Тема 4. Операторы цикла

При написании кода программы может возникнуть необходимость выполнения одних и тех же действий несколько раз. Для этих целей используются циклы.

Цикл – управляющая конструкция в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций. Также циклом может называться любая многократно исполняемая последовательность инструкций, организованная любым способом (например, с помощью условного перехода по меткам).

Циклы делятся изначально на 2 типа:

1. ***Определенный цикл*** – это цикл, в котором заранее известно количество повторений действий.
2. ***Неопределенный цикл*** – это цикл, в котором заранее не известно количество повторений действий, но известно условие окончания цикла.

4.1. Цикл while

Цикл **while** относится к неопределенным циклам, так как его можно использовать в тех случаях, когда заранее неизвестно количество повторений блока действий.

Цикл **while** в Python выполняет блок кода программы, пока условие, указанное в цикле, имеет значение **True**.

Синтаксис цикла while

while *<условие>*:

<блок действий>

else:

<блок, если не использовался оператор break>

Условия в цикле строятся по тому же принципу, что рассматривался в алгоритме ветвления.

Последовательность выполнения цикла while

1. Переменным присваиваются начальные значения.
2. Проверяется условие, и, если оно истинно, выполняется блок кода внутри цикла, иначе выполнение цикла завершается.
3. Переменные изменяются по требованию задачи.
4. Переход к пункту 2.

5. Если внутри цикла не использовался оператор **break**, то после завершения выполнения цикла будет выполнен блок кода после инструкции **else**. Этот блок не является обязательным.

Пример:

Вывести в столбик числа от 1 до 5.

```
count = 1
while count <= 5:
    print(count)
    count += 1
```

Результат:

```
1
2
3
4
5
```

4.2. Функция range

Перед тем, как разбирать оператор цикла **for**, рассмотрим функцию **range**. Она нам понадобится для организации цикла.

range() является универсальной функцией Python для создания списков **(list)**, содержащих арифметическую прогрессию. Чаще всего она используется в цикле **for**. Функция **range()** может принимать от одного до трех аргументов, при этом аргументами должны быть целые числа **(int)**.

range([*старт*,] *стоп*[, *шаг*]) – так выглядит стандартный вызов функции **range**().

По умолчанию *старт* = 0, *шаг* = 1.

Возвращает список целых чисел в форме [*старт*, *старт* + *шаг*, *старт* + *шаг**2...].

Если ***шаг* положительное число**, последним элементом списка будет наибольшее *старт* + **i** * *шаг*, меньшее числа стоп. Здесь **i** – количество шагов.

Если ***шаг* отрицательное число**, то последний элемент будет наименьшее *старт* + **i** * *шаг*, большее числа *стоп*.

Стоп в список не включается. *Шаг* не должен равняться нулю, иначе возникнет **ValueError** (ошибка). Причем, если *шаг* положительное число, то *старт* должен быть меньше *стоп*, а если *шаг* отрицательное число, то старт должен быть больше *стоп*.

Примеры использования функции range():

Код	Диапазон чисел
`range(5)  # указываем только верхнюю границу`	`[0, 1, 2, 3, 4]`
`range(2, 5)  # указываем и нижнюю и верхнюю границу`	`[2, 3, 4]`

`range(0, 10, 2) # указываем и нижнюю и верхнюю` `# границу и шаг. Если шаг не указан, то он равен 1.`	`[0, 2, 4, 6, 8]`
`range(0, -5, -1)`	`[0,-1,-2,-3,-4]`

4.3. Цикл for

Рассмотрим определенный цикл, который также называют циклом со счетчиком. Реализуется данный цикл с помощью оператора цикла **for**.

Цикл **for** в Python перебирает объекты в переданных ему последовательностях, таких как списки, строки, диапазоны.

Синтаксис цикла for

for *<переменная>* ***in*** *<последовательность>:*
    <блок действий>
else*:*
    <блок, если не использовался оператор break>

Последовательность выполнения цикла for

1. Переменной-счетчику присваивается начальное значение из диапазона.
2. Выполняется блок кода внутри цикла.
3. Если не все значения из диапазона были перебраны, то переменная-счетчик принимает следующее значение из диапазона, переход к пункту 2, иначе выход из цикла.
4. Если внутри цикла не использовался оператор **break**, то после завершения выполнения цикла будет выполнен блок кода в инструкции **else**. Этот блок не является обязательным.

Обратите внимание, что здесь также, как и в других операторах, блок действий должен быть написан с одинаковым отступом.

На данном этапе изучения мы будем использовать только одну последовательность, которая называется диапазон (**range**). Остальные рассмотрим, когда будем изучать строки, списки и т. д. В данном случае мы будем использовать оператор цикла **for** для создания классического цикла со счетчиком, подобно тому, как оператор **for** используется, например, в языке Pascal.

Пример 1:

Решим ту же самую задачу, которую решали при помощи цикла **while**.

Вывести в столбик числа от 1 до 5.

```
for index in range(1, 6):
    print(index)
```

Результат:

```
1
2
3
4
5
```

В этом примере мы не увеличиваем на каждом шаге цикла **index** на 1, так как по умолчанию в **range** шаг равен 1. Также не забываем про верхнюю границу.

Пример 2:

Вывести на экран четные числа от 0 до 10.

```
for index in range(0, 11, 2):
    print(index)
```

Результат:

```
0
2
4
6
8
10
```

Пример 3:

Вывести на экран числа от 10 до 1. Здесь нам нужно использовать отрицательный шаг.

```
for index in range(10, 0, -1):
    print(index)
```

Результат:

```
10
9
8
7
6
5
4
3
2
1
```

4.4. Вложенные циклы

В программах часто возникают случаи, когда в теле цикла нужно выполнить другой цикл. Такая структура называется «вложенные циклы». Количество вложенных циклов и уровень вложенности могут быть различными.

Для примера решим очень простую задачу.

Нужно вывести на экран таблицу чисел следующего вида:

1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9

Понятно, что такую задачу можно решить и без цикла. Но стоит подумать, а что, если у нас потребуют тысячу таких строк? Тут уже решение задачи без использования цикла является явно неоптимальным.

```
for i in range(1, 6):
    for j in range(1, 10):
        print(j, end=' ')
    print()
```

Обратите внимание, что первая команда **print()** будет работать во вложенном цикле, и именно она выводит числа, а вторая команда **print()** нужна только для перехода на новую строку.

Порядок изменения переменных i и j во время работы этой программы:

i=1
j=1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9
i=2
j=1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9
i=3
j=1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9
i=4
j=1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9
i=5
j=1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9

Здесь через дефис перечисляются значения переменной j на каждой итерации вложенного цикла. Обратите внимание на то, что с каждой итерацией цикла со счетчиком **i**, вложенный цикл начинает свою работу заново.

Вложенные конструкции цикла и алгоритма ветвления

При изучении различных алгоритмов в программировании нужно понимать то, что, хоть алгоритмы и делят на различные виды, они редко используются в сложных

задачах отдельно друг от друга. При написании программы может возникнуть ситуация, что алгоритм ветвления будет вложенной инструкцией для цикла и наоборот, цикл может оказаться вложенной инструкцией для алгоритма ветвления.

Примеры:

Алгоритм ветвления вложен в цикл

Вывести на экран все двухзначные числа, у которых сумма цифр равна 10.

```
for i in range(10, 100):
    if (i // 10 + i % 10) == 10:
        print(i)
```

В результате получим:

```
19
28
37
46
55
64
73
82
91
```

Цикл вложен в алгоритм ветвления

Вводится целое число. Если оно положительное, то найти факториал этого числа, а если не положительное, то вывести соответствующее сообщение.

```
a = int(input('Введите целое число: '))
if a > 0:
    f = 1
    for i in range(1, a + 1):
        f *= i
    print(f)
else:
    print('Число не положительное')
```

4.5. Оператор break

Оператор **break** служит для прерывания цикла, т. е. для остановки выполнения команд, даже если условие выполнения цикла не приняло значение **False**, в случае с циклом **while**, или последовательность элементов не закончилась, в случае с циклом **for**.

Пример 1:

С клавиатуры вводятся целые числа, посчитать их сумму. Условием окончания ввода служит введенное число 0.

```
sum = 0
while True:
    a = input('Введите целое число: ')
    a = int(a)
    if a == 0:
        break
    sum += a
print('Сумма чисел = ', sum)
```

Пример 2:

С клавиатуры вводится двузначное число. Посчитать сумму его цифр.

Эту задачу решим с проверкой ввода.

```
while True:
    a = input('Введите двухзначное число: ')
    a = int(a)
    if (a > 9) and (a < 100):
        break
    if (a < 10) or (a > 99):
        print('Это не двухзначное число, введите нужное число')
sum = (a % 10) + (a // 10)
print('Сумма цифр = ', sum)
```

4.6. Оператор continue

Оператор **continue** используется для того, чтобы пропустить все оставшиеся команды в текущем блоке и продолжить цикл с начала следующей итерации.

Для наглядности разберем пример.

Пример:

Вывести все числа из диапазона [1, 10], кроме 4 и 7.

```
for i in range(1, 11):
    if (i == 4) or (i == 7):
        continue
    print(i, end=' ')
```

4.7. Оператор else в циклах

Слово **else**, примененное в цикле **for** или **while**, проверяет, был ли произведен выход из цикла инструкцией **break**, или же «естественным» образом. Блок инструкций внутри **else** выполнится только в том случае, если выход из цикла произошел без помощи **break**.

Пример:

Вводится слово на русском языке. Нужно проверить, присутствует ли в нем русская буква «а».

Как уже было сказано выше, цикл **for** может работать с различными, не только числовыми последовательностями. Воспользуемся этим при решении задачи.

```
s = input('Введите слово на русском языке: ')
for i in s:
  if i == 'а':
    print('Буква "а" входит в слово')
    break
else:
  print('Буква "а" в слово не входит')
```