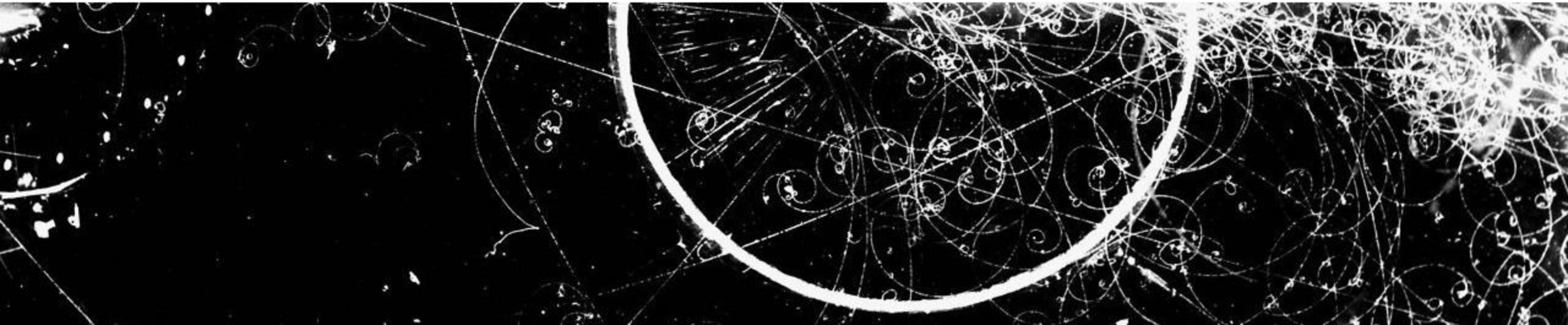
Язык Python

Ветвления и циклы — Часть I

Меджидов Р.Г.



if в C и в Python

По сравнению с языком C и другими СИ-подобными языками, оператор if в Python требует набора меньшего объема символов.

```
1  if (x < y)
2  {
3      x =
1;
4      y =
2;
5  }
```

```
1  if x < y:
2      x = 1
3      y = 2
```

Круглые скобки, в которые можно заключить условие, необязательны, но и ошибкой не являются.

То, что конец строки является концом оператора, а отступ — заменяет блок, заключенный в фигурные скобки, добавляет компактности коду.

Двоеточие

Новым, по сравнению с языком C, компонентом синтаксиса if в Python является символ двоеточия.

Все составные операторы в Python следуют общему шаблону:

строка_заголовка: вложенный_блок_кода

Отступы и нужный if в C

К какому из операторов if относится else (язык C)?

```
1    if (x)
2        if (y)
3
оп_1;
4    else
5        оп_2;
```

Ответ: if(y). Блок else в языке C относится к последнему оператору if.

Неправильный ответ можно дать, благодаря неверным (логически, но не синтаксически) отступам перед else для языка C.

То есть для понятности кода, отступы надо изменить, но язык позволяет и такое написание.

Отступы и нужный if в Python

В языке Python подобная ошибка возникнуть не может, так как блок else относится не к последнему оператору if, а к соответствующему по отступам:

```
1   if x:
2       if y:
3
оп_1
4   else:
5       оп_2
```

В данном случае else относится к первому оператору if x.

Область видимости Python

В языке Python операторы ветвлений и циклов не создают собственную область видимости, как это происходит в операторах ветвлений, циклов и блоках кода {} в языках C и C#:

СИ

```
1  if (1)
2  {
3      int x = 55;
4  }
5  printf("%d", x);
```

Error: 'x' undeclared

Python

```
1  if 1:
2      x = 55
3  print(x)
```

55

Ошибки нет.

C#

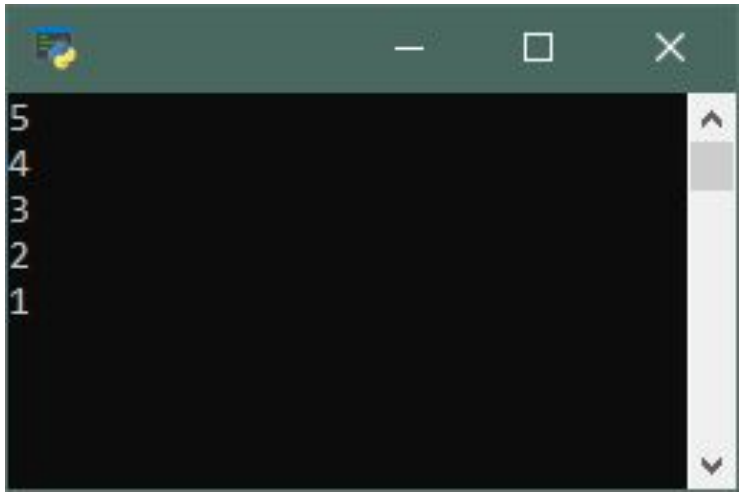
```
1  if (true)
2  {
3      int x = 55;
4  }
5  Console.WriteLine(x.ToString());
```

Error: The name 'x' does not exist in the current context

Оператор while

Оператор while многократно выполняет блок операторов до тех пор, пока проверка в заголовочной части оценивается как истинное значение:

```
1   x = 5
2   while x > 0:
3       print(x)
4       x -= 1
```



Оператор do while

В языке СИ существует не только оператор с предусловием while, но и оператор с постусловием do while (оператор repeat until в Pascal):

```
do {
    тело цикла
} while (условие);
```

В языке Python подобного цикла нет. При возникновении необходимости, можно осуществлять проверки «вручную», например:

```
while True:
    тело цикла
    if not условие:
        break
```

for в языке Python

Оператор for в языке Python, предназначен для прохода по элементам в последовательности или в другом итерируемом объекте и выполнения блока кода для каждого элемента.

То есть, другими словами, for больше не является универсальным циклом
как в языке C, но позволяет совершать действия над каждым элементом строки, списка, файла и прочих объектов.

Также стоит отметить, что цикл for *выполняется быстрее* аналогичного while.

for в языке C

```
for (инициализация; условия; изменение)
{
    делаем что угодно;
}
```

for в Python

```
for переменная in объект
    действия
```

Примеры работы оператора for

Оператор for языка Python во многом напоминает оператор foreach, таких языков как Java и C#, однако может содержать блок else подобно оператору while в Python.

Для понимания работы оператора for рассмотрим пример:

```
1    for i in '12345':
2        print(int(i) * 2, end='
')
```

2 4 6 8 10

Где int(i) — явное привидение к целочисленному типу.

for i in range(n)

Для повторения цикла некоторое заданное число раз n можно использовать цикл for вместе с функцией range:

```
1  for i in range(5):
2      print(i , end=' ')
```

```
0 1 2 3 4
```

Функция range может также принимать не один, а два или три параметра. Вызов range(a, b) означает, что индексная переменная будет принимать значения от a до b - 1.

Третий параметр означает величину изменения (шаг) индексной переменной.

```
1  for i in range(1, 11,
2):
2      print(i , end=' ')
```

```
1 3 5 7 9
```

Сравнение скорости циклов

На скриншоте время выполнения функций сложения целых чисел от 0 до 100 000 000:

```
while loop      15.8540316
for pure        10.620606200000001
sum range       6.4504783
numpy sum       0.8016000000000005

Process finished with exit code 0
```

1. Сложение при помощи цикла while
2. Сложение при помощи цикла for (+range)
3. Сложение при помощи встроенной функции sum (+range)
4. Сложение при помощи библиотеки numpy написанной на C

Подробнее — https://www.youtube.com/watch?v=Qgevy75co8c

Как нельзя

Код, который изменяет коллекцию во время обхода этой же коллекции, может привести к неожиданному результату.

```
1  s = '?????'
2  for i in s:
3      s = 'k'
4      print(s, end = ' ')
```

k k k k k

В данном примере ничего страшного не произошло, однако неочевидно, что цикл выполнится len(«исходной s») = 5 раз, при этом выводя «новую s» длины 1.

Намного проще и безопаснее выполнить цикл над копией коллекции.

continue и break

Данные операторы аналогичны одноименным операторам СИ:

- Оператор break переходит за пределы ближайшего заключающего его цикла, то есть начинают выполняться строки кода после всего оператора цикла, в котором расположен этот break.
- Оператор continue переходит в начало ближайшего заключающего цикла на строку заголовка цикла, то есть позволяет игнорировать все тело цикла, находящееся после continue.

```
1 for i in 'hello world':
2         if i == 'a':
3                 print('Буква A обнаружена')
4                 break
```

Задача

Определить число знаков (порядок) целого положительного числа.

```
1  x = 88005553535  # исходное число
2  res = 0
3  while x:      # аналог на C: while (x != 0)
4      res += 1      # инкремент (++res для C)
5      x //= 10      # краткая форма записи для x = x // 10
6  print(res)
```

Язык Python

Ветвления и циклы — Часть I

Меджидов Р.Г.