

Оператор цикла while

На этом занятии начнем знакомиться с еще одним ключевым элементом компьютерных программ – циклами. Вначале, давайте я на простом примере покажу, о чем идет речь. Представим, что нам нужно вычислить сумму целых чисел от 1 до N. Причем, N может быть сколько угодно большой: тысяча, миллион и так далее. Понятно, что мы не можем здесь просто записать операторы сложения чисел друг за другом для вычисления этой последовательности. Тем более, что на момент написания программы, N может быть неизвестна. Вот в таких ситуациях нам на помощь, как раз, и приходят циклы.



Циклы позволяют реализовывать некие повторяющиеся действия. Например, предположим, что маленькие панды прыгают с горки в течение часа, пока мама-панда не позовет всех к столу – кушать. На уровне текста это можно записать, так:

цикл (пока не прошел час):
 прыгаем с горки

То есть, пока истинно условие, цикл работает, как только условие становится ложным – прошел час, цикл завершается. Ровно так работает цикл `while`, о котором и пойдет речь на нашем занятии. Он имеет, следующее определение (синтаксис):

```
while <условие> :  
    оператор 1  
    оператор 2  
    ...  
    оператор N
```

заголовок

тело цикла

Вначале записывается ключевое слово `while`, затем, условие работы цикла, ставится двоеточие для определения блока операторов, работающих внутри этого цикла. Такой блок еще называют телом цикла, а ключевое слово `while` с условием – заголовком цикла.

Обратите внимание на форматирование. Здесь также, как и в условных операторах, набор операторов внутри тела цикла должны иметь одинаковые отступы относительно ключевого слова `while`.

Давайте вернемся к исходной задаче – вычисления суммы чисел от 1 до `N` и посмотрим, как здесь нам поможет цикл `while`. Вначале определим три вспомогательные переменные: `N` – значение последнего слагаемого; `s` – для хранения вычисленной суммы (начальное значение 0); `i` – значение текущего слагаемого (начинается с 1):

| | |
|-----------------------|---|
| <code>N = 1000</code> | Далее, так как сумму нужно вычислять, пока слагаемое <code>i</code> |
| <code>s = 0</code> | не достигнет значения <code>N</code> , то условие цикла можно |
| <code>i = 1</code> | определить, следующим образом: |

`while i <= N:` А внутри цикла будем выполнять следующие действия:

`s += i`

`i += 1`

Вначале `i` равна 1 и эта единица прибавляется к сумме `s`. После чего `i` увеличивается на 1 и становится равной 2. Затем, выполняется проверка условия цикла. Пока оно у нас истинно, поэтому снова попадаем внутрь тела цикла и к `s` прибавляется уже значение 2, а `i` опять увеличиваем на 1 и оно становится равным 3. И так до тех пор пока `i` не станет больше `N`. К этому моменту мы просуммируем все числа и результат будет храниться в переменной `s`. Вот принцип работы циклов, причем, во всех языках программирования, не только в Python.

```
s = 0
i = 1
N = 1000
while i <= N:
    s += i
    i += 1
```

↓ итерация цикла

Также однократное выполнение тела цикла в программировании называют **итерацией**. Я буду часто использовать этот термин, поэтому привел его, чтобы вы меня правильно понимали.

Давайте реализуем теперь эту программу на Python и посмотрим как она работает.

Возможно, у вас возник вопрос: а какие условия можно прописывать в циклах? В действительности, все те же самые, что и в условных операторах. В том числе и составные. Например, давайте будем вычислять сумму пока не дойдем до слагаемого N или до значения 50. Как цикл работает, пока истинно условие, то его следует записать, так:

```
while i <= N and i <= 50:  
    ...
```

Смотрите, мы здесь указали делать цикл пока i меньше или равна N и меньше или равна 50. Если хотя бы одно из этих подусловий

окажется ложным, то и все составное условие также станет ложным и цикл завершится. В результате, i достигнет или N или 50. Вот это нужно хорошо себе представлять: в циклах прописываются условия их работы, а не завершения.

Используя этот же цикл, мы легко можем поменять условие задачи и вычислить сумму чисел, которые меняются через один:

1, 3, 5, 7, ...

Для этого достаточно счетчик i увеличивать не на 1, а сразу на два: `i += 2`

При этом, условие цикла можно оставить прежним. Он завершится, как только i превысит N или 50. Здесь уже нет гарантии, что последнее слагаемое будет именно N или 50, но нам это и не нужно, мы лишь указываем завершить цикл, когда превысим одно из этих значений.

Конечно, внутри тела цикла можно записывать любые операторы, в том числе и функцию print(). Давайте, например, отобразим в консоли цифры от 0 до 9:

```
i = 0  
  
while i < 10:  
    print(i)  
    i += 1
```

Я здесь указал условие i меньше 10, а не $i \leq 9$, так как оператор $<$ работает несколько быстрее оператора \leq . Поэтому на практике предпочтительно, по возможности, применять не составные, а простые операторы: меньше, больше, равно и не равно. Хотя использование \leq или \geq не критично и вполне допустимо. Но, все же, по возможности, лучше прописывать простые операторы в условиях циклов.

Если нам нужно реализовать убывающую последовательность чисел, например:

-1, -2, -3, ..., -N то это делается аналогично, только с уменьшающимся счетчиком:

```
N = -10
i = -1

while i >= N:
    print(i)
    i -= 1
```

Здесь нет никаких ограничений.

В заключение этого занятия приведу еще пару примеров с оператором цикла `while`. Предположим, что пользователь вводит пароль, до тех пор, пока не введет верно. Это можно сделать, следующим образом:

```
pass_true = 'password'
ps = ''

while ps != pass_true:
    ps = input("Введите пароль: ")

print("Вход в систему осуществлен")
```

Наконец, внутри цикла `while` можно прописывать, например, и условные операторы. Давайте выведем все числа, кратные 3, которые нам встретятся при переборе целых значений от 1 до N:

```
N = 20
i = 1

while i <= N:
    if i % 3 == 0:
        print(i)

    i += 1
```