## Протокол

## Лабораторна робота №3. Варіант 14

### Завдання

Надрукувати всі слова, які відрізняються від першого слова. Перед друком подвоїти першу літеру, якщо в слові парна кількість літер, та видалити останню літеру, якщо в слові непарна кількість літер. Замінити в словах всі літери ’a’на’е’.

### Реалізація

class Node:

    # node initialization

    def \_\_init\_\_(self, data=None):

        self.data = data

        self.next = None

        self.prev = None

Клас **Node** слугує для створення вузлів у двозв'язному списку, де кожен вузол тримає якісь дані та посилання на наступний і попередній вузли в списку. Це дозволяє елементам списку бути впорядкованими так, що кожен з них знає про свого сусіда зпереду і ззаду, полегшуючи навігацію туди-сюди по списку.

class DoublyLinkedList:

    # doublyLinkedList initialization

    def \_\_init\_\_(self):

        self.head = None

        self.tail = None

    def append(self, data):

        # method to append data at the end of the list

        new\_node = Node(data)

        if not self.head:  # if list is empty

            self.head = new\_node

            self.tail = new\_node

        else:

            self.tail.next = new\_node

            new\_node.prev = self.tail

            self.tail = new\_node

    # method to process words through the modify\_function

    def process\_words(self, modify\_function):

        if not self.head:

            print("Список порожній.")

            return

        first\_word = self.head.data

        current = self.head.next

        while current:

            if current.data != first\_word:

                print(f"{modify\_function(current.data)}")

            current = current.next

    # making the list iterable

    def \_\_iter\_\_(self):

        current = self.head

        while current:

            yield current

            current = current.next

Клас **DoublyLinkedList** створює двозв'язний список, де кожен елемент (вузол) знає про свого наступника і попередника. Він дозволяє додавати елементи в кінець списку, обробляти дані в кожному вузлі згідно з певною функцією та перебирати елементи списку. Якщо список порожній, голова **(head)** і хвіст **(tail)** списку не вказують ні на що. При додаванні першого елементу, він стає і головою, і хвостом. Для кожного нового елемента створюється вузол, який додається в кінець списку, при цьому оновлюються посилання на наступний і попередній вузли. Метод **process\_words** дозволяє застосувати задану функцію до даних кожного вузла, починаючи з другого, і вивести результат. Також клас має спеціальний метод **\_\_iter\_\_** для ітерації по елементах, що робить можливим використання вузлів списку в циклах for та інших ітераційних контекстах.

class UserInterface:

    # method for data input with validation and retry option

    @staticmethod

    def input\_data():

        while True:

            data = input("Введіть текст: ").strip()

            if not data:

                print("Помилка: Пусте введення.")

                if UserInterface.ask\_retry():

                    continue

                else:

                    return None, False

            if not UserInterface.validate\_data(data):

                print("Помилка: Введені дані не відповідають вимогам. Спробуйте ще раз.")

                if UserInterface.ask\_retry():

                    continue

                else:

                    return None, False

            return data.strip(".").split(), True

    # method for input validation

    @staticmethod

    def validate\_data(data):

        return data.replace(".", "").replace(" ", "").isalpha()

    # method to print user instructions

    @staticmethod

    def print\_instructions():

        print(

           "Інструкції: Введіть слова, розділені пробілами. В кінці дозволяється ставити крапку.\n"

           "Всі символи в словах мають бути літерами; цифри, спеціальні символи, або знаки пунктуації (крім крапки в кінці) не дозволяються."

        )

    # method to ask user if they want to retry input

    @staticmethod

    def ask\_retry():

        retry = input("Спробувати знову? (так/ні): ").lower()

        return retry in ["так", "yes"]

Клас **UserInterface** допомагає спілкуватися з користувачем: запитує ввести текст, перевіряє, чи правильно введено, і питає, чи хоче користувач спробувати знову, якщо щось не так. Також каже, як правильно вводити дані.

def modify\_word(word):

    # function to modify word as per given logic

    modified\_word = word.replace('a', 'e')

    if len(modified\_word) % 2 == 0:

        modified\_word = modified\_word[0] \* 2 + modified\_word[1:]

    else:

        modified\_word = modified\_word[:-1]

    return modified\_word

Ця функція змінює слово за певним правилом. Спочатку вона замінює всі 'a' на 'e' у слові. Потім перевіряє, чи кількість букв у зміненому слові парна. Якщо так, то подвоює першу букву слова. Якщо ні, то видаляє останню букву слова. У результаті ми отримуємо змінене слово.

def print\_modified\_words\_with\_list(words):

    # function to print modified words except the first word in the list

    if not words:

        print("Помилка: Список слів порожній.")

        return

    first\_word = words[0]

    for word in words[1:]:

        if word != first\_word:

            print(f"{modify\_word(word)}")

Ця функція бере список слів і працює з кожним словом, окрім першого. Спочатку вона перевіряє, чи список не порожній. Якщо він порожній, функція повідомляє про помилку і закінчує роботу. Якщо в списку є слова, функція переходить до кожного слова після першого і змінює його за певним правилом, але тільки якщо це слово не співпадає з першим словом у списку. Після зміни кожного такого слова, вона виводить його на екран.

def main():

    UserInterface.print\_instructions()

    words\_list, input\_valid = UserInterface.input\_data()

    if words\_list and input\_valid:

        start\_time = time.time()

        list\_memory\_usage = sys.getsizeof(words\_list) + sum(sys.getsizeof(word) for word in words\_list)

        print("Динамічний масив:")

        print\_modified\_words\_with\_list(words\_list)

        array\_processing\_time = time.time() - start\_time

        print(f"Час виконання (масив): {array\_processing\_time} секунд.")

        print(f"Об’єм пам’яті (масив): {list\_memory\_usage} байтів.")

        start\_time = time.time()

        dll = DoublyLinkedList()

        for word in words\_list:

            dll.append(word)

        dll\_memory\_usage = sys.getsizeof(dll) + sum(sys.getsizeof(node) + sys.getsizeof(node.data) for node in dll)

        print("\nДвозв'язний список:")

        dll.process\_words(modify\_word)

        list\_processing\_time = time.time() - start\_time

        print(f"Час виконання (список): {list\_processing\_time} секунд.")

        print(f"Об’єм пам’яті (список): {dll\_memory\_usage} байтів.")

    else:

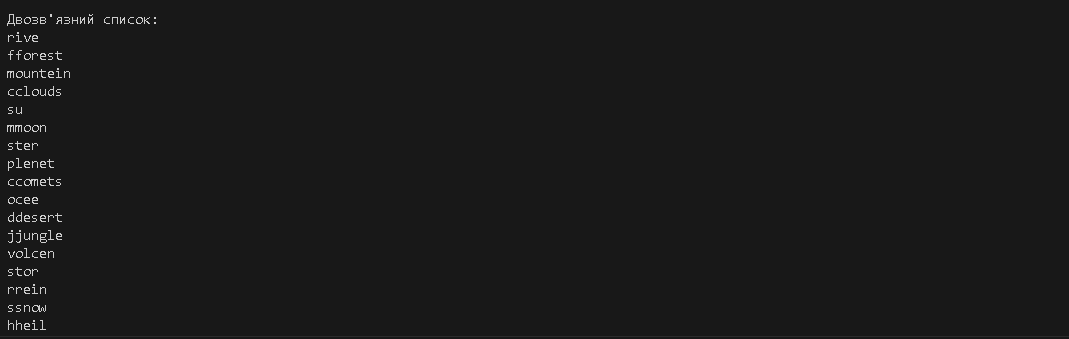
        print("Обробка даних не відбулась через невідповідність умовам введення.")

Функція main запускає програму, спочатку показуючи користувачу, як правильно вводити слова. Потім вона просить ввести список слів і перевіряє, чи введення відповідає правилам. Якщо все гаразд, програма розпочинає обробку: вимірює, скільки пам'яті займає список слів і як довго триває обробка слова. Вона виводить змінені слова, крім першого у списку, і показує, скільки часу і пам'яті це зайняло. Потім робить те ж саме для двозв'язного списку, створеного з цих слів, і знову показує час обробки та використану пам'ять. Якщо введені дані не відповідають правилам, програма повідомляє про це і закінчує роботу.

### Результат

### 







Час обробки близький, але обробка двозв'язного списку трохи довша, що може бути зумовлено додатковими операціями з посиланнями між вузлами.

Двозв'язний список використовує більше пам'яті порівняно з динамічним масивом через додаткові посилання у кожному вузлі на попередній та наступний елементи.

**Висновок:** Обидві структури даних ефективно обробляють введені дані, але динамічний масив виявляється більш ефективним за рахунок меншого використання пам'яті та трохи швидшої обробки. Вибір між ними залежить від конкретних вимог до швидкості обробки та економії пам'яті.