Aula 2 - Operações matemáticas, variáveis

Karl Jan Clinckspoor

15 de julho de 2018

Sumário

1 Operações matemáticas

Nesta aula serão introduzidos dois novos tipos de Python, números inteiros, ou *int*, e números decimais, ou *float*. Serão realizadas algumas operações entre números, depois isso será abstraído para variáveis e, por final, será mostrado como se faz a conversão de um tipo para outro.

```
In [2]: print(1 + 2)  # Soma
        print(1 - 2)  # Subtração
        print(1 * 2)  # Multiplicação
        print(1 / 2)  # Divisão
```

```
3
-1
2
0.5
```

Note que o Python mostra números negativos e decimais da mesma maneira que nós. As partes que possuem '#' são chamadas de comentários. Tudo à direita de um # é ignorado pelo interpretador. Isso é essencial para que você não se perca no seu código, e também é útil para remover reversívelmente partes problemáticas do seu código.

Veja se você nota a diferença entre essas duas operações:

```
In [1]: print(1 + 2)    # Int
        print(1. + 2)   # Float

3
3.0
```

Supostamente os resultados seriam iguais. Porém, a diferença é que no segundo caso, o número gerado é um *float*, porque um dos dois também é um float. Caso o resultado gere um número decimal, o resultado será convertido para um float, como visto no caso de 1/2. Isso é exclusivo de Python 3, em Python 2, o resultado seria 0.

Para saber qual é o tipo de um objeto, é só utilizar a função **type**.

```
In [9]: print(type(1))
        print(type(1.))
        print(type('1'))

<class 'int'>
<class 'float'>
<class 'str'>
```

Veja que ele mostra que os tipos desses números são inteiros (int), float e strings (str).

```
In [6]: print(10 % 3)   # Retorna o resto da divisão. Em inglês chamado de "modulo"
        print(10 // 3)  # Retorna a divisão cortando as casas decimais. Chamado de "floor divisi
        print(10 ** 3)  # Potência

1
3
1000
```

Note que eu coloco um espaço entre os números e as operações. Isso é mais para facilitar a leitura das operações do que algo estritamente necessário. É uma convenção que eu acho bastante útil.

Operadores matemáticos, da maior para menor precedência:

- **: Exponencial
- %: Módulo/resto
- //: Divisão inteira
- /: Divisão
- *: Multiplicação
- -: Subtração
- +: Adição

1.1 Operações envolvendo strings

Alguns operadores matemáticos podem ser utilizados em outros objetos, que não números, dependendo se a operação em si foi definida no objeto. Qual seria o resultado de aplicar uma operação de soma entre duas strings?

```
In [7]: print('ABC' + 'def')
```

```
ABCdef
```

Veja que ele juntou as duas strings em uma só. Porém, veja o seguinte:

```
In [8]: print('ABC' - 'ABC')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

<ipython-input-8-42a4ec3a2677> in <module>()
----> 1 print('ABC' - 'ABC')


TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

1.2 Erros

O resultado da operação foi um erro. Erros são bastante comuns em programação. Python em especial pode produzir mensagens de erro compridíssimas, porém ele consegue, frequentemente, mostrar exatamente a linha onde o erro aconteceu. Junto da linha do erro, é mostrado o tipo do erro. Nesse caso, um 'TypeError', ou seja, um erro associado ao tipo. Aqui, ele fala que ele não sabe o que fazer quando uma subtração é utilizada entre duas strings (str).

Caso você receba uma mensagem de erro difícil de ser interpretada, você pode copiar e colar a mensagem no Google ou DuckDuckGo e obter informações mais detalhadas sobre o seu erro. Veja aqui alguns erros comuns, mas não se preocupe em entender a nomenclatura:

- TypeError: operação ou função aplicada a um tipo incompatível
- IndexError: índices não presentes em listas
- SyntaxError: Python não entendeu o que está escrito na linha
- IndentationError: Falta ou há indentação extra
- NameError: erro de escopo ou variável não foi declarada
- ZeroDivisionError: divisão por zero

2 Variáveis

Variáveis são maneiras de se atribuir valores a nomes. Python é uma linguagem onde o tipo da variável é fluído, ou seja, não é necessário defini-lo de antemão, e é possível realizar operações

sem conhecer o tipo da variável. O processo de atribuição de um valor à uma variável é chamado de **declaração**, e é realizado com um sinal de igual, `=`, por exemplo, `nome=valor`. Isso não significa que é uma equação, ou seja, `valor=nome` é totalmente diferente de `nome=valor`!

```
In [13]: a = 1
         b = 2
         c = a + b
         print(c)
```

```
3
```

```
In [14]: a = '1'
         b = '2'
         c = a + b
         print(c)
```

```
12
```

Veja que essencialmente as mesmas operações foram feitas nos dois casos. Porém, em um caso, somou-se dois números e no outro caso, duas strings foram concatenadas.

2.1 Conversão de uma variável

É possível converter uma variável de um tipo para outro utilizando as funções específicas para cada tipo. Por exemplo, **int()** converte qualquer tipo a um número inteiro, desde que isso seja possível. Veja.

```
In [15]: a = '1'
         b = '2'
         a = int(a)
         b = int(b)
         c = a + b
         print(c)
```

```
3
```

O oposto também pode ser feito.

```
In [16]: a = 1
         b = 2
         a = str(a)
         b = str(b)
         c = a + b
         print(c)
```

```
12
```

2.1.1 Junção de números e strings

Isso é bastante útil quando se deseja escrever os resultados de, por exemplo, um ajuste, para um arquivo. Como arquivos de texto conhecem somente texto, é necessário então transformar todos os números em texto (strings) antes de serem gravados. Por exemplo.

```
In [26]: resultado = 10 / 3
         print('O resultado do ajuste foi %s' % resultado)

O resultado do ajuste foi 3.3333333333333335
```

Neste caso, utilizou-se o operator '%' para fazer outra tarefa, que foi achar todos os locais que continham o sinal de % e colocar o valor que seguia % no lugar. Há duas outras maneiras de se fazer isso, e eu, pessoalmente, acho essa maneira mostrada aqui pouco elegante. Os dois métodos são: Utilizando **.format** e **f-strings**.

Veja que foi colocado um número muito longo depois da vírgula. Isso é frequentemente indesejado, então é possível cortar esse excesso.

```
In [27]: print('O resultado do ajuste foi {}'.format(round(resultado, 2)))

O resultado do ajuste foi 3.33
```

.format é um método que coloca números que recebe como argumentos nos locais que contém chaves {}. Aqui, ele colocou o resultado a função de arredondamento do Python **round**, dentro das chaves. Como essa sintaxe é um pouco feia, há outra maneira.

```
In [34]: print('O resultado do ajuste foi {0:.2f}'.format(resultado))

O resultado do ajuste foi 3.33
```

Agora, dentro das chaves colocou-se um código que significa "pegue o primeiro resultado e, depois da vírgula, coloque somente duas casas decimais, e trate o número como se fosse um float". Esse tipo de representação pode parecer complicado, e não é estritamente necessário saber todos os detalhes. Sempre que surgir a dúvida, é só consultar na internet.

A terceira maneira, exclusiva para versões mais recentes do Python, são as f-strings. Da mesma maneira que *raw* strings, um caracter, f, precede as aspas. A sintaxe depois é muito semelhante à sintaxe de .format. Dentro dos colchetes, pode ser colocada qualquer expressão de Python, desde que tenha uma única linha e seja válida. Porém, evite de colocar expressões muito complicadas dentro.

```
In [40]: print(f'O resultado do ajuste foi {resultado:.2f}')

O resultado do ajuste foi 3.33
```

3 Comparações

Comparações são utilizadas para testar as relações entre duas variáveis. As comparações existentes são:

- ==, igualdade. Note a presença de dois '='!
- !=, desigualdade.
- >, maior.
- <, menor.
- >=, maior ou igual
- <=, menor ou igual

```
In [4]: a = 1
        b = a
        c = 2
        d = 3

        print(a == b)
        print(a != b)
        print(a < c)
        print(a <= c)
        print(a > c)
        print(a >= c)
        print(a < c < d) # Funciona como na matemática, mas tente evitar, porque isso não funcio
```

```
True
False
True
True
False
False
True
```

Essas comparações também servem para strings. Tome cuidado, porém, com o significado de maior ou menor para strings. Isso tem a ver com os valores dos caracteres na tabela ASCII.

```
In [49]: a = 'abc'
         b = 'abc'
         c = 'def'

         print(a == b)
         print(a != b)
         print(a < c)
         print(a <= c)
         print(a > c)
         print(a >= c)
```

```
True
False
```

```
True
True
False
False
True
```

4 Exercícios

Explore os métodos para juntar strings e números (mais informações sobre coisas como *%d* e *:.2f* podem ser encontradas aqui e aqui).