Lição 3

Estruturas condicionais

1. Operadores relacionais

Normalmente, as condições em Python são escritas com expressões relacionais que se utiliza de um ou mais dos seguintes operadores:

Símbolo	Operador	Descrição
<	Menor que	A condição é verdadeira se o lado esquerdo for menor que o lado direito
>	Maior que	A condição é verdadeira se o lado esquerdo for maior que o lado direito
<=	Menor ou igual a	A condição é verdadeira se o lado esquerdo for menor ou igual ao lado direito
>=	Maior ou igual a	A condição é verdadeira se o lado esquerdo for menor ou igual ao lado direito
==	Igual a	A condição é verdadeira se o lado esquerdo for igual ao lado direito
!=	Diferente de	A condição é verdadeira se o lado esquerdo for diferente do lado direito

Por exemplo, a condição **x * x < 1000** significa "o valor da expressão **x * x** é menor que **1000**", e a condição **2 * x != y** significa "o valor duplicado da variável **x** é diferente do valor da variável **y**".

Os operadores relacionais (ou de comparação) em Python podem ser agrupados assim: **a == b == c** ou **x <= y >= 10**. É uma coisa rara entre as linguagens de programação.

2. Operadores lógicos

Python tem os seguintes operadores lógicos:

Símbolo	Operador	Descrição
and	E	O operador **e** é um operador binário que avalia como verdadeiro (**True**), se e somente se os lados esquerdo e direito forem verdadeiros
or	Ou	O operador **ou** é um operador binário avaliado como verdadeiro (**True**) se pelo menos um de seus lados for verdadeiro
not	Não	O operator **não** é uma negação unária, é seguida por algum valor. É avaliado como verdadeiro (**True**) se esse valor for falso (**False**) e vice-versa

Por exemplo, a condição **x >= 1 and x <= 100** significa "o valor da variável x é maior ou igual a 1 e , mesmo tempo, menor ou igual a 100", ou seja, "x está no intervalo entre 1 e 100, inclusive". Diferentemente da maioria das linguagens de programação, estas condições poderiam também ser agrupadas e escritas da seguinte forma: **1 <= x <= 100**.

Às vezes, você precisa verificar várias condições ao mesmo tempo. Por exemplo, você pode verificar se um número **n** é divisível por **2** usando a condição **n% 2 == 0** (**n** ao ser dividido por **2** resulta no resto **0**). Se você precisar verificar se dois números **n** e **m** são divisíveis por **2**, você deve escrever **n% 2 == 0** e **m% 2 == 0**. Para fazer isso, você os une usando um operador **and** (e lógico) : **n% 2 == 0 and m% 2 == 0**.

3. Objetos tipo **bool** e operadores lógicos

Quando somamos dois objetos inteiros usando o operador **+**, como **2 + 5**, obtemos um novo objeto: **7**. Da mesma forma, quando comparamos dois inteiros usando o operador **<**, como **2 < 5**, obtemos um novo objeto: **True**.

```
1 print(2 < 5)
2 print(2 > 5)
```

Os objetos lógicos **True** e **False** têm um tipo especial chamado **bool**. Como todo nome de tipo pode ser usado para converter objetos nesse tipo, vamos ver o que a função de conversão **bool()** resulta para os valores:

```
1 print(bool(-10))    # True
2 print(bool(0))      # False - zero é o único número falso
3 print(bool(10))     # True
4
5 print(bool(''))     # False - string vazio é o único
6                     # string falso
7 print(bool('abc'))  # True
```

4. Comando condicional: **if**

4.1. Sintaxe

Todos os programas da primeira lição foram executados sequencialmente, linha após linha. Nenhuma linha pode ser pulada.

Vamos considerar o seguinte problema: dado o inteiro **X** ele sempre assumirá seu valor absoluto, que será mostrado. Se **X < 0**, o programa deve alterar o valor **X** para positivo. Este comportamento não pode ser alcançado usando o programa sequencial. O programa deve selecionar condicionalmente a próxima etapa. É aí que as condições ajudam:

```
1 x = int(input())
2 if x < 0:
3     x = -x
3 print(x)
```

Este programa usa uma declaração condicional **if**. Depois de **if** colocamos uma condição (**x < 0**) seguida de dois pontos. Depois disso, colocamos um bloco de instruções que será executado apenas se a condição for verdadeira (ou seja, avaliada como **True**. O bloco deve ser recuado para a direita com espaços. A instrução print (x) é executada em qualquer situação, porque não é **indentada**, portanto não pertence ao "*bloco do verdadeiro*".

Para resumir, a instrução condicional **if** em Python tem a seguinte sintaxe:

```
if condição:
    bloco do verdadeiro:
    uma ou várias instruções que são executadas
```

```
    se a condição for avaliada como True
```

O recuo é uma maneira geral em Python de separar blocos de código. Todas as instruções dentro do mesmo bloco devem ser **indentadas** da mesma maneira, ou seja, devem ter o mesmo número de espaços no início da linha. Recomenda-se usar 4 espaços para **indentação**.

O recuo (**indentação**) é o que torna o Python diferente da maioria das outras linguagens, nas quais as chaves {...} são usadas para formar os blocos.

A propósito, há uma função interna para valor absoluto em Python:

```
1  x = int(input())
2  print(abs(x))
```

5. Comando condicional: **if-else**

5.1. Sintaxe

Vamos considerar o seguinte problema: para o inteiro dado **X** imprima seu valor absoluto. Se **X > 0**, o programa deve imprimir o valor **X**, caso contrário, deve imprimir **-X**. Este comportamento mais uma vez, não pode ser alcançado usando o programa sequencial. O programa deve selecionar condicionalmente a próxima etapa:

```
1  x = int(input())
2  if x > 0:
3      print(x)
3  else:
4      print(-x)
```

Este programa usa uma declaração condicional **if**. Depois de **if** colocamos uma condição (**x > 0**) seguida de dois pontos. Depois disso, colocamos um bloco de instruções que será executado apenas se a condição for verdadeira ("bloco do verdadeiro), ou seja, avaliada como **True**. Este bloco é seguido pela palavra **else**, dois pontos e outro bloco de instruções que será executado somente se a condição for falsa, ou seja, avaliada como **False**. Quando a condição é falsa, então o "*bloco do falso*" é executado. Cada bloco deve ser recuado com a mesma quantidade de espaços para a direita.

Para resumir, a instrução condicional em Python tem a seguinte sintaxe:

```
if condição:
    bloco do verdadeiro:
    uma ou várias instruções que são executadas
    se a condição for avaliada como True
else:
    bloco do falso:
    uma ou várias instruções que são executadas
    se a condição for avaliada como False
```

Vamos verificar se pelo menos um dos dois números termina com 0:

```
1  a = int(input())
2  b = int(input())
3  if a%10 == 0 or b%10 == 0:
4      print("SIM")
5  else:
6      print("NÃO")
```

Vamos verificar se o número a é positivo e o número b não é negativo:

```
1  a = int(input())
2  b = int(input())
3  if a > 0 and not(b < 0):  # Em vez de not(b < 0),
4                            # poderia ser escrito  b >= 0
5      print("SIM")
6  else:
7      print("NÃO")
```

6. Comando condicional: **if-elif-else**

6.1. Sintaxe

Se você tiver mais de duas opções para diferenciar usando operadores condicionais, você pode usar a instrução **if... elif...else...**

Vamos mostrar como funciona reescrevendo o exemplo com o ponto (x, y) no plano e quadrantes de cima:

```
1  x = int(input())
2  y = int(input())
3  if x>0 and y>0:
4      print("I Quadrante")
5  elif x>0 and y<0:
6      print("IV Quadrante")
7  elif y>0:
8      print("II Quadrante")
9  else:
10     print("III Quadrante")
```

Nesse caso, as condições em **if** e **elif** são verificadas uma após a outra até que a primeira condição verdadeira seja encontrada. Então, apenas o "*bloco do verdadeiro*" para aquela condição será executado. Se todas as condições forem falsas, o bloco "**else**" (*bloco do falso*) será executado, se estiver presente.

```
if condição_1:
    bloco verdadeiro_1:
    uma ou várias instruções que são executadas
```

```
    se a condição_1 for avaliada como True
elif condição_2:
    bloco do verdadeiro 2:
    uma ou várias instruções que são executadas
    se a condição_2 for avaliada como True
...
elif condição_n:
    bloco do verdadeiro n:
    uma ou várias instruções que são executadas
    se a condição_n for avaliada como True
else:
    bloco do falso (opcional):
    uma ou várias instruções que são executadas
    se todas as condições forem avaliadas como False
```

7. Estruturas condicionais aninhadas

Qualquer instrução Python pode ser colocada em "*blocos dos verdadeiros*" e "*blocos dos falsos*", incluindo outra instrução condicional. Dessa forma, obtemos condições aninhadas. Os blocos de condições internas são recuados usando duas vezes mais espaços (por exemplo, 8 espaços). Vamos ver um exemplo. Dadas as coordenadas do ponto no plano, imprima seu quadrante.

```
 1 x = int (input())
 2 y = int (input())
 3 if x>0:
 4     if y>0:
 5         # x é maior que 0, y é maior que 0
 6         print("I Quadrante")
 7     else:
 8         # x é maior que 0, y é menor ou igual a 0
 9         print("IV Quadrante")
10 else:
11     if y>0:
12         # x é menor ou igual a 0, y é maior que 0
13         print("II Quadrante")
14     else:
15         # x é menor ou igual a 0, y é menor ou igual a 0
16         print("III Quadrante")
```

Neste exemplo, usamos os comentários: o texto explicativo que não tem efeito na execução do programa. Este texto começa com o "#" e vai até o final da linha.