



# Программирование В ЛИНГВИСТИКЕ

Структурное программирование. Циклы

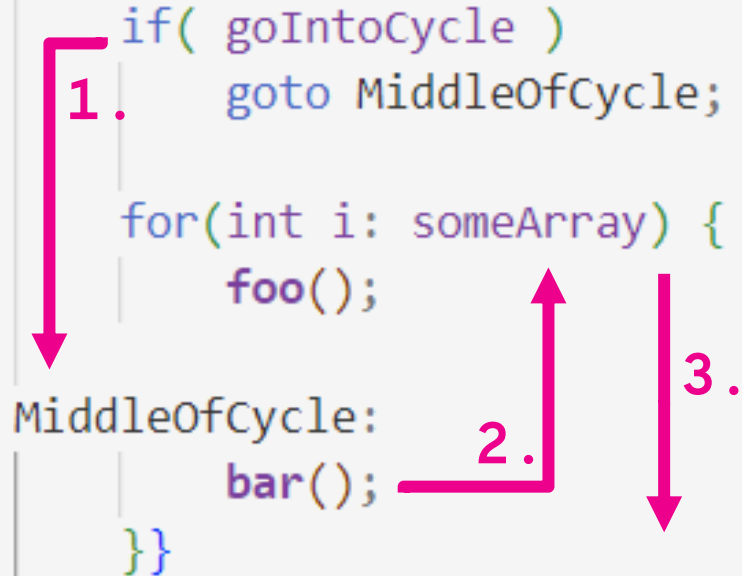
Верещагина Анна Дмитриевна (Аня)

# goto VS. структурирование

```
1. if( goIntoCycle )
    goto MiddleOfCycle;


    for(int i: someArray) {
        foo();
    }

MiddleOfCycle:
    bar();
}
```



C++ go to

```
bar()
for i in someArray:
    foo()
    bar()
```



Python

# Структурирование: понятия

1.  
2.



Последователь-  
ность

Команды выполняются в том порядке, в котором они представлены в коде

`if`



Ветвление

Условные конструкции, создающие ветки



Цикл

Команды выполняются повторно либо заданное кол-во раз, либо до тех пор, пока условие равно True

# Цикл for

Тело цикла:

```
for <переменная цикла> in <итерируемый объект>:  
    команда  
    команда  
else:  
    команда
```

Циклу **for** нужен итерируемый объект – объект, который может возвращать элементы по одному.

+ Списки, строки, кортежи, множества, словари, итераторы, генераторы

Используется, когда:

- Мы знаем, сколько раз нам повториться
- Мы перебираем элементы итерируемого объекта

Итерируемся по строке:

```
for letter in "abcd":  
    print(letter)
```

a  
b  
c  
d

# Цикл `for`: пример

```
for letter in "abc":  
    print(letter)  
else:  
    print("end")
```

```
a  
b  
c  
end
```

Для каждого символа из строки 'abc'  
выполнить:

напечатать символ

По завершении:

напечатать 'end'

- > `for` – элемент синтаксиса Python, предполагающий начало цикла;
- > `letter` – переменная цикла, меняющая свое значение на следующее в итерируемом объекте при каждой итерации (проходе по телу цикла);
- > `in` – элемент синтаксиса Python, спрашивающий «через что итерироваться?»;
- > `'abcd'` – строка, пример итерируемого объекта, каждый из элементов которой последовательно передастся в переменную цикла в процессе итерации;
- > `print(letter)` – тело цикла. В данном случае функция `print` печатает по текущей букве в каждой итерации;
- > `else (опционально)` – элемент синтаксиса Python, спрашивающий «Что сделать по завершении?».

# range()

```
for i in range(3, 0, -1):  
    print(i)
```

3  
2  
1

Range

Команда `range()` создает итератор, который возвращает целые числа в заданном диапазоне с заданным шагом:

`range(начало, конец, шаг)`

Диапазон и шаг работают как срезы в строке:

тут 3 = начало, 0 = конец, -1 = шаг.

```
print("0123"[3:0:-1])
```

321

Срезы строки

Полезно при итерировании по индексам строк, списков и кортежей.

# Цикл while

## Тело цикла:

```
while <условие выполнения>:  
    команда  
    команда  
else:  
    команда
```

Циклу `while` нужно условие, которое будет заставлять его повторять команды. Если условие изначально `False`, срабатывает `else`. Если условие становится `False` в процессе работы цикла, то цикл останавливается по завершении итерации и приступает к телу `else`.

## Используется, когда:

- Мы не знаем, сколько раз нам повториться
- Есть условие для выполнения цикла

## Пример такого цикла:

```
number = input()  
while not number.isdigit():  
    number = input('Введите число!\n')  
else:  
    print('Наконец-то число!')
```

# Цикл while: пример

```
number = input()
while not number.isdigit():
    number = input('Введите число!\n')
else:
    print('Наконец-то число!')
```

```
a
Введите число!
1
Наконец-то число!
```

запросить ввод

Пока ввод не является цифрой:

запросить ввод

Когда ввод оказался цифрой:

напечатать 'Наконец-то число!'

> `number = input()` – запрос ввода. Используется до цикла для объявления `number` заранее, чтобы мы могли в условии цикла обратиться к этой переменной;

> `while` – элемент синтаксиса Python, предполагающий начало цикла («пока...»);

> `not number.isdigit()` – условие выполнения цикла. Проверяется в начале каждого прохода. Когда не выполняется, цикл прерывается;

> `number = input(...)` – тело цикла. Запрашивает ввод до тех пор, пока условие не будет нарушено;

> `Else (опционально)` – элемент синтаксиса Python, спрашивающий «Что сделать в случае невыполнения условия `while`?»;

> `print(...)` – тело `else`. Выполняется в случае невыполнения условия `while`.



# Вложенный цикл for

```
string = "ab"  
for letter in string:  
    for i in range(len(string)):  
        print(f'{letter}{string[i]}')
```

aa  
ab  
ba  
bb

Первый цикл итерируется напрямую по символам в строке 'ab'.

Вложенный цикл итерируется по символам в строке 'ab', но уже по их индексам (для разнообразия).

Переменные обоих циклов сочетаются в print, и на каждой итерации вложенного цикла печатаются различные комбинации символов.

# Цикл for, вложенный в while

```
string = input("Введите строку: ")
while string:
    for letter in string:
        print(letter)
    string = input("Введите строку: ")
else:
    print("Цикл закончен")
```

```
Введите строку: эй
эй
й
Введите строку:
Цикл закончен
```

Цикл **while** запрашивает строку до тех пор, пока пользователь не нажимает на Enter (это будет пустая строка = **False**).

Цикл **for** печатает эту строку побуквенно.

Когда пользователь ввел пустую строку, программа печатает «Цикл закончен».

# Отступы в Python

```
string = input("Введите строку: ")
while string:
    if len(string) <= 2:
        for letter in string:
            print(letter)
        else:
            print(f"Мы напечатали букв: {len(string)}")
    else:
        print("Слишком много букв")
        string = input("Введите строку: ")
else:
    print("Цикл закончен")
```

Ваша IDE (кроме Google Colab) проведет палочки, помогая Вам определить, на каком уровне Вы находитесь.

# Отступы в Python

```
string = input("Введите строку: ")
while string:
    if len(string) <= 2:
        for letter in string:
            print(letter)
        else:
            print(f"Мы напечатали букв: {len(string)}")
    else:
        print("Слишком много букв")
    string = input("Введите строку: ")
else:
    print("Цикл закончен")
```

Красное – цикл  
while

Синее – проверка

Зеленое – цикл for

# While True

Исполнение цикла `while` требует `bool`, возвращающего `True`. Таким образом, если мы зададим неизменяемое условие наподобие `while True`, то из цикла мы сможем выйти только с помощью команды `break` или вручную прекратив исполнение кода. `while False`, как можно догадаться, никогда не выполнится.

## ОПАСНО!

```
spisok = []  
while True:  
    spisok.append("a")
```

Подобный код будет бесполезно и беспощадно грузить Python-ядро, а также бесконечно потреблять оперативную память (с каждой итерацией все больше). Может вызвать зависание компьютера или даже вылет из системы.

Единственный способ прекратить его выполнение после запуска – прервать исполнение кода вручную.

# break и continue

```
while True:
    number = input("Введите число: ")
    if number.isdigit():
        print("Молодчина!")
        break
    print('Я сказал число!')    break
```

Команда **break** прерывает работу цикла.

Команды, следующие за ней в теле цикла, не выполняются. Следующая итерация не наступает.

Так, здесь будет бесконечно запрашиваться ввод до тех пор, пока пользователь не введет число.

Команда **continue** прерывает текущую итерацию и переходит к следующей. То, что следует за **continue**, не выполнится, но цикл не остановится.

Так, здесь, если пользователь ввел пробельный символ, цикл не дойдет до последнего **print'a**.

```
while True:
    number = input()
    if number.isdigit():
        break
    if number.isspace():
        print('нам не нужен пробел!')
        continue
    print('Вы ввели букву!')    continue
```

Обе команды применимы как для цикла **while**, так и для цикла **for**.

**Спасибо за внимание!**

`admvereshchagina@gmail.com`