Звіт з практичної роботи No9

Тема роботи: Створення універсальних функцій

Виконав: Кравченко Сергій, ТЦР-12

Дата: 02.06.2025

Мета роботи:

Розробити набір універсальних функцій на мові Python, які можуть гнучко

обробляти різні вхідні дані та виконувати схожі операції над ними,

мінімізуючи дублювання коду та підвищуючи його повторне використання.

Хід роботи:

Постановка завдання:

1. Функція для обробки колекцій даних:

- Створіть функцію під назвою `process\_data` яка б могла приймати

різні типи колекцій даних (наприклад, списки, кортежі, можливо навіть

словники) та застосовувати до їх елементів певну операцію.

- Функція повинна приймати щонайменше два аргументи:

- `data`: Сама колекція даних, яку потрібно обробити.

- `operation`: Функція, яку необхідно застосувати до кожного

елемента колекції `data`. Ця функція повинна приймати один аргумент

(елемент колекції) та повертати результат обробки. ( згадують можливість

передавати функції як аргументи).

- Функція `process\_data` повинна повертати нову колекцію (того ж

типу, що й вхідна `data`), яка містить результати застосування `operation`

до кожного елемента вхідної колекції.

- Обробка різних типів колекцій: Забезпечте, щоб функція коректно

працювала зі списками та кортежами. Для словників ви можете вирішити

обробляти лише ключі, лише значення, або обидва (можливо, передаючи

додатковий параметр, який вказує, що саме обробляти).

2. Функція для фільтрації даних:

- Створіть функцію під назвою `filter\_data` яка б приймала колекцію

даних та функцію-предикат.

- Функція повинна приймати два аргументи:

- `data`: Колекція даних для фільтрації.

- `predicate`: Функція, яка приймає один аргумент (елемент колекції)

та повертає булеве значення (`True` якщо елемент повинен бути включений

до результату, `False` інакше).

- Функція `filter\_data` повинна повертати нову колекцію, що містить

лише ті елементи з вхідної колекції, для яких функція `predicate` повернула

`True`. Знову ж таки, тип вихідної колекції має відповідати типу вхідної.

3. Функція з динамічною кількістю аргументів:

- Створіть функцію під назвою `combine\_values` яка б могла

приймати довільну кількість числових або рядкових аргументів та

об'єднувати їх. ( згадують `\*args` для обробки довільної кількості

позиційних аргументів).

- Функція повинна визначати тип першого переданого аргументу та

намагатися об'єднати всі наступні аргументи відповідно до цього типу

(додавання для чисел, конкатенація для рядків).

- Передбачте можливість передачі необов'язкового

параметра-роздільника (для рядків) або початкового значення (для чисел),

використовуючи аргументи за замовчуванням. ( описує аргументи за

замовчуванням).

4. Обробка помилок та винятків:

- У всіх створених функціях використовуйте блоки `try-except` для

обробки можливих помилок, таких як:

- Передача функції неправильного типу в якості аргументу

`operation` або `predicate`.

- Спроба об'єднання значень різних несумісних типів у функції

`combine\_values`.

- Інші потенційні помилки, пов'язані з обробкою даних.

- У разі виникнення помилки, функції повинні повертати

інформативне повідомлення про помилку або обробляти її іншим

прийнятним способом (наприклад, пропускати проблемні елементи).

5. Документування функцій:

- Для кожної створеної функції додайте докстрінг, який би чітко

описував її призначення, параметри та значення, що повертаються. (

підкреслює важливість документування за допомогою докстрінгів).

Код програми:

def process\_data(data, operation):

if isinstance(data, list):

return [operation(x) for x in data]

elif isinstance(data, tuple):

return tuple(operation(x) for x in data)

elif isinstance(data, dict):

return {k: operation(v) for k, v in data.items()}

return data

def filter\_data(data, predicate):

if isinstance(data, list):

return [x for x in data if predicate(x)]

elif isinstance(data, tuple):

return tuple(x for x in data if predicate(x))

elif isinstance(data, dict):

return {k: v for k, v in data.items() if predicate(v)}

return data

def combine\_values(\*args):

if not args:

return None

if all(isinstance(x, (int, float)) for x in args):

return sum(args)

return ''.join(str(x) for x in args)