Циклы

Существует 2 вида циклов `for` и `while`. `for` используется в случаях когда заранее известно количество итераций (циклов). `while` ипользуется когда количество итераций не известно и нужно повторить определенный блок кода пока собледено определенное условие.

For

`for` используется, когда нужно итерировать через списки `list`, строки `str`, кортежи `tuple` или любой другой итератор (например `range`).

```
for i in range(5):
    print(i)

>> 0
>> 1
>> 2
>> 3
>> 4
```

`for` циклы можно сочетать с условными инструкциями `if-else`.

```
for i in range(1, 11):
    if i % 2 == 0:
        print('Even number:', i)
    else:
        print('Odd number: ', i)

>> Odd number:  1
>> Even number: 2
>> Odd number:  3
>> Even number: 4
>> Odd number:  5
>> Even number: 6
>> Odd number:  7
```

```
>> Even number: 8
>> Odd number:  9
>> Even number: 10
```

---

zip(*iterables, strict=False)

Позволяет итерировать через несколько итераторов параллельно, создавая в каждом цикле ёё с элементом из каждого итератора.

```
even = [0, 2, 4, 6, 8]
odd = [1, 3, 5, 7, 9]

for item in zip(even, odd):
  print(item)

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

Стоит учесть, что итераторы не всегда могут быть одного размера. Если размеры итераторов разные, то, по умолчанию, фукция сработает наименьшее количество раз.

```
even = [0, 2, 4, 6, 8, 10, 12]
odd = [1, 3, 5, 7, 9]

for item in zip(even, odd):
  print(item)

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

Если задать параметр , то будет выдана ошибка `ValueError`.

```
even = [0, 2, 4, 6, 8, 10, 12]
odd = [1, 3, 5, 7, 9]

for item in zip(even, odd, strict=True):
    print(item)

>> (0, 1)
>> (2, 3)
>> (4, 5)
>> (6, 7)
>> (8, 9)
>> Traceback (most recent call last):
    File "<pyshell#5>", line 1, in <module>
      for item in zip(even, odd, strict=True):
ValueError: zip() argument 2 is shorter than argument 1
```

Генераторы

Генераторы это упрощенный способ создания коллекций (списков, кортежей, словарей). На уровне процессора генераторы работают быстрее и используют меньше оперативной памяти, также они упрощают код, так как все можно уместить в ожну строку кода.

> ### Например: Мы хотим создать список из первых десяти чисел возведенные в квадрат

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

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

> ### Анологично:

```
lst = []
for x in range(10):
    lst.append(x**2)
print(lst)

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

Также генераторы можно сочетать с инструкцией `if`.

> **Например: Мы хотим создать список из первых десяти чисел возведенные в квадрат, только если число четное.**

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

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

Также можно генерировать `dict`.

```
names = ['Ayaz', 'Seymur', 'Imad', 'Rufat']
ages = [22, 30, 25, 26]

dct = {name: age for name, age in zip(names, ages)}
print(dct)

>> {'Ayaz': 22, 'Seymur': 30, 'Imad': 25, 'Rufat': 26}
```

While

`while` используется в случае, если нужно повторить блок кода пока соблюдено определенное условие. Если условие не соблюдено цикл закончится.

```
count = 1
while count < 5:
    print(count)
    count += 1

>> 1
>> 2
>> 3
>> 4
```

`while` циклы можно сочетать с условными инструкциями `if-else`.

```
n = 10

while n > 0:
    if n%2 == 0:
        print('Even number:', n)
    else:
        print('Odd number:', n)

    n -= 1

>> Even number: 10
>> Odd number: 9
>> Even number: 8
>> Odd number: 7
>> Even number: 6
>> Odd number: 5
>> Even number: 4
>> Odd number: 3
>> Even number: 2
>> Odd number: 1
```

Инструкции контроля циклами (loop control statements)

Существует 3 ключевых слова для конроля циклами: `break`, `continue` и `pass`.

`break`: останавливает цикл на месте. Оно используется для незамедлительного выхода из цикла.

> **Например: У нас есть строка, и мы хотим расмечатать каждый символ в этой строке до тех пор пока не попадется пробел. Если попался пробел, мы хотим остановить программу**

```
text = 'azazel_guduruf is the nickname on Steam'

for symbol in text:
    if symbol.isspace():
        break

    print(symbol, end=' ') # параметр "end" задает символ, который будет распечатан после
```

```
 того, что мы печатаем (text[i]). По умолчанию end='\n'. Поэтому, по умолчанию, каждый сле
дующий текст печатается на новой строке в консоли

>> a z a z e l _ g u d u r u f
```

`continue`: **Пропускает текущий цикл и сразу прыгает на следующий.**

> **Например: Мы хотим найти обратное значение чисел (1/x) из коллекции чисел. Проблема в том, что на 0 делить нельзя. Поэтому, если попадается 0, мы сразу прыгаем на следующий цикл.**

```
for x in range(10):
    if x == 0:
        continue

    print(1/x, end=' | ')

>> 1.0 | 0.5 | 0.3333333333333333 | 0.25 | 0.2 | 0.16666666666666666 | 0.14285714285714285
| 0.125 | 0.1111111111111111 |
```

`pass`: **Ничего не делает. Он используется, в основном, как заполнитель. Бывают случаи, когда был задан какой-то метод, но пока не был имплементирован. Также импользуется для обработки ошибок.**

```
text = 'azazel_guduruf is the nickname on Steam'

for symbol in text:
    if symbol.isspace(): # Например, мы пока не решили или не написали, что делать если на
м попдется пробел, подтому мы пока нисего не делаем
        pass
    else:
        print(symbol, end=' ')

>> a z a z e l _ g u d u r u f   i s   t h e   n i c k n a m e   o n   S t e a m
```

```
for x in range(10):
    if x == 0:
        pass
```

```
    else:
        print(1/x, end=' | ')

>> 1.0 | 0.5 | 0.3333333333333333 | 0.25 | 0.2 | 0.16666666666666666 | 0.14285714285714285
| 0.125 | 0.1111111111111111 |
```