MC102-2018s1-Aula02-180228

Arthur J. Catto, PhD

01 de março de 2018

2 Dados simples em Python

2.1 Comandos, expressões e variáveis

Programas são compostos por **comandos** que dizem ao computador o que ele deve fazer.

Nesta aula vamos conhecer alguns comandos básicos do Python e ver como eles podem ser usados para resolver alguns problemas matemáticos simples.

A construção de **expressões matemáticas** envolvendo **variáveis** é uma das tarefas mais comuns em programação e também é um ótimo caminho para se começar a aprender a programar.

É também a base de quase tudo o que vamos ver nesta disciplina.

2.2 Objetos, valores e tipos de dados

Os elementos básicos que um programa manipula e transforma são chamados **objetos**.

Todo objeto pertence a uma **classe**, que caracteriza o objeto e define o **tipo de valores** que ele pode assumir e as **operações** possíveis sobre ele.

Python possui uma função `type(`*arg*`)` que dá o tipo do objeto *arg* passado como argumento.

2.2.1 Exemplos

```
In [1]: print(type(12345))
```

```
<class 'int'>
```

```
In [2]: print(type(3.1416))
```

```
<class 'float'>
```

```
In [3]: abcde = '123'
        print(type(abcde))
        abcde = 123
        print(type(abcde))
```

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

```
In [4]: print(type('3.1416'))

<class 'str'>
```

2.2.2 É fácil exibir valores...

```
In [5]: print(12345)

12345
```

```
In [6]: print(3,1416)

3 1416
```

O que aconteceu?
`print` exibe qualquer número de valores, desde que eles estejam separados por vírgulas. Um espaço é usado para separar os valores na apresentação.

```
In [7]: print(1, 2, 3,4,5)

1 2 3 4 5
```

```
In [8]: print(3.1416,3,1416)

3.1416 3 1416
```

Em Python, cadeias de caracteres podem ser delimitadas por aspas simples, aspas duplas ou "aspas triplas".

```
In [9]: casa = 1
        amarela = 2
        print(casa, amarela)

1 2
```

```
In [10]: print("abcde")

abcde
```

```
In [11]: print("""abcde""", '''fghij''')

abcde fghij
```

E se a cadeia a ser impressa contiver aspas?

```
In [12]: print('copo d'água')
```

```
  File "<ipython-input-12-03f53dee3440>", line 1
    print('copo d'água')
                  ^
SyntaxError: invalid syntax
```

Uma cadeia delimitada por aspas simples pode conter aspas duplas.
Uma cadeia delimitada por aspas duplas pode conter aspas simples.
Uma cadeia delimitada por aspas triplas pode conter aspas simples e duplas.

```
In [13]: print("Quero um copo d'água.", 'Acho que vou "pagar o pato".')
```

```
Quero um copo d'água. Acho que vou "pagar o pato".
```

```
In [14]: print("""Tomei um copo d'água""", '''em Santa Bárbara d'Oeste.''')
```

```
Tomei um copo d'água em Santa Bárbara d'Oeste.
```

Além disso, uma cadeia delimitada por aspas triplas pode se estender por mais do que uma linha.

```
In [16]: print("""copo
         muito cheio
         d'água""")
```

```
copo
muito cheio
d'água
```

```
In [17]: print("copo
              d'água")
```

```
  File "<ipython-input-17-dd4dc6baa698>", line 1
    print("copo
               ^
SyntaxError: EOL while scanning string literal
```

2.3 Funções de conversão de tipos

As funções `int`, `float` e `str` tentam converter seus argumentos aos respectivos tipos.

```
In [18]: print(int(3.1416), int(2.999), int('1234'))
```

```
3 2 1234
```

```
In [22]: print(float(1234), float('3.1416'))
```

```
1234.0 3.1416
```

```
In [20]: print(str(1234), str(3.1416))
```

```
1234 3.1416
```

```
In [21]: print(int(float('3.1416')))
```

```
3
```

Uma exceção é gerada quando o argumento não permite a conversão.

```
In [23]: print(int('3.1416'))
```

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

<ipython-input-23-18b956ab43c0> in <module>()
----> 1 print(int('3.1416'))


ValueError: invalid literal for int() with base 10: '3.1416'
```

```
In [24]: print(float('abcde'))
```

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

<ipython-input-24-4e80dcd9b10e> in <module>()
----> 1 print(float('abcde'))


ValueError: could not convert string to float: 'abcde'
```

2.4 Variáveis

Uma dos recursos mais poderosos de um programa é a sua capacidade de manipular **variáveis**.

Uma variável é um **nome** que se refere a um objeto.

Uma variável é criada por um **comando de atribuição**, que também serve para associá-la a um objeto.

```
In [26]: mensagem = 'Olá pessoal!'
         total = 512
         pi = 3.14159265
         print(mensagem, total, pi)
```

```
Olá pessoal! 512 3.14159265
```

O **operador de atribuição** = associa a variável que está à esquerda ao objeto que está à direita.

Assim, se tentarmos executar o comando abaixo, causaremos uma exceção.

```
In [27]: 3.1416 = pi
```

```
          File "<ipython-input-27-f8bfb86d016b>", line 1
        3.1416 = pi
                 ^
    SyntaxError: can't assign to literal
```

A associação entre uma variável e um objeto permanece, até que seja redefinida por uma nova atribuição.

```
In [28]: dia = 1
         print(dia)
         dia = 'quinta-feira'
         print(dia)
```

```
1
quinta-feira
```

Quando aplicada a uma variável, a função `type()` retorna o tipo do objeto que está associado a ela nesse momento.

```
In [29]: dia = 1
         print(dia, type(dia))
         dia = 'quinta-feira'
         print(dia, type(dia))
```

```
1 <class 'int'>
quinta-feira <class 'str'>
```

2.5 Nomes de variáveis e palavras-chave

O nome de uma variável pode ser qualquer combinação de letras, algarismos e "sublinhados" (_) mas *não* pode começar por um algarismo.
- O manual de estilo de Python recomenda não usar letras maiúsculas, embora nem todos respeitem essa recomendação. - São exemplos de nomes válidos: - dia_da_semana - diaDaSemana - turma15 - turma_15

Defina seu estilo de nomeação e atenha-se a ele.

Algumas palavras, denominadas **palavras-chave**, são reservadas e não podem ser usadas como nomes de variáveis.
Você encontrará na Ajuda a lista completa de palavras-chave de Python.

Não há qualquer relação implícita entre o nome de uma variável e os objetos aos quais ela pode ser associada.
Assim, os comandos abaixo estão absolutamente corretos, embora pareçam não fazer sentido...

```
In [30]: dia = 45
         mes = 21
         ano = -32
         print(dia, mes, ano)
```

```
45 21 -32
```

2.6 Comandos e expressões

Um **comando** é uma instrução que o interpretador Python consegue executar - Por exemplo, o comando de atribuição que acabamos de conhecer.

Uma **expressão** é uma combinação de valores, variáveis, operadores e chamadas de função que - respeita as regras sintáticas da linguagem de programação - pode ser avaliada, gerando um resultado que é um objeto.

Exemplo

```
In [31]: x = 'bom dia'
         y = -15
         print(len(x), abs(y))
```

```
7 15
```

Exemplo

```
In [32]: pi = 3.14159265
         raio = 4
         circunferencia = 2 * pi * raio
         print(raio, circunferencia)
```

```
4 25.1327412
```

2.7 Operadores e operandos

Nós estamos acostumados a lidar com expressões matemáticas. Python não é muito diferente...

Operadores são tokens especiais que representam operações como adição, multiplicação e divisão.
Operandos são os valores com os quais os operadores trabalham.

```
In [33]: 4 + 13

Out[33]: 17
```

Os operadores `+`, `-`, `*` comportam-se como esperado.
A exponenciação é indicada por `**`.
Parênteses `(` e `)` também são usados da maneira habitual para agrupar sub-expressões e permitir a alteração da ordem de avaliação dos resultados.

```
In [34]: a = 1.0
         b = 5.0
         c = 6.0
         delta = b ** 2 - 4 * a * c
         print(a, b, c, delta)

1.0 5.0 6.0 1.0
```

O operador `/` indica uma divisão em ponto flutuante (`float`) e `//` indica uma divisão inteira.
Uma divisão inteira sempre arredonda seu resultado **para baixo**.

```
In [35]: print(17 / 3)

5.666666666666667


In [36]: print(17 // 3)

5


In [37]: print(-17 // 3)

-6


In [38]: print(17 // -3)

-6
```

O operador *modulo* `%` produz o resto da divisão inteira.
O sinal do resultado sempre acompanha o sinal do divisor.

```
In [39]: print(17 % 3)
```

```
2
```

```
In [40]: print(-17 % 3)
```

```
1
```

```
In [41]: print(17 % -3)
```

```
-1
```

Para ser preciso, sendo a e b dois números diferentes de zero, o resultado da operação % sempre torna verdadeira a seguinte afirmação:
$(b \cdot (a \ // \ b) + a \ \% \ b)$ é igual a a.

2.7.1 Ordem de precedência dos operadores aritméticos

Precedência	Operador	Associatividade	Operação
Mais alta	**	À direita	Exponenciação
	-	À esquerda	Negação
	*, /, //, %	À esquerda	Multiplicação, divisão, divisão inteira e resto
Mais baixa	+, -	À esquerda	Soma e subtração

Ao contrário dos demais operadores, ** associa à direita.

```
In [42]: print(2 ** 2 ** 3)
```

```
256
```

```
In [43]: print(4.0, (15.0 // 10.0))
```

```
4.0 1.0
```

```
In [44]: print(4.0 * (5.0 // 10.0))
```

```
0.0
```

2.8 Entrada

Quase qualquer programa precisa obter dados do usuário.
Uma das maneiras de se fazer isso em Python é usando o comando `input`.

`input` lê e retorna uma sequência de caracteres digitados no teclado.

```
In [45]: entrada = input()
         print(entrada, type(entrada))

12345
12345 <class 'str'>
```

Veja que o tipo de `entrada` é `<class 'str'>`, ou seja, `entrada` é uma `string`.
Portanto, se quisermos usar `entrada` como um valor numérico, será preciso convertê-la usando `int` ou `float`.

```
In [46]: entrada = int(input())
         print(entrada, type(entrada))

12345
12345 <class 'int'>
```

```
In [47]: entrada = float(input())
         print(entrada, type(entrada))

12.345
12.345 <class 'float'>
```

É possível ajudar o usuário, incluindo uma mensagem explicativa em `input`.

```
In [48]: nome = input('Por favor, digite seu nome: ')
         print('Olá,', nome, ', tudo bem?')

Por favor, digite seu nome: Arthur
Olá, Arthur , tudo bem?
```

Na mensagem de saída, há um espaço extra entre o nome e a vírgula que o segue. Você sabe explicar por que?

2.9 Saída

Já vimos que `print` aceita uma lista de argumentos, que separa os valores com espaços em branco e que muda de linha ao final do processamento.

```
In [49]: print(10,    'abc',     1.23)
         print('isto sai em outra linha')

10 abc 1.23
isto sai em outra linha
```

É possível mudar o comportamento de `print` ao final do processamento, acrescentando-se `end='x'` ao final da lista, onde `x` é o que se deseja que `print` use no lugar da mudança de linha.

```
In [50]: print(10, 'abc', 1.23, end=' ')
         print('isto sai na mesma linha')

10 abc 1.23 isto sai na mesma linha


In [51]: print('abc', end='')
         print('def', '<<< deve ter saído "abcdef" sem separação')

abcdef <<< deve ter saído "abcdef" sem separação
```

Agora já podemos melhorar a mensagem do exemplo da seção anterior...

```
In [52]: nome = input('Por favor, digite seu nome: ')
         print('Olá,', nome, end='')
         print(', tudo bem?')

Por favor, digite seu nome: José
Olá, José, tudo bem?
```

2.10 Exemplo de um primeiro programa completo

2.10.1 Quantos dias, horas, minutos e segundos há num dado número de segundos?

Como resolver este problema?

- Ler o número de segundos a serem convertidos
- Calcular o número de dias
- Calcular o número de horas restantes
- Calcular o número de minutos restantes
- Calcular o número de segundos restantes
- Mostrar o resultado

Vamos ler a entrada...

```
In [53]: segs = int(input('Qual o número de segundos que você quer converter? '))

Qual o número de segundos que você quer converter? 90061
```

Vamos calcular algumas constantes úteis...

```
In [54]: segs_no_minuto = 60
         segs_na_hora = 60 * segs_no_minuto
         segs_no_dia = 24 * segs_na_hora
```

Quantos dias em segs segundos?

```
In [55]: dias = segs // segs_no_dia
```

Quantas horas? Quantos minutos? Quantos segundos sobram?

```
In [56]: segs_restantes = segs % segs_no_dia
         horas = segs_restantes // segs_na_hora

In [57]: segs_restantes = segs_restantes % segs_na_hora
         minutos = segs_restantes // segs_no_minuto

In [58]: segs_restantes = segs_restantes % segs_no_minuto
```

Agora é só mostrar o resultado...

```
In [59]: print(segs, 'segundos =', dias, 'dia(s),', horas,'hora(s),',
                minutos, 'minuto(s) e', segs_restantes, 'segundo(s).')

90061 segundos = 1 dia(s), 1 hora(s), 1 minuto(s) e 1 segundo(s).
```