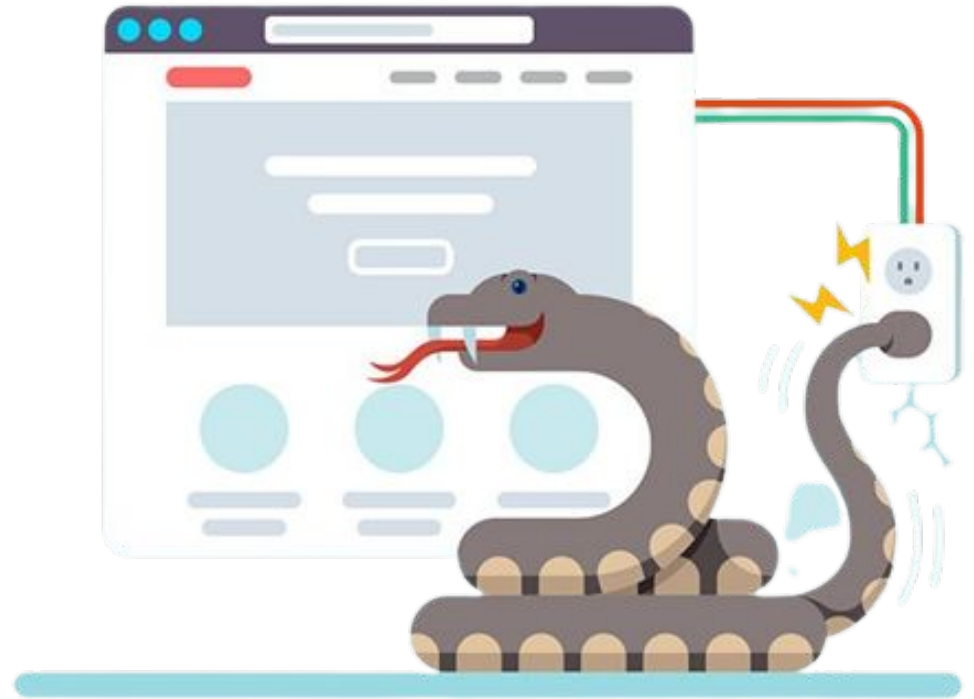


*Моя Школа*

сертифицированные IT курсы

# Цикл `while`





```
1 s = input()
2
3 print(s[2]) # Сначала выведите третий символ этой строки.
4 print(s[-2]) # Во второй строке выведите предпоследний символ этой строки.
5 print(s[:4]) # В третьей строке выведите первые пять символов этой строки.
6 print(s[:-2]) # В четвертой строке выведите всю строку, кроме последних двух символов.
7 print(s[::2]) # В пятой строке выведите все символы с четными индексами
8               # (считая, что индексация начинается с 0, поэтому символы выводятся начиная с первого)
9 print(s[1::2]) # В шестой строке выведите все символы с нечетными индексами, то есть начиная со
10               # второго символа строки.
11 print(s[::-1]) # В седьмой строке выведите все символы в обратном порядке.
12 print(s[::-2]) # В восьмой строке выведите все символы строки через один в обратном порядке, начиная
13               # с последнего.
14 print(len(s)) # В девятой строке выведите длину данной строки.
```



# План занятия

1

Отладчик PyCharm

2

Цикл while

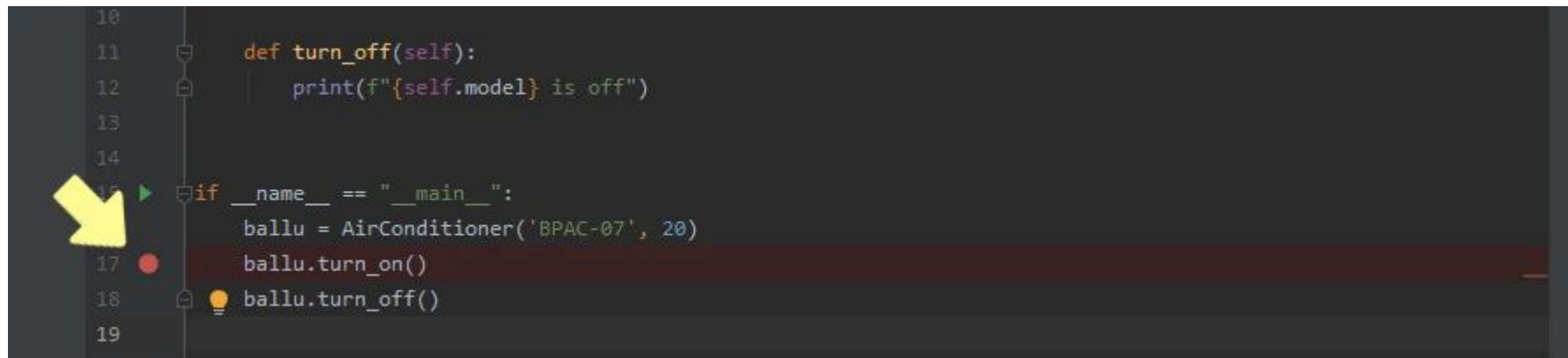
3

Конструкция  
for-else, while-else

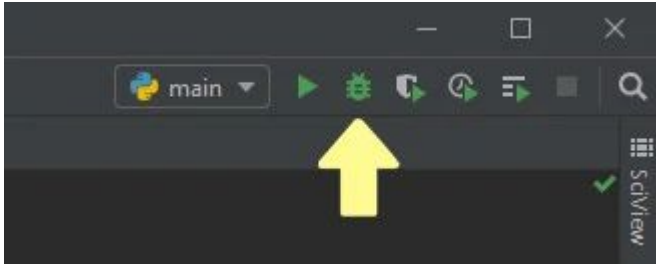
**Отладка** — чрезвычайно полезный инструмент. С помощью него можно пошагово исполнять программный код. Интерпретатор переходит в особый режим, фиксируя на каждом этапе выполнения текущее состояние программы. Т.е. для каждой строки кода вам будет представлен мини-отчёт, в котором можно посмотреть актуальные значения всех переменных и проследить таким образом весь процесс их изменения.

Находясь в руках опытного разработчика, отладчик сокращает время поиска ошибок на порядки.

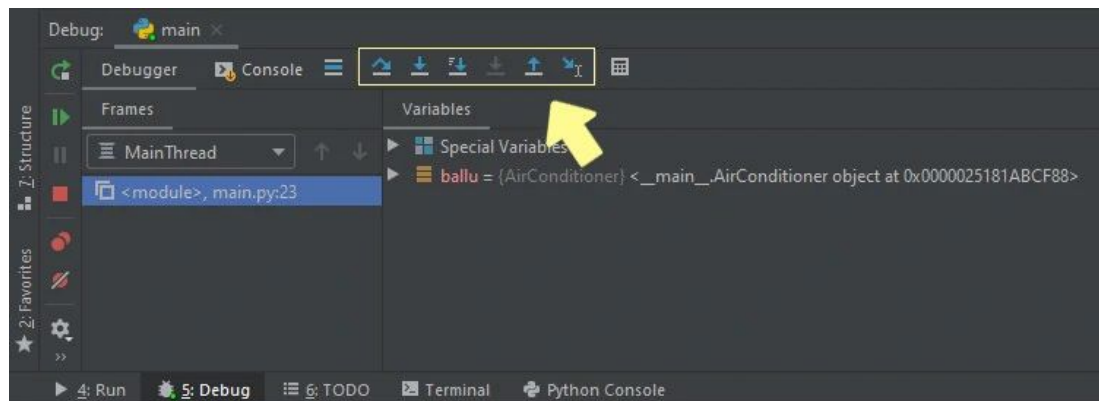
Для старта отладки, нужно поставить так называемую точку останова. Это красный кружок напротив строки кода.



Вы можете установить её где захотите, но чаще всего точку следует ставить там, где интерпретатор увидел ошибку. Для запуска кода в режиме отладки нажмите на значок в "жука" в левом верхнем углу:



Теперь, нажимая F8, можно последовательно шагать по строчкам кода и смотреть текущее состояние всей программы. "Шагать" можно и через F7, но в таком случае дебаггер будет совершать "Шаг с заходом". Т.е. при встрече вызова функции он будет заходить в её описание и последовательно проходить по инструкциям. Все варианты навигации в режиме отладки можно посмотреть в окне "Debugger":



Цикл **while** позволяет выполнить одну и ту же последовательность команд, пока проверяемое условие истинно.

Цикл **while** состоит из двух частей:

- Условие (Часто используют условие `while True`)
- Тело цикла (команды, которые будут выполняться внутри цикла)

Условие записывается до тела цикла и проверяется до выполнения тела цикла. Синтаксис цикла **while** в простейшем случае выглядит так:

```
while условие:  
    команда1  
    команда2  
    .....  
    командаN
```

При выполнении цикла `while` сначала проверяется условие. Если условие истинно, то выполняются команды внутри цикла. После чего условие проверяется снова и все повторяется. Так продолжается до тех пор, пока условие будет истинно. Как только условие станет ложно, работа цикла завершится и управление передастся следующей команде после цикла.

Давайте взглянем на простой пример:

```
i = 0
while i < 10:
    print(i)
    i = i + 1
```

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

```
Process finished with exit code 0
```



Бесконечные циклы известны как логические ошибки, и их нужно избегать. Существует другой способ вырваться из цикла, для этого нужно использовать встроенную функцию `break`.

Давайте посмотрим, как это работает:

```
i = 0
while i < 10:
    print(i)

    if i == 5:
        break

    i += 1
```

```
0
1
2
3
4
5

Process finished with exit code 0
```

## Рассмотрим еще один пример:

Необходимо вычислить сумму чисел от 1 до 50 и результат вывести на экран.

```
i = 1
result = 0
while i <= 50:
    result += i
    i += 1
print(result)
```

1275

*Process finished with exit code 0*

Давайте разберем эту программу по командам:

- 1) `i = 1` и `result = 0`, создаем две переменные и присваиваем им начальные значения.
- 2) `while i <= 50:` - прописываем цикл, который будет выполняться пока переменная `i` будет меньше или равна 50, как только `i` станет 51 программа начнет выполнять команду `print(result)`, которая выведет результат на экран
- 3) `result += i` - каждую итерацию цикла мы увеличиваем значение переменной `result` на `i`
- 4) `i += 1` - каждую итерацию цикла мы увеличиваем значение переменной `i` на `+1`

## Задание №1

Квадраты всех целых чисел от 1 до 10.

```
i = 1
while i <= 10:
    print(i ** 2)
    i += 1
```

```
1
4
9
16
25
36
49
64
81
100
```

### Задание №2

Перемножить все чётные значения в диапазоне от 0 до 125;  
результат вывести на экран.

```
i = 1
result = 0
while i <= 9435:
    if i % 2 == 0:
        result = i*i
    i += 1
print(result)
```

89000356

*Process finished with exit code 0*

### Задание №3

Вывести числа от 1 до 15 в порядке убывания



```
i = 15
while i != 0:
    print(i)
    i -= 1
```

```
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
```

*Process finished with exit code 0*

### Задание №4

Пользователь вводит два числа с клавиатуры, необходимо вывести на экран все отрицательные числа, лежащие между ними. Например пользователь ввел -5 и 3, на экране вывелось -4, -3, -2, -1

```
a = int(input('Введите первое число: '))
b = int(input('Введите второе число: '))

while a < b:
    a += 1
    if a == 0:
        break
print(a)
```

Введите первое число: -5

Введите второе число: 3

-4

-3

-2

-1

*Process finished with exit code 0*

Циклы for и while могут иметь блок else и многие не знакомы с этим фактом. Блок else выполняется, когда цикл завершается в нормальном режиме. Т.е. не был вызван break.

Блок else после циклов относится не к самому циклу, а к оператору break!

## Пример условия else в цикле for:

```
for i in range(3):  
    print(i)  
else:  
    print('Готово')
```

```
0  
1  
2  
Готово
```

*Process finished with exit code 0*

## Пример условия else в цикле while:

```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print('Готово')
```

```
0
1
2
Готово

Process finished with exit code 0
```

Условие else не выполняется, если цикл завершается принудительно (например, с помощью оператора break или путем вызова исключения):

```
for i in range(3):  
    print(i)  
    if i == 1:  
        break  
else:  
    print('Готово')
```

```
0  
1  
  
Process finished with exit code 0
```

### Задание №5

Необходимо, чтоб программа выводила на экран вот такую последовательность(не использовать готовый массив):

7 14 21 28 35 42 49 56 63 70 77 84 91 98

Добавить в массив и найти его длину.



```
a = 0
mas = []
while a < 98:
    a += 7
    mas.append(a)

print(mas, 'Длина: ', len(mas))
```

```
[7, 14, 21, 28, 35, 42, 49, 56, 63, 70, 77, 84, 91, 98] Длина: 14
```

```
Process finished with exit code 0
```

## Задание №6

Простейший калькулятор с введёнными двумя числами вещественного типа.

Ввод с клавиатуры: операции  $+$   $-$   $*$   $/$  и два числа.

Обработать ошибку: “Деление на ноль”

Ноль использовать в качестве завершения программы (сделать как отдельную операцию).

```
# Практическое 6

print("Ноль в качестве знака операции завершит работу программы")
x = float(input("x="))
y = float(input("y="))
while True:
    s = input("Знак (+,-,*,/): ")
    if s == '0':
        break
    elif s == '+':
        print(x+y)
    elif s == '-':
        print(x - y)
    elif s == '*':
        print(x * y)
    elif s == '/':
        if y != 0:
            print(x / y)
        else:
            print("Деление на ноль!")
```

## Задание №7

Массив из 7 цифр. Если четных цифр в нем больше чем нечетных, то найти сумму всех его цифр, если нечетных больше, то найти произведение 1 3 и 6 элемента.

Казино. Компьютер генерирует числа от 1 до 10 и от 1 до 2-х соответственно. Цифры от одного до 10 отвечают за номера, а от 1 до 2 за цвета (1 красный, 2 черный).

Пользователю дается 5 попыток угадать номер и цвет (Прим. введения с клавиатуры: 3 красный). В случае неудачи вывести на экран правильную комбинацию.