word: len(word))
print(words)
```

```
['кіт', 'код', 'яблуко', 'університет']
```

🛠️ Функції, як інструмент для організації коду

Функції в Python - це потужний інструмент для організації та повторного використання коду. Вони дозволяють нам розбивати складні задачі на менші, більш керовані частини, що робить наш код чистішим і легшим для розуміння.

```
def function_name(parameters):
    # Тіло функції
    ...
    # return expression (не обов'язково)
```

👨‍💻 Live Coding 1: Робот кур'єр

Контекст: Складський робот переміщується між стелажами по ортогональній сітці (може їхати тільки прямо і повертати на 90 градусів). Тому відстань між ним та ціллю обчислюється не по прямій, а за формулою мангеттенської відстані: $d = | x_1 - x_2 | + | y_1 - y_2 |$.

На кожну пройдену комірку сітки робот витрачає 1.5% заряду батареї. Необхідно написати функцію, яка розрахує загальну витрату заряду на поїздку від поточної позиції до заданої комірки.

Вхідні дані:

- Координати старту: `x_start=2`, `y_start=2`
- Координати цілі: `x_target=5`, `y_target=6`.

Очікуваний результат: Відсоток витраченої батареї. Якщо передані некоректні координати — повідомлення про помилку.

🛠️ Guided Practice:

Контекст: Наш робот отримує нове замовлення на доставку товару. Ми можемо порахувати скільки відсотків заряду витратить робот на доставку, але що якщо йому не вистачить заряду для виконання замовлення та повернення на базу? (координати бази завжди 0, 0)

Напишіть функцію, яка визначить, чи вистачить роботу поточного заряду на весь цей маршрут.

Вхідні дані:

- Поточні координати робота: `current_x=2`, `current_y=2`.
- Координати цілі: `target_x=5`, `target_y=6`.
- Поточний рівень заряду (%): `current_battery=50`

Очікуваний результат: `True` або `False` — чи вистачить заряду.

> 💡 Порада
>
> Для обчислення загальної відстані, яку має проїхати робот, можна скористатися функцією, яку ми написали в попередньому завданні для розрахунку відсотка витраченої батареї. Пам'ятайте, що робот має повернутися на базу після доставки товару, тому потрібно врахувати відстань від поточної позиції до цілі та від цілі до бази.

🔃 Рекурсивні функції

Рекурсія - це техніка програмування, при якій функція викликає сама себе для розв'язання задачі. Вона складається з двох основних компонентів: базового випадку, який зупиняє рекурсію, та рекурсивного кроку, який розбиває задачу на менші підзадачі.

Вона часто використовується для розв'язання задач, які мають природну рекурсивну структуру, таких як обчислення факторіала, фібоначчі, обходу дерев та алгоритмів типу "розділяй та володарюй".

```python
def fibonacci_recursive(n):
    if n <= 1:  # Базовий випадок: F(0) = 0, F(1) = 1
        return n
    else:
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)

print(fibonacci_recursive(10))
```

```
55
```

👨‍💻 Live Coding 2: Факторіали та рекурсії

Контекст: Багато математичних операцій можна реалізувати за допомогою простих рекурсивних функцій. Однією з таких тривіальних задач = обчислення факторіала числа `n`, який визначається як добуток усіх цілих чисел від 1 до `n` включно.

Вхідні дані: Ціле число `n`, введене користувачем

Очікуваний результат: Результат обчислення $n!$ або повідомлення про помилку, якщо `n` є від'ємним числом.

🛠️ Guided Practice: Піднесення до степеня

Контекст: Ще однією класичною задачею, яка ілюструє концепцію рекурсії, є піднесення числа `x` до степеня `n`. Це можна реалізувати за допомогою рекурсивної функції, яка базується на визначенні степеня: $x^n = x \cdot x^{n-1}$ для `n > 0`

Вхідні дані: Два числа: основа `x` та показник степеня `n`, введені користувачем.

Очікуваний результат: Результат обчислення x^n або повідомлення про помилку, якщо `n` є від'ємним числом.

> 💡 Порада
>
> Перед написанням рекурсивної функції, подумайте, яким буде базовий випадок та крок рекурсії

❔ Анонімні функції та λ-вирази

Анонімні функції, або λ-вирази, - це компактний спосіб створення функцій без імені. Вони особливо корисні, коли нам потрібно передати функцію як аргумент до іншої функції, наприклад, для сортування або фільтрації даних.

Найчастіше λ-вирази використовуються з вбудованими функціями, такими як `sorted()`, `map()`, `filter()`, де вони дозволяють визначити логіку обробки даних без необхідності створювати окрему функцію за допомогою `def`.

👨‍💻 Live Coding 3: Обробка даних - від класичних функцій до λ-виразів

Контекст: Працюючи з даними, часто виникає потреба у швидкому редагуванні цих елементів за певним правилом. Наприклад, маючи список температур у Цельсіях, ми хочемо перетворити їх у Фаренгейти.

Вхідні дані: Список температур у Цельсіях: `celsius = [0, 20, 30, 100]`

Очікуваний результат: Список температур у Фаренгейтах: `fahrenheit = [32.0, 68.0, 86.0, 212.0]`

🛠️ Guided Practice: Бортовий журнал робота

Контекст: Наш робот кур'єр з попередніх завдань веде журнал своїх поїздок, де кожен запис містить відсоток заряду, витрачений на доставку.

Ми хочемо швидко проаналізувати ці дані, спочатку відсортувавши їх за витратою заряду, а потім знайти всі поїздки, де витрачено більше 30% заряду.

Вхідні дані: Список витрат заряду на поїздки: `battery_usage = [25, 40, 15, 35, 10]`

Очікуваний результат: Список відсортованих витрат заряду: `[10, 15, 25, 35, 40]`, список поїздок з витратою більше 30%: `[35, 40]`

> ⚠️ Обмеження
>
> Для цієї задачі, використовуйте λ-функції та вбудовані функції `sorted()` та `filter()`, а не цикли чи інші методи.