

Цикл `while` позволяет выполнять одну и ту же последовательность действий, пока проверяемое условие истинно. В этом заключается основное отличие от цикла `for`, основное назначение которого – «пробегать» по итерируемым объектам.

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 – в противном случае разумнее использовать `for`, это удобнее и быстрее.

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 передается следующей инструкции после цикла.

Например, следующий фрагмент программы напечатает на экран квадраты всех целых чисел от 1 до 10.

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

Видно, что этот цикл свободно заменяем на

```
for i in range(1, 11):  
    print(i)
```

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`:

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

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))`.

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:

```
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`, использование которой внутри цикла приводит к немедленному прекращению цикла, и при этом не исполняется ветка `else`. Разумеется, инструкцию `break` разумно вызывать только из инструкции `if`, то есть она должна выполняться только при выполнении какого-то особенного условия.

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.

Инструкции `break`, `continue` и ветка `else`: можно использовать и внутри цикла `for`. Тем не менее, увлечение инструкциями `break` и `continue` не поощряется, если можно обойтись без их использования. Вот типичный пример плохого использования инструкции `break`.

```
while True:
    length += 1
    n //= 10
    if n == 0:
        break
```

Пример для инструкции `continue`:

```
while x <= 10:
    if x == -1:
        x+=1
        continue
    print("{:+7.4f}".format(1/(x+1)))
    i+=1
```

Здесь пропускается вычисление выражения $\frac{1}{x+1}$ в точке, в которой знаменатель обращается в ноль.