Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення
Дисципліна: Скриптові мови програмування (Python)

Лабораторна робота №4
Тема: «РОБОТА З РЯДКАМИ У PYTHON»

Виконав: ст. гр. КН-24
Куріщенко П. В.
Перевірив: ассистент
Ткаченко О.С.

Кропивницький 2025

Варіант - 1

Мета роботи - набути навичок роботи з вбудованими функціями для роботи з рядками у Python.

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

Завдання 1:

Створіть програму, яка буде складати випадкові фрази на основі трьох списків зі словами. З кожного списку вона повинна брати випадковим чином слова і поєднувати їх в одну фразу.

Функція `genRanSentence(*listOfWords())`

Призначення:

Функція genRanSentence генерує одне випадкове речення, складаючи його із випадково вибраного прикметника, іменника та дієслова.

Принцип роботи:

Функція приймає три списки слів як аргументи. За допомогою функції random.choice із кожного списку вибирається одне слово. Потім ці слова об'єднуються в рядок за шаблоном "прикметник + іменник + дієслово" і виводяться на екран у рамках повідомлення "Завдання 1. Твоя випадкова фраза: ...".

Проектні рішення:

- Для генерації випадкового вибору використовується стандартний модуль Python random.
- Функція не повертає результат, а виводить його безпосередньо через print, оскільки мета — миттєво показати користувачу результат роботи.

- Параметри функції зроблені гнучкими: genRanSentence приймає списки як аргументи, що дозволяє її використовувати з іншими наборами слів за потреби.

Блок схема:

Лістинг функції:

```
def genRanSentence(adjectives, nouns, verbs):
    """
    Генерує випадкове речення з трьох частин: прикметник, іменник, дієслово.
    """
    import random
    print("Завдання 1. Твоя випадкова фраза:", f"{random.choice(adjectives)}
{random.choice(nouns)} {random.choice(verbs)}")
```

Результати роботи:

```
Завдання 1. Твоя випадкова фраза: підозрілий котлета жонглює
Завдання 1. Твоя випадкова фраза: шумний папуга регоче
Завдання 1. Твоя випадкова фраза: підозрілий буряк стрибає
```

Завдання 2.1:

Візьміть текстовий файл, що містить Вашу улюблену художню книгу та визначте загальну кількість символів у тексті з пробілами та без пробілів.

2. Визначте загальну кількість слів у тексті, загальну кількість різних слів (без повторів) та кількість унікальних слів, що зустрічаються тільки один раз.

Функція `amountOfSymbols(filePath)`

Призначення:

Функція amountOfSymbols визначає кількість символів у текстовому файлі, як із пробілами, так і без них.

Принципи роботи:

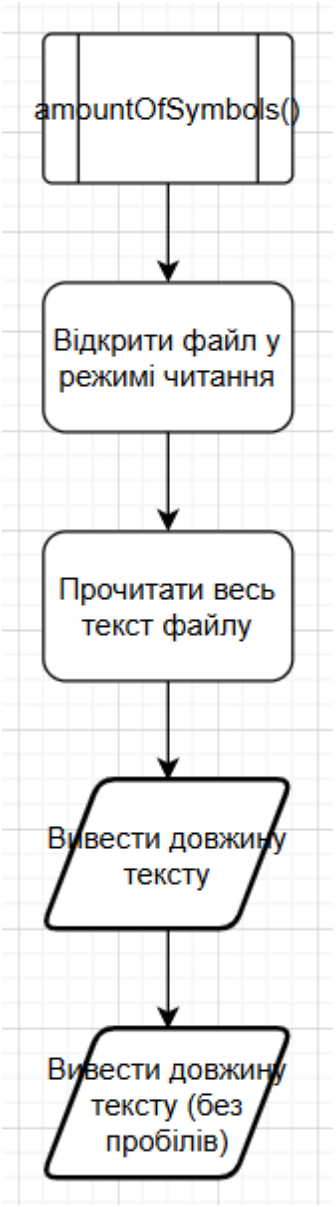
Функція приймає шлях до текстового файлу (filePath) як аргумент. Відкриває файл у режимі читання з кодуванням UTF-8, читає весь його вміст у змінну text, після чого:

- Рахує загальну кількість символів за допомогою len(text).
- Рахує кількість символів без урахування пробілів, видаляючи всі пробіли методом replace(" ", "") і обчислюючи довжину отриманого рядка. Результати виводяться на екран у вигляді оформленого повідомлення.

Проектні рішення:

- Використання конструкції with open(...) гарантує автоматичне закриття файлу після роботи з ним, що підвищує безпечність та зручність обробки файлів.
- Кодування utf-8 обране для коректної роботи з українськими та іншими юнікодними символами.
- Функція не повертає значення, а відразу виводить результати за допомогою print, оскільки основною задачею є відображення статистики користувачу.

Блок схема:



Лістинг функції:

```
def amountOfSymbols(filePath):
    """
    Повертає кількість символів у текстовому файлі.
    """
    with open(filePath, "r", encoding='utf-8') as file:
        text = file.read()
        print("Завдання 2.1) Кількість символів у файлі(з пробілами):   ",
len(text))
        print("                Кількість символів у файлі(без пробілів): ",
len(text.replace(" ", "")))
```

Результат роботи:

```
Завдання 2.1) Кількість символів у файлі(з пробілами):   99133
              Кількість символів у файлі(без пробілів):  83384
```

Завдання 2.2:

Візьміть текстовий файл, що містить Вашу улюблену художню книгу та визначте загальну кількість слів у тексті, загальну кількість різних слів (без повторів) та кількість унікальних слів, що зустрічаються тільки один раз.

Функція `amountOfWords(filePath)`

Призначення:

Функція amountOfWords визначає статистику за кількістю слів у текстовому файлі, включаючи загальну кількість слів, кількість різних слів (без повторень) та кількість унікальних слів (що зустрічаються тільки один раз).

Принцип роботи:

Функція приймає шлях до текстового файлу (filePath) як аргумент. Вона викликає допоміжну функцію getCleanedWords, яка читає текстовий документ і повертає його слова без інших символів. Далі виконується:

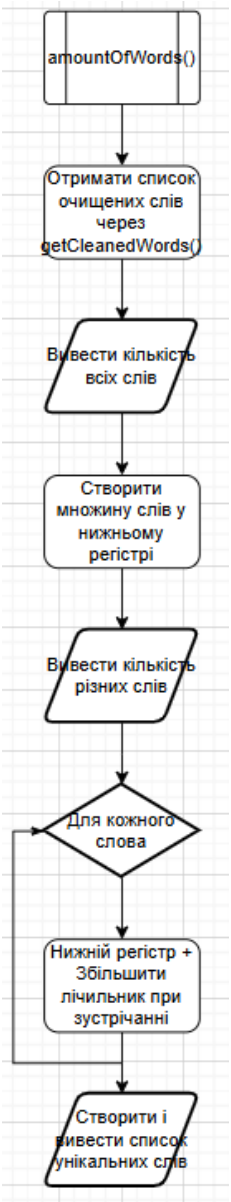
- Виведення загальної кількості слів (len(words)).
- Створення множини різних слів у нижньому регістрі (set(word.lower() for word in words)) та виведення їхньої кількості.
- Формування словника word_counts, який підраховує кількість входжень кожного слова (у нижньому регістрі).
- Визначення унікальних слів (тих, які зустрічаються лише один раз) і підрахунок їхньої кількості.

Проектні рішення:

- Для уніфікації підрахунків усі слова переводяться до нижнього регістру (lower()), щоб "Слово" і "слово" вважалися одним і тим самим словом.
- Для підрахунку кількості слів без повторень використовується структура set, яка автоматично усуває дублікати.

- Для підрахунку кількості унікальних слів використовується словник (dict), що дозволяє ефективно зберігати частотність кожного слова.
- Функція виводить результати через print, оскільки головна мета — оперативно показати користувачу отриману статистику.

Блок схема:



Лістинг функції:

```
def amountOfWords(filePath):
    """
    Повертає кількість слів у текстовому файлі.
    """
    words = getCleanedWords(filePath)
    print("           2) Кількість слів у файлі:              ", len(words))
```

```
    different_words = set(word.lower() for word in words)
    print("                        Кількість  різних  слів(без  повторів):    ",
len(different_words))
    word_counts = {}
    for word in words:
        word_lower = word.lower()
        word_counts[word_lower] = word_counts.get(word_lower, 0) + 1
    unique_words = [word for word, count in word_counts.items() if count == 1]
    print("                      Кількість  унікальних  слів  у  файлі:      ",
len(unique_words))
```

Результат роботи:

```
2) Кількість слів у файлі:                15702
   Кількість різних слів(без повторів):   4894
   Кількість унікальних слів у файлі:     3431
```

Завдання 3(варіант 1):

Знайдіть у тексті найдовшу послідовність слів, що повторюється більше одного разу.

Функція `findRepSeq(filePath)`

Призначення:

Функція findRepSeq знаходить і виводить найдовшу послідовність слів, яка повторюється у текстовому файлі.

Принцип роботи:

Функція приймає шлях до текстового файлу (filePath) як аргумент. Вона викликає допоміжну функцію getCleanedWords, яка повертає очищений текст. Далі виконується:

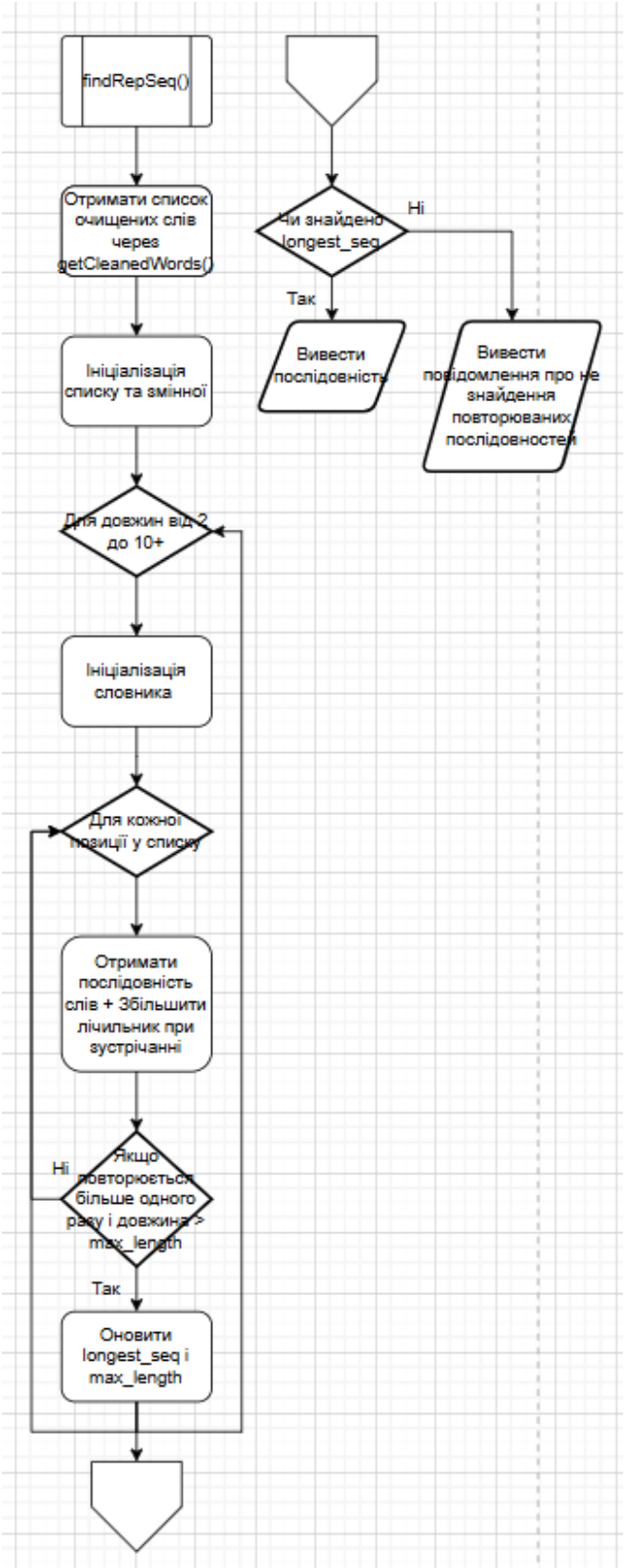
- Ініціалізація змінних для зберігання найдовшої знайденої повторюваної послідовності слів (longest_seq) та її довжини (max_length).
- Перебір можливих довжин послідовностей від 2 до 10 слів (або менше, якщо файл коротший).
- Для кожної довжини створюються всі можливі підпослідовності (seq) і підраховується кількість їхніх входжень у тексті за допомогою словника sequences.

- Якщо підпослідовність зустрічається більше одного разу і є довшою за попередньо знайдену, вона зберігається як новий найдовший збіг.
- Після завершення пошуку:
    - Якщо повторювана послідовність знайдена — вона виводиться на екран.
    - Якщо ні — виводиться відповідне повідомлення.

Проектні рішення:

- Пошук обмежений підпослідовностями довжиною максимум 10 слів, що дозволяє зменшити час обробки для великих текстів.
- Для ефективного збереження і підрахунку послідовностей використовується структура dict, де ключами є кортежі слів (tuple).
- Слова в тексті попередньо очищаються і уніфікуються через getCleanedWords, що підвищує якість пошуку (уникнення впливу розділових знаків).
- Результати виводяться через print, що дозволяє миттєво показати результат користувачу без додаткових кроків.

Блок схема:



Лістинг функції:

```
def findRepSeq(filePath):
    """
    Повертає найдовшу повторювану послідовність слів у текстовому файлі.
```

```
    """
    words = words = getCleanedWords(filePath)

    longest_seq = []
    max_length = 0

    for length in range(2, min(11, len(words))):
        sequences = {}

        for i in range(len(words) - length + 1):
            seq = tuple(words[i:i + length])
            sequences[seq] = sequences.get(seq, 0) + 1
            if sequences[seq] > 1 and length > max_length:
                longest_seq = seq
                max_length = length

    if longest_seq:
        print("Завдання 3. Найдовша повторювана послідовність слів:", "
".join(longest_seq))
    else:
        print("Завдання 3. Повторюваних послідовностей не знайдено")
```

Результат роботи:

```
Завдання 3. Найдовша повторювана послідовність слів: що тулуб його занадто широкий
```

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які базові операції роботи з рядками є у мові Python?

- **Конкатенація (додавання)**: об'єднання рядків за допомогою оператора +.
- **Дублювання рядка**: повторення рядка кілька разів оператором *.
- **Визначення довжини рядка**: функція len().
- **Доступ до символів за індексом**: через квадратні дужки, наприклад, S[0].
- **Отримання зрізу рядка**: за допомогою синтаксису S[start:stop] або S[start:stop:step].

2. Як можна одержувати зрізи рядків? Якими бувають зрізи? Наведіть приклади.

- **Отримання зрізу** здійснюється через індекси: рядок[початок:кінець:крок].
- Види зрізів:
  - З вказанням початку і кінця: s[1:4] → символи з 1 до 3 індексу.
  - З пропущеним початком: s[:5] → від початку до 4 індексу.
  - З пропущеним кінцем: s[2:] → від другого індексу до кінця рядка.
  - Повний зріз: s[:] → увесь рядок.
  - З кроком: s[::2] → кожен другий символ.
  - У зворотному порядку: s[::-1] → всі символи у зворотному порядку.

Приклади:

```
s = 'spameggs'
print(s[1:4])   # 'pam'
print(s[::-1])  # 'sggemaps'
print(s[:6])    # 'spameg'
```

3. Які методи роботи з рядками є у мові Python?

Основні методи для роботи з рядками:

- find(), rfind(), index(), rindex() – пошук підрядків.
- replace() – заміна частини рядка.
- split() – розбиття рядка.
- upper(), lower(), swapcase(), capitalize(), title() – зміна регістру символів.
- startswith(), endswith() – перевірка початку/кінця рядка.
- strip(), lstrip(), rstrip() – видалення пробілів.
- join() – об'єднання списку рядків в один рядок.
- count() – підрахунок входжень підрядка.
- isdigit(), isalpha(), isalnum(), islower(), isupper(), isspace(), istitle() – перевірка складу рядка.
- partition(), rpartition() – розбиття рядка на частини.

- expandtabs() – заміна табуляцій пробілами.
- zfill(), ljust(), rjust(), center() – форматування рядків за шириною.

4. Що таке словники? Як з ними працювати? Для чого вони потрібні?

- **Словники** в Python — це невпорядковані колекції пар "ключ-значення".
- Створити словник можна через {}, функцію dict(), метод fromkeys(), або генератор словників.
- Доступ до значень відбувається через ключі: d[key].
- Додавання нової пари — простим присвоєнням d[new_key] = value.
- Використовуються для:
  - Швидкого доступу до даних за унікальним ключем.
  - Створення асоціативних масивів.
  - Ефективного зберігання і обробки даних.

5. Які методи роботи зі словниками є у мові Python?

Основні методи словників:

- clear() — очищення словника.
- copy() — створення копії словника.
- fromkeys(seq, value) — створення нового словника із заданими ключами.
- get(key, default) — отримання значення за ключем без виникнення помилки.
- items() — отримання пар (ключ, значення).
- keys() — отримання всіх ключів.
- values() — отримання всіх значень.
- pop(key, default) — видалення ключа і повернення його значення.
- popitem() — видалення і повернення останньої пари (ключ, значення).

- setdefault(key, default) — отримання значення за ключем або вставка нового.
- update(other) — оновлення словника парами з іншого словника.