Introdução à programação Python

Python Notebooks[1]

Uma forma bastante profícua para interação com scripts Python é através dos notebooks. O programa IPython fornece uma interface interativa para o Python, de forma que outros programas (IDEs) possam incorporar os resultados das chamadas, sejam na forma numérica, de texto ou de gráfico, apresentando-os de modo elegante.

Um desses programas é o Jupyter, que fornece uma interface web para a entrada de comandos e acessar os resultados. Além do mais, Jupyter entende a formatação de texto markdown, permitindo a criação de apostilas e relatórios de um modo elegante e interativo. Um resumo das regras de formatação do markdown pode ser encontrada em:

- http://svmiller.com/blog/2016/02/svm-r-markdown-manuscript/ (http://svmiller.com/blog/2016/02/svm-r-markdown-manuscript/)
- https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet)

Para iniciar o Jupyter, usamos a linha de comando:

- `jupyter notebook`

Na janela do navegador que foi aberta, você pode criar um novo notebook ou navegar entre as pastas até encontrar o notebook que quer abrir

---

[1]: Se quiser editar a célula, basta clicar duas vezes

Estruturas de dados fundamentais

Em Python podemos armazenar conjuntos de dados em *listas*, *tuplas* e *dicionários*.

Listas e Tuplas

Uma lista é uma sequência imutável de valores separados por vírgulas dentro de colchetes []

In [1]:

```
lista1 = [1, 2.0, False]
outra_lista = ['3', 2.0, 1]
lista_simples = ['A']
```

- Os valores podem ser de tipos distintos, incluído ainda outras estruturas complexas.
- Para saber o comprimento da lista: `len(lista)`

Acessando elementos singulares de uma tupla através de índices inteiros:

In [9]:

```
dois_ponto_zero = outra_lista[1] # começa em 0
um = outra_lista[-1] # conta a partir do último
print(dois_ponto_zero)
print(um)
```

```
2.0
1
```

Acessando sub-listas de elementos a partir de slices

- Notação: `lista[start : stop : step]`
  - `start`: índice de início (inclusive); 0 por omissão
  - `stop`: índice de parada (exclusive); `len(lista)` por omissão
  - `step`: opcional, incremento do índice
- Exemplos:

In [3]:

```
um_dois = lista1[0:2] # tomas os índices 0 e 1
lista_reversa = lista1[::-1] # inverte a lista
um_tres = outra_lista[::2] # toma só os índices pares
print("um_dois: {}".format(um_dois))
print("lista_reversa: {}".format(lista_reversa))
print("um_tres: {}".format(um_tres))
```

```
um_dois: [1, 2.0]
lista_reversa: [False, 2.0, 1]
um_tres: ['3', 1]
```

Além da lista, a tupla também é uma sequência de valores separados por vírgulas, porém não precisam de delimitadores.

- Opcionalmente, os elementos ficam dentro de parênteses
- Exemplos:

In [4]:

```
tupla1 = 1, 2.0, False
outra_tupla = ('3', 2.0, 1)
tupla_simples = ('A',) # necessario ter a vírgula no final
```

Para saber o comprimento da tupla: `len(tupla)`

Os elementos da tupla usam a mesma notação de índice e slices que a lista

- A diferença entre tuplas e listas é que as primeiras são imutáveis

Dicionários

Dicionários são sequências de itens rotulados que armazenam pares de chave-valor

- Cada chave é separada de seu valor por dois pontos
- Os itens são separados por vírgulas
- O dicionário é delimitado por chaves
- Um dicionário vazio é definido por `{}`.

As chaves de um dicionário são exclusivas, enquanto os valores podem não ser. Os valores de um dicionário podem ser de qualquer tipo, mas as chaves devem ser de um tipo de dados imutável, como strings, números ou tuplas

- Exemplos:

In [5]:

```
dic1 = {'um': 1, 2.0: [2], False: None}
dic2 = {(0,0):'a', (0,1):'b', (1,0):'c'}
```

Para acessar os elementos de um dicionário usamos a chave no lugar do índice:

In [6]:

```
print(dic1['um'])
print(dic2[0, 1])
```

```
1
b
```

Estruturas de bloco

Um bloco em Python é um conjunto de linhas de código dentro da seguite estrutura:

```
palavra-chave | complemento:
    linha 1
    linha 2
    ...
    última linha
```

A delimitação do código dentro do bloco se dá pela indentação, e não por caracteres delimitadores. Os caracteres usados na indentação das linhas devem ser sempre os mesmos, seja o caractere de tabulação, seja um certo número de caracteres de espaço.

Funções

Funções em Python são definidas de acordo com a seguinte sintaxe:

```
def nome_funcao(arg0, arg1,..., argopt0=val0, argopt1=val1,...):
    <Corpo da função>
    return valor_retorno
```

Exemplo:

```
def soma2(a, b=2):
    return a + b
```

A definição de função, bem como as demais definições em blocos, é delimitada através da indentação das linhas de código.

- Os argumentos `arg0`, `arg1`, ... são obrigatórios
- Os argumentos `argopt0`, `argopt1`, ... são opcionais

A invocação de uma função é semelhante a outras linguagens, exceto que os argumentos podem ser identificados por sua posição ou pelo identificador do argumento. Por exemplo:

```
var_retorno = nome_funcao(val0, arg1=val1, argopt4=val2)
```

Estruturas de controle

Laço de contagem.

- Para executar um código iterando sobre os elementos de uma sequência (listas, tuplas, etc.):

```
for i in seq:
  <Código a ser executado>
```

- Para criar uma sequência de inteiros, usamos uma das versões da função range:
  - `range(stop)`: cria uma sequência de `0` a `stop - 1`,
  - `range(start, stop)`: cria uma sequência iniciando em `start` e terminando em `stop - 1`,
  - `range(start, stop, step)`: cria uma sequência de `start` (inclusive) a `stop` (exclusive), com passo de incremento de `step`
- Exemplos:

In [12]:

```
for i in range(0, 20, 2):
    print(i) # imprime os números pares

soma = 0.0
for n in [2.0,3.5,-1,4,5]:
    soma += n #soma os valores da lista
print("Soma = {}".format(soma))
```

```
0
2
4
6
8
10
12
14
16
18
Soma = 13.5
```

Estrutura de desvio condicional

```
if <condição>:
    <Código a ser executado>
    elif <outra-condição>: #opcional, quantas vezes for necessario
    <Código a ser executado>
    else <condição>: #opcional
<Código a ser executado>
```

- Exemplo:

In [13]:

```
n = int(input('Entre com um número natural: '))
if n % 2 == 0:
    print("O número {} é par".format(n))
else:
    print("O número {} é ímpar".format(n))
```

```
Entre com um número natural: 20
O número 20 é par
```

- Para executar um código com condição de parada:

```
while <condição>:
  <Código a ser executado>
```