‹ Занятие 6. Цикл while

1. Цикл while

Цикл `while` (“пока”) позволяет выполнить одну и ту же последовательность действий, пока проверяемое условие истинно. Условие записывается до тела цикла и проверяется до выполнения тела цикла. Как правило, цикл `while` используется, когда невозможно определить точное значение количества проходов исполнения цикла.

Синтаксис цикла `while` в простейшем случае выглядит так:

```
while условие:
    блок инструкций

```

При выполнении цикла `while` сначала проверяется условие. Если оно ложно, то выполнение цикла прекращается и управление передается на следующую инструкцию после тела цикла `while` . Если условие истинно, то выполняется инструкция, после чего условие проверяется снова и снова выполняется инструкция. Так продолжается до тех пор, пока условие будет истинно. Как только условие станет ложно, работа цикла завершится и управление передастся следующей инструкции после цикла.

Например, следующий фрагмент программы напечатает на экран квадраты всех целых чисел от 1 до 10. Видно, что цикл `while` может заменять цикл `for ... in range(...)` :

запустить выполнить пошагово

```
i = 1
while i <= 10:
    print(i ** 2)
    i += 1

```

В этом примере переменная `i` внутри цикла изменяется от 1 до 10. Такая переменная, значение которой меняется с каждым новым проходом цикла, называется счетчиком. Заметим, что после выполнения этого фрагмента значение переменной `i` будет равно `11` , поскольку именно при `i == 11`

условие `i <= 10` впервые перестанет выполняться.

Вот еще один пример использования цикла `while` для определения количества цифр натурального числа `n`:

запустить выполнить пошагово

```
n = int(input())
length = 0
while n > 0:
    n //= 10  # это эквивалентно n = n // 10
    length += 1
print(length)

```

В этом цикле мы отбрасываем по одной цифре числа, начиная с конца, что эквивалентно целочисленному делению на 10 (`n //= 10`), при этом считаем в переменной `length`, сколько раз это было сделано.

В языке Питон есть и другой способ решения этой задачи:
`length = len(str(i))`.

2. Инструкции управления циклом

После тела цикла можно написать слово `else:` и после него блок операций, который будет выполнен *один раз* после окончания цикла, когда проверяемое условие станет неверно:

запустить выполнить пошагово

```
i = 1
while i <= 10:
    print(i)
    i += 1
else:
    print('Цикл окончен, i =', i)

```

Казалось бы, никакого смысла в этом нет, ведь эту же инструкцию можно просто написать *после* окончания цикла. Смысл появляется только вместе с инструкцией `break`. Если во время выполнения Питон встречает инструкцию `break` внутри цикла, то он сразу же прекращает выполнение этого цикла и выходит из него. При этом ветка `else` исполняться не будет. Разумеется, инструкцию `break` осмысленно вызывать только внутри

инструкции `if`, то есть она должна выполняться только при выполнении какого-то особенного условия.

Приведем пример программы, которая считывает числа до тех пор, пока не встретит отрицательное число. При появлении отрицательного числа программа завершается. В первом варианте последовательность чисел завершается числом 0 (при считывании которого надо остановиться).

запустить выполнить пошагово

```
a = int(input())
while a != 0:
    if a < 0:
        print('Встретилось отрицательное число', a)
        break
    a = int(input())
else:
    print('Ни одного отрицательного числа не встретилось')

```

Во втором варианте программы сначала на вход подается количество элементов последовательности, а затем и сами элементы. В таком случае удобно воспользоваться циклом `for`. Цикл `for` также может иметь ветку `else` и содержать инструкции `break` внутри себя.

запустить выполнить пошагово

```
n = int(input())
for i in range(n):
    a = int(input())
    if a < 0:
        print('Встретилось отрицательное число', a)
        break
else:
    print('Ни одного отрицательного числа не встретилось')

```

Другая инструкция управления циклом — `continue` (продолжение цикла). Если эта инструкция встречается где-то посередине цикла, то пропускаются все оставшиеся инструкции до конца цикла, и исполнение цикла продолжается со следующей итерации.

> Если инструкции `break` и `continue` содержатся внутри нескольких вложенных циклов, то они влияют лишь на исполнение самого

внутреннего цикла. Вот не самый интеллектуальный пример, который это демонстрирует:

запустить выполнить пошагово

```
for i in range(3):
    for j in range(5):
        if j > i:
            break
        print(i, j)

```

Увлечение инструкциями `break` и `continue` не поощряется, если можно обойтись без их использования. Вот типичный пример *плохого использования* инструкции `break` (данный код считает количество знаков в числе).

запустить выполнить пошагово

```
n = int(input())
length = 0
while True:
    length += 1
    n //= 10
    if n == 0:
        break
print('Длина числа равна', length)

```

Гораздо лучше переписать этот цикл так:

запустить выполнить пошагово

```
n = int(input())
length = 0
while n != 0:
    length += 1
    n //= 10
print('Длина числа равна', length)

```

Впрочем, на Питоне можно предложить и более изящное решение:

запустить выполнить пошагово

```
n = int(input())
print('Длина числа равна', len(str(n)))

```

3. Множественное присваивание

В Питоне можно за одну инструкцию присваивания изменять значение сразу нескольких переменных. Делается это так:

```
1  a, b = 0, 1
2
```

Этот код можно записать и так:

```
1  a = 0
2  b = 1
3
```

Отличие двух способов состоит в том, что множественное присваивание в первом способе меняет значение двух переменных одновременно.
Если слева от знака «=» в множественном присваивании должны стоять через запятую имена переменных, то справа могут стоять произвольные выражения, разделённые запятыми. Главное, чтобы слева и справа от знака присваивания было одинаковое число элементов.

Множественное присваивание удобно использовать, когда нужно обменять значения двух переменных. В обычных языках программирования без использования специальных функций это делается так:

запустить выполнить пошагово

```
1  a = 1
2  b = 2
3  tmp = a
4  a = b
5  b = tmp
6  print(a, b)
7  # 2 1
8
```

В Питоне то же действие записывается в одну строчку:

запустить выполнить пошагово

```
1  a = 1
2  b = 2
3  a, b = b, a
4  print(a, b)
5  # 2 1
6
```

Ссылки на задачи доступны в меню слева. Эталонные решения теперь доступны на странице самой задачи.

Показать мои решения задач этого урока