
Яндекс

Академия Яндекса

Цикл for. Диапазоны

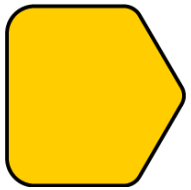
Именованные аргументы функции **print**

Именованные аргументы функции `print`

Мы уже пользовались тем, что функция **`print`** при выводе разделяет аргументы пробелами, а в конце переходит на новую строчку.
Часто это удобно. Но что, если от этого нужно избавиться?

```
measures = 7
cuts = 1
print('Количество отмеров:', measures, ', количество отрезов:', cuts)
```

```
Количество отмеров: 7 , количество отрезов: 1
```



Для такой тонкой настройки вывода у функции **`print`** существуют **необязательные именованные аргументы**.

Именованные аргументы функции `print`

Функция **`print`** наряду с другими аргументами может (вместе или по отдельности) принимать таких два аргумента: **`sep`** — *разделитель аргументов* (по умолчанию пробел) и **`end`** — *то, что выводится после вывода всех аргументов* (по умолчанию — символ начала новой строки).

```
print('При')
print('вет!')
print('При', end='')
print('вет!')
print('Раз', 'два', 'три')
print('Раз', 'два', 'три', sep='--')
```

```
При
вет!

Привет!
Раз два три
Раз--два--три
```

PEP8

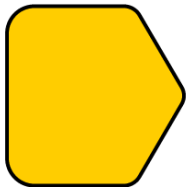
Не используйте пробелы вокруг знака =, если он используется для обозначения именованного аргумента.

Правильно: `print('При', end='')`

Неправильно: `print('При', end = '')`

Специальные символы в строках

Специальные символы в строках



Если внутри кавычек встречается символ "\" (обратная косая черта, обратный слэш, бэкслэш), то он вместе со следующим после него символом образует **экранирующую последовательность** (escape sequence) и воспринимается интерпретатором как **единый специальный символ**.

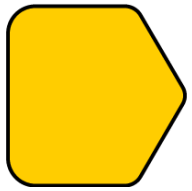
\n	\t	\'	/homepage/materials1course/\\
символ начала новой строки	табуляция	кавычка	просто бэкслэш

```
print('восход\t07:15\nзакат\t22:03')
print('Предыдущая строка этой программы выглядит так: print(\'восход\\t07:15\\nзакат\\t22:03\')')
```

```
восход  07:15
закат   22:03
Предыдущая строка этой программы выглядит так: print('восход\t07:15\nзакат\t22:03')
```

Специальные символы в строках

Таким образом, значения именованных аргументов функции `print` по умолчанию такие: `print(..., sep=' ', end='\n')`.



При этом, если приписать букву `r` перед открывающей строку кавычкой, то бэкслэши будут считаться обычными символами.

А если открывать и закрывать строку не одной, а тремя кавычками подряд, то внутри можно делать обычные переводы строки (внутри одинарных кавычек так делать нельзя).

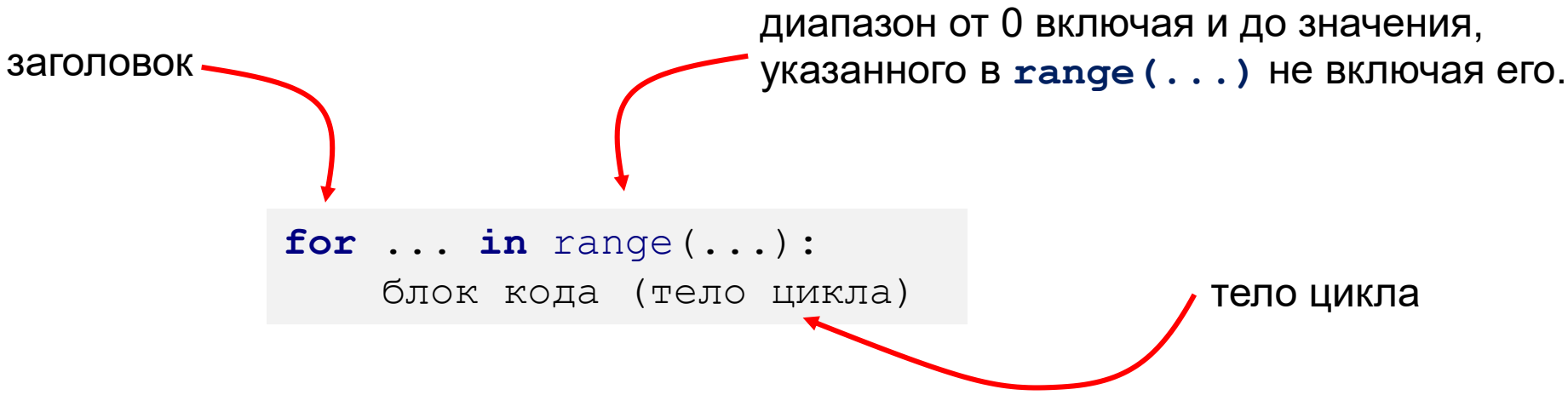
```
print(r'\\\\\\\nnnnn <- забор, переходящий в низкую изгородь')
print('''Нужно сказать много важного.
Одной строки для этого мало.
Зато три - в самый раз.''')
```

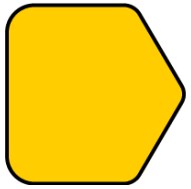
```
\\\\\\\nnnnn <- забор, переходящий в низкую изгородь
Нужно сказать много важного.
Одной строки для этого мало.
Зато три - в самый раз.
```

Цикл **for**

Цикл **for**

Цикл **`for`** выполняет блок кода заданное количество раз





`Range` означает «диапазон», то есть **`for i in range(n)`** читается как «для (всех) **`i`** в диапазоне от **`0`** (включительно) до **`n`** (не включительно)...». Цикл выполняется **`n`** раз.

Цикл **for**

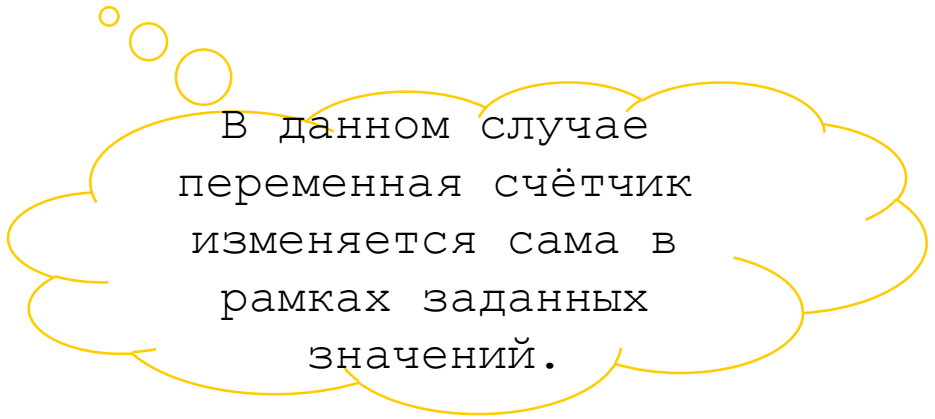
Давайте вспомним задачу, где мы три раза получали цены на товар и вычисляли общую цену товара.

Вот так мы ее записали через цикл **while**:

```
count = 0
total = 0
while count < 3:
    price = float(input())
    total = total + price
    count = count + 1
print('Сумма введённых чисел равна', total)
```

Теперь мы ее можем записать через цикл **for**, который будет выполняться 3 раза:

```
total = 0
for i in range(3):
    price = float(input())
    total = total + price
print('Сумма введённых чисел равна', total)
```



Соглашения об именовании переменных

Соглашения об именовании переменных

В программах, решающих абстрактные, математические задачи, допустимо называть переменные короткими и непонятными именами типа `n` и `i`. Однако этого лучше избегать.

Стоит соблюдать общепринятые договорённости:

`n` обычно обозначают количество чего-либо (например, итераций цикла).

Буквами `i` и `j` (по-русски они традиционно читаются как «и» и «жи») обычно обозначают итераторы цикла for.

При этом, если есть хоть какая-то определённость (например, речь идёт о количестве автомобилей), то стоит и переменную назвать более понятно (например, `cars`).

Начальное значение и шаг итератора в **range**

Начальное значение и шаг итератора в `range`

В скобках после слова **range** можно записать не одно, а два или три числа. Эти числа будут интерпретироваться как начальное значение итератора, конечное и его шаг (может быть отрицательным).

- ✓ Если для **range** задано одно число, то итератор идет от 0 до заданного значения (не включая его).
- ✓ Если задано два числа, то это начальное значение итератора и конечное.
- ✓ Если задано три числа, то это не только начальное и конечное значение итератора, но и шаг итератора.

```
for i in range(n):
```

```
for i in range(1, n):
```

```
for i in range(5, n, 2):
```

```
for i in range(1, 11):
    print(i)  # выведет на отдельных строчках числа
    # от 1 (включительно) до 11 (не включительно)
for i in range(1, 11, 2):
    print(i)  # выведет (на отдельных строчках) 1, 3, 5, 7, 9
for i in range(10, 0, -1):
    print(i)  # выведет 10, 9, ..., 1
```

Подведение итогов

Подведение итогов

- Цикл **`while`** нужен, когда какой-то кусок кода должен выполниться несколько раз, причём заранее неизвестно, сколько именно.
- Цикл **`for`** нужен, когда какой-то кусок кода должен выполниться несколько раз, при этом известно, сколько раз, еще до начала цикла.


Яндекс