Циклы while и for

Общий формат while

```
while проверка:       # Проверка цикла
   операторы          # Тело цикла
else:                 # Необязательная часть else
   операторы   # Выполняются, если не произведен выход из цикла с помощью break
```

```
>>> while True:
...     print('Type Ctrl-C to stop me!')
```

```
>>> a=0; b=10
>>> while a < b:                  # Один способ написания циклов с подсчетом
...     print(a, end=' ')
...     a += 1                     # Или a = a + 1
...
0 1 2 3 4 5 6 7 8 9
```

do until

- Python не располагает тем, что в других языках называется оператором цикла “do until”

```
while True:
    ...цело цикла...
    if exitTest(): break
```

break, continue, pass и конструкция else цикла

`break`	Переходит за пределы ближайшего заключающего цикла (после всего оператора цикла).
`continue`	Переходит в начало ближайшего заключающего цикла (на строку заголовка цикла).
`pass`	Вообще ничего не делает: это пустой оператор-заполнитель.
Блок `else` цикла	Выполняется тогда и только тогда, когда происходит нормальный выход из цикла (т.е. без выполнения оператора `break`).

```
while проверка:
    операторы
    if проверка: break          # Выход из цикла с пропуском else, если есть
    if проверка: continue       # Переход на проверку в начале цикла
else:
    операторы                   # Выполняется, если не было break
```

Оператор continue/break

```
x = 10
while x:
    x = x-1                          # Либо x -= 1
    if x % 2 != 0: continue          # Нечетное? Тогда пропустить print
    print(x, end=' ')
```

```
>>> while True:
...     name = input('Enter name:')  # Использовать raw_input() в Python 2.X
...     if name == 'stop': break
...     age = input('Enter age: ')
...     print('Hello', name, '=>', int(age) ** 2)
...
```

Конструкция else цикла

```
x = y // 2                                  # Для значений y > 1
while x > 1:
    if y % x == 0:                          # Остаток от деления
        print(y, 'has factor', x)           # Имеет сомножитель
        break                               # Пропуск else
    x -= 1
else:                                       # Нормальный выход
    print(y, 'is prime')                    # Является простым
```

Конструкция else цикла

```
found = False
while x and not found:
    if match(x[0]):          # Значение в начале?
        print('Ni')
        found = True
    else:
        x = x[1:]            # Усечь в начале и повторить
if not found:
    print('not found')       # Значение не найдено
```

```
while x:                     # Выйти, когда x пусто
    if match(x[0]):
        print('Ni')
        break                # Выйти, пропустить else
    x = x[1:]
else:
    print('Not found')       # Выполняется только в случае исчерпания x
```

Циклы for

```
for цель in объект:                # Присваивает цели элементы объекта
    операторы                      # Повторяемое тело цикла: использует цель
else:                              # Необязательная часть else
    операторы                      # Если не встречался оператор break
```

```
for цель in объект:                # Присваивает цели элементы объекта
    операторы
    if проверка: break             # Выход из цикла с пропуском else
    if проверка: continue          # Переход в начало цикла
else:
    операторы                      # Если не встречался оператор break
```

Циклы for

```
>>> for x in ["spam", "eggs", "ham"]:
...     print(x, end=' ')
...
spam eggs ham
```

```
>>> sum = 0
>>> for x in [1, 2, 3, 4]:
...     sum = sum + x
...
>>> sum
10
>>> prod = 1
>>> for item in [1, 2, 3, 4]: prod *= item
...
>>> prod
24
```

Циклы for

```
>>> S = "lumberjack"
>>> T = ("and", "I'm", "okay")

>>> for x in S: print(x, end=' ')        # Итерация по строке
...
l u m b e r j a c k

>>> for x in T: print(x, end=' ')        # Итерация по кортежу
...
and I'm okay
```

Присваивание кортежей в циклах for

```
>>> T = [(1, 2), (3, 4), (5, 6)]
>>> for (a, b) in T:                    # Присваивание кортежей в действии
...     print(a, b)
...
1 2
3 4
5 6
```

```
>>> D = {'a': 1, 'b': 2, 'c': 3}
>>> for key in D:
...     print(key, '=>', D[key])    # Использование итерации по ключам
...                                 #   словаря и индексации
...
a => 1
c => 3
b => 2

>>> list(D.items())
[('a', 1), ('c', 3), ('b', 2)]

>>> for (key, value) in D.items():
...     print(key, '=>', value)     # Итерация сразу по ключам и значениям
...
a => 1
c => 3
b => 2
```

Присваивание кортежей в циклах for

```
>>> ((a, b), c) = ((1, 2), 3)  # Работает также и для вложенных последовательностей
>>> a, b, c
(1, 2, 3)
>>> for ((a, b), c) in [((1, 2), 3), ((4, 5), 6)]: print(a, b, c)
...
1 2 3
4 5 6
```

```
>>> a, *b, c = (1, 2, 3, 4)    # Расширенное присваивание последовательностей
>>> a, b, c
(1, [2, 3], 4)

>>> for (a, *b, c) in [(1, 2, 3, 4), (5, 6, 7, 8)]:
...     print(a, b, c)
...
1 [2, 3] 4
5 [6, 7] 8
```