Урок Цикл while

Знакомство с циклом while

Аннотация

В этом уроке мы познакомимся с оператором цикла while. Цикл позволяет организовать многократное повторение одних и тех же действий. Кроме того, мы сделаем акцент вот на чем: в одной и той же строчке программы на разных итерациях цикла переменные могут иметь разное значение.

1. Цикл while

Сегодня мы научимся повторять заданные действия несколько раз. Для этого существуют операторы циклов. Мы разберем оператор цикла `while`. Он выполняет блок кода, **пока истинно** какое-то условие.

Напомним, условный оператор `if` проверяет условие и, в зависимости от того, истинно оно или ложно, выполняет либо не выполняет следующий записанный с отступом блок. После этого программа в любом случае выполняется дальше (там еще может быть `elif` или `else`, но сути это не меняет).

Цикл while

Оператор `while` («пока») тоже проверяет условие и тоже, в случае его истинности, выполняет следующий блок кода (**тело цикла**). Однако после выполнения этого блока кода выполняется не то, что идет после него, а снова проверяется условие, записанное после `while`.

Ведь при выполнении тела цикла значения каких-то переменных могли измениться — в результате

условие цикла может уже не быть истинным. Если условие все еще истинно, тело цикла выполняется снова. Как только условие цикла перестало выполняться (в том числе если оно с самого начала не было выполнено), программа идёт дальше — выполняются команды, записанные после тела цикла.

Условие цикла записывается как и для `if`: с помощью операций отношения (`>`, `>=`, `<`, `<=`, `!=`, `==`). Сложные условия можно составлять с помощью логических операций `not`, `and`, `or`.

Действия, расположенные в теле цикла (блок кода), записываются со смещением вправо на четыре пробела относительно начала слова `while`. Переменные, входящие в условие, должны на момент проверки условия цикла иметь значения.

```
while условие:
    блок кода (тело цикла)
```

Важно!

Один шаг цикла (выполнение тела цикла) еще называют **итерацией**.

Используйте цикл `while` всегда, когда какая-то часть кода должна выполниться несколько раз, причем невозможно заранее сказать, сколько именно.

Давайте посмотрим программу, в которой цикл будет выполняться, пока не введут число, меньшее или равное 0:

```
number = int(input())
while number > 0:
    print('Вы ввели положительное число! Вводите дальше.')
    number = int(input())
    print('Так-так, что тут у нас...')
print('Вы ввели отрицательное число или ноль. Всё.')
```

Разберемся, как будет работать эта программа.

Сначала выполняется первая строчка: `number = int(input())` — пользователь вводит целое число. (Мы предполагаем, что пользователь действительно ввел число, и программа не вылетела с ошибкой.) Предположим, он ввел число 10. Оно записано в переменной `number`.

Выполняется вторая строчка: `while number > 0`: — «пока number > 0» — здесь проверяется, выполнено ли условие `number > 0`. Поскольку мы предположили, что `number` в этот момент равно 10, тогда условие выполнено, поэтому дальше выполняется блок, записанный с отступом, — тело цикла.

Третья строчка программы выводит на экран строку, тут все понятно.

Четвертая строчка вновь считывает с клавиатуры число и сохраняет его в переменную `number`. Пусть пользователь ввел 2.

Когда выполнение программы доходит до конца тела цикла, происходит возврат к заголовку цикла (второй строчке программы) и повторная проверка условия. Поскольку `2 > 0`, снова выполняется тело цикла.

Третья строчка снова выводит на экран сообщение, четвертая строчка снова считывает число (пусть это будет число 3), пятая строчка снова выводит на экран сообщение...

Закончив тело цикла, опять проверяем условие в заголовке. `number` равно 3, `3 > 0`, поэтому продолжаем.

Третья строчка опять выводит на экран сообщение, четвертая строчка опять считывает число. Пусть теперь это будет −1. Обратите внимание: переменная `number` на каждой итерации цикла приобретает новое значение! Пятая строчка опять выводит на экран сообщение...

Вновь вернувшись на вторую строчку, получаем, что `-1 > 0` — ложно. Поэтому цикл завершается, тело цикла больше не выполняется, прыгаем сразу на следующую после цикла строчку программы — шестую. Она выводит последнее сообщение.

Все.

2. Составной оператор присваивания

Напомним, что в операторе присваивания одно и то же имя переменной может стоять и справа (в составе какого-то выражения), и слева. В этом случае сначала вычисляется правая часть со старым значением переменной, после чего результат становится новым значением этой переменной.
Ни в коем случае не воспринимайте такой оператор присваивания как уравнение!

```
number = int(input())  # например, 5
number = number + 1  # тогда здесь number становится равным 6
print(number)
```

Важно!

Для конструкций вида `number = number + 1` существует и сокращенная форма записи оператора присваивания: `number += 1`. Аналогично оператор `x = x + y` можно записать как `x += y`, оператор `x = x * y` — как `x *= y`, и так для любого из семи арифметических действий.

3. Сигнал остановки

Рассмотрим такую задачу: пользователь вводит числа. Пусть это будут цены на купленные в магазине товары, а наша программа — часть программного обеспечения кассового аппарата. Ввод "−1" — сигнал остановки. Нужно сосчитать сумму всех введенных чисел (сумму чека).

Поскольку требуется повторить нечто (ввод очередной цены) неизвестное количество раз, потребуется цикл `while`. Нам понадобится как минимум две переменные: `price` для цены очередного товара и `total` — для общей суммы.

Если бы мы знали точно, что пользователю надо купить ровно три товара, цикл (и ввод −1 как условие его прерывания) был бы не нужен. Тогда программа могла бы выглядеть так:

```
total = 0
```

```
price = float(input())
total = total + price
price = float(input())
total = total + price
price = float(input())
total = total + price
print('Сумма введённых чисел равна', total)
```

Обратите внимание: мы назвали переменные осмысленно. Это очень облегчит жизнь программисту, который будет читать наш код позже, даже если это будете вы сами неделю спустя. Однако интерпретатор Python к этому факту совершенно равнодушен. Чтобы значения переменных соответствовали названиям и тому смыслу, который мы в них закладываем, нужно поддерживать переменные в актуальном состоянии. И только вы, программист, можете это сделать.

С переменной `price` все относительно понятно: ее значение обновляется при считывании с клавиатуры на каждой итерации цикла, как это делалось во многих других задачах. `total` сначала равно нулю: до начала ввода цен их сумма, конечно, ноль. Однако значение переменной `total` устаревает каждый раз, когда пользователь вводит цену очередного товара. Поэтому нам нужно прибавить к значению `total` только что введенную цену, чтобы эта переменная по-прежнему обозначала сумму цен всех купленных товаров.

Если бы мы хотели сократить запись, можно было бы организовать цикл, который выполнился бы ровно три раза. Для этого нам потребуется переменная-счетчик, которая внутри цикла будет считать каждую итерацию цикла. А условием выхода обозначим выполнение нужного количества итераций:

```
count = 0
total = 0
while count < 3:
    price = float(input())
    total = total + price
    count = count + 1
print('Сумма введённых чисел равна', total)
```

Обратите внимание: `total` и `count` должны обнуляться до цикла.

Однако у нас в задаче количество товаров неизвестно, поэтому понадобится цикл до ввода сигнала остановки (–1). С учетом сказанного выше программа будет выглядеть так:

```
total = 0
print('Вводите цены; для остановки введите -1.')
price = float(input())
while price > 0:
    total = total + price  # можно заменить на аналогичную запись
    # total += price
    price = float(input())
print('Общая стоимость равна', total)
```

4. Подсчет количества элементов, удовлетворяющих условию

А теперь рассмотрим еще одну задачу.

Пользователь вводит целые числа. Ввод чисел прекращается, если введено число 0. Необходимо определить, сколько чисел среди введенных оканчивались на 2 и были кратны числу 4. Теперь нам надо проверять последовательность чисел.

Для каждого введенного числа надо делать проверку, соответствует ли оно условию. Если оно подходит под условие, увеличиваем счетчик таких чисел.

И уже после цикла, когда остановился ввод чисел, выводим результат — посчитанное количество нужных чисел.

```
count = 0
number = int(input())
while number != 0:
    if number % 10 == 2 and number % 4 == 0:
        count += 1
    number = int(input())
print('Количество искомых чисел:', count)
```

Обратите внимание: до цикла необходимо задать начальное значение для переменной `count`. Ведь когда придет первое подходящее под условие число, у нас `count` будет увеличиваться на 1 относительно предыдущего значения. А значит, это значение должно быть задано.

Давайте посмотрим, как будет работать эта программа для последовательности чисел: 12, 3, 32, 14, 0.

Шаг	Действие	Пояснение	Цикл
1	`count = 0`	`count = 0`	
2	`number = int(input())`	`number = 12`	
3	`while number != 0:`	`12 != 0` (Истина)	Вход в цикл, 1-я итерация
4	`if number % 10 == 2 and number % 4 == 0:`	`12 % 10 == 2 and 12 % 4 == 0` (Истина)	Заходим в if
5	`count += 1`	`count = 1`	
6	`number = int(input())`	`number = 3`	
7	`while number != 0:`	`3 != 0` (Истина)	2-я итерация
8	`if number % 10 == 2 and number % 4 == 0:`	`3 % 10 == 2 and 3 % 4 == 0` (Ложь)	Пропускаем if
9	`number = int(input())`	`number = 32`	
10	`while number != 0:`	`32 != 0` (Истина)	3-я итерация
11	`if number % 10 == 2 and number % 4 == 0:`	`32 % 10 == 2 and 32 % 4 == 0` (Истина)	Заходим в if

Шаг	Действие	Пояснение	Цикл
12	`count += 1`	`count = 2`	
13	`number = int(input())`	`number = 14`	
14	`while number != 0:`	`14 != 0` (Истина)	4-я итерация
15	`if number % 10 == 2 and number % 4 == 0:`	`14 % 10 == 2 and 14 % 4 == 0` (Ложь)	Пропускаем if
16	`number = int(input())`	`number = 0`	
17	`while number != 0:`	`0 != 0` (Ложь)	Выход из цикла
18	`print('Количество искомых чисел:', count)`	Вывод вычисленного `count`	

5. Поиск максимума и минимума

Очень часто в задачах приходится использовать различные статистические алгоритмы: поиск максимума, минимума, среднего значения, медианы и моды чисел, главный из которых — определение максимального и минимального значений на множестве данных.

Рассмотрим алгоритм в общем виде.

1. Заведем отдельную переменную для хранения максимума и минимума. В качестве начального значения можно задать:
   - Заведомо малое для анализируемых данных значения, для максимума это будет совсем маленькое число: например, если мы вычисляем максимальный балл за экзамен, можно взять `maximum = 0`, тогда гарантированно произойдет замена максимума. Минимуму же, наоборот, присваивается заведомо большое значение
   - Первый элемент данных
2. В теле цикла каждый подходящий элемент данных обрабатывается операторами по принципу:
   - Если текущий элемент больше максимума, меняем максимум
   - Если текущий элемент меньше минимума, заменяем минимум

Рассмотрим пример. Витя анализировал список литературы и решил, что хочет начать с самой большой по объему книги. Напишем программу, которая поможет мальчику определить, сколько страниц ему предстоит прочитать. Витя последовательно вводит количество страниц каждой книги из списка, а окончанием ввода служит ввод любого отрицательного числа (или 0).

```
biggest_book = 0
n = int(input())
while n > 0:
    if n > biggest_book:
        biggest_book = n
    n = int(input())
print(biggest_book)
```

Так как книга не может содержать в себе 0 страниц, для значения максимума мы можем взять 0.

После этого Витя начинает вводить количество страниц: например, он вводит 148. `148 > 0` — условие цикла выполняется, и мы переходим к операции сравнения. На данном шаге `148 > 0`, значит, `biggest_book = 148`. Снова считываем число.

Предположим, теперь введено 120. `120 > 0` — продолжаем работать в цикле. `120 > 148` — условие не выполняется, переходим к вводу новых данных, `biggest_book` все еще равен 148.

В этот раз мальчик ввел 486, мы заходим в цикл `486 > 148`, производим замену `biggest_book = 486`. Продолжаем ввод. И так далее до тех пор пока не будет введено отрицательное число или 0.

При решении задачи мы можем использовать особенность языка Python 3.8 — моржовый оператор или, как его еще называют, «оператор-морж» (из за ассоциации обозначения оператора `:=` с животным). Моржовый оператор позволяет присвоить значение переменной в условии `if` или `while`. Используя его, можно записать нашу программу следующим образом:

```
biggest_book = 0
while (n := int(input())) > 0:
    if n > biggest_book:
        biggest_book = n
print(biggest_book)
```

В этой программе значение переменной `n` задается один раз, в условии цикла, в отличие от ее предыдущей версии.