STEP IT ACADEMY

Занятие №5

Управляющие выражения. Блоки, условия, циклы

Что такое цикл while в Python?

Цикл while используется в Python для неоднократного исполнения определенной инструкции до тех пор, пока заданное условие остается истинным.
Этот цикл позволяет программе перебирать блок кода.

```
while test_expression:
    body of while
```

Рассмотрим пример, чтобы лучше понять.

```
a = 1

while a < 10:
    print('Цикл выполнился', a, 'раз(а)')
    a = a+1
print('Цикл окончен')
```

Результат программы

```
Цикл выполнился 1 раз
Цикл выполнился 2 раз
Цикл выполнился 3 раз
Цикл выполнился 4 раз
Цикл выполнился 5 раз
Цикл выполнился 6 раз
Цикл выполнился 7 раз
Цикл выполнился 8 раз
Цикл выполнился 9 раз
Цикл окончен
```

Бесконечный цикл while в Python

Бесконечный цикл while — это **цикл**, в котором условие никогда не становится ложным. Это значит, что тело исполняется снова и снова, а цикл никогда не заканчивается.

```
a = 1

while a==1:
    b = input('Как тебя зовут?')
    print('Привет', b, ', Добро пожаловать')
```

Если запустить этот код, то программа войдет в бесконечный цикл и будет снова и снова спрашивать имена. Цикл не остановится до тех пор, пока не нажать Ctrl + C.

Else в цикле while

В этом примере блок в else исполняется, когда условие цикла становится ложным.

```
a = 1

while a < 5:
   print('условие верно')
   a = a + 1
else:
   print('условие неверно')
```

Результат программы:

условие верно
условие верно
условие верно
условие верно
условие неверно

Программа исполняет код цикла while до тех, пока условие истинно, то есть пока значение a меньше 5.

Поскольку начальное значение a равно 1, а с каждым циклом оно увеличивается на 1, условие станет ложным, когда программа доберется до четвертой итерации — в этот момент значение a изменится с 4 до 5.

Программа проверит условие еще раз, убедится, что оно ложно и исполнит блок else, отобразив «условие неверно».

Прерывания цикла while в Python

Break — ключевое слово break прерывает цикл и передает управление в конец цикла:

```
a = 1
while a < 5:
    a += 1
    if a == 3:
        break
    print(a) # 2
```

Прерывания цикла while в Python

Continue — ключевое слово continue прерывает текущую итерацию и передает управление в начало цикла, после чего условие снова проверяется. Если оно истинно, исполняется следующая итерация.

```
a = 1

while a < 5:
    a += 1
    if a == 3:
        continue
    print(a)  # 2, 4, 5
```

Цикл For

Цикл for уже чуточку сложнее, чуть менее универсальный, но выполняется гораздо быстрее цикла while.

Этот цикл проходится по любому итерируемому объекту (например строке или списку), и во время каждого прохода выполняет тело цикла.

Цикл for, также называемый циклом с параметром.

```
>>> for i in 'hello world':
...     print(i * 2, end='')
...
hheelllloo  wwoorrlldd
```

Синтаксис

```
for <переменная> in <последовательность>:
    <действие>
else:
    <действие>
```

Запустите программу.

```
i = 1
for color in 'red', 'orange', 'yellow', 'green', 'cyan', 'blue', 'violet':
    print('#', i, ' color of rainbow is ', color, sep = '')
    i += 1
```

Что вы получили на выходе?

В этом примере переменная color последовательно принимает значения 'red', 'orange'. В теле цикла выводится сообщение, которое содержит название цвета, то есть значение переменной color, а также номер итерации цикла число, которое сначала равно 1, а потом увеличивается на один (инструкцией i += 1 с каждым проходом цикла.

Инструкция i += 1 эквивалентна конструкции i = i + 1 (это просто сокращенная запись). Такую сокращенную запись можно использовать для всех арифметических операций: *=, -=, /=, %=...

Протестируйте код программы

```
>>> languages = ["C", "C++", "Perl", "Python"]
>>> for x in languages:
...     print(x)
...
C
C++
Perl
Python
>>>
```

Функция range

Для работы функции range указываются 2 или 3 числа:

•Первое число — **start** — с него функция начинает отсчет.

•Второе число называется **stop** и обозначает конец выбранного промежутка чисел.
В примере это цифра 10, поэтому функция не может показать число больше 10.

•Третье число называется **step**: это шаг, который делает функция при переборе чисел.
Можно не указывать шаг, и тогда функция покажет все числа от 5 до 10.

Для повторения цикла некоторое заданное число раз n можно использовать цикл for вместе с функцией range:

```
for i in range(4):  # равносильно инструкции for i in 0, 1, 2, 3:

    # здесь можно выполнять циклические действия
    print(i)
    print(i ** 2)

# цикл закончился, поскольку закончился блок с отступом
print('Конец цикла')
```

```
sum = 0
n = 5
for i in range(1, n + 1):
    sum += i
print(sum)
```

В этом примере переменная i принимает значения 1, 2, ..., n, и значение переменной sum последовательно увеличивается на указанные значения.

Особенности работы с функцией range()

Мы можем получить доступ ко всем элементам, но индекс элемента остается недоступным. Есть способ получить доступ как к индексу элемента, так и к самому элементу. Для этого используйте функцию range() в сочетании с функцией длины len():

```
fibonacci = [0,1,1,2,3,5,8,13,21]

for i in range(len(fibonacci)):
    print(i,fibonacci[i])
```

Сверьте результат программы

0 0

1 1

2 1

3 2

4 3

5 5

6 8

7 13

8 21

Оператор прерывания в python — break

Если в программе цикл for должен быть прерван оператором break, цикл будет завершен, и поток программы будет продолжен без выполнения действий из else.

Протестируйте код программы

```
edibles = [«отбивные", "пельмени", "яйца", "орехи"]

for food in edibles:
    if food == "пельмени":
        print("Я не ем пельмени!")
        break
    print("Отлично, вкусные " + food)
else:
    print("Хорошо, что не было пельменей!")
print("Ужин окончен.")
```

Оператор пропуска python — continue

Предположим, нам «пельмени» нам нужно просто пропустить и продолжить прием пищи. Тогда нужно использовать оператор continue, для перехода к следующему элементу.

В следующем маленьком скрипте python мы используем continue, чтобы продолжить, итерацию по списку, когда мы сталкиваемся с пельменями.

Протестируйте код

```
edibles = ["отбивные", "пельмени", "яйца", "орехи"]

for food in edibles:
    if food == "пельмени":
        print("Я не ем пельмени!")
        continue
    print("Отлично, вкусные " + food)
else:
    print("Ненавижу пельмени!")
print("Ужин окончен.")
```

Какая разница между двумя результатами программы?

Первый результат

Отлично, вкусные отбивные

Я не ем пельмени!

Ужин окончен.

Второй результат

Отлично, вкусные отбивные

Я не ем пельмени!

Отлично, вкусные яйца

Отлично, вкусные орехи

Ненавижу пельмени!

Ужин окончен.

Циклы в PYTHON

Enumerate в python 3

Enumerate — встроенная функция Python.
Позволяет нам автоматически считать итерации цикла.

Функция enumerate принимает необязательный аргумент (значение начала отсчета, по умолчанию 0).

Запустите программный код

```
for counter, value in enumerate(some_list):
    print(counter, value)
```

Протестируйте код

```
my_list = ['яблоко', 'банан', 'вишня', 'персик']

for c, value in enumerate(my_list, 1):
    print(c, value)

# Результат:
# 1 яблоко
# 2 банан
# 3 вишня
# 4 персик
```

Вложенные циклы

Вложенный цикл - цикл который выполняется внутри другого цикла.

Результат работы

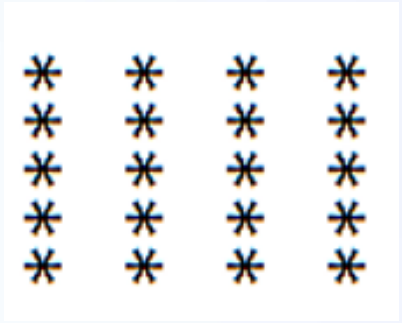

Схема работы

- При каждой итерации внешнего цикла внутренний цикл будет выполнен полностью.
- Внутренний цикл должен завершить все свои итерации, прежде чем внешний цикл сможет перейти к следующей итерации.

Иллюстрируемый пример работы вложенного цикла

Пример

```
for i in range(1, 10):
    for j in range(1, 10):
        print(i * j, end="\t")
    print()
```

Внешний цикл

Вложенный цикл

Тело вложенного цикла

Тело внешнего цикла

Вывод

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

Программный код к результату на 5 слайде

```
n = int(input())
m = int(input())
for i in range(n):
    for j in range(m):   # вложенный цикл
        print('*', end='')
    print()
```

Проверьте результат программы

Запустите программу и составьте условие для этой задачи

```
k = int(input())
for i in range(1, 10):
    print(i, '*', k, '=', k * i, sep='', end='\t')
```

Условие: Вывести в строку таблицу умножения числа n.

Ввод:

5

Вывод:

1*5=5 2*5=10 3*5=15 4*5=20 5*5=25 6*5=30 7*5=35 8*5=40 9*5=45

Проверь себя:

Как написать цикл **for** в Python?

Как использовать **else**, связанное с циклом **for**?

Что такое итераторы и итерируемые объекты?

Как создать итератор и итерируемый объект?

Как работает цикл **for**?

Как используя цикл **while** имитировать цикл **for**?

Как читать и понимать разобранные инструкции?

Какой цикл называется вложенным?


STEP
IT ACADEMY