Разбор домашнего задания

Роман Булгаков
Спикер курса

Skillbox

Задача «Стек»

Стек — это абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»).

stack = [ ]

Добавление: stack.append()

Удаление: stack.pop()

Тарелок: 0

Тарелок: 1

3 недоступны

Тарелок: 4

Итераторы

Роман Булгаков
Спикер курса

Skillbox

Итерируемые объекты и итераторы

```
items = [10, 20, 30]

for i_elem in items:
    print(i_elem)
```

← Итерируемый объект

← Цикл for работает с итератором объекта items

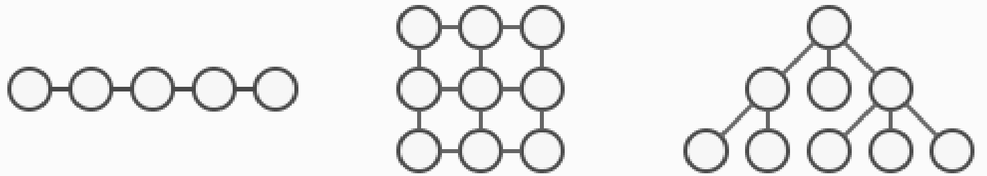
Итерируемый объект (iterable) — это объект, который способен возвращать элементы по одному с помощью цикла. Итерируемый объект содержит итератор.

Итератор — это объект, который позволяет перебирать элементы коллекции.

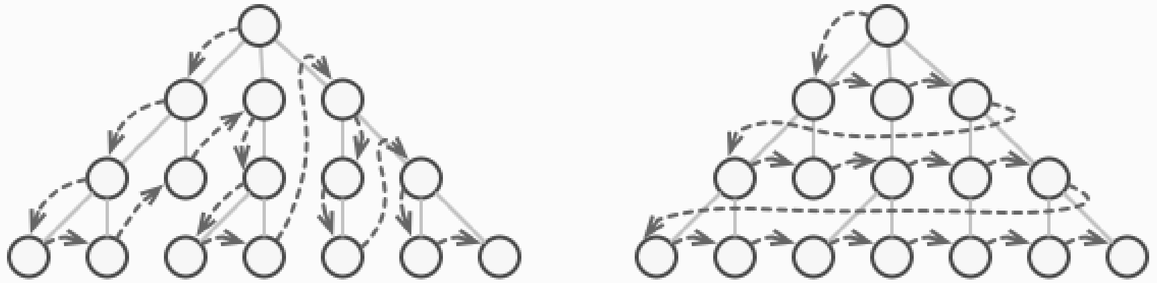
Коллекции

```python
my_list = [10, 20, 30]

my_str = 'Some string'

my_tuple = (10, 20, 30)

my_dict = {1: 'a', 2: 'b'}
```

Базовые коллекции



Разные типы коллекций



Обход одной коллекции разными способами

Паттерны проектирования

Реализация итераторов

Роман Булгаков
Спикер курса

Skillbox

Задача «Матрица»

Условия задачи:

- класс RandomNumbers
- атрибут limit — количество чисел

Выходные данные:

- итерируемый объект класса

```
class RandomNumbers:
    def __init__(self, limit):
        self.limit = limit
```

Числа Фибоначчи

```python
def fibonacci(number):
    result = []
    cur_val = 0
    next_val = 1
    for _ in range(number):
        result.append(cur_val)
        cur_val, next_val = next_val, cur_val + next_val

    return result
```

```python
fib_seq = fibonacci(10000000)
print(8 in fib_seq)
```

0 1 1 2 3 5 8 13 ....

Остальные числа занимают много памяти (а также увеличивается время работы программы).

Генераторы и их реализация

Роман Булгаков
Спикер курса

Skillbox

Числа Фибоначчи

```python
def fibonacci(number):
    result = []
    cur_val = 0
    next_val = 1
    for _ in range(number):
        result.append(cur_val)
        cur_val, next_val = next_val, cur_val + next_val

    return result
```

Обычная функция

```python
class Fibonacci:
    """ Итератор последовательности Фибоначчи из N элементов """

    def __init__(self, number):
        self.counter = 0
        self.cur_val = 0
        self.next_val = 1
        self.number = number

    def __iter__(self):
        self.counter = 0
        self.cur_val = 0
        self.next_val = 1
        return self

    def __next__(self):
        self.counter += 1
        if self.counter > 1:
            if self.counter > self.number:
                raise StopIteration()
            self.cur_val, self.next_val = self.next_val, self.cur_val + self.next_val

        return self.cur_val
```

Итератор

Генераторы

```python
def fibonacci(number):
    cur_val = 0
    next_val = 1
    for _ in range(number):
        yield cur_val
        cur_val, next_val = next_val, cur_val + next_val
```

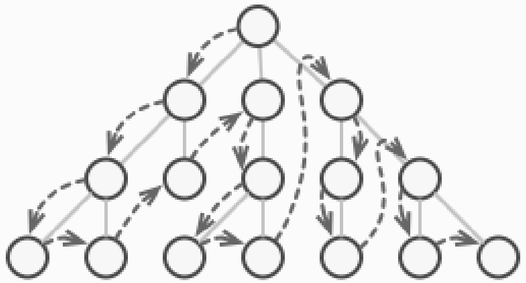
Генераторы — это итераторы, реализованные в виде функции, по которым можно итерироваться только один раз.

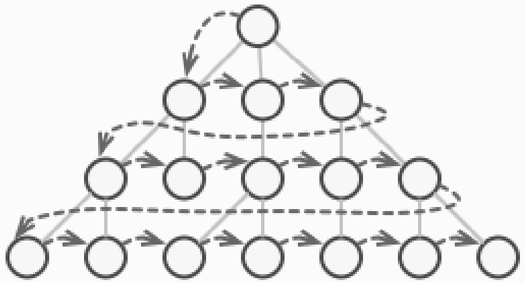
Генераторы — это функции, которые используют выражение yield

Методы __iter__ и __next__ реализуются автоматически

Отличие: итераторы — в основном для структур данных, генераторы — для генерации данных «на лету».

Генераторы





Сложный обход той или иной структуры

```python
class Family:
    def __init__(self):
        self.dad = 'Tom'
        self.mom = 'Alisa'
        self.counter = 0

    def __iter__(self):
        self.counter = 0
        return self

    def __next__(self):
        self.counter += 1
        if self.counter == 1:
            return 'Папа - {dad}'.format(dad=self.dad)
        if self.counter == 2:
            return 'Папа - {mom}'.format(mom=self.mom)
        if self.counter == 3:
            return 'Конец'
        raise StopIteration()
```

В любой класс можно добавить итератор

Итоги урока

- Генераторы — это итераторы, реализованные в виде функции
-
```
def fibonacci(number):
    . . . . .
    yield cur_val
```
- Ленивые вычисления как в итераторе (но код намного короче)
-
```
cubes_gen = (num ** 3 for num in range(10))
```

Аннотации типов

Роман Булгаков
Спикер курса

Skillbox

Итоги модуля

- ```
  def __iter__(self):
      . . . . .
  def __next__(self):
  ```
- ```
  def fibonacci(number):
      . . . . .
      yield cur_val
  ```
- lazy evaluation
- cubes_gen = (num ** 3 for num in range(10))
- def __init__(self, name: str, age: int, friends: list) -> None: