

## ✓ Часть 1. Цикл for

На прошлом занятии мы рассмотрели, как в зависимости от условия выполнять различные блоки кода. Теперь рассмотрим такое понятие как цикл.

Циклы в языках программирования — это конструкции, которые позволяют выполнять один и тот же блок кода несколько раз. Они являются важным инструментом в программировании и используются для множества задач.

В программировании существует несколько основных типов циклов, которые позволяют выполнять повторяющиеся операции. Основными двумя типами являются цикл с параметром (for) и цикл с условием (while).

Цикл **for** в языке программирования Python предназначен для перебора элементов структур данных и других составных объектов, чтобы сделать с каждым из этих элементов какое-либо действие. Синтаксис оператора следующий:

```
for переменная in набор_значений:  
    инструкции
```

После ключевого слова **for** идет название переменной, в которую будут помещаться значения. Затем после оператора **in** указывается набор значений и двоеточие. А со следующей строки располагается блок инструкций цикла, которые также должны иметь отступы от начала цикла.

В Python for это не цикл со счетчиком, каковым он является во многих других языках. Например, у нас есть список, состоящий из ряда элементов. Интерпретатор сначала берет из него первый элемент, затем второй, потом третий и так далее. С каждым элементом он выполняет одни и те же действия в теле for. Здесь нет необходимости извлекать элементы по их индексам и заботиться на каком из них список заканчивается и следующая итерация бессмысленна. Цикл for сам переберет элементы и определит конец.



В качестве списка значений, например, можно рассматривать строку, которая по сути представляет набор символов. Посмотрим на пример:

```
1 message = "Привет"
2
3 for element in message:
4     print(element)
```

```
⇒ П
   р
   и
   в
   е
   т
```

После ключевого слова `for` используется переменная под именем `element`, которое может быть любым; нередко используют `i`. На каждой итерации цикла `for` переменной будет присвоен очередной элемент из строки (списка). Так, при первой итерации (первом шаге) цикла идентификатор `element` связан с буквой **П**, на второй – с буквой **р**, и так далее. Когда элементы в списке заканчиваются, цикл `for` завершает свою работу.


На русский язык синтаксис можно перевести так: для каждого элемента в списке делать следующее (что указано в теле цикла). В примере мы выводили на экран каждый элемент строки.

## ✓ Функция range

Однако если нам всё же нужен индекс для итерации, на помощь приходит функция **range**. Range означает «диапазон», то есть range(n) читается как «для всех чисел в диапазоне от 0 (включительно) до n (не включительно)». То есть функция генерирует последовательность чисел в указанном диапазоне. Так, range(5, 11) сгенерирует последовательность 5, 6, 7, 8, 9, 10. Однако это будет не структура данных типа "список". Функция range() производит объекты своего класса — диапазоны (итераторы).

Вместе с циклом for они образуют эффективный тандем. Возьмем пример — задача Фридриха Гаусса о быстром сложении чисел от 1 до 100. С помощью Python мы можем решить её не менее быстро:

```
1 num = 0
2 for i in range(101):
3     num += i
4 print(num)
5
```

 5050

Можно инициализировать объект range() с двумя аргументами, чтобы указать, с какого числа начинать итерироваться и каким закончить:

```
1 for i in range(4, 7):
2     print(i)
```

 4  
5  
6

Если же проинициализировать range() с тремя аргументами, последним из них будет шаг итерации:

```
1 for i in range(1, 10, 2):
2     print(i)
```

 1  
3  
5  
7  
9

Также есть возможность итерироваться по числам в порядке уменьшения, используя отрицательное значение шага:

```
1 for i in range(11, 6, -1):
2     print(i)
```

```
⇒ 11
   10
   9
   8
   7
```

Если имеется какой-нибудь простой цикл, не требующий осмысленного наименования переменных, в качестве индекса зачастую используют символ подчеркивания:

```
1 for _ in range(5):
2     print("Hello from ", _)
```

```
⇒ Hello from 0
   Hello from 1
   Hello from 2
   Hello from 3
   Hello from 4
```

## ✓ Функция enumerate

Если мы итерируемся, например, по строке, иногда нам необходимо знать не только букву, но и её номер. Можно для этого завести отдельный счётчик, который будет увеличиваться на каждой итерации цикла:


```
1 message = "Привет"
2 counter = 0
3 for element in message:
4     print(counter, element)
5     counter = counter + 1
```

```
⇒ 1 П
   2 р
   3 и
   4 в
   5 е
   6 т
```

Однако для этой цели можно воспользоваться функцией **enumerate**. Эта функция используется в цикле `for`, чтобы получать индекс и значение элемента из

последовательности (например, строки или списка) во время итерации. Она возвращает пару, состоящую из индекса и соответствующего элемента на каждой итерации. Например:

```
1 message = "Привет"
2
3 for i, element in enumerate(message):
4     print(i, element)
```



```
1 П
2 р
3 и
4 в
5 е
6 т
```

Нумерация элементов в Python, как и во многих языках программирования, начинается с нуля.

## ✓ Часть 2. Операторы break, continue и else

Для циклов существует ещё несколько полезных команд:

- **pass** (определяет пустой блок, который ничего не делает),
- **break** (позволяет выйти из цикла досрочно),
- **continue** (позволяет перейти к следующей итерации цикла без выполнения оставшихся инструкций),
- **else** (проверяет цикл на экстренный выход).

```
1 if 15 % 3 == 0:
2     ...
```

## ✓ Оператор break

Инструкция для прерывания цикла называется **break**. После её выполнения работа цикла прекращается (как будто не было выполнено условие цикла). Осмысленное использование конструкции break возможно, только если выполнено какое-то условие, то есть break должен вызываться только внутри if (находящегося внутри цикла). Несмотря на то, что иногда break может быть полезным, его использование — плохой тон в программировании и, по возможности, следует обходиться без него.

```
1 for char in 'Python':
2     if char == 'h':
3         break
4     print(char)
```


 P  
y  
t

В данном случае оператор `break` ведет сразу к окончанию работы цикла. При этом получается, что совершается экстренный выход из цикла, т.к. мы не переберем все элементы списка.

## ✓ Оператор `continue`

Оператор **`continue`** позволяет начать следующий проход цикла, минуя оставшиеся на данном шаге инструкции:

```
1 for char in 'Python':
2     if char == 'h':
3         continue
4     print(char)
```

 P  
y  
t  
o  
n

Мы перебираем последовательность символов и когда наша переменная станет хранить в себе символ 'h', мы используем оператор `continue`, чтобы пропустить дальнейшую инструкцию `print(char)`.

## ✓ Оператор `pass`

**`Pass`** — оператор-заглушка, равноценный отсутствию операции. В ходе исполнения данного оператора ничего не происходит, поэтому он может использоваться в тех местах, где это синтаксически необходимо.

Зачастую `pass` используется там, где код пока ещё не появился, но планируется. Кроме этого, иногда его используют при отладке, разместив на строчке с ним точку остановки.

```
1 for char in 'Python':  
2     pass # TODO: позже придумаем что хотим делать в цикле
```

**TODO** — это комментарии в исходном коде, которые используются для обозначения незавершенных задач или задач, которые предстоит выполнить в будущем. Они являются важной частью разработки программного обеспечения и служат для напоминания себе или своим коллегам о незаконченной задаче, либо как рекомендация к действию при написании отзыва на чужой код.

Важно помнить, что TODO комментарии должны использоваться осторожно и регулярно обновляться, чтобы обеспечить актуальность информации о задачах.

## ✓ Оператор else

В языке Python к циклу for можно написать блок **else**. Команды в этом блоке будут выполняться, если цикл завершил свою работу нормальным образом (т.е. условие в какой-то момент перестало быть истинным) и не будут выполняться только в случае, если выход из цикла произошел с помощью команды break.

```
1 for char in 'Python':  
2     if char == 'a':  
3         break  
4 else:  
5     print('Символа "a" нет в слове Python')
```

➡ Символа "a" нет в слове Python

Оператор else проверяет цикл на экстренный выход (break). Блок инструкций внутри else выполнится только в том случае, если выход из цикла произошел без помощи break.

Операторы continue, break и else работают с циклами for и while.

## ✓ Часть 3. Пример: нахождение простых чисел

Задача: вывести все простые числа, меньшие данного натурального числа. Натуральное число называется простым, если оно имеет только два различных делителя: единицу и само себя.

Простейшая реализация включает полный перебор чисел и подсчет для каждого числа его делителей:

```

1 n = int(input())
2 for i in range(2, n):
3     count = 0
4     for j in range(2, i):
5         if i % j == 0:
6             count += 1
7     if count == 0:
8         print(i)

```

# пробегаем все числа от 2 до n  
# в count будем хранить количество делителей  
# перебираем все числа от 2 до текущего  
# ищем количество делителей  
# если делителей нет, выводим число на экран

```

⇒ 16
   2
   3
   5
   7
  11
  13

```

Оптимизируем код. Можно заметить, что если найден хотябы один делитель, то число не простое, и дальнейшую проверку можно прервать конструкцией `break`. Также, можно перебирать делители не превосходящие корня из искомого: если у числа  $n$  имеется делитель  $p$ , то имеется делитель  $q$ , такой, что  $p \cdot q = n$ .

```

1 n = int(input())
2 for i in range(2, n):
3     count = 0
4     for j in range(2, int(i ** 0.5) + 1):
5         if i % j == 0:
6             count += 1
7             break
8     if count == 0:
9         print(i)

```

# пробегаем все числа от 2 до n  
# в count будем хранить количество делителей  
# перебираем все числа от 2 до корня квадр  
# ищем количество делителей; если найден х  
# если делителей нет, выводим число на экран

```

⇒ 16
   2
   3
   5
   7
  11
  13

```

Получаем небольшую прибавку в скорости.

## Вывод

Итак, циклы предоставляют программистам мощный инструмент для автоматизации обработки данных, и являются неотъемлемой частью практически всех программ на любых



языках программирования.

Основные сценарии для применения циклов такие:

- Итерация по данным.

Циклы позволяют перебирать элементы внутри коллекций, таких как списки, кортежи, словари и строки. Это позволяет обрабатывать данные, выполнять вычисления и применять операции к каждому элементу коллекции.

- Автоматизация повторяющихся задач.

- Работа с матрицами и массивами.

В вычислительных задачах и анализе данных циклы могут использоваться для манипуляций с элементами матриц и многомерных массивов.

- Итерация по числовым последовательностям.

Циклы часто применяются для генерации числовых последовательностей и выполнения операций на каждой итерации. Это может использоваться при высокопроизводительных расчетах и моделировании физических процессов.

- Оптимизация кода.

В некоторых случаях циклы могут использоваться для оптимизации кода, чтобы избежать дублирования и повысить читаемость.

- Управление потоком выполнения.

Циклы позволяют управлять потоком выполнения программы, например, с помощью операторов `break` (для прерывания цикла) и `continue` (для перехода к следующей итерации).