



**universidade de aveiro**  
escola superior de tecnologia e  
gestão de água

# Introdução à Programação

Instruções condicionais

# Divisão, divisão arredondada e resto da divisão

O operador de divisão, `//`, divide dois números e arredonda para um número inteiro.

Por exemplo, suponhamos que o tempo de duração de um filme seja de 105 minutos.

Para saber quanto tempo isso é em horas. A divisão convencional retorna um número de ponto flutuante:

```
>>> minutes = 105
>>> minutes / 60
1.75
```

Normalmente não escrevemos horas com pontos decimais. A divisão arredondada retorna o número inteiro de horas, arredondando para baixo:

```
>>> minutes = 105
>>> hours = minutes // 60
>>> hours
1
```

Para obter o restante, temos de subtrair uma hora em minutos

```
>>> remainder = minutes - hours * 60
>>> remainder
45
```

Uma alternativa é usar o operador de módulo, `%`, que divide dois números e retorna o resto.

```
>>> remainder = minutes % 60
>>> remainder
45
```

# Exemplo de utilização do operador módulo divisão

Verificar se um número é divisível por outro

Se  $x \% y$  é zero, então  $x$  é divisível por  $y$ .

Extrair o dígito ou dígitos mais à direita de um número

$x \% 10$  produz o dígito mais à direita de  $x$  (na base 10). Ex:  $12345 \% 10 = 5$

Da mesma forma,  $x \% 100$  produz os dois últimos dígitos. Ex:  $12345 \% 100 = 45$

# Expressões booleanas

Uma expressão booleana é uma expressão que é verdadeira ou falsa.

Os exemplos a seguir usam o operador `==`, que compara dois valores e produz `True` se forem iguais e `False` caso contrário:

```
>>> 5 == 5
True
>>> 5 == 6
False
```

`True` e `False` são valores especiais que pertencem ao tipo `bool`; não são strings:

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

# Operadores relacionais

- ▶  $x \neq y$       # x não é igual a y
- ▶  $x > y$         # x é maior que y
- ▶  $x < y$         # x é menor que y
- ▶  $x \geq y$         # x é maior ou igual a y
- ▶  $x \leq y$         # x é menor ou igual a y

Um erro comum é usar um único sinal de igual (=) em vez de um duplo sinal de igual (==).

**ATENÇÃO:** Lembrar que = é um operador de atribuição e == é um operador relacional. Não existe =< ou =>.

# Operadores lógicos

Existem três operadores lógicos:

And (e), Or (ou), e Not (não). A semântica (significado) desses operadores é semelhante ao seu significado em inglês.

Por exemplo,  $x > 0$  **and**  $x < 10$  é verdadeiro somente se  $x$  for maior que 0 e menor que 10.

$n\%2 == 0$  **or**  $n\%3 == 0$  é verdadeiro se uma ou ambas as condições forem verdadeiras, ou seja, se o número for divisível por 2 ou 3.

Finalmente, o operador **not** nega uma expressão booleana, então **not**  $(x > y)$  é verdadeiro se  $x > y$  for falso, ou seja, se  $x$  for menor ou igual a  $y$ .

Estritamente falando, as operações dos operadores lógicos devem ser expressões booleanas, mas o Python não é muito rigoroso. Qualquer número diferente de zero é interpretado como True:

```
>>> 42 and True  
True
```

Essa flexibilidade pode ser útil, mas perigosa ao mesmo tempo. Devemos evitar estas situações (a menos que se saiba o que se está a fazer).

# Execução condicional

Para escrever programas úteis, quase sempre precisamos da capacidade de verificar as condições e alterar o comportamento do programa. Declarações condicionais dão essa habilidade. A forma mais simples é a instrução **if**:

```
if x > 0:
```

```
    print('x é positivo')
```

A expressão booleana a seguir ao **if** é chamada de condição.

Se for verdadeiro, a instrução indentada é executada. Se não, nada acontece.

As instruções **if** têm a seguinte estrutura: um cabeçalho seguido por um corpo indentado.

Declarações como esta são chamadas de declarações compostas.

# Execução condicional

Não há limite para o número de declarações que podem aparecer no corpo, mas deve haver pelo menos uma.

Ocasionalmente, é útil ter um corpo sem declarações (geralmente como um marcador de lugar para código que ainda não foi escrito). Nesse caso, você pode usar a instrução **pass**, que não faz nada.

if  $x < 0$ :

    pass      # TODO: need to handle negative values!

# Execução alternativa

Uma segunda forma da instrução if é a “execução alternativa”, na qual existem duas possibilidades e a condição determina qual delas é executada. A sintaxe fica assim:

```
if x % 2 == 0:  
    print('x é par')  
else:  
    print('x é impar')
```

Se o resto da divisão de x dividido por 2 for 0, então sabemos que x é par e o programa exibe a mensagem apropriada.

Se a condição for falsa, o segundo conjunto de instruções será executado. Como a condição deve ser verdadeira ou falsa, exatamente uma das alternativas será executada. As alternativas são chamadas de ramificações, pois são ramificações no fluxo de execução.

# Condicionais encadeados

Às vezes, há mais de duas possibilidades e precisamos de mais de duas ramificações. Uma maneira de expressar uma computação como essa é uma condicional encadeada:

```
if x < y:  
    print('x is less than y')  
elif x > y:  
    print('x is greater than y')  
else:  
    print('x and y are equal')
```

**elif** é uma abreviação de “else if”. Novamente, exatamente uma ramificação será executada.

Não há limite para o número de instruções elif.

Se houver uma cláusula else, esta deve estar no final, mas não é obrigatório que haja uma.

Cada condição é verificada por ordem. Se o primeiro for falso, o próximo é verificado e assim por diante.

Se um deles for verdadeiro, a ramificação correspondente é executada e a instrução termina.

Mesmo se mais de uma condição for verdadeira, somente a primeira ramificação verdadeira será executada.

# Condicionais compostas

Uma condicional também pode estar contida dentro de outra. Poderíamos ter escrito o exemplo na seção anterior assim:

```
if x == y:
    print('x and y are equal')
else:
    if x < y:
        print('x is less than y')
    else:
        print('x is greater than y')
```

A condicional externa contém duas ramificações.

A primeira ramificação contém uma instrução simples.

A segunda ramificação contém outra instrução if, que possui duas ramificações próprias.

Essas duas ramificações são instruções simples, embora também possam ter ser instruções condicionais.

# Condicionais compostas

Embora a indentação das instruções torne a estrutura evidente, as condicionais compostas tornam-se difíceis de ler muito rapidamente. É uma boa ideia evitá-las sempre que se puder.

Os operadores lógicos geralmente fornecem uma maneira de simplificar instruções condicionais compostas. Por exemplo, podemos reescrever o código a seguir usando uma única condicional:

```
if 0 < x:  
    if x < 10:  
        print('x is a positive single-digit number.')
```

A instrução print é executada apenas se passarmos pelas duas condicionais, para que possamos obter o mesmo efeito com o operador and:

```
if 0 < x and x < 10:  
    print('x is a positive single-digit number.')
```

Para esse tipo de condição, o Python oferece uma opção mais concisa:

```
if 0 < x < 10:  
    print('x is a positive single-digit number.')
```