Механизм итерации

In [1]:

```
res = [1, 2, 3, 4]
for i in res:
    print(i)
```

```
1
2
3
4
```

In [2]:

```
it = iter(res)
it
```

Out[2]:

```
<list_iterator at 0x7f43dd608710>
```

In [3]:

```
print(next(it))
print(next(it))
print(next(it))
print(next(it))
print(next(it))
```

```
1
2
3
4

----------------------------------------------------------------------
-----
StopIteration                             Traceback (most recent call
 last)
<ipython-input-3-6ced74e2ed96> in <module>()
      3 print(next(it))
      4 print(next(it))
----> 5 print(next(it))

StopIteration:
```

Примеры

In [4]:

```
it = reversed([1, 2, 3, 4])
it
```

Out[4]:

```
<list_reverseiterator at 0x7f43dd634c18>
```

In [5]:

```
it.__next__()
```

Out[5]:

```
4
```

In [6]:

```
dir(it)
```

Out[6]:

```
['__class__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__length_hint__',
 '__lt__',
 '__ne__',
 '__new__',
 '__next__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__setstate__',
 '__sizeof__',
 '__str__',
 '__subclasshook__']
```

In [7]:

```
print(next(it))
print(next(it))
print(it.__next__())
print(it.__next__())
print(next(it))
```

```
3
2
1

------------------------------------------------------------------------
-----
StopIteration                             Traceback (most recent call
 last)
<ipython-input-7-84bd0e09cd35> in <module>()
      2 print(next(it))
      3 print(it.__next__())
----> 4 print(it.__next__())
      5 print(next(it))

StopIteration:
```

In [8]:

```
it = {'a': 1, 'b': 2, 'c': 3}
it = iter(it)
```

In [9]:

```
d = {'a': 1, 'b': 2, 'c': 3}
for _, v in d.items():
    print(v)
```

```
1
2
3
```

In [10]:

```
print(next(it))
print(next(it))
print(next(it))
```

```
a
b
c
```

In [11]:

```
it = enumerate("параллелограм")
it
```

Out[11]:

```
<enumerate at 0x7f43dcd9e8b8>
```

In [12]:

```
print(next(it))
print(next(it))
print(next(it))
```

```
(0, 'п')
(1, 'а')
(2, 'р')
```

In [13]:

```
it = map(lambda x: 'e' + str(x), [1, 2, 3])
it
```

Out[13]:

```
<map at 0x7f43dcd9ddd8>
```

In [14]:

```
print(next(it))
print(next(it))
print(next(it))
```

```
e1
e2
e3
```

In [15]:

```
with open("files/untitled.py", 'r') as f_script:
    for i in f_script:
        print(i, end='')
```

```
import sys
print(sys.PATH)
x = 2
print(2 ** 8)
```

Генераторы

Скорость

In [16]:

```
L = list(range(1_000_000))
```

In [17]:

```
%%timeit

res = []
for i in L:
    res.append(i + 10)
```

```
103 ms ± 7.7 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

In [18]:

```
%%timeit

res = [i + 10 for i in L]
```

```
63.5 ms ± 7.84 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Лаконичность

In [19]:

```
with open("files/untitled.py", 'r') as f_script:
    result = f_script.readlines()

result
```

Out[19]:

```
['import sys\n', 'print(sys.PATH)\n', 'x = 2\n', 'print(2 ** 8)']
```

In [20]:

```
result = []

with open("files/untitled.py", 'r') as f_script:
    for line in f_script:
        if line.startswith('p'):
            result.append(line.rstrip().upper())

result
```

Out[20]:

```
['PRINT(SYS.PATH)', 'PRINT(2 ** 8)']
```

In [21]:

```
with open("files/untitled.py", 'r') as f_script:
    result = [line.rstrip().upper() for line in f_script if line.startswith('p')]

result
```

Out[21]:

```
['PRINT(SYS.PATH)', 'PRINT(2 ** 8)']
```

In [22]:

```
result = []

for w in 'abc':
    for f in '123':
        result.append(w + '-' + f)

result
```

Out[22]:

```
['a-1', 'a-2', 'a-3', 'b-1', 'b-2', 'b-3', 'c-1', 'c-2', 'c-3']
```

In [23]:

```
result = [w + '-' + f for w in 'abc' for f in '123']
result
```

Out[23]:

```
['a-1', 'a-2', 'a-3', 'b-1', 'b-2', 'b-3', 'c-1', 'c-2', 'c-3']
```

Правила "Бойцовского клуба"

Первое правило клуба: генераторы нельзя переиспользовать.

Второе правило клуба: генераторы нельзя переиспользовать.

In [24]:

```
gen = (chr(10 + i) for i in range(ord('a'), ord('f')))
gen
```

Out[24]:

```
<generator object <genexpr> at 0x7f43dcda15c8>
```

In [25]:

```
for i in gen:
    print(i)
```

```
k
l
m
n
o
```

In [26]:

```
next(gen)
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call
 last)
<ipython-input-26-6e72e47198db> in <module>()
----> 1 next(gen)

StopIteration:
```

Выражения-генераторы

In [27]:

```
gen = (x ** 2 for x in range(10))
gen
```

Out[27]:

```
<generator object <genexpr> at 0x7f43dcda1a40>
```

In [28]:

```
list(gen)
```

Out[28]:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Как конвертировать циклы в генераторы?

In [29]:

```
numbers = [1, 2, 3, 4, 5, 6]

odds_2 = []
for n in numbers:
    if n % 2 == 1:
        odds_2.append(2 * n)

odds_2
```

Out[29]:

```
[2, 6, 10]
```

In [30]:

```
numbers = [1, 2, 3, 4, 5, 6]

odds_2 = [2 * n for n in numbers if n % 2 == 1]

odds_2
```

Out[30]:

```
[2, 6, 10]
```

Простейшие генераторы

In [31]:

```
[x ** 2 for x in range(10)]
```

Out[31]:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [32]:

```
[2 ** i for i in range(13)]
```

Out[32]:

```
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]
```

Генераторы с условиями

In [33]:

```
s = [x ** 2 for x in range(10)]
[x for x in s if x % 2 == 0]
```

Out[33]:

```
[0, 4, 16, 36, 64]
```

In [34]:

```
[2 * i if i % 2 else i // 2 for i in range(10)]
```

Out[34]:

```
[0, 2, 1, 6, 2, 10, 3, 14, 4, 18]
```

Генераторы с множественной итерацией

In [35]:

```
a = []
for i in range(2):
    for j in range(3):
        a.append((i,j))
a
```

Out[35]:

```
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2)]
```

In [36]:

```
[(i, j) for i in range(2) for j in range(3)]
```

Out[36]:

```
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2)]
```

Генерация других объектов (не списков)

In [37]:

```
res = {'key1': 'value1', 'key3': 'value3', 'key2': 'value2'}
tuple((v, k) for k, v in res.items())
```

Out[37]:

```
(('value1', 'key1'), ('value3', 'key3'), ('value2', 'key2'))
```

In [57]:

```
{k: v for k, v in zip("python", [0, -1, 1, 2, -2, 3])}
```

Out[57]:

```
{'p': 0, 'y': -1, 't': 1, 'h': 2, 'o': -2, 'n': 3}
```

Функции-генераторы

In [38]:

```
def triangle(n):
    n = n + 1
    for i in range(1, n):
        yield ''.join('*' if n - i < j < n + i else ' ' for j in range(2 * n))

print(*triangle(5), sep='\n', end='')
```

```
      *
     ***
    *****
   *******
  *********
```

In [39]:

```
def gen_func():
    for w in 'abc':
        for f in '123':
            yield w + '-' + f

gen_func()
```

Out[39]:

```
<generator object gen_func at 0x7f43dcda18e0>
```

In [40]:

```
gen = gen_func()

print(next(gen))
print(next(gen))
print(next(gen))
```

```
a-1
a-2
a-3
```

In [41]:

```
for i in gen_func():
    print(i)
```

```
a-1
a-2
a-3
b-1
b-2
b-3
c-1
c-2
c-3
```

In [55]:

```python
def accumulator():
    total = 0
    while True:
        value = yield total
        print(f"Accepted: {value}")

        if not value:
            break
        else:
            total += value
    yield total
```

In [56]:

```python
from time import sleep

gen = accumulator()

print('Sum: {}'.format(next(gen)))
sleep(1)
print('Sum: {}'.format(gen.send(1)))
sleep(1)
print('Sum: {}'.format(gen.send(2)))

next(gen)
```

```
Sum: 0
Accepted: 1
Sum: 1
Accepted: 2
Sum: 3
Accepted: None
```

Out[56]:

```
3
```

In [44]:

```python
def gen_squares():
    return (i ** 2 for i in range(3))

def complex_gen():
    for i in "sphere":
        yield 'Letter:', i
        yield from gen_squares()
```

In [45]:

```
for i in complex_gen():
    print(i)
```

```
('Letter:', 's')
0
1
4
('Letter:', 'p')
0
1
4
('Letter:', 'h')
0
1
4
('Letter:', 'e')
0
1
4
('Letter:', 'r')
0
1
4
('Letter:', 'e')
0
1
4
```

In [46]:

```
def complex_gen(n):
    for i in range(n + 1):
        yield from (i for j in range(i))

print(*complex_gen(5))
```

```
1 2 2 3 3 3 4 4 4 4 5 5 5 5 5
```

In [47]:

```
def recursive_gen(a_curr, a_delta, a_max=5):
    yield a_curr

    if a_delta > 0:
        if a_curr + a_delta < a_max:
            yield from recursive_gen(a_curr + a_delta, a_delta)
        else:
            yield from recursive_gen(a_curr + a_delta, -a_delta)
    elif a_delta < 0 and a_curr + a_delta >= 0:
        yield from recursive_gen(a_curr + a_delta, a_delta)

def complex_gen(a_delta, a_max):
    yield from recursive_gen(0, a_delta, a_max)

print(*complex_gen(1, 5))
```

```
0 1 2 3 4 5 4 3 2 1 0
```

```
In [ ]:
```