Цикл while

Цикл `while` (“пока”) позволяет выполнять 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 заранее 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` (после блока инструкций). Если *условие* истинно, то последовательной выполняется блок инструкций данного цикла (инструкций может быть несколько), после чего условие проверяется снова и снова выполняется блок инструкций. Так продолжается до тех пор, пока условие будет истинно. Как только *условие* станет ложным, работа цикла завершится и управление передастся следующей инструкции после цикла.

Следующий фрагмент программы напечатает на экран квадраты всех целых чисел от 1 до 10:

```
i = 1
while i <= 10:
    print(i)
    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. Видно, что цикл `while` может заменять уже известный вам цикл `for ... in range(...)`

Но если *условие* оказывается истинным всегда (например, когда соответствующие переменные никак не изменяются внутри тела цикла), то цикл штатным образом закончиться не может, при этом говорят, что программа “зацикливается”.

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

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

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

В языке Python есть и другой способ решения конкретно этой задачи:

```
length = len(str(i)).
```

Обработка последовательностей неизвестной длины

В уроке по циклам `for` мы уже рассматривали задачи на обработку числовых последовательностей. С использованием цикла `while` подобные задачи можно решать и для случая, когда заранее не известно число элементов последовательности, а ввод ограничен тем или иным образом. Рассмотрим пример такой задачи.

В программу вводится последовательность натуральных чисел, заканчивающаяся нулем. Требуется найти сумму произведение этих чисел (очевидно, что ноль не должен участвовать в нахождении произведения). Решение может выглядеть так:

```
a = int(input())
p = 1
while a != 0:
  p *= a
  a = int(input())
print(p)
```

Подумайте, что выдаст эта программа, когда сразу будет введено значение 0 и что скорее всего логично выдавать в этом случае.

Окончание ввода может быть и более сложным. Например, чтобы 0 сам по себе мог быть членом последовательности, то окончанием ввода могут служить уже два нуля подряд. Что вынуждает хранить не только текущее считанное значение, но и предыдущее:

```
b = int(input())
a = int(input())
…
while a != 0 or b != 0:
  …# обработка b
  b = a
  a = int(input())
```

В этом случае последние два нуля в обработке принимать участия не будут.

Инструкции управления циклом

После тела цикла как и в условном операторе 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 этот же блок инструкций можно просто написать **после** окончания цикла. Смысл появляется только вместе с инструкцией `break`, использование которой внутри цикла приводит к немедленному прекращению цикла, при этом не исполняется и ветка `else`. Разумеется, инструкцию `break` осмысленно 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 (в цикле `while` — с новой проверки условия).

Инструкции `break`, `continue` и ветку `else:` можно использовать и внутри цикла `for`, причем именно там использование `break` часто бывает оправданным. 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
```

Здесь `True` — логическая константа, означающая, что *условие* всегда истинно, а цикл закачивается по инструкции `break`. Важно также понимать, что в случае вложенных циклов `break` завершает выполнение только одного цикла, а не всех сразу. Завершить выполнение, не только внутреннего, но и внешнего из двух циклов можно, например, с использованием логической переменной:

```
f = False
for i in range(N):
    for j in range(M):
        …
        if i*j > K:
            f = True
            break
        … # основное тело цикла
    if f:
        break
```

Начальное присваивание для переменной f является обязательным, в противном случае программа не сможет выполнять строку `if f`, так как переменной f может не существовать (в других языках программирования при описании f ей будет присвоено или значение по умолчанию `false` или случайное значение).

Задачи и указания к ним

A: 3642 В этой задаче решения с использованием break будут проигнорированы.

B: 3645 В этой задаче степени двойки вычисляются последовательным домножением результата на 2. Не забудьте, что по определению $2^0 = 1$, то есть 1 является степенью двойки. В этой задаче решения с использованием break будут проигнорированы.

C: 3798 Докажите утверждение из условия задачи, что если у числа нет делителей до корня из этого числа, то число простое. Используйте данный факт для ускорения работы программы. Оформлять решение с помощью функции не нужно. В этой задаче решения с использованием break будут проигнорированы.

D: 3647 Обратите внимание, что входные данные в этой задаче – действительные числа, считывание которых следует организовывать так `x = float(input())`. В этой задаче решения с использованием break будут проигнорированы.

E: 3660 В этой задаче решения с использованием break будут проигнорированы.

F: 3651 В этой задаче решения с массивами (списками и другими структурами данных, для тех, кто их знает) будут проигнорированы.

G. 3656 Обратите внимание, что для входных данных
1
1
0
ответом будет 1. В этой задаче решения с массивами (списками и другими структурами данных, для тех, кто их знает) будут проигнорированы.

H: 3657 В этой задаче решения с массивами (списками и другими структурами данных, для тех, кто их знает) будут проигнорированы.

I: 3665 В этой задаче решения с массивами и аналогичными структурами данных будут проигнорированы.

J: 3017 Докажите, что количество подряд идущих одинаковых цифр не может быть больше 9. Тогда для хранения последовательности можно использовать строку из цифр.