

Операторы цикла и списки.

FOR

Цикл for позволяет в одной переменной перебрать значения из какого-то множества или диапазона и для каждого из этих значений выполнить какое-то действие.

Множество значений задается структурой, позволяющей перечислить элементы в каком-то порядке. Такими структурами являются, например, списки, строки и диапазоны.

Функция range() может принимать от одного до трех параметров и возвращает диапазон значений следующим образом:

range(n)	n > 0	0, 1, 2, ..., n-1	range(5) = 0, 1, 2, 3, 4
range(a, b)	a < b	a, a+1, ..., b-1	range(2, 6) = 2, 3, 4, 5
range(a, b, step)	step > 0 => a < b step < 0 => a > b	a, a+s, ... a+k*s a+k*s - максимальное число, меньшее b	range(2, 10, 3) = 2, 5, 8 range(4, -1, -1) = 4, 3, 2, 1, 0

Параметры range() должны иметь тип int() - целые числа.

Результат функции range() имеет свой тип данных - range.

С помощью диапазонов и функции list() можно получать списки: l = list(range(5))

Циклы необходимы при работе с массивами, так как зачастую приходится перебирать значительный объем переменных и констант.

i = 1 for digit in 'one', 'two', 'three', 'four', 'five', 'six', 'seven': print(i, '~', digit) i += 1	В качестве множества значений используется кортеж.
for i in 1, 2, 3, 'four', 'five', 'six': print(i)	В первых 3 проходах по циклу переменная i будет принимать тип Integer, а в последующих String

WHILE

Цикл while (“пока”) позволяет выполнить одну и ту же последовательность действий, пока проверяемое условие истинно.

Условие записывается до тела цикла и проверяется до выполнения тела цикла. Как правило, цикл while используется, когда невозможно определить точное значение количества проходов исполнения цикла.

Синтаксис цикла while в простейшем случае выглядит так:

while *условие*:
 блок инструкций

<pre>i = 1 while i <= 10: print(i) i += 1</pre>	Данная программа выведет все натуральные числа от 1 до 10 включительно
<pre>n = int(input()) length = 0 while n > 0: n //= 10 length += 1 print(length)</pre>	В результате будет выведено кол-во цифр в числе n

Список (массив)

Для хранения большого количества переменных удобно использовать списки, специальный тип данных, который представляет собой последовательность элементов, пронумерованных от 0 до $n - 1$ (где n - количество элементов в списке).

Один из способов задать список - это перечисление элементов в явном виде.	<pre>l = [1, 2, 3] print(l)</pre>	[1, 2, 3]
Элементы списка могут иметь разный тип.	<pre>l = [1, '2', 3] print(l)</pre>	[1, '2', 3]
Мы можем положить в список значение переменной или даже другой список.	<pre>a = 5 l = [1, [2, 3, 4], a] print(l)</pre>	[1, [2, 3, 4], 5]
Длину списка можно узнать с помощью функции <code>len()</code> (результат - целое число).	<pre>l = [1, [2, 3, 4], 5] print(len(l))</pre>	3

Работа с элементами списка:

<pre>primes = [2, 3, 5, 7, 11, 13] rainbow = ['Red', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Violet']</pre>		
Обратиться к отдельному элементу списка можно с помощью адресации по индексу.	<pre>print(primes[0]) print(primes[2])</pre>	2 5
Можно использовать отрицательные числа (<code>lst[-1]</code> равносильно <code>lst[len(lst) - 1]</code> , то есть последнему элементу)	<pre>print(primes[-1]) print(rainbow[0]) print(rainbow[-2])</pre>	13 Red Indigo
Элементы списка можно менять так же, как обычные переменные.	<pre>primes[2] = 19 print(primes)</pre>	[2, 3, 19, 7, 11, 13]

Также в питоне есть возможность добавить новый элемент в конец списка с помощью функции `append()`.

поиск делителей числа	<pre>n = int(input()) d = [] for i in range(2, n + 1): if (n % i == 0): d.append(i) print(d)</pre>	12	[2, 3, 4, 6, 12]
		13	[13]

Срезы

Срезы чем-то напоминают функцию range().

Имеют вид `lst[x:y:step]`. Любой из трех аргументов можно опускать (по умолчанию `x = 0`, `y = len(lst) + 1`, `step = 1`).

Срезы создают новый список, при этом старый никак не меняется.

`lst[2]` - это не срез, а обращение к элементу списка.

```
a = [0, 1, 2, 3, 4, 5]
```

<code>print(a[:])</code>	<code>[0, 1, 2, 3, 4, 5]</code>	<code>print(a[1, 4, -1])</code>	<code>[]</code>
<code>print(a[1:])</code>	<code>[1, 2, 3, 4, 5]</code>	<code>print(a[::-1])</code>	<code>[5, 4, 3, 2, 1, 0]</code>
<code>print(a[0:2])</code>	<code>[0, 1]</code>	<code>print(a[:2])</code>	<code>[0, 2, 4]</code>
<code>print(a[:3])</code>	<code>[0, 1, 2]</code>	<code>print(a[-1:1:-1])</code>	<code>[5, 4, 3, 2]</code>

примеры использования

<pre>a = [1,2,3,4,5] List = a[:3] print(' '.join(map(str, List)))</pre>	1 2 3
<pre>a = [1,2,3,4,5] List = a[:3] print(' '.join(map(str, List)))</pre>	1 4

Обратите внимание, что при создании списка `a = []`, в переменной `a` хранится указатель на данный список, поэтому, если попробовать сохранить значение `a` в другую переменную `b`:

```
a = []
```

```
b = a
```

```
a.append(1)
```

```
print(b)
```

то на выходе мы получим `[1]`, так как теперь и `a` и `b` являются указателями на один и тот же массив.

Для копирования массива следует использовать другие способы. Например

```
a = [1, 2]
```

```
b = a[:]
```

```
a.append(3)
```

```
print(b)
```

На выходе будет уже `[1, 2]`

Считывание списков:

<pre>n = int(input()) lst = [] for i in range(n): lst.append(int(input())) print(lst)</pre>	<pre>3 1 2 3</pre>	[1, 2, 3]
<pre>n = int(input()) lst = list(map(int, input().split())) print(lst)</pre>	<pre>3 1 2 3</pre>	
<pre>n = int(input()) lst = [int(x) for x in input().split()] print(lst)</pre>		
<pre>n = int(input()) a = input().split() for i in range(len(a)): a[i] = int(a[i]) print(a)</pre>		

Вывод списков

<pre>a = [1,2,3] for elem in range(len(a)): print(a[elem])</pre>	Стандартный вывод данных из массива
<pre>a = [1,2,3] for elem in a: print(elem)</pre>	Эти 2 кода эквивалентны
<pre>a = [1,2,3] print(a)</pre>	На выходе мы получим: [1, 2, 3]

<pre>a = ['red', 'green', 'blue'] print(' '.join(a)) print("".join(a)) print('***'.join(a))</pre>	На выходе мы соответственно получим: <pre>red green blue redgreenblue red***green***blue</pre>
<pre>print(' '.join(map(str, a)))</pre>	Если наш список состоит из чисел, то придется сначала их привести к типу String Слева показан пример вывода числового списка разделенного пробелами

Методы	Что делает
list.append(x)	Добавляет элемент в конец списка
list.extend(L)	Расширяет список list, добавляя в конец все элементы списка L
list.insert(i, x)	Вставляет на i-ый элемент значение x
list.remove(x)	Удаляет первый элемент в списке, имеющий значение x. ValueError, если такого элемента не существует
list.pop([i])	Удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент
list.index(x, [start [, end]])	Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end)
list.count(x)	Возвращает количество элементов со значением x
list.sort([key=функция])	Сортирует список на основе функции
list.reverse()	Разворачивает список
list.copy()	Поверхностная копия списка
list.clear()	Очищает список

x in list	Проверить, содержится ли элемент в списке. Возвращает True или False
x not in list	То же самое, что not(x in A)
min(list)	Возвращает минимальный элемент в списке
Max(list)	Возвращает максимальный элемент в списке