Python para Ciência de Dados

Luiz Alberto

Ciência da Computação

May 9, 2019

Tipos

- String
  - "Data Science", "Programming", "Python"
- Inteiros
  - -11, 7, 500, 700, 0, -80
- Ponto flutuante
  - 1.5, 0.5679, 2.909, -3.4560

Strings (1)

- Um **string** é uma cadeia de caracteres envolvida por aspas simples ou duplas
  - "Data Science", 'Programming', "Python"
- Mudando para letras maiúsculas e minúsculas

```
1       nome = "ada Lovelace"
2       print(nome.title())
3       Ada Lovelace
4       print(nome.upper())
5       ADA LOVELACE
6       print(nome.lower())
7       ada lovelace
8
```

Strings (2)

- Combinando ou concatenando strings

```
1       primeiro_nome = "ada"
2       segundo_nome = "lovelace"
3       nome_completo = primero_nome + " " + segundo_nome
4       mensagem = "Ola  , " + nome_completo.title() + "!"
5       print(mensagem)
6       Boa tarde, Ada Lovelace!
7
```

- Acrescentando espaços com tabulações ou quebras de linhas

```
1       print("\tAda Lovelace")
2         Ada Lovelace
3       print("Ada\nLovelace")
4         Ada
5         Lovelace
6
```

Strings (3)

- Removendo espaços em branco

```
1       nome = " Ada Lovelace "
2       print("["+nome.rstrip()+"]")
3       [ Ada Lovelace]
4       print("["+nome.rstrip()+"]")
5       [Ada Lovelace ]
6       print("["+nome.strip()+"]")
7       [Ada Lovelace]
8
```

Inteiros

- Python trata números de várias maneiras diferentes

```
>>> 2 + 3
5
>>> 2 * 3
6
>>> 3 / 2
1.5
>>> 3 ** 2
9
>>> 10 ** 6
1000000
>>> 2 + 3 * 4
14
>>> (2 + 3) * 4
20

```

Pontos flutuantes

- Python chama qualquer número com um ponto decimal de *número de ponto flutuante* (float)

```
>>> 0.1 + 0.1
0.2
>>> 0.2 + 0.2
0.4
>>> 2 * 0.1
0.2
>>> 2 * 0.2
0.4

```

Convertendo números para string

```
>>> mensagem = "Dia " + str(11) + "de maio."
>>> print(mensagem)
Dia 11 de maio.

```

Comentários

- Um comentário permite escrever notas em seus programas em linguagem natural.

```
1       # o que a pessoas acham sobre o Python.
2       print('Python is cool!')
3
```

Zen do Python

1. Bonito é melhor do que feio
2. Simples é melhor que complexo
3. Complexo é melhor que complicado
4. Legibilidade conta
5. Deve haver -e, de preferência, apenas uma maneira óbvia de fazer algo
6. Agora é melhor do que nunca

```
1       >>> import this
2       The Zen of Python, by Tim Peters
3
4       Beautiful is better than ugly
5
```

Introdução às Listas

- Coleção de itens em uma ordem particular
- Itens podem ser letras, dígitos e etc.
  - não precisam estar relacionados de nenhum modo em particular
- Em Python, colchetes([]) indicam uma lista, e elementos individuais da lista são separados por vírgula
- A posição dos índices de uma lista começa em 0 e não em 1

Acessando elementos da lista (1)

- Escreva o nome da lista seguido do índice do item entre colchetes

```
>>> mestres = ['yoda', 'qui-gon', 'kenobi'
    , 'luke']
>>> print(mestres[0])
yoda
>>> print(mestres[0].title())
Yoda

```

- Python tem uma sintaxe especial para acessar o último elemento de uma lista

Acessando elementos da lista (2)

```
>>> print(mestres[-1].title())
Luke

```

Acrescentando elementos no final da lista (1)

```
>>> mestres = ['yoda', 'qui-gon', 'kenobi'
    , 'luke']
>>> mestres.append('windu')
>>> print(mestres)
['yoda', 'qui-gon', 'kenobi', 'luke', 'windu']
```

- O método **append()** acrescenta o elemento 'windu' no final da lista

Acrescentando elementos no final da lista (2)

- O método **append()** facilita a criação de listas dinamicamente

```
1       casas = []
2       casas.append('starks')
3       casas.append('greyjoy')
4       casas.append('tyrell')
5       casas.append('lannister')
6       print(casas)
7       ['starks', 'greyjoy', 'tyrell', 'lannister']
8
```

Inserindo elementos em uma lista

- O método **insert** permite adicionar um elemento em qualquer posição da lista

```
1       mestres = ['yoda', 'qui-gon', 'kenobi', 'luke']
2       mestres.insert(0, 'vader')
3       print(mestres)
4       ['vader', 'yoda', 'qui-gon', 'kenobi', 'luke']
5
```

Removendo elementos da com **del**

- O comando **del** remove item da lista

```
1       mestres = ['yoda', 'qui-gon', 'kenobi', 'luke']
2       del(mestres[0])
3       print(mestres)
4       ['qui-gon', 'kenobi', 'luke']
5
```

- Não é possível reutilizar o elemento removido

Removendo elementos da lista com **pop**

- O método **pop** remove o último item da lista por padrão, sendo que elemento removido pode ser reutilizado

```
1 mestres    = ['yoda', 'qui-gon', 'kenobi', 'luke']
2 mais_novo = mestres.pop()
3 print(mais_novo.title() + ' e o Mestre Jedi mais novo.')
4 # output: Luke e o Mestre Jedi mais novo.
```

- De fato, o método **pop** pode ser utilizado para remover qualquer elemento da lista, basta passar o índice do elemento

```
1 mestres    = ['yoda', 'qui-gon', 'kenobi', 'luke']
2 mais_velho = mestres.pop(0)
3 print(mais_velho.title() + ' e o Mestre Jedi mais velho.')
4 # output: Yoda e o Mestre Jedi mais velho.
```

Ordenando a lista com **sort**

- O método **sort** altera a forma da lista permanentemente

```
mestres    = ['yoda', 'qui-gon', 'kenobi', 'luke']
mestres.sort()
print(mestres)
# output: ['kenobi', 'luke', 'qui-gon', 'yoda']
```

- O método **sort** permite ordenar a lista em ordem alfabética inversa

```
mestres    = ['yoda', 'qui-gon', 'kenobi', 'luke']
mestres.sort(reverse=True)
print(mestres)
# output: ['yoda', 'qui-gon', 'luke', 'kenobi']
```

Ordenando a lista com **sorted**

- O método **sorted** mantém a forma original da lista

```
1 mestres    = ['yoda', 'qui-gon', 'kenobi', 'luke']
2 print(sorted(mestres))
3 # output: ['luke', 'kenobi', 'qui-gon', 'yoda']
```

Exibindo uma lista em ordem inversa com **reverse**

- O método **reverse** mantém a forma original da lista

```
1 mestres   = ['yoda', 'qui-gon', 'kenobi', 'luke']
2 mestres.reverse()
3 print(mestres)
4 # output: ['luke', 'kenobi', 'qui-gon', 'yoda']
```

Percorrendo uma lista com um laço

- Um laço **for** permite que se percorra uma lista de início até o final

```
1 mestres = ['yoda', 'qui-gon', 'kenobi', 'luke']
2 for mestre in mestres:
3   print(mestre)
4 # output: yoda
5 #         qui-gon
6 #         kenobi
7 #         luke
```

- a linha 2 deverá estar **indentada** para ser executada pelo laço for

Criando listas numéricas (1)

- A função **range** permite a geração de uma série de números

```
1 numeros = list(range(1, 6))
2 print(numeros)
3 # output: [1, 2, 3, 4, 5]
```

- A função range **NÃO** exibe o limite superior do intervalo.
- Um salto pode ser fornecido para a função **range** para que a função ignore alguns números no intervalo

```
1 numeros = list(range(2, 11, 2))
2 print(numeros)
3 # output: [2, 4, 6, 8, 10]
```

Criando listas numéricas (2)

- Outro Exemplo: Imprimindo os 10 primeiros quadrados perfeitos

```
1 quadrados = []
2 for n in range(1, 11):
3   quadrados.append(n ** 2)
4 print(quadrados)
5 # output: [2, 4, 6, 8, 10]
```

Estatísticas simples com lista de números (1)

- **min()**, **max()** e **sum()** são funções específicas para listas de números:

```
1 digitos = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
2 print(min(digitos))
3 # output: 0
4 print(max(digitos))
5 # output: 9
6 print(sum(digitos))
7 # output: 45
```

List comprehensions

- Combina o laço **for** e a criação de novos elementos em uma lista, e concatena cada novo elemento automaticamente

```
1 quadrados = [n ** 2 for n in range(1, 11)]
2 print(quadrados)
```

Fatiando uma lista (1)

- É necessário especificar o índice do primeiro e do último elemento desejado

```
1 mestres = ['yoda', 'qui-gon', 'kenobi', 'luke']
2 print(mestres[0:3])
3 # ['yoda', 'qui-gon', 'kenobi']
```

- O intervalo 0:3 faz com os elementos 0, 1 e 2 sejam impressos
- Se o primeiro índice de uma fatia for omitido, Python comecará automaticamente do inicio da lista

```
1 mestres = ['yoda', 'qui-gon', 'kenobi', 'luke']
2 print(mestres[:3])
3 # ['yoda', 'qui-gon', 'kenobi']
```

Fatiando uma lista (2)

- Todos os elementos a partir de qualquer posição podem ser apresentados, até mesmo a partir do final

```
1 mestres = ['yoda', 'qui-gon', 'kenobi', 'luke']
2 print(mestres[-3:])
3 # ['qui-gon', 'kenobi', 'luke']
```

Percorrendo uma fatia com um laço

- Pode-se utiliza um laço **for** para percorrer os elementos de uma fatia:

```
1 mestres = ['yoda', 'qui-gon', 'kenobi', 'luke']
2 for mestre in mestres[:3]:
3   print(mestre.title())
4 # output: Yoda
5 #         Qui-gon
6 #         Knobi
```

Copiando uma lista

- Pode-se criar uma fatia que inclua a lista original inteira omitindo o primeiro e o segundo índice ([:])

```
1 mestres = ['yoda', 'qui-gon', 'kenobi', 'luke']
2 copia_dos_mestres = mestres[:]
3 print(mestres)
4 # output: ['yoda', 'qui-gon', 'kenobi', 'luke']
5 print(copia_dos_mestres)
6 # output: ['yoda', 'qui-gon', 'kenobi', 'luke']
```

- Isto não funciona!

```
1 mestres = copia_dos_mestres
```

Tuplas (1)

- Tuplas são listas *imutáveis* em Python. Exemplo:

```
1 dimensoes = (200, 50)
2 print(dimensoes[0])
3 print(dimensoes[1])
4 # output: 200
5 #         50
```

- Erro ao tentar alterar o conteúdo da uma tupla

```
1 dimensoes = (200, 50)
2 dimensoes[0] = 250
3 # outuput:  ...
4 #   TypeError: 'tuple' object does not support item
      assigment
```

Tuplas (2)

- Percorrendo todos os valores de uma tupla com um laço

```
1 dimensoes = (200, 50)
2 for dimensao in dimensoes:
3   print(dimensao)
```

Instrução condicional IF (1)

- Testes condicionais

```
1  carros = ['audi', 'bmw', 'subaru', 'toyota']
2  carro  = carros[1]
3  ano  = 2014
4  print(carro == 'bmw')     # igualdade
5  print(carro != 'bmw')     # diferenca
6  print(ano <   2014)       # menor
7  print(ano <=  2014)       # menor ou igual
8  print(ano >   2014)       # maior
9  print(ano >=  2014)       # maior ou igual
10 print(ano >= 2014 and ano <= 2018)  # conectivo E
11 print(ano == 2014 or ano == 21)     # conectivo OR
12 print('bmw' in carros)       # checa se esta na lista
13 print('bmw' not in carros)  # checa se nao esta na lista
14 # output:  True
15 #          False
16 #          False
17 #          True
```

Instrução condicional IF (2)

```
18 #           False
19 #           True
20 #           True
21 #           True
22 #           True
23 #           False
```

- Exemplo 1:

```
1  carros = ['audi', 'bmw', 'subaru', 'toyota']
2  for carro in carros:
3    if carro == 'bmw':
4      print(carro.upper())
5    else:
6      print(carro.title())
7  # output: Audi
8  #         BMW
9  #         Subaru
10 #         Toyota
```

Instrução condicional IF (3)

- Exemplo 2:

```
1  ano = 1998
2  if ano < 1980:
3    imposto = 0
4  elif ano < 1990:
5    imposto = 5
6  elif ano < 2000:
7    imposto = 10
8  elif ano < 2010:
9    imposto = 15
10 else:
11   imposto = 20
12 print("O valor do imposto e " + str(imposto) + ".")
13 # output: O valor do imposto e 10.
```

Instrução IF com listas (1)

- Verificando se uma lista está vazia

```
1  carros = []
2  if carros:
3    for carro in carros:
4      if carro == 'bmw':
5        print(carro.upper())
6      else:
7        print(carro.title())
8  else:
9    print("O estoque de carros esta vazio!")
10
11 # output: O estoque de carros esta vazio!
```

Instrução IF com listas (2)

- Utilizando diversas listas:

```
1  carros  = ['audi', 'bmw', 'subaru', 'toyota']
2  pedidos = ['bmw', 'ferrari']
3  for pedido in pedidos:
4    if pedido in carros:
5      print("Requisitanto " + pedido.title() + ".")
6    else:
7      print("Infelizmente, nao temos " + pedido.upper() + "!")
8
9  print("\nO processamento de pedidos foi finalizado!")
10
11 # output: Requisitando Bmw.
12 #         Infelizmente, nao temos FERRARI
13 #
14 #   O processamento de pedidos foi finalizadoS
```

Dicionários

- Dicionários permitem conectar informações **relacionadas** e modelar uma diversidade de objetos do mundo real
  - podemos armazenar, por exemplo, o nome, ano, cor e fabricante de uma caro em uma única estrutura, ao invés de quatro estruturas separadas
- Um dicionário é uma coleção de pares *chave-valor*. Cada chave é conectada a um valor:

```
1 populacoes = {'afeganistao':30.55, 'albania':2.77
2   , 'algeria': 39.21}
3 print(populacoes)
4 # output:
5 # {'afeganistao':30.55, 'albania':2.77, 'algeria': 39.21}
```

Acessando valores do dicionário

```
1 populacoes = {'afeganistao':30.55, 'albania':2.77
2   , 'algeria': 39.21}
3 print("A populacao da Algeria e de "
4   + str(populacoes['algeria'])
5   + " milhoes de habitantes.")
6 # output:
7 # A populacao da Algeria e de 39.21 milhoes de habitantes.
```

Começando um dicionário vazio

```
1 populacoes = {}
2 populacoes['afeganistao'] = 30.55
3 populacoes['albania'] = 2.77
4 populacoes['algeria'] = 39.21
5 print(populacoes)
6 # output:
7 # {'afeganistao':30.55, 'albania':2.77, 'algeria': 39.21}
```

Um exemplo de utilização de dicionário

```
1  figura = {'x': 0, 'y': 25, 'velocidade': 'media'}
2  print("x original: " + str(figura['x']))
3
4  # move a figura para direita
5  # determina a distancia que a figura deve se descolocar
6  # de acordo com sua velocidade atual.
7  if figura['velocidade'] == 'baixa':
8    incremento_x = 1
9  elif figura['velocidade'] == 'media':
10   incremento_x = 2
11 else:
12   incremento_x = 3
13
14 # a nova posicao e a posicao antiga somada ao incremento
15 figura['x'] = figura['x'] + incremento_x
16
17 print("x incrementado: " + str(figura['x']))
```

Percorrendo todos os pares chave-valor

```
1  populacoes = {'afeganistao':30.55, 'albania':2.77
2    , 'algeria': 39.21}
3  for key, value in populacoes.items():
4    print("Chave: " + key)
5    print("Valor: " + str(value))
6
7  # output:
8  # Chave: afeganistao
9  # Valor: 30.55
10 # Chave: albania
11 # Valor: 2.77
12 # Chave: algeria
13 # Valor: 39.21
```

Percorrendo todos as chaves

```
1 populacoes = {'afeganistao':30.55, 'albania':2.77
2   , 'algeria': 39.21}
3 for key in populacoes.keys():
4   print("Chave: " + key)
5
6 # output:
7 # Chave: afeganistao
8 # Chave: albania
9 # Chave: algeria
```

Percorrendo todos as chaves em ordem

```
1 populacoes = {'albania':2.77
2   , 'algeria': 39.21, 'afeganistao':30.55}
3 for key in sorted(populacoes.keys()):
4   print("Chave: " + key)
5
6 # output:
7 # Chave: afeganistao
8 # Chave: albania
9 # Chave: algeria
```

Percorrendo todos os valores

```
1 populacoes = {'albania':2.77
2   , 'algeria': 39.21, 'afeganistao':30.55}
3 for populacao in sorted(populacoes.values()):
4   print("Populacao: " + str(populacao))
5
6 # output:
7 # Populacao: 2.77
8 # Populacao: 30.55
9 # Populacao: 39.21
```

Uma lista de dicionarios

```
1  america  = {'brasil': 207, 'argentina': 44.293
2    , 'uruguai': 3.36}
3  europa   = {'portugal':10.83, 'espanha': 49.95
4    , 'italia':62.13}
5  asia     = {'filipinas': 104.25, 'malasia': 31.38
6    ,  'tailandia': 68.41}
7  continentes = [america, europa, asia]
8
9  for continente in continentes:
10   print(continentes)
11
12 # output:
13 # [{'brasil': 207, 'argentina': 44.293, 'uruguai': 3.36}]
14 # [{'portugal':10.83, 'espanha': 49.95, 'italia':62.13}]
15 # [{'filipinas': 104.25, 'malasia': 31.38,  'tailandia':
     68.41}]
```