Лабораторна робота 6

Тема: Ітератори, генератори та декоратори.

Мета: Навчитися керувати споживанням ітератора, створювати генератори опрацювання даних і застосовувати простий декоратор.

Зміст роботи

Завдання 1. Поточний стан ітератора

Введіть щонайменше два слова, створіть список та його ітератор через iter().

1. Отримайте два елементи окремими викликами next() та виведіть перше й друге слова.

2. Через for обійдіть решту того самого ітератора, виведіть ці слова й кількість елементів, що залишилися.

3. Виведіть next(iterator, "Кінець") після завершення. Створіть новий ітератор початкового списку та виведіть усі його слова повторним обходом.

Дані коректні. Перевірте власні дані та рівно два слова.

Таблиця 1 — Контрольна перевірка стану ітератора

Вхід	**Виведення: перший обхід**	**Виведення: завершення і повтор**
код мова дані	Перше: код Друге: мова Решта: дані Кількість решти: 1	Після завершення: Кінець Новий обхід: код, мова, дані
код мова	Перше: код Друге: мова Решта: немає Кількість решти: 0	Після завершення: Кінець Новий обхід: код, мова

Завдання 2. Генератор добору та перетворення

Створіть генератор для свого варіанта таблиці 3. Числа або слова введіть одним рядком; слова нормалізуйте до нижнього регістру. Вхідні дані коректні.

1. У генераторі переберіть дані й через yield поверніть лише відповідні перетворені елементи. Не створюйте готовий список результатів.

2. Отримайте й виведіть перший результат через next(generator, None). Решту виводьте циклом for по тому самому генератору. Виведіть повну кількість результатів та підсумок за наведеними далі правилами, урахувавши окремо спожитий перший результат.

3. Повторіть for по вже вичерпаному генератору й покажіть, що нових елементів 0. Для повного повтору створіть новий генератор.

Порожні дані або відсутність збігів дають перший результат None, кількість і підсумок 0. None не враховуйте. Перевірте власні дані, один результат і відсутність результатів; для варіантів 4 і 14 останнє означає порожній набір. У таблиці 3 наведено вхідні дані та очікуване виведення. Кожен рядок «Далі» відповідає наступному результату циклу; готовий список результатів будувати не потрібно. Лапки позначають межі рядків, зокрема " — порожній рядок.

Для чисел підсумок — сума значень; для рядків — сума їхніх довжин; для пар варіанта 4 — сума других компонентів. У варіанті 13 знаки «+» і «-» входять до довжини рядка. Порожній рядок також є результатом: він збільшує кількість на 1, а підсумок — на 0.

У порівнянні з попереднім елементом перший не розглядайте; у порівнянні з наступним не розглядайте останній. Для початків серій перший елемент відбирайте окремо. Позиції відлічуйте від 1, як зазначено у варіанті.

Таблиця 2 — Спільна контрольна перевірка генератора

Випадок	**Очікуваний результат**	**Що перевіряється**
Порожній набір або жодного збігу.	Перший: None Кількість: 0 Підсумок: 0	Безпечне next()
Один результат. Вхід для варіанта 9: я Вхід для варіанта 10: 10	Варіант 9: перший " Варіант 10: перший 0 В обох: кількість 1, підсумок 0; решта порожня.	Нуль і порожній рядок не втрачені
Повторний for по тому самому генератору.	Нових результатів 0.	Вичерпання
Новий виклик генератора на тих самих даних.	Знову повна контрольна послідовність варіанта.	Новий незалежний обхід

Таблиця 3 — Правила генераторів і контрольні послідовності

Варіант	**Що відбирати**	**Що повертати**	**Вхід і очікуване виведення**
1	Цілі: додатні парні	квадрати.	**Вхід:** -3 -2 0 2 3 4 **Виведення:** Перший: 4 Далі: 16 Кількість: 2 Підсумок: 20

Варіант	Що відбирати	Що повертати	Вхід і очікуване виведення
2	Цілі: від’ємні	модулі.	**Вхід:** -3 -2 0 2 3 4 **Виведення:** Перший: 3 Далі: 2 Кількість: 2 Підсумок: 5
3	Цілі: елементи більші за попередній	різниці з ним.	**Вхід:** 3 5 5 2 6 **Виведення:** Перший: 2 Далі: 4 Кількість: 2 Підсумок: 6
4	Цілі: перший елемент і початки нових серій	пара (позиція від 1, значення). Підсумок — сума значень.	**Вхід:** 3 3 1 1 1 2 **Виведення:** Перший: (1, 3) Далі: (3, 1) Далі: (6, 2) Кількість: 3 Підсумок: 6
5	Цілі: непарні	подвійні значення зі збереженням знака.	**Вхід:** -3 -2 0 2 3 4 **Виведення:** Перший: -6 Далі: 6 Кількість: 2 Підсумок: 0
6	Слова: довжина ≥ 4	верхній регістр.	**Вхід:** код дані мова python я **Виведення:** Перший: 'ДАНІ' Далі: 'МОВА' Далі: 'PYTHON' Кількість: 3 Підсумок: 14
7	Слова: містять «а»	зворотний запис.	**Вхід:** код дані мова python я **Виведення:** Перший: 'інад' Далі: 'авом' Кількість: 2 Підсумок: 8
8	Слова: довші за попереднє	перші 2 літери.	**Вхід:** код дані мова python я **Виведення:** Перший: 'да' Далі: 'ру' Кількість: 2 Підсумок: 4

Варіант	Що відбирати	Що повертати	Вхід і очікуване виведення
9	Слова: довжина непарна	слово без першої літери.	**Вхід:** код дані мова python я **Виведення:** Перший: 'од' Далі: '' Кількість: 2 Підсумок: 2
10	Цілі: модуль ≥ 3	остання цифра модуля.	**Вхід:** -3 -2 0 2 3 4 **Виведення:** Перший: 3 Далі: 3 Далі: 4 Кількість: 3 Підсумок: 10
11	Цілі: парні позиції від 1	значення мінус попереднє.	**Вхід:** -3 -2 0 2 3 4 **Виведення:** Перший: 1 Далі: 2 Далі: 1 Кількість: 3 Підсумок: 4
12	Слова: не містять «а» й довжина ≥ 3	перша та остання літери разом.	**Вхід:** код дані мова python я **Виведення:** Перший: 'кд' Далі: 'рп' Кількість: 2 Підсумок: 4
13	Цілі: ненульові	«+число» для додатних, звичайний запис для від’ємних. Підсумок — сума довжин рядків.	**Вхід:** -3 -2 0 2 3 4 **Виведення:** Перший: '-3' Далі: '-2' Далі: '+2' Далі: '+3' Далі: '+4' Кількість: 5 Підсумок: 10
14	Слова: перший елемент і початки нових серій	title().	**Вхід:** код код мова мова дані **Виведення:** Перший: 'Код' Далі: 'Мова' Далі: 'Дані' Кількість: 3 Підсумок: 11

Варіант	Що відбирати	Що повертати	Вхід і очікуване виведення
15	Цілі: елементи, більші за наступний. Останній не відбирати.	квадрати.	**Вхід:** -3 -2 4 2 3 0 **Виведення:** Перший: 16 Далі: 9 Кількість: 2 Підсумок: 25

Завдання 3. Декоратор звичайної функції

Для свого варіанта створіть звичайну функцію з одним аргументом — набором даних. Вона створює генератор із завдання 2, повністю обходить його й повертає підсумок, визначений у таблиці 3.

1. Створіть декоратор: він приймає функцію, повертає обгортку з одним аргументом, друкує аргумент і результат та повертає результат без змін.

2. Застосуйте декоратор до звичайної функції. Викличте її на контрольних даних, порожньому наборі та повторно на початкових даних. Кожне повернене значення присвойте змінній і виведіть.

Не застосовуйте цей декоратор до функції-генератора. Підсумок беріть зі свого рядка таблиці 3; для порожнього набору він дорівнює 0. Таблиця 4 ілюструє варіант 1: для власної перевірки підставте дані й підсумок свого варіанта. Параметризовані декоратори та *args/**kwargs не потрібні. Підписи виведення можна обрати самостійно.

Таблиця 4 — Контрольна перевірка декоратора

Дані та дія	Очікуваний результат	Що перевіряється
Приклад для варіанта 1. **Вхід:** -3 -2 0 2 3 4 Виклик декорованої функції.	**Виведення:** Аргумент: [-3, -2, 0, 2, 3, 4] Результат: 20 Повернено: 20	Збереження return
Вхід: порожній набір. Виклик декорованої функції.	**Виведення:** Аргумент: [] Результат: 0 Повернено: 0	Порожні дані
Повторний виклик на тих самих початкових даних варіанта 1.	**Виведення:** Аргумент: [-3, -2, 0, 2, 3, 4] Результат: 20 Повернено: 20	Відсутність спільного вичерпаного генератора

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

Ітерований об’єкт та поточний стан ітератора

Список і рядок — ітеровані об’єкти, з яких iter() створює ітератор. next() просуває його на один елемент. Цикл for продовжує з поточної позиції, а не з початку. Після вичерпання next(iterator) піднімає StopIteration, проте next(iterator, default) повертає default.

Для нового повного обходу списку створіть iter(values) повторно. Присвоєння another = iterator дає інше ім’я того самого стану, а не новий незалежний перебір.

for продовжує після вже спожитого елемента

Код Python

```
words = "код дані мова".split()
iterator = iter(words)
print(next(iterator))
for word in iterator:
    print(word)
print(next(iterator, "Кінець"))
print(next(iter(words)))
```

Результат виконання

```
код
дані
мова
Кінець
код
```

Перший next() спожив «код», тому for почав із «дані». Вичерпаний ітератор повертає типове значення, а новий iter(words) знову починає з першого слова.

Генератор, добір і перетворення

Функція з yield створює генератор. Її тіло починає виконуватися тільки при next() або переборі. yield value повертає елемент і призупиняє виконання;

наступний запит продовжує після yield. Умова if визначає добір, вираз після yield — перетворення.

Перший результат через next(generator, None) уже спожито. Його потрібно показати й урахувати окремо; цикл for обробить лише решту. Перевіряйте first is not None, а не if first: нуль і порожній рядок теж можуть бути правильними результатами.

За відсутності збігів next() поверне None, цикл не виконається, кількість і підсумок залишаться нульовими. Повторний for по тому самому генератору дасть 0 нових елементів. Для пар позиція, значення підсумовуйте другий компонент. Для рядків підсумовуйте len(item).

Початок серії — перший елемент або елемент, який відрізняється від попереднього. Для слів порівнюйте вже нормалізовані значення. У знаковому записі f"{value:+d}" додає «+» до додатного цілого, а знак від'ємного зберігає; наприклад, +3 та -3.

yield призупиняє обчислення

Код Python

```
def increased_positive(values):
    for value in values:
        if value > 0:
            yield value + 1

generator = increased_positive([0, -2, 1, 3])
print(next(generator, None))
for value in generator:
    print(value)
print(next(generator, None))
```

Результат виконання

```
2
4
None
```

Умова визначає добір, вираз value + 1 — перетворення. Перший результат уже спожито перед for. Останнє None указує на вичерпання, а не на повернений елемент.

Нуль і порожній рядок є результатами

Код Python

```
first = next(iter([0]), None)
print(first is not None)
second = next(iter([""]), None)
print(second is not None)
empty = next(iter([]), None)
print(empty is not None)
```

Результат виконання

```
True
True
False
```

Перевірка if first була б хибною для 0 або порожнього рядка. Порівняння з None правильно відрізняє відсутність елемента; це суттєво для варіанта, де слово з однієї літери перетворюється на порожній рядок.

Пари та рядки потребують різного підсумку

Код Python

```
pairs = [(1, 2), (3, 4)]
total = 0
for position, value in pairs:
    total += value
print(total)
print(sum(len(text) for text in ["-3", "+2"]))
```

Результат виконання

```
6
4
```

Розпакування виділяє значення з пари, а для рядка показником може бути його довжина. Така сама логіка застосовується під час обходу генератора без готового списку його результатів.

Безпечний перебір сусідніх елементів

Щоб поточний елемент мав попередній, починайте індекси з 1. У прикладі кожен рядок показує пару сусідніх значень без звернення за межі списку.

Код Python

```
values = [4, 1, 3]
for index in range(1, len(values)):
    print(values[index - 1], values[index])
```

Результат виконання

```
4 1
1 3
```

Для порівняння з наступним елементом перебирайте індекси від 0 до len(values) - 2 включно: наступний має індекс index + 1. Позиція у звичайному рахунку від 1 дорівнює index + 1. Для порожнього списку або одного елемента пар сусідів немає.

Функції як об’єкти та простий декоратор

Функцію можна передати аргументом іншій функції та повернути як результат. Декоратор отримує початкову функцію й повертає wrapper — обгортку. @trace над оголошенням означає заміну імені функції результатом trace(function).

Обгортка з одним аргументом друкує аргумент, викликає original(argument), друкує результат і обов’язково повертає його. Якщо не написати return result, зовнішній виклик отримає None. Підсумкова звичайна функція створює свіжий генератор на кожен виклик.

Виклик функції-генератора лише створює об’єкт генератора; його елементи ще не обчислено. Тому обгортку для аргументу та готового підсумку застосовують до звичайної функції, що повністю обходить генератор, а не до створення генератора.

Обгортка повертає результат початкової функції

Код Python

```
def trace(function):
    def wrapper(number):
        print(f"Аргумент: {number}")
        result = function(number)
        print(f"Результат: {result}")
        return result
    return wrapper

@trace
def triple(number):
    return 3 * number

answer = triple(4)
print(f"Повернено: {answer}")
```

Результат виконання

```
Аргумент: 4
Результат: 12
Повернено: 12
```

trace повертає функцію wrapper, а wrapper після виклику triple повертає число. @trace застосовується під час оголошення. У завданні звичайна функція має спочатку повністю опрацювати новий генератор.

Часті помилки

Не втрачайте перший спожитий елемент у кількості й сумі. Не перетворюйте генератор на готовий список, якщо завдання вимагає поетапного споживання. Не очікуйте результатів від повторного проходу. Декоратор має повертати обгортку, а обгортка — результат початкової функції.

Приклади виконання завдань

Приклад до завдання 1

Навчальна задача. Послідовно отримуйте слова через ітератор і виводьте їх парами. Якщо кількість слів непарна, останнє виведіть окремо.

Код Python

```
iterator = iter(input("Слова: ").split())
pairs = 0
while True:
    first = next(iterator, None)
    if first is None:
        break
    second = next(iterator, None)
    if second is None:
        print(f"Без пари: {first}")
        break
    pairs += 1
    print(f"Пара {pairs}: {first} / {second}")
print(f"Повних пар: {pairs}")
```

Приклад виконання в консолі

```
Слова: мова Python дані код приклад
Пара 1: мова / Python
Пара 2: дані / код
Без пари: приклад
Повних пар: 2
```

Пояснення до прикладу

Кожен next() споживає одне слово того самого ітератора. Перші два виклики дають «мова» та «Python», наступні — «дані» та «код». На останньому повторенні first містить «приклад», а second — None.

None не є введеним словом, тому підходить для перевірки вичерпання. Порожній рядок одразу завершить цикл із кількістю пар 0. У завданні 1 організуйте окреме виведення перших двох слів і перебір решти через for.

Приклад до завдання 2

Генератор послідовно утворює повні фрагменти по два символи без перекриття й повертає лише фрагменти без пробілів у верхньому регістрі. Останній одиночний символ пропускає. Перший результат споживаємо окремо.

Код Python

```python
def compact_chunks(text):
    for position in range(0, len(text) - 1, 2):
        chunk = text[position:position + 2]
        if " " not in chunk:
            yield chunk.upper()

text = input("Текст: ")
generator = compact_chunks(text)
first = next(generator, None)
count = 0
total = 0
print(f"Перший: {first}")
if first is not None:
    count += 1
    total += len(first)
for chunk in generator:
    print(f"Далі: {chunk}")
    count += 1
    total += len(chunk)
print(f"Кількість: {count}; символів: {total}")
second_count = 0
for chunk in generator:
    second_count += 1
print(f"Повторний прохід: {second_count}")
print("Новий генератор:")
for chunk in compact_chunks(text):
    print(chunk)
```

Приклад виконання в консолі

```
Текст: мова код
Перший: МО
Далі: ВА
Далі: ОД
Кількість: 3; символів: 6
Повторний прохід: 0
Новий генератор:
МО
ВА
ОД
```

Пояснення до прикладу

Крок 2 задає початки фрагментів: 0, 2, 4, 6. Верхня межа len(text) - 1 залишає лише повні пари символів; останній одиночний символ не обробляється. Із тексту «мова код» отримуємо «мо», «ва», « к», «од». Фрагмент із пробілом пропускається, решта перетворюється на «МО», «ВА», «ОД».

next() уже спожив «МО», тому його кількість і довжину враховано перед циклом. for обробляє два наступні результати: разом 3 фрагменти й 6 символів. Перевірка first is not None відрізняє результат від його відсутності. Другий прохід не дає елементів, а новий виклик compact_chunks(text) створює незалежний генератор.

Приклад до завдання 3

Генератор добирає слова довжиною не більше двох символів. Звичайна функція повністю обходить його й повертає кількість таких слів. Декоратор показує аргумент і готовий підсумок та зберігає повернене значення. Перевіримо також порожній рядок і повторний виклик.

Код Python

```
def show_call(function):
    def wrapper(text):
        print(f"Аргумент: {text!r}")
        result = function(text)
        print(f"Результат: {result}")
        return result
    return wrapper

def short_words(text):
    for word in text.split():
        if len(word) <= 2:
            yield word

@show_call
def short_word_count(text):
    count = 0
    for word in short_words(text):
        count += 1
    return count

text = input("Речення: ")
answer = short_word_count(text)
print(f"Повернено: {answer}")
empty_answer = short_word_count("")
print(f"Порожній рядок: {empty_answer}")
again = short_word_count(text)
print(f"Повторний виклик: {again}")
```

Приклад виконання в консолі

```
Речення: я і мова
Аргумент: 'я і мова'
Результат: 2
Повернено: 2
Аргумент: ''
Результат: 0
Порожній рядок: 0
Аргумент: 'я і мова'
Результат: 2
Повторний виклик: 2
```

Пояснення до прикладу

Під час оголошення @show_call замінює ім’я short_word_count обгорткою wrapper, але ще не викликає її. При зверненні short_word_count(text) обгортка друкує аргумент, викликає початкову функцію, друкує її підсумок і повертає те саме число. Отже, show_call повертає функцію, а wrapper — результат обчислення.

Звичайна функція при кожному виклику створює новий short_words(text) і повністю споживає його в циклі. У «я і мова» підходять «я» та «і», тому результат 2. Порожній рядок дає 0, а повторний виклик на початковому тексті знову дає 2. Формат !r показує лапки навколо аргументу, роблячи порожній рядок видимим. Значення answer, empty_answer та again підтверджують збереження return.

Контрольні питання

1. Чим ітерований об’єкт відрізняється від ітератора?
2. З якого місця for продовжує після next()?
3. Для чого потрібне типове значення next()?
4. Коли починає виконуватися тіло генератора?
5. Як yield відрізняється від return?
6. Чому first is not None точніше за if first?
7. Як урахувати окремо спожитий перший елемент?
8. Що дає повторний обхід вичерпаного генератора?
9. Що повертає декоратор і що повертає його обгортка? Поясніть на прикладі.
10. Чому декорування створення генератора не показує готовий підсумок?