Общая терминология циклов

> **Тело цикла** — это набор команд, находящихся на одном (четыре пробела) и более отступе от отступа самого цикла.
> Другими словами, **тело цикла** — это те команды, которые находятся внутри него и будут повторяться.

> **Итерация** — это один шаг цикла, повторное применение операции, прописанной в теле цикла.

В Python предусмотрены две встроенные конструкции, которые организуют цикл: **for** и **while**.

Цикл for

Для работы с циклом **for** используется следующая конструкция:

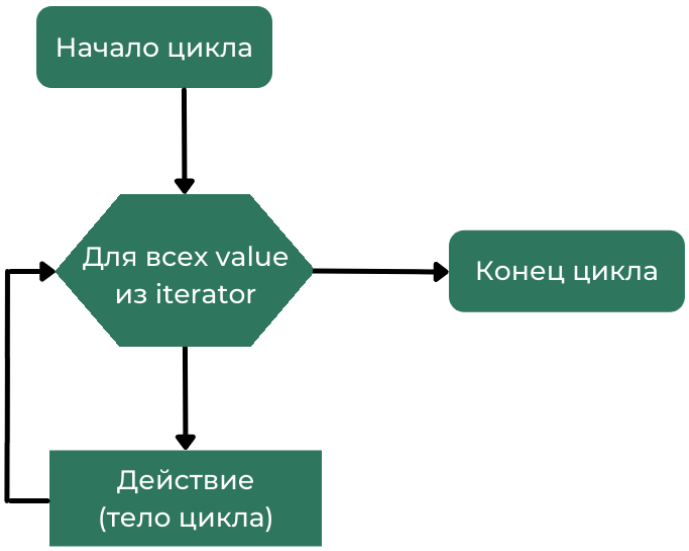
```
for value in iterator:
    # Начало блока кода с телом цикла
    ...
    ...
    ...
    # Конец блока кода с телом цикла
# Код, который будет выполняться после
цикла
```

Здесь:

- `for` — ключевое слово, с которого начинается цикл, отправная точка.
- `value` — переменная цикла (может иметь другое название), в которой на каждом шаге цикла (итерации) содержится текущее значение из итерируемого объекта `iterator`.
- `in` — оператор принадлежности, который указывает, откуда берутся значения для переменной `value`.

- `iterator` — итерируемый объект, из которого на каждой итерации извлекаются элементы (например, список, словарь, кортеж, строка и т. д.).

Блок-схема цикла for



Алгоритм создания цикла for

1. Определяем итерируемый объект, по которому будем проходить циклом (это может быть список, строка, кортеж, словарь или любой другой объект-итератор).
2. Когда вы создаёте цикл, скорее всего, вам нужно совершить какое-то действие с каждым элементом итерируемого объекта. Можно заранее определить для себя это действие, а можно придумать его при отладке программы.
3. Оборачиваем данную конструкцию в код. Для этого берём шаблон для создания цикла. Придумываем имя для переменной цикла, в которую будем помещать значения из итерируемого объекта.

Пример — последовательный вывод элементов списка на экран:

```
# Определяем итерируемый объект
```

```
my_list= [5, 9, 19]
# Подставляем его в шаблон для цикла и записываем имя переменной цикла
for element in my_list:
    # Указываем необходимые действия в теле цикла
    print('Element', element)
```

Совмещённые операторы

Длинная запись	Сокращённая запись
`value = value + a`	`value += a`
`value = value - a`	`value -= a`
`value = value / a`	`value /= a`
`value = value * a`	`value *= a`
`value = value ** a`	`value **= a`

Функция range()

Функция `range()` позволяет создавать последовательность целых чисел.

Общий синтаксис:

```
range(START, STOP, STEP)
```

Здесь:

- `START` — число, с которого начинается последовательность. По умолчанию последовательность начинается с 0.
- `STOP` — верхняя граница последовательности. Обязательный аргумент. **Важно!** `STOP` не включается в итоговый диапазон чисел.
- `STEP` — шаг последовательности. По умолчанию последовательность идёт с шагом 1.

Пример — список целых чисел от 0 до 5 (не включая 5):

```
list(range(5))
```

Пример — список целых чисел от -4 до 10 (не включая 10):

```
list(range(-4, 10))
```

Пример — список целых чисел от -4 до 10 (не включая 10) с шагом 3:

```
list(range(-4, 10, 3))
```

Способы обхода списка

1. **Цикл по элементам списка.**

```
weight_of_products = [10, 42.4, 240.1, 101.5, 98, 0.4, 0.3, 15]

max_weight = 100
num = 1

for weight in weight_of_products:
    if weight < max_weight:
        print('Product {}, weight: {} -passenger car'.format(num,
weight))
    else:
        print('Product {}, weight: {} -truck'.format(num, weight))
    num += 1
```

2. **Цикл по индексам списка.**

```
weight_of_products = [10, 42.4, 240.1, 101.5, 98, 0.4, 0.3, 15]
max_weight = 100
N = len(weight_of_products)
for i in range(N):
    if weight_of_products[i] < max_weight:
        print('Product {}, weight: {} -passenger car'.format(i+1,
weight_of_products[i]))
    else:
        print('Product {}, weight: {} -truck'.format(i+1,
```

```
weight_of_products[i]))
```

Цикл while

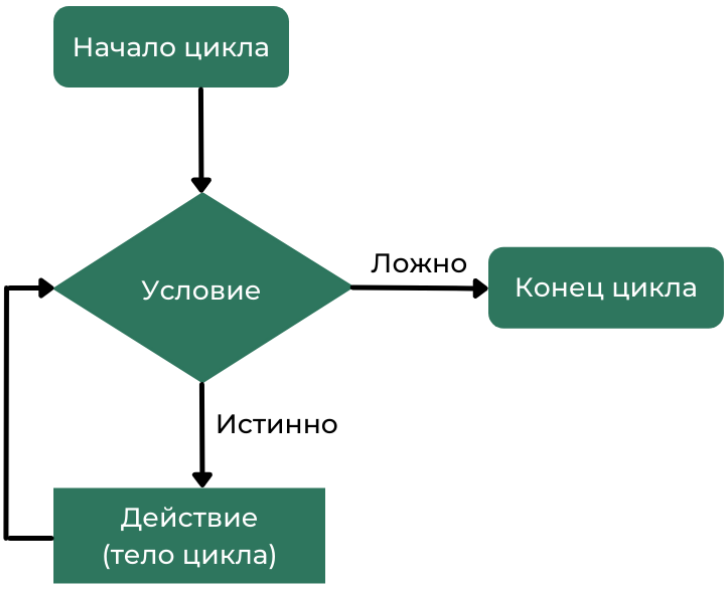
Цикл `while` выполняется до тех пор, пока истинно его условие. Как только оно становится ложным, цикл прерывается.

> **Условие** — это логическое выражение, которое может принимать значение True и False.

Для работы с циклом `while` используется следующая конструкция:

```
while # условие:
    # Начало блока кода с телом цикла
    # Пока условие истинно, цикл выполняется
    ...
    ...
    ...
    # Конец блока кода с телом цикла
# Код, который будет выполняться после цикла
```

Блок-схема цикла while



Алгоритм создания цикла while

1. Определяем, какое условие будет проверять цикл на каждой итерации. Это может быть любое выражение, которое возвращает булево значение — True или False.
2. Определяем действия, которые мы будем совершать внутри цикла.
3. Оборачиваем эту конструкцию в код. Для этого используем шаблон для создания цикла.

Пример — считаем, сколько раз к x нужно прибавить 2, чтобы x стал больше либо равен y:

```
x = 21
y = 55
# Задаём начальное значение количества итераций
count = 0
# Записываем условное выражение в цикл
while x < y:
    # Увеличиваем значение переменной x на 2
    # Равносильно x = x + 2
    x += 2
    # Увеличиваем количество итераций на 1
    # Равносильно count = count + 1
    count += 1
# Выводим результирующее количество итераций
print('Number of iterations', count)
```

Бесконечный цикл

При работе с циклом `while` нужно быть внимательными, ведь если условие остановки будет всегда True, то цикл никогда не завершится.

Пример:

```
# Плохо
n = 1
```

```
# В данной программе условие всегда True, цикл будет бесконечным
while n < 10:
    print("Hello World")
```

Чтобы остановить выполнение цикла, используется ключевое слово `break`:

```
# Хорошо
n = 1
# В данной программе условие всегда True, цикл будет бесконечным
while True:
    print("New client !")
    n += 1
    # Условие, при достижении которого цикл while будет принудительно завершён
    if n > 10:
        break
```

Особенность использования такого цикла `while` с условием внутри заключается в том, что тело цикла точно выполнится один раз, в отличие от цикла с предусловием. Такой цикл ещё называют **циклом с постусловием**.

Вложенные циклы

Циклы могут быть вложены друг в друга. Такие конструкции могут пригодиться при работе с вложенными структурами.

Пример — вложенный цикл по элементам внешнего и внутренних списков:

```
matrix = [
    [1, 2],
    [3, 4],
    [5, 6]
]
for row in matrix:
    print('Current row', row)
    for elem in row:
        print('Current elem', elem)
    print()
```

Первый цикл называют **циклом по первому уровню вложенности**, второй цикл — **циклом по второму уровню вложенности**. Если уровней вложенности всего два, то удобнее называть первый цикл **внешним**, а второй — **внутренним** (вложенным).

Пример — вложенный цикл по индексам внешнего и внутренних списков:

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

N = len(matrix)
M = len(matrix[0])
for i in range(N):
    print('Current i', i)
    print('Current row', matrix[i])
    for j in range(M):
        print('Current j', j)
        print('Current elem', matrix[i][j])
    print()
```

При работе с вложенными циклами стоит учитывать количество итераций.

Количество итераций во вложенном цикле = количество элементов в первой итерируемой последовательности * количество элементов во второй итерируемой последовательности

Чем больше итераций, тем больше времени необходимо на обработку кода.

Старайтесь избегать использования вложенных циклов в задачах, где в них нет необходимости.

Пример — подсчитать сумму элементов в первом столбце матрицы:

```
# Плохой способ:
table = [
    [1, 3, 6],
```

```
    [4, 6, 8],
    [10, 33, 53]
]
N = len(table)
M = len(table[0])
S = 0
count = 0
for i in range(N):
    for j in range(M):
        if j == 0:
            S += table[i][j]
        count += 1

# Лучше:
table = [
    [1, 3, 6],
    [4, 6, 8],
    [10, 33, 53]
]
N = len(table)
S = 0
count = 0
for i in range(N):
    S += table[i][0]
    count += 1
```

Функция enumerate()

Функция `enumerate()` часто используется в цикле `for` и позволяет выдавать одновременно индекс элемента последовательности (порядковый номер, начиная от 0), а также сам элемент итерируемого объекта.
Синтаксис такой конструкции будет следующим:

```
for index, value in enumerate(iterator):
    ...
```

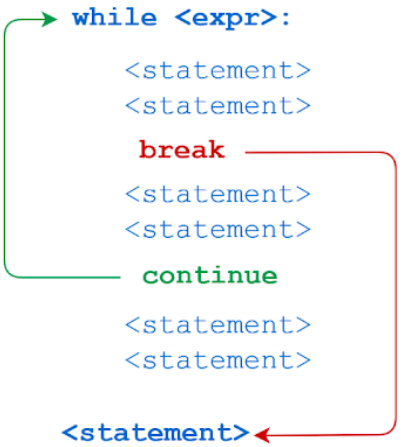
На каждой итерации в переменную `index` заносится индекс текущего элемента из `iterator`, а в `value` — значение текущего элемента.

Пример — одновременный проход по элементам списка и их индексам:

```
my_list = [5, 9, 13]
for index, value in enumerate(my_list):
    print(index, value)
```

Break

Любой цикл, встречая ключевое слово `break`, преждевременно заканчивает своё выполнение и переходит к основному коду программы. Ниже приведён пример для цикла `while`:



Python "while" Loops (Indefinite Iteration) – Real Python

Пример — добавление элементов в инвентарь, размер инвентаря — 3:

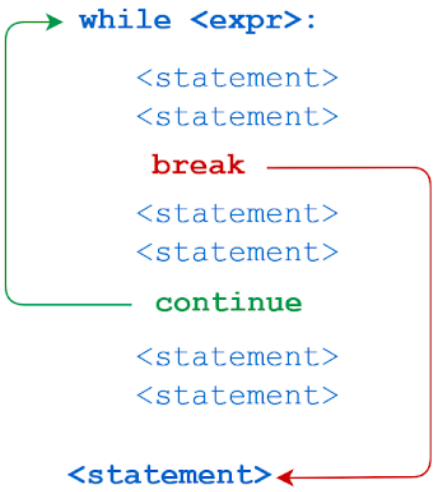
```
to_inventory = ['Blood Moon Sword', 'Sunset-colored sword', 'Bow of
Stars', 'Gain Stone']
inventory = []

for item in to_inventory:
    if len(inventory) == 3:
        print('Inventory is full!')
        break
    else:
        inventory.append(item)
```

Continue

Если в теле цикла встречается ключевое слово `continue`, то цикл пропускает весь код до конца тела цикла и переходит на следующий шаг.

Ниже приведён пример для цикла `while`:



Python "while" Loops (Indefinite Iteration) – Real Python

Пример:

```
client_status = {
    103303: 'yes',
    103044: 'no',
    100423: 'yes',
    103032: 'no',
    103902: 'no'
}

for user_id in client_status:
    if client_status[user_id] == 'no':
        continue
    else:
        print('Send present user', user_id)
```

Примечание. Ключевые слова `break` и `continue` могут использоваться как в цикле `while`, так и в цикле `for`.

Pass

Помимо ключевых слов `break` и `continue`, существует ключевое слово `pass`. Заглушка `pass` означает «ничего не делать». Обычно мы используем её, так как Python не позволяет создавать класс, функцию, цикл или оператор `if` без кода внутри.

Пример — цикл, в котором ничего не происходит, если элемент списка больше 3:

```
lst = [1,2,3,4,5]
for elem in lst:
    if elem > 3:
        pass
    else:
        print(i)
```