Aula 1 - Hello world, strings, obtendo ajuda

Karl Jan Clinckspoor

2 de julho de 2018

Sumário

1 Primeiro programa

É praticamente uma regra que qualquer curso introdutório de programação mostre o mesmo exemplo: Como mostrar os caracteres "Hello world!"na tela. Este curso não será diferente. Para executar o conteúdo de uma cela no Jupyter Notebook, aperte CTRL+Enter, ou SHIFT+Enter para executar e criar uma nova cela abaixo.

```
In [1]: print('Hello world!')
```

```
Hello world!
```

Somente em uma única linha já temos bastante do que conversar.

2 Funções

Em suma, o que ocorre é que esta linha chama a função *print*, e fornece a string 'Hello world!' para a função. Em retorno, a função imprime na tela justamente o que foi fornecido a ela.

O que seria uma função? Uma função é basicamente um pacote de código, que recebeu um nome específico para tornar seu uso mais fácil. Ao invés de uma série de comandos ter que ser escrita repetidas vezes, é comum encapsular os comandos numa função, e depois chamá-la diretamente.

2.1 Argumentos

Uma função pode ou não receber argumentos. Necessariamente, após o nome de uma função é necessário colocar parênteses. Caso a função não tenha argumentos, não haverá nada dentro dos parênteses. Algumas funções aceitam mais de um argumento, e esses devem ser separados por vírgulas. Em Jupyter Notebooks, se você apertar Shift+Tab dentro dos parênteses de uma função, aparecerá um balão de ajuda mostrando o que a função em especial pode chamar. Apertar Tab (com shift apertado) múltiplas vezes expande o conteúdo desse balão, até quatro vezes, onde uma janela aparece na parte debaixo da tela.

Neste caso, o único argumento é um conjunto de caracteres, ou string, que contém as letras 'Hello world!'. Em Python, strings são demarcadas por aspas, tanto simples (') quanto duplas ("), mas somente uma ou outra dentro de uma string. Há um caso especial, utilizando aspas triplas (''' ou """), que é utilizado para linhas mais compridas de texto. Python é uma linguagem onde a manipulação de strings é bastante fácil.

Veja os exemplos a seguir:

```
In [5]: print("Hello world!")
        print()
        print('Hello world!')

Hello world!

Hello world!
```

Veja que o uso de aspas simples e duplas não faz diferença. Eu costumo utilizar sempre aspas simples, e somente troco para aspas duplas quando é necessário colocar aspas simples dentro da string. Por exemplo, "Don't mention it".

Também, quando a função *print* foi chamada com nenhum argumento, ela imprimiu somente uma linha em branco.

```
In [9]: print('Hello world', '!')

Hello world !
```

Agora, note que foram fornecidos dois argumentos, separados por vírgula. O que o comando *print* fez foi juntá-los com um espaço e colocar na mesma linha.

2.2 *keyword arguments*

```
In [7]: print('Hello world', '!', sep='')

Hello world!
```

Note aqui que foi fornecido um terceiro argumento para a função. Esse tipo de argumento recebe o nome de *keyword argument* ou *kwarg*. Eles são, por definição, opcionais e possuem um valor padrão. Na função print, seu valor padrão é um espaço, ' ', mas aqui esse valor padrão foi substituído por nada, '', então não há um espaço entre 'world' e '!'.

```
In [8]: print('abc', end='')
        print('def')

abcdef
```

Este é um segundo exemplo de um *keyword argument*, desta vez trocando o caractere colocado no final de uma string. Geralmente, uma nova linha é colocada sempre após os comandos print.

3 Strings

Strings são conjuntos imutáveis (não se preocupe com o termo agora) de caracteres. Python 3 utiliza, por padrão, o conjunto de caracteres *Unicode*, ou *UTF-8*. Isso significa que falantes de línguas como Português, que possuem diacríticos (á, à, ã, ç), podem facilmente utilizar strings com os caracteres nativos. Como já escrito, strings são sempre envolvidas por aspas simples ou duplas.

3.1 *Escape sequences*

Um *escape sequence* é um conjunto de caracteres, geralmente precedidos por uma barra para a esquerda () que possuem um significado especial. É bastante útil que você se familiarize com esses caracteres, pois eles podem ajudá-lo muito na hora de consertar arquivos de texto.

```
\n: Cria uma nova linha
\t: Cria um tab
\r: Retorna ao início da linha
\\: Escreve a barra para a esquerda em si.
```

O mais utilizado é '
n', mas é bom saber que esses códigos especiais existem. Veja o exemplo:

```
In [10]: print('Hello\nworld!')

Hello
world!
```

No lugar do espaço, foi colocado '
n', e isso causou que a linha fosse dividida em duas. Isso é fundamental para quem está utilizando um script que escreve um arquivo de texto, por exemplo.

3.2 *raw* strings

Suponha que você deseja imprimir justamente a string 'Hello
nworld!', sem separação de texto. Isso pode ser feito utilizando uma *raw string*, que é criada simplesmente colocando um *r* antes das aspas.

```
In [13]: print(r'Hello\nworld!')
         print(r'\abc')
```

```
Hello\nworld!
\abc
```

Caso você deseje escrever somente uma barra simples, é necessário fazer algo um pouco estranho: colocar duas barras. A primeira barra fala "considere o próximo caracter como sendo especial"e a segunda barra fala "o caractere especial é uma barra mesmo, então use-a". Com *raw strings*, pode ser utilizada somente uma barra. Isso será útil bastante no futuro, quando fizermos plots utilizando codigos matemáticos do estilo LaTeX para legendas, etc.

```
In [19]: print('ABC\\DEF')
         print(r'ABC\DEF')

ABC\DEF
ABC\DEF
```

3.3 Métodos de strings

Tudo possui um *tipo* em Python. E todo tipo possui um conjunto de *métodos* e *propriedades* associadas a ele. É possível obter informações sobre um tipo utilizando o comando **dir**.

```
In [20]: dir('Hello')

Out[20]: ['__add__',
          '__class__',
          '__contains__',
          '__delattr__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getitem__',
          '__getnewargs__',
          '__gt__',
          '__hash__',
          '__init__',
          '__init_subclass__',
          '__iter__',
          '__le__',
          '__len__',
          '__lt__',
          '__mod__',
          '__mul__',
          '__ne__',
          '__new__',
          '__reduce__',
```

```
'__reduce_ex__',
'__repr__',
'__rmod__',
'__rmul__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'capitalize',
'casefold',
'center',
'count',
'encode',
'endswith',
'expandtabs',
'find',
'format',
'format_map',
'index',
'isalnum',
'isalpha',
'isdecimal',
'isdigit',
'isidentifier',
'islower',
'isnumeric',
'isprintable',
'isspace',
'istitle',
'isupper',
'join',
'ljust',
'lower',
'lstrip',
'maketrans',
'partition',
'replace',
'rfind',
'rindex',
'rjust',
'rpartition',
'rsplit',
'rstrip',
'split',
'splitlines',
'startswith',
'strip',
'swapcase',
```

```
 'title',
 'translate',
 'upper',
 'zfill']
```

O resultado desse comando é uma lista enorme de tudo que está associado a strings. Para acessar um método, utiliza-se a notação de ponto. Por exemplo:

```
In [21]: '1'.isdecimal()

Out[21]: True
```

A função *isdecimal* retorna se a string possui números. Aqui, ela retornou **True**, ou seja, '1' contém somente números.

4 Obtendo ajuda

Para obter mais informações sobre uma função, é possível chamar a documentação utilizando o comando **help** ou, em ambientes IPython/Jupyter Notebook, colocando um ponto de interrogação antes do comando.

```
In [23]: help('1'.isdecimal)

Help on built-in function isdecimal:

isdecimal(...) method of builtins.str instance
    S.isdecimal() -> bool

    Return True if there are only decimal characters in S,
    False otherwise.



In [27]: ?print
```

A medida que você explora novos tipos, tenha em mente esses comandos. É uma maneira rápida e fácil de obter ajuda especificamente sobre uma função que você deseja utilizar.

Alguns comandos úteis para strings:

- **endswith**
- **startswith**
- **split**
- **lower**
- **rstrip**
- **replace**

Exemplos:

```
In [28]: print('Hello world!'.endswith('!'))
         print('Hello world!'.startswith('a'))
         print('Hello world!'.split(' '))
         print('Hello world!'.lower())
         print('Hello world!      '.rstrip())
         print('Hello world!'.replace('l','r'))

True
False
['Hello', 'world!']
hello world!
Hello world!
Herro worrd!
```

Note aqui que é possível chamar funções dentro de chamadas de outras funções. Isso pode parecer complicado de acompanhar de início, mas com o tempo você irá se acostumar e até irá preferir essa maneira à maneira mais longa.

Essas ferramentas de manipulação de strings são bastante úteis para quem precisa manipular arquivos de texto. Como muitos dados científicos acabam vindo de arquivos de texto, é útil conhecer um pouco de funções de string.