Цикл for. Диапазоны

Аннотация

В уроке рассматриваются именованные аргументы функции print, специальные символы в строке и конструкция for ... in range(...):.

1.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? В примере ниже пробелы появляются не только после двоеточий (что хорошо), но и перед запятой (что плохо).

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

Необязательные именованные аргументы

Для такой тонкой настройки вывода у функции `print` существуют необязательные именованные аргументы — такие удобные инструменты бывают и у других функций, мы познакомимся с ними позже.

Обычно при вызове функции мы пишем имя функции, а затем в скобках ее аргументы через запятую. Стандартный способ сообщить функции, что и с какими аргументами делать (например, какой из аргументов функции `print` вывести первым, какой вторым и т. д.), — это передать аргументы в нужном порядке. Например, функция `print` выводит аргументы именно в том порядке,

в котором их ей передали. Однако есть и другой способ — именованные аргументы. Чтобы при вызове функции передать ей именованный аргумент, нужно после обычных аргументов написать через запятую имя аргумента, знак = и значение аргумента.

sep и end

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

В частности, если `end` сделать пустой строкой, то print не перейдет на новую строчку, и следующий `print` продолжит вывод прямо на этой же строке.

```
print('При')
print('вет!')
# эти две строки кода выведут "При" и
"вет!" на отдельных строках
print('При', end='')
print('вет!') # эти две строки кода выведут
"Привет!"
print('Раз', 'два', 'три') # выведет "Раз два
три"
print('Раз', 'два', 'три', sep='--') # выведет
"Раз--два--три"
```

Обратите внимание: знак `=` здесь не выполняет никакого присваивания, переменных `end` и `sep` не появляется.

PEP 8

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

Правильно:

```
print('При', end='')
```

~~Неправильно:~~

```
print('При', end = '')
```

2. Специальные символы в строках

Можно задаться вопросом: как указать значение `end` по умолчанию — символ начала новой строчки? Ведь это специальный символ, который нельзя

сделать частью строки, просто поместив его между кавычек, это вызовет ошибку.

Экранирующая последовательность

Если внутри кавычек встречается символ \ — обратная косая черта, обратный слеш, бэкслеш, он вместе с идущим после него символом образует экранирующую последовательность (escape sequence) и воспринимается интерпретатором как **единый специальный символ**.

В частности, \n — символ начала новой строки. Кроме того, \t — табуляция, \' — кавычка, \\ — просто бэкслеш. Использование экранирующих последовательностей вместо специальных символов называется их экранированием.

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

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

Важно!

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

А если открывать и закрывать строку не одной, а тремя кавычками подряд, 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.''')
```

3. Цикл for

Сегодня мы изучим еще один оператор цикла. Цикл `for` выполняет блок кода заданное количество раз.

Синтаксис

```
for ... in range(...):
    блок кода (тело цикла)
```

Как и у `while`, у цикла `for` есть заголовок, заканчивающийся

двоеточием, и тело цикла, которое записывается с отступом в четыре пробела. В цикле вида `for ... in range(...):` вместо первого многоточия указывается какая-то переменная, которая на начальной итерации принимает значение 0, на следующей — 1, и так далее, до значения указанного в `range(...)`, само это значение переменная не принимает. Диапазон значений переменной-итератора от 0 включая и до значения, указанного в `range(...)`, не включая его.

Вот программа, которая выводит на экран подряд (на отдельных строчках) целые числа от 0 (включительно) до n (не включительно).

```
n = int(input())
for i in range(n):
    print(i)
```

Range

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.

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 три раза:

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

В данном случае переменная-счетчик изменяется сама в рамках заданных значений.

Запустите эту программу с отладчиком и пройдите ее пошагово. Можно поставить breakpoint на первую же строчку или начать выполнение программы кнопкой F7. Следите за тем, как меняется

значение переменной `i`. Обратите внимание, что цикл for присваивает переменной `i` (она называется итератором цикла) значения (0, потом 1...), хотя нигде нет оператора присваивания = или его родственников типа +=.

4.Соглашения об именовании переменных

В программах, решающих абстрактные, математические задачи, допустимо называть переменные короткими и непонятными именами типа `n` или `i`. Однако этого лучше избегать. Кроме того, стоит соблюдать общепринятые договоренности: буквой `n` обычно обозначают количество чего-либо (например, итераций цикла). При этом если есть хоть какая-то определенность (например, речь идет о количестве автомобилей), то стоит и переменную назвать более понятно (например, `cars`). Буквами `i` и ;`j` (по-русски они традиционно читаются как «и» и «жи») обычно обозначают итераторы цикла `for`.

Еще один пример: программа, подсчитывающая сумму всех целых чисел, которые меньше данного.

```
n = int(input())
total = 0
for i in range(n):
    print('Рассматриваем число', i)
    total += i
    print('Промежуточная сумма равна', total)
print('Итоговая сумма всех этих чисел
 равна', total)
```

5.Начальное значение и шаг итератора в range

Однако это не все возможности цикла `for`.

Предположим, вам нужен цикл, выполняющийся 10 итераций. При этом нужно, чтобы итератор пробегал значения не подряд (0, 1, ..., 9), а, скажем, 10, 20, ..., 100. Конечно, с помощью уже известной нам конструкции `for` можно организовать цикл, в котором некая дополнительная переменная будет последовательно принимать именно такие значения (проверьте себя: как?).

Однако для этого есть и специальная встроенная в язык конструкция. В скобках после слова `range` можно записать не одно, а два или три числа. (Правда, очень похоже на функцию? Это не случайность, `range` — тоже функция, но об этом позже). 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(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
```

6. Когда какой цикл использовать

-Цикл while нужен, когда какой-то кусок кода должен выполниться несколько раз, причем 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