skypro

Включения и генераторы. Шпаргалка

Процессы итерации

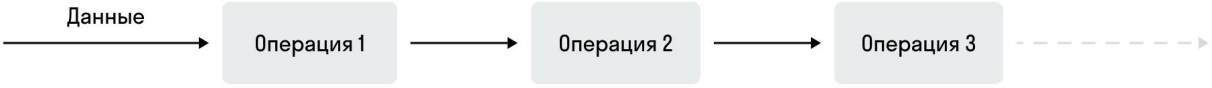
Пример итерации:

```
for i in range(10):
      print(i)

>>> 0
    1
    2
    3
    4
    5
    6
    7
    8
    9
```

Конвейерная разработка

Конвейер для данных — это структура, которая представляет собой последовательность операций обработки данных, где результат каждой операции передается на вход следующей.



Функция map()

Функция `map()` — применяет указанную в аргументе функцию ко всем элементам входной последовательности и возвращает итератор с результатами.

```
# Удвоить каждый элемент списка
numbers = [1, 2, 3, 4, 5]
doubled_numbers = map(lambda x: x * 2, numbers)
```

Функция filter()

Функция `filter()` — фильтрует элементы из входной последовательности на основе заданного условия.

```
# Оставить только четные числа
numbers = [1, 2, 3, 4, 5]
even_numbers = filter(lambda x: x % 2 == 0, numbers)
```

Функция chain()

Функция `chain()` — объединяет несколько последовательностей в одну.

Чтобы работать с функцией `chain`, ее необходимо импортировать из модуля `itertools`:

```
from itertools import chain

# Объединить два списка в один
list1 = [1, 2, 3]
list2 = [4, 5, 6]
combined_list = list(chain(list1, list2))
```

Генераторы списков

Генераторы списков — способ создания списков в Python, с помощью которого можно: создавать список, используя другой список или итератор и применяя к каждому элементу определенное выражение/функцию; заменять циклы `for` и `if`; упрощать создание списков, преобразование и анализ данных.

```
result = [x for num in range(20) for x in [num, num] if num % 2 == 0]
# [0, 0, 2, 2, 4, 4, 6, 6, 8, 8, 10, 10, ..., 18, 18]
```

Синтаксис генераторов списка

```
[выражение for переменная in источник if условие]
```

Генераторные выражения

Генераторные выражения — способ создания итераторов в Python, с помощью которого можно: генерировать элементы последовательности без хранения их всех в памяти; работать с большими объемами данных; снижать использование памяти и ускорять выполнение программы.

```
# Генераторное выражение использует круглые скобки
(x * x for x in range(10))

>>> <generator object <genexpr> at 0x7fe76f7e5db0>
```

Генераторный объект (generator object) — является итератором, который позволяет откладывать вычисление элементов последовательности до появления необходимости в них.

```
def print6(xs):
    for i, x in enumerate(xs):
        print(x)
        if i == 5:
            break

i = (x * x for x in range(10))
print6(i) # Вызываем функцию для выдачи 6 элементов
>>> 0
    1
    4
    9
    16
    25
print6(i)  # Продолжаем перебирать элементы с той точки, где остановились
>>> 36
    49
    64
    81
print6(i)  # Больше ничего не осталось
```

Пример с функцией any

```
any(x > 100 for x in range(1000000))

>>> True
```

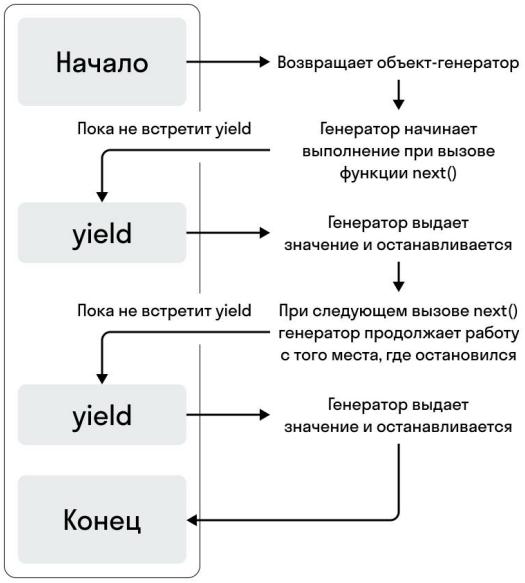
Функция `any` — принимает итерируемый объект (в данном случае генераторное выражение) и возвращает `True`, если хотя бы один элемент в этом объекте истинный (не равен нулю, не является пустым, соответствует условию и т. д.).

Функции-генераторы (yield)

Генераторная функция — это функция, использующая ключевое слово `yield`, чтобы возвращать итерируемый объект, который может генерировать значения по запросу, вместо того чтобы создавать список значений заранее.

Ключевое слово yield

`yield` — это ключевое слово, используемое внутри функций-генераторов для создания объектов, которые могут быть итерированы.




```
def simple_generator():
    yield 1
    yield 2
    yield 3


gen = simple_generator()

print(next(gen))
>>> 1
print(next(gen))
>>> 2
print(next(gen))
>>> 3
```

Генератор в цикле

```
for value in simple_generator():
    print(value)

>>> 1
    2
    3
```

Инициализация, приостановка и завершение генерации

Этапы генерации:

1. Инициализация.
2. Приостановка и получение значений.
3. Завершение генерации.

```
def f():
    print('Initializing...')
    yield 'one'
    print('Continue...')
    yield 'two'
    print('Stopping...')

i = f()
>>> Initializing...
print(next(i))
>>> one
print(next(i))
>>> Continue...
    two
print(next(i))
>>> Stopping...
StopIteration
```

Тестирование генераторов с помощью pytest

Тестирование генераторных выражений

Генераторное выражение для создания списка квадратов четных чисел от 0 до 9

```
squares = (x * x for x in range(10) if x % 2 == 0)
```

Тестирование генераторного выражения:

```
import pytest

def test_squares():
    expected_result = [0, 4, 16, 36, 64]
    result = list((x * x for x in range(10) if x % 2 == 0))
    assert result == expected_result
```

Тестирование функций-генераторов

Генераторная функция для создания бесконечной последовательности натуральных чисел:

```
def infinite_sequence(start=1):
    while True:
        yield start
        start += 1
```

Тестирование функции-генератора:

```
import pytest

def test_infinite_sequence():
    generator = infinite_sequence()
    assert next(generator) == 1
    assert next(generator) == 2
    assert next(generator) == 3
    # Дополнительные проверки по мере необходимости
```