Ітератори

№ уроку: 6 **Курс:** Python Essential

Засоби навчання: PyCharm

Огляд, ціль та призначення уроку

Після завершення уроку учні матимуть уявлення про механізм ітераторів, навчаться створювати власні ітератори.

Вивчивши матеріал даного заняття, учень зможе:

- Мати уявлення про внутрішні механізми роботи циклу for у Python
- Створювати та використовувати ітератори

Зміст уроку

1. Що таке ітератор?
2. Ітерування через ітератор
3. Як цикл for працює з ітераторами?
4. Створення власних ітераторів

Резюме

- *Контейнер* – це тип даних, який інкапсулює у собі значення інших типів.
- *Ітерабельний об'єкт* (в оригінальній термінології – іменник «*iterable*») – це об'єкт, який може повертати значення по одному за раз. Приклади: всі контейнери та послідовності (списки, рядки тощо), файли, а також екземпляри будь-яких класів, в яких визначено метод_iter_() или_getitem_ ().
- Метод__iter__() повертає об'єкт-ітератор. Метод__getitem__() повертає елемент контейнера за ключем або індексом.
- Ітерабельні об'єкти можуть бути використані всередині циклу for, а також у багатьох інших випадках, коли очікується послідовність (функції sum(), zip(), map() тощо).
- Коли ітерабельний об'єкт передається у вбудовану функцію iter(), вона повертає ітератор для об'єкта, який дозволяє один раз пройти по значенням ітерабельного об'єкта.
- При використанні ітерабельних об'єктів зазвичай не потрібно викликати функцію iter() або працювати з ітераторами безпосередньо, так як цикл for робить це автоматично.
- *Ітератор (iterator)* – це об'єкт, який представляє поток даних. Повторювані виклики методу _next_() (next() у Python 2) ітератора або передача його вбудованій функції next() повертає наступні елементи потоку.
- Якщо більше не залишилося даних, викидається виняток StopIteration. Після цього ітератор вичерпано і будь-які подальші виклики його методу__next__() знову генерують виняток StopIteration.
- Ітератори повинні мати метод__iter_, який повертає сам об'єкт ітератора, отже будь-який ітератор також є ітерабельним об'єктом і може бути використаний майже скрізь, де приймаються ітерабельні об'єкти. Одним із винятків є код, який проходить по ітератору кілька разів. Контейнери (наприклад, список) щоразу створюють новий ітератор кожного разу, коли вони передаються у функцію iter() або використовуються в циклі for. Спроба зробити це з ітератором поверне вичерпаний ітератор із попереднього циклу, і він виглядатиме як порожній контейнер.

План заняття:

CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Ітератори в Python всюди. Вони елегантно вбудовані в for цикли, включення, генератори тощо, але зачасту ми їх просто не помічаємо. Більшість вбудованих типів даних ітерабельні "з коробки": списки, кортежі, рядки. Кожен із цих типів ми можемо "перебрати" у циклі.
Ітератори в Python — це звичайні об'єкти, які можна ітерувати. Такий об'єкт повертатиме по одному елементу даних за один раз.
Технічно, ітератор повинен вміщати в собі два спеціальні методи: : __iter__() та __next_(), які зазвичай називаються протоколом ітератора.
Об'єкт називається ітерабельним, якщо ми можемо отримати з нього ітератор. Python має вбудовану функцію iter(), яка повертає ітератор об'єкта.
Насправді функція iter() просто викликає метод __iter__() об'єкта.

Ітерування через ітератор
Ми використовуємо next(), щоб вручну ітерувати всі елементи ітератора. Коли ми отримаємо всі елементи та ітератор буде порожнім, ми отримаємо виняток StopIteration:

```
# Створюємо список
my_list = ["One", "piece", "per", "time"]

# Отримуємо ітератор за допомогою
iter()my_iter = iter(my_list)

# Ітеруємо об’єкт за допомогою next()

# Виведення: One
print(next(my_iter))

# next(obj) is same as obj.__next__()

# Виведення: piece
print(my_iter.__next__())

# Виведення: per
print(my_iter.__next__())

# Виведення: time
print(my_iter.__next__())

# наступний виклик викене виключення StopIteration, так як елементи закінчилися
next(my_iter)
One
piece
per
time
Traceback (most recent call last):
  File "temp.py", line 27, in <module>
    next(my_iter)
StopIteration
```

Ми вже знайомі зі способом ітерувати об'єкт елегантніше за допомогою циклу for. З його допомогою ми можемо ітерувати будь-який об'єкт, який вміє повертати ітератор: списки, рядки, файли тощо:

```
my_list = ["One", "piece", "per", "time"]

for item in my_list:
    print(item)
```

Як цикл for працює з ітераторами?
Цикл for вміє автоматично виконувати ітерацію за списком. Фактично цикл for може виконувати ітерацію будь-якого ітератора. Давайте докладніше розглянемо, як насправді реалізований цикл for у Python

```
for element in iterable:
    pass
```


CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
    # якісь операції над елементом
```

Насправді, приклад вище аналогічний наступному:

```
# отримання ітератора ітерабельного об’єкту
iter_obj = iter(iterable)

# нескінченний цикл
while True:
    try:
        # отримання наступного елементу
        element = next(iter_obj)
        # якісь операції над елементом
    except StopIteration:
        # Якщо отримали StopIteration, выходимо з циклу
        break
```

Таким чином, всередині циклу for створюється об'єкт-ітератор iter_obj шляхом виклику iter() для об'єкта, що ітерується.
Фактично, цикл for насправді є нескінченним циклом while.
Всередині циклу він викликає next() для отримання наступного елемента та виконує тіло циклу for з цим значенням. Після того, як всі елементи вичерпані, викликається виключення StopIteration, яке перехоплюється всередині, і цикл закінчується. Зауважте, що будь-які інші винятки не будуть оброблені.

Створення власних ітераторів
Створити ітератор у Python просто. Нам просто потрібно реалізувати методи __iter__() та __next__().
Метод __iter_() повертає сам об'єкт ітератора. При необхідності можна виконати певну ініціалізацію.
Метод __next__() повинен повертати наступний елемент у послідовності. Після досягнення кінця та при наступних викликах він повинен викликати StopIteration.
Приклад, який повертає наступне число з ряду Фібоначчі на кожній ітерації:

```
class Fibonacci:

    def __init__(self, max):
        self.max = max

    def __iter__(self):
        self.count = 0
        self.last_numbers = (0, 1)

        return self

    def __next__(self):
        last_number, current_number = self.last_numbers
        last_number, current_number = current_number, last_number + current_number

        self.last_numbers = last_number, current_number
        self.count += 1

        if self.count > self.max:
            raise StopIteration

        return last_number

fibo = Fibonacci(10)

for number in fibo:
    print(number)
```


CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Закріплення матеріалу

- Що таке ітерабельний об'єкт?
- Що таке ітератор?
- Як створити власний ітератор?

Додаткове завдання

Завдання
Напишіть ітератор, який повертає елементи заданого списку у зворотному порядку (аналог reversed).

Самостійна діяльність учня

Завдання 1
Реалізуйте цикл, який перебиратиме всі значення ітерабельного об'єкту iterable

Завдання 2
Взявши за основу код прикладу example_5.py, розширте функціональність класу MyList, додавши методи очищення списку, додавання елемента у довільне місце списку, видалення елемента з кінця та довільного місця списку.

Рекомендовані ресурси

Документація Python
https://docs.python.org/3/tutorial/classes.html#iterators

Статті у Вікіпедії про ключові поняття, розглянуті на цьому уроці
https://ru.wikipedia.org/wiki/Итератор

CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com