## Лабораторна робота 5

**Тема:** Функції та модулі.

**Мета:** Навчитися проєктувати функції з явними параметрами й поверненими результатами та повторно використовувати їх у модулі.

### Зміст роботи

#### Завдання 1. Параметри та повернення результату

Створіть дві функції: перша очищує крайні пробіли імені й застосовує title(); друга формує підпис з імені та групи, використовуючи першу функцію. Роздільник між ім’ям і групою — параметр зі значенням за замовчуванням « — ».

1. Введіть ім’я та групу в головній програмі. Виведіть підпис із типовим роздільником.
2. Повторіть виклик з іменованим аргументом для роздільника « / ». Функції мають повертати рядки й не виконувати input() або print().

Перевірте власні дані та покажіть обидва способи виклику. В усіх завданнях введення вважається коректним; обробка помилок не потрібна. У таблиці лапки позначають межі рядка й не входять до введення.

Таблиця 1 — Контрольна перевірка параметрів функції

| Вхід | Спосіб виклику | Очікуване виведення |
|---|---|---|
| Ім’я: «  оЛЕНА  »<br>Група: КН-02 | Без аргументу роздільника | Олена — КН-02 |
| Ім’я: «  оЛЕНА  »<br>Група: КН-02 | Іменований роздільник « / » | Олена / КН-02 |

#### Завдання 2. Індивідуальне опрацювання даних функціями

Розв’яжіть задачу свого варіанта таблиці 2. Розділіть її на 2–3 змістовні функції, самостійно обравши назви, параметри та розподіл роботи.

1. Введіть дані в головній програмі. Передавайте їх функціям явно; кожна функція повертає свій результат.
2. Обчисліть і виведіть усі результати варіанта. Не змінюйте передані колекції й не використовуйте глобальний стан.
3. Перевірте контрольний приклад, власні дані й порожній набір.

Числа — цілі, слова — літери без розділових знаків, їхній регістр зведіть до нижнього. Числа й слова вводьте одним рядком. Для пар введіть $n \geq 0$, потім n рядків назва;число; назви нормалізуйте до нижнього регістру. Порожні результати — [], {}, 0 відповідно до типу; найдовша серія порожнього набору — 0. Зберігайте порядок появи, якщо не задано інше; ключі словників виводьте за зростанням.

Створюйте лише групи, які мають дані. У контрольному стовпці окремо подано вхідні дані та очікувані результати. Для записів спочатку вказано n, далі — кожний запис з нового рядка. Підписи під час виведення можна обрати самостійно.

Таблиця 2 — Індивідуальні задачі для декомпозиції

| **Варіант** | **Вхідні дані** | **Потрібні результати** | **Контрольний приклад** |
|---|---|---|---|
| 1 | Рядок цілих чисел | Унікальні значення в порядку появи та їхні частоти. | **Вхід:**<br>2 1 2 3 1<br>**Виведення:**<br>Унікальні: [2, 1, 3]<br>Частоти: {1: 2, 2: 2, 3: 1} |
| 2 | Рядок цілих чисел | Стиснути сусідні повторення та знайти довжину найдовшої серії. | **Вхід:**<br>2 2 1 1 1 2<br>**Виведення:**<br>Стиснутий список: [2, 1, 2]<br>Найдовша серія: 3 |
| 3 | Рядок цілих чисел | Пари (значення, довжина серії) для всіх серій і кількість серій. | **Вхід:**<br>1 1 3 3 3<br>**Виведення:**<br>Серії: [(1, 2), (3, 3)]<br>Кількість серій: 2 |
| 4 | Рядок слів | Групи за першою літерою та перше найдовше слово кожної групи. | **Вхід:**<br>код кіт дані книга<br>**Виведення:**<br>Групи: {'д': ['дані'], 'к': ['код', 'кіт', 'книга']}<br>Найдовші слова: {'д': 'дані', 'к': 'книга'} |
| 5 | Рядок слів | Частоти та список слів, що трапилися рівно один раз, у порядку появи. | **Вхід:**<br>код мова код дані<br>**Виведення:**<br>Частоти: {'дані': 1, 'код': 2, 'мова': 1}<br>Рівно один раз: ['мова', 'дані'] |

| Варіант | Вхідні дані | Потрібні результати | Контрольний приклад |
|---|---|---|---|
| 6 | Рядок цілих чисел | Для кожної пари сусідніх чисел знайти різницю «наступне − попереднє» та кількість строго зростаючих переходів. | **Вхід:**<br>3 5 5 2<br>**Виведення:**<br>Різниці: [2, 0, -3]<br>Зростаючі переходи: 1 |
| 7 | Рядок цілих чисел | Суми префіксів та найбільша з них; порожній набір → 0. | **Вхід:**<br>2 -5 4<br>**Виведення:**<br>Префіксні суми: [2, -3, 1]<br>Найбільша сума: 2 |
| 8 | n записів назва;число | Суми за назвою та назви із сумою > 0 за алфавітом. | **Вхід:**<br>n = 3<br>а;2<br>б;-1<br>а;3<br>**Виведення:**<br>Суми: {'а': 5, 'б': -1}<br>Назви із сумою > 0: ['а'] |
| 9 | Рядок слів | Групи за довжиною та кількість різних слів у кожній групі. | **Вхід:**<br>код кіт код мова<br>**Виведення:**<br>Групи: {3: ['код', 'кіт', 'код'], 4: ['мова']}<br>Різних слів: {3: 2, 4: 1} |
| 10 | Рядок цілих чисел | Від’ємні, нулі й додатні окремими списками та сума кожної групи. | **Вхід:**<br>-2 0 3 -1<br>**Виведення:**<br>Групи: {-1: [-2, -1], 0: [0], 1: [3]}<br>Суми: {-1: -3, 0: 0, 1: 3} |
| 11 | n записів назва;число | Найбільше число за назвою та список назв за спаданням цих чисел; за рівності — алфавіт. | **Вхід:**<br>n = 3<br>а;2<br>б;5<br>а;4<br>**Виведення:**<br>Максимуми: {'а': 4, 'б': 5}<br>Порядок назв: ['б', 'а'] |
| 12 | Рядок слів | Унікальні слова в порядку появи. Для цих унікальних слів — список пар (слово, довжина), відсортований за довжиною, потім за словом. | **Вхід:**<br>код я код мова<br>**Виведення:**<br>Унікальні: ['код', 'я', 'мова']<br>Пари: [('я', 1), ('код', 3), ('мова', 4)] |
| 13 | Рядок цілих чисел | Вибрати елементи, більші за середнє всього набору, й обчислити їхню суму. | **Вхід:**<br>1 2 6<br>**Виведення:**<br>Відібрані: [6]<br>Сума: 6 |

| Варіант | Вхідні дані | Потрібні результати | Контрольний приклад |
|---|---|---|---|
| 14 | Рядок слів | Частоти перших літер та слова з найчастішою першою літерою. За рівності частот оберіть меншу літеру. | **Вхід:**<br>код кіт дані<br>**Виведення:**<br>Частоти літер: {'д': 1, 'к': 2}<br>Відібрані слова: ['код', 'кіт'] |
| 15 | Рядок цілих чисел | Видалити всі значення з частотою > 1; повернути решту в порядку появи та їхню суму. | **Вхід:**<br>2 1 2 3<br>**Виведення:**<br>Залишені: [1, 3]<br>Сума: 4 |

**Завдання 3. Модуль і два набори даних**

Продовжіть свій варіант завдання 2. Розмістіть функції в окремому власному модулі й імпортуйте їх у головну програму.

1. Введіть два незалежні набори одного формату. Через імпортовані функції обчисліть і виведіть усі результати свого варіанта окремо для першого та другого наборів.

2. Об’єднайте початкові набори послідовно: спочатку перший, потім другий. Обчисліть заново й виведіть усі результати об’єднаного набору з підписом «Об’єднаний». Зіставте їх із результатами окремих наборів, щоб простежити вплив об’єднання.

3. У модулі під захистом __name__ == "__main__" додайте коротку демонстрацію на готових даних. Перевірте прямий запуск та імпорт: імпорт не має друкувати демонстрацію.

У головній програмі покажіть також початкові набори після викликів: вони не повинні змінитися. Перевірте два непорожні набори, порожній перший, порожній другий та обидва порожні. Для додаткового контролю візьміть дані свого рядка таблиці 2 як обидва набори. У серіях сусідні рівні значення на межі наборів утворюють одну серію.

Таблиця 3 — Контрольна перевірка роботи модуля

| Дія або дані | Очікуваний результат | Що перевіряється |
|---|---|---|
| Імпортувати власний модуль. | Функції доступні; демонстрація не друкується. | Захист прямого запуску |
| Запустити модуль без імпорту. | Виконується коротка демонстрація. | __name__ == "__main__" |

| Дія або дані | Очікуваний результат | Що перевіряється |
|---|---|---|
| Варіант 2.<br>**Вхід:**<br>Перший: 2 2 1<br>Другий: 1 1 2 | **Виведення:**<br>Перший: [2, 1]; найдовша серія 2<br>Другий: [1, 2]; найдовша серія 2<br>Об’єднаний: [2, 1, 2]; найдовша серія 3 | Злиття серій на межі |
| Порожній перший або другий набір; обидва порожні. | Об’єднання дає результат непорожнього набору. Якщо порожні обидва — порожні результати свого варіанта. Початкові дані незмінні. | Порожній набір і відсутність мутації |

## Оформлення звіту

Звіт оформлюють за чинними вимогами курсу (документ «Вимоги до оформлення»).

Для кожного завдання у звіті подають формулювання, текст програми та скріншот результату в консолі.

## Методичні рекомендації

### Параметри, return і значення за замовчуванням

def оголошує функцію. Параметри отримують дані виклику, return повертає результат і завершує функцію. Без return повертається None. print() лише виводить текст і не замінює повернення результату.

У def label(text, separator=" / ") параметр separator має типове значення. label("код") використовує його, а label("код", separator=" — ") передає іменований аргумент. Обов’язкові параметри записують перед типовими. Типове значення обчислюється один раз під час оголошення функції, тому змінний список у цій ролі може накопичувати дані між викликами.

Локальні змінні належать окремому виклику. Колекцію передають як об’єкт, тому append() усередині функції може змінити передані дані. Для нового результату створіть окрему колекцію. Призначення функції можна пояснити через docstring — рядок у потрійних лапках, записаний першим у тілі функції. Це документація функції; вона не виводиться сама.

**Значення за замовчуванням і повернення рядка**

**Код Python**

```
def greet(name, ending="!"):
    """Повертає вітання з указаним завершенням."""
    return f"Вітаю, {name}{ending}"

first = greet("Олено")
second = greet("Олено", ending=".")
print(first)
print(second)
```

**Результат виконання**

```
Вітаю, Олено!
Вітаю, Олено.
```

Функція повертає рядок; друкує його головна програма. Перший виклик використовує типовий аргумент, другий — іменований. Рядок у потрійних лапках пояснює призначення greet() і не змінює результатів. Повернений рядок можна зберегти та використати далі.

**Незалежний список для кожного виклику**

Якщо потрібен новий список за відсутності аргументу, використайте типове значення None. Перевірка items is None визначає цей випадок; для переданого списку copy() створює незалежну копію.

**Код Python**

```
def collect(value, items=None):
    result = [] if items is None else items.copy()
    result.append(value)
    return result

print(collect("а"))
print(collect("б"))
```

**Результат виконання**

```
['а']
['б']
```

Два виклики без аргументу items отримали окремі списки: другий не містить «а» з першого виклику. Якщо передати готовий список, елемент додається до його копії; початковий список зберігається.

**Новий результат не змінює аргумент**

**Код Python**

```
def shifted(values, offset):
    return [value + offset for value in values]

source = [1, 2]
result = shifted(source, 5)
print(result)
print(source)
```

**Результат виконання**

```
[6, 7]
[1, 2]
```

Спискове включення утворило нову колекцію. Передавання source у функцію не означає копіювання; незмінність забезпечено способом реалізації.

**Змістовна декомпозиція й алгоритми**

Одна функція може утворювати проміжні дані, друга — підраховувати показники або добирати результат. Функції, які лише викликають одна одну без власної роботи, не дають змістовної декомпозиції. Введення і виведення залишайте в головній програмі.

Унікальні елементи в початковому порядку накопичують у списку з перевіркою value not in result. Частоти — у словнику. Для серій зберігайте попереднє значення та довжину поточної серії; після циклу врахуйте останню серію. Порожній набір перевірте перед зверненням до першого елемента.

Для префіксних сум почніть total = 0, додавайте черговий елемент і зберігайте поточний total. Для різниць сусідів перебирайте індекси від 1. Для групування застосуйте словник окремих списків. Середнє обчислюйте лише для непорожніх даних; у порожньому випадку повертайте визначені умовою результати.

**Дві функції мають різну відповідальність**

**Код Python**

```
def lengths(words):
    return [len(word) for word in words]

def average(numbers):
    return sum(numbers) / len(numbers) if numbers else 0

sizes = lengths(["я", "код"])
print(sizes)
print(average(sizes))
```

**Результат виконання**

```
[1, 3]
2.0
```

Перша функція готує похідні дані, друга обчислює показник. Обидві отримують дані через параметри та повертають значення. У своєму варіанті оберіть потрібні проміжні результати й функції.

**Серія завершується також наприкінці даних**

**Код Python**

```
values = [2, 2, 3]
lengths = []
current = 0
for j, value in enumerate(values):
    if j > 0 and value != values[j - 1]:
        lengths.append(current)
        current = 0
    current += 1
if current:
    lengths.append(current)
print(lengths)
```

**Результат виконання**

```
[2, 1]
```

При зміні значення зберігається довжина попередньої серії. Після циклу потрібно зберегти останню; інакше зникне серія з числом 3. Це демонстрація стану, а декомпозицію та повернені результати своєї задачі визначає студент.

**Префіксна сума накопичується поступово**

**Код Python**

```
total = 0
prefixes = []
for number in [2, -5, 4]:
    total += number
    prefixes.append(total)
print(prefixes)
```

**Результат виконання**

```
[2, -3, 1]
```

Кожен результат містить суму всіх елементів від початку до поточного. total зберігає стан між ітераціями; у функції цей стан має бути локальним.

**Порядок результатів та короткі перевірки**

sorted() приймає key — функцію, що повертає ключ порівняння. Для пар слово, довжина можна визначити окрему функцію з return (record[1], record[0]); спочатку порівнюється довжина, потім слово. Для спадання числа й алфавіту назви ключем може бути (−value, name). Lambda не обов’язкова.

Коротка перевірка assert actual == expected корисна на готових контрольних даних. Наприклад, assert len([]) == 0. AssertionError указує на порушене очікування. assert не замінює перевірку введення; для роботи з користувачем потрібні звичайні умови або винятки.

**Сортування за двома ознаками через key**

**Код Python**

```
def order_key(record):
    word, length = record
    return length, word

records = [("кіт", 3), ("я", 1), ("код", 3)]
ordered = sorted(records, key=order_key)
print(ordered)
assert ordered[0] == ("я", 1)
```

**Результат виконання**

```
[('я', 1), ('код', 3), ('кіт', 3)]
```

Кортеж ключа задає порядок: спочатку довжина, за рівності — слово. sorted() повертає новий список. assert перевіряє очікування на готових даних; це не перевірка введення користувача.

**Модуль та незалежні набори**

Модуль — окремий файл Python. Для навчального прикладу модуль study_tools зберігають як study_tools.py у тій самій папці, що й головну програму; from study_tools import function імпортує функцію. Це пояснення організації прикладу, а назви власних модулів ви обираєте самостійно.

У модулі __name__ дорівнює "__main__" при прямому запуску та назві модуля при імпорті. Блок if __name__ == "__main__": містить демонстрацію, яка не виконається під час імпорту. Функції оголошують поза цим блоком.

Об’єднання first + second створює новий список. Підсумок об’єднаних даних потрібно перерахувати: частоти, унікальність, середнє й серії можуть змінитися, а серія на межі наборів може злитися. Повернені результати й початкові списки порівнюйте окремо.

**Блок демонстрації залежить від __name__**

**Код Python**

```
def cube(number):
    return number ** 3

if __name__ == "__main__":
    print(cube(2))
```

**Результат виконання**

```
8
```

Результат показано для прямого запуску. Під час імпорту __name__ містить назву модуля, тому print() не виконається, а cube залишається доступною. Повну організацію модуля та головної програми наведено в аналогічному прикладі до завдання 3.

**Часті помилки**

Не використовуйте глобальні списки замість параметрів. Не друкуйте результат усередині функції, якщо він потрібен для наступних обчислень. Не втрачайте останню серію. Не вважайте, що результат для об’єднання завжди дорівнює сумі окремих результатів. Не розміщуйте імпортовані функції під захистом прямого запуску.

## Приклади виконання завдань

### Приклад до завдання 1

Навчальна задача. Переведіть температуру з градусів Цельсія у градуси Фаренгейта та сформуйте підпис двома способами. Використайте формулу F = 9C / 5 + 32.

### Код Python

```python
def to_fahrenheit(celsius):
    return 9 * celsius / 5 + 32

def temperature_label(celsius, suffix="°F"):
    value = to_fahrenheit(celsius)
    return f"{value:.1f} {suffix}"

celsius = float(input("Температура °C: "))
print(temperature_label(celsius))
print(temperature_label(suffix="градусів Фаренгейта", celsius=celsius))
```

### Приклад виконання в консолі

```
Температура °C: 10
50.0 °F
50.0 градусів Фаренгейта
```

### Пояснення до прикладу

to_fahrenheit() виконує обчислення, а temperature_label() викликає її й формує рядок. Для 10 °C отримуємо 50 °F. Обидві функції повертають результат, не вводячи й не друкуючи даних.

Перший виклик використовує suffix за замовчуванням, другий передає аргументи за іменами. У завданні 1 самостійно сформуйте функції очищення полів і складання підпису.

**Приклад до завдання 2**

Навчальна задача. Виберіть слова, що закінчуються введеним фрагментом без урахування регістру, та визначте сумарну кількість літер відібраних слів.

**Код Python**

```python
def has_suffix(word, suffix):
    return word.lower().endswith(suffix.lower())

def select_words(words, suffix):
    result = []
    for word in words:
        if has_suffix(word, suffix):
            result.append(word)
    return result

def total_length(words):
    total = 0
    for word in words:
        total += len(word)
    return total

words = input("Слова: ").split()
suffix = input("Закінчення: ").strip()
selected = select_words(words, suffix)
print(f"Відібрані слова: {selected}")
print(f"Усього літер: {total_length(selected)}")
print(f"Початковий список: {words}")
```

**Приклад виконання в консолі**

```
Слова: мова школа код
Закінчення: А
Відібрані слова: ['мова', 'школа']
Усього літер: 9
Початковий список: ['мова', 'школа', 'код']
```

**Пояснення до прикладу**

has_suffix() повертає логічне значення. select_words() викликає її для кожного слова й додає відповідні слова до нового списку. Початковий words залишається незмінним.

«мова» та «школа» закінчуються на «а» й мають 4 та 5 літер, разом 9. total_length() накопичує суму довжин у циклі. У завданні 2 самостійно розподіліть обчислення свого варіанта між функціями та поверніть потрібні результати.

**Приклад до завдання 3**

Модуль обчислює літери, спільні для всіх слів набору. Порівняємо два набори та їхнє послідовне об’єднання. Тут інший алгоритм, ніж у варіантах.

**Код Python — навчальний модуль study_tools**

```
def common_letters(words):
    if not words:
        return set()
    result = set(words[0])
    for word in words[1:]:
        result &= set(word)
    return result

def describe(letters):
    return ", ".join(sorted(letters)) if letters else "немає"

if __name__ == "__main__":
    print(describe(common_letters(["мова", "мед"])))
```

**Код Python — головна програма**

```
from study_tools import common_letters, describe
first = input("Перший набір слів: ").lower().split()
second = input("Другий набір слів: ").lower().split()
for label, words in (("Перший", first), ("Другий", second),
                     ("Об’єднаний", first + second)):
    letters = common_letters(words)
    print(f"{label}: {describe(letters)}")
print(f"Початкові: {first}; {second}")
```

**Приклад виконання в консолі**

```
Перший набір слів: кіт код
Другий набір слів: мова мед
Перший: к
Другий: м
Об’єднаний: немає
Початкові: ['кіт', 'код']; ['мова', 'мед']
```

**Прямий запуск модуля: результат**

```
м
```

**Пояснення до прикладу**

common_letters() починає з множини літер першого слова. Для кожного наступного слова операція &= залишає лише спільні літери. Для «кіт» і «код» це «к», для «мова» й «мед» — «м»; спільної літери всіх чотирьох слів немає. Перевірка if not words повертає порожню множину до звернення до words[0].

describe() сортує літери й об’єднує їх у текст, а для порожньої множини повертає «немає». Вираз first + second створює новий список, тому початкові набори залишаються незмінними. Результат об’єднаного набору обчислюється заново: готові результати окремих наборів не додаються.

При імпорті стають доступними функції, але демонстрація під if __name__ == "__main__": не виконується. За прямого запуску модуля вона працює з готовими словами «мова» й «мед» і друкує «м». У своєму модулі відокремте таку демонстрацію від функцій, а введення двох наборів залиште в головній програмі.

**Контрольні питання**

1. Чим return відрізняється від print()?
2. Як діють позиційні й іменовані аргументи?
3. Коли використовується значення за замовчуванням?
4. Чому змінний список небезпечний як типовий аргумент?
5. Як уникнути зміни переданої колекції?
6. Що робить функцію змістовною частиною декомпозиції?
7. Як урахувати останню серію однакових значень?
8. Як sorted() використовує key?
9. Чим результат об’єднання може відрізнятися від окремих підсумків?
10. Для чого потрібен захист __name__ == "__main__"?