INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Introdução à Python

Altino Dantas

PARTE I

INSTITUTO DE INFORMÁTICA
UFG

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Objetivos

- Conhecer as principais estruturas de dados do Python
- Realizar as principais operações em cada estrutura

Listas

- Provavelmente a principal e mais utilizada estrutura de dados em Python;
- Espécie de *Array* com mais funcionalidades;

```
lista_vazia  = []
lista_ints = [1,2,3,5]
lista_heterogenea = [20, "dias", True]

tamanho_da_lista = len(lista_ints)
suma_da_lista = sum(lista_ints)
```

Listas

- Você pode configurar o enésimo elemento de uma lista com colchetes:

```
x = range(10)   # é a lista [0, 1, 2, ... 9]
zero = x[0]     # é igual a zero porque as listas são indexadas em 0
um = x[1]       # é igual a 1
nove = x[-1]    # é igual a 9, forma pythonica para último elemento
oito = x[-2]    # é igual a 8
x[0] = -1       # agora x é [-1, 1, 2, ... 9]
```

- Você pode usar os colchetes para repartir listas:

```
primeiros_tres = x[:3]           # [-1, 1, 2]
apos_o_terceiro = x[3:]          # [3, 4, 5, 6, 7, 8, 9]
um_ou_quatro = x[1:5]            # [1, 2, 3, 4]
ultimos_tres = x[-3:]            # [7,8,9]
sem_primeiro_e_ultimo = x[1:-1] # [1, 2, 3, 4, 5, 6, 7, 8]
copia_de_x = x[:]                # [-1, 1, 2, 3, 4, 5, 6, 7, 8]
```

Listas

- Busca em lista com operador in

```
1 in [1,2]      # verdadeiro
0 in [1,2]      # falso
```

- Concatenação de listas:

```
x = [1,2,3]
x.extend([4,6])  # [1, 2, 3, 4, 6]

x = [1,2,3]
y = x + [4,6]    # y é [1, 2, 3, 4, 6], mas x não foi alterada

x = [1,2,3]
x.append(4)      # x agora é [1, 2, 3, 4]
```

Codificando

- Crie uma lista vazia;
- Preencha a lista como os números pares de 0 a 100;
  - Use o operador % para descobrir os números pares e um estrutura de repetição para colocar cada número na lista;
  - Use o comando ***lista.append(i)*** para adicionar o número à lista;
- Por fim, imprima a lista com ***print(lista).***

List Comprehension

- No Python temos uma construção chamada List Comprehension que gera uma lista de forma **clara e concisa**;

- Como conseguimos colocar qualquer objeto dentro de uma lista, também é possível colocar uma **expressão** e obter um resultado a partir disto;

- A *List Comprehension* sempre irá retornar uma lista como o resultado.

List Comprehension

- O List Comprehension irá gravar o resultado da expressão que foi aplicada dentro da lista.
- É uma maneira simples e legível de criar expressões e gravar seu resultado, ou seja, conseguimos utilizar;
- um for e/ou if dentro de uma lista e o seu resultado poderá ser utilizado na aplicação posteriormente;
- Utilizar List Comprehension é também uma forma de deixar o seu código mais limpo e organizado;
- https://docs.python.org/3/tutorial/datastructures.html

List Comprehension

Sintaxe

- Para conseguirmos chegar ao resultado, podemos utilizar da seguinte forma:

```
[<expressão> for <item> in <sequencia>]
```

Ainda podemos combinar o uso do for com if:

```
[<expressão> for <item> in <sequencia> if <condição>]
```

List Comprehension

Utilizando List Comprehension

Imagine que temos gerar uma lista de n números sequenciais:

```
lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9 , 10]
print (num)
```

Podemos transformar isto em:

```
lista = [ item for item in range(11) ]
print (num)
```

List Comprehension

Utilizando List Comprehension

Imagine que temos uma lista e dela pretende-se copiar apenas os números negativos para outra lista :

```
lista = [0, -1, 2, -3, -4, 5, 6, 7, -8, 9 , 10]
Lista_neg = []
for (i in lista):
    if i < 0:
        lista_neg.append(i)
```

Podemos transformar isto em apenas uma linha:

```
Lista_neg = [ item for item in lista if item < 0]
```

Codificando

- Implemente a lista de números pares usando *list comprehension;*
- Veja a quantidade de linhas usadas antes e agora, diminuiu?!

Tuplas

- Similares às listas porém com imutáveis;
- Especifica-se com parênteses (ou nada) em vez de colchetes;

```
lista = [1,3]
tupla = (1,3)
outra_tupla = 4, 3
lista[1] = 20

try:
    tupla[1] = 3
except TypeError:
    print("Impossível modificar uma tupla")
```

```
Impossível modificar uma tupla
```

Tuplas

- Forma eficiente para retornar múltiplos valores numa função:

```
def soma_e_produto(x, y):
    return (x+y), (x*y)

sp = soma_e_produto(2,3)     # é igual a (5,6)
s,p = soma_e_produto(5,10)    # s é igual a 15 e p igual a 50
```

Dicionários

- Uma eficiente estrutura de dados do tipo chave valor;
- Permite recuperar rapidamente um valor relacionado a uma chave;
- Forma de definição:

```
dic_vazio = {}                                  # pythonic
dic_vazio_2 = dict()                            # menos pythonic
escores = {'joel': 80, 'marcos': 98}    # dicionário literal
```

- Recuperação de valores

```
nota_joel = escores['joel']                 # é igual a 80
```

Dicionários

- Pode ocorrer Exceções quando se tentar acessar uma chave inexistente:

```
try:
    notas_dias = escores['dias']
except KeyError:
    print ("Não existe nota para Dias")
```

```
Não existe nota para Dias
```

- É possível checar se existe uma determinada chave

```
joel_tem_nota = "joel" in escores;    # verdadeiro
dias_tem_notas = "dias" in escores;   # falso
```

Dicionários

- Capturando valores através do get(key):

```
nota_joel = escores.get('joel')              # é igual a 80
nota_dias = escores.get('dias')              # retorna "none"
nota_marcos = escores.get('marcos')          # é igua a 98
```

- Adicionando ou atualizando um dicionário:

```
escores['joel'] = 95                 # atualiza o escores existente
escores['maria'] = 100               # adiciona uma nova chave com o respectivo valor
qtd_estudantes = len(escores)        # é igual a 3

escores.update({'maria': 98})  # atualiza a nota de maria
escores.update({'dias': 81})   # insere dias e sua nota
qtd_estudantes = len(escores)  # é igual a 4
```

Dicionários

- Retornando algumas listas a partir de um dicionários:

```
lista_chaves = escores.keys()   # lista de chaves
lista_valores= escores.values() # lista de valores-chave
lista_itens = escores.items()   # lista de tuplas chave-valor
```

- Buscando nas listas e no dict diretamente:

```
"maria" in lista_chaves         # verdadeiro, mas lento por usar in de lista
"maria" in escores              # mais pythonic e mais rápido por usar in de dict
81 in lista_valores             # verdadeiro pois há 81 entre as notas
```

Exercício

Codificando

```
document = list(open("caminho\\alice.txt").read().lower().split())
palavras = {}
```

- Considerando o código acima, document possui uma lista das palavras do texto de “caminho\\alice.txt”, na ordem em que estão no texto;
- Apresente a contagem do total de palavras no texto;
- Apresente a contagem do total de palavras únicas no texto;
- Apresente a contagem da quantidade de ocorrências de cada palavra no texto.

Conjuntos (set)

- É uma coleção de elementos distintos;
- Consulta com in é mais eficiente do que em listas;
- Ideal para armazenar elementos únicos;

```
s = set()     # criar um conjunto
s.add(1)      # s agora é {1}
s.add('A')    # s agora é {1, 'A'}
s.add(3)      # s agora é {1, 'A', 3}
s.add(3)      # s continua é {1, 'A', 3}
len(s)        # tamanho de s é 3

y = 2 in s    # y é False
z = 3 in s    # z é True
```

Ordenação

- As listas em Python possuem dois meios de ordenação:
- Método lista.sort() da própria lista, que altera diretamente na lista;
- Método sorted(lista) que retorna uma lista ordenada;

```
x = [4,1,5,3]
y = sorted(x)      # y é [1, 3, 4, 5] e x não mudou
x.sort()           # x agora é [1, 3, 4, 5]
```

Ordenação

- Dicionários não são ordenáveis, mas é possível obter uma cópia ordenada através do método sorted();
- Método sorted() possui um parâmetro para a função de ordenação.

```
wc = sorted(palavras.items(), key=lambda chave_valor:chave_valor[1], reverse=True)
```

```
[('the', 66),
 ('she', 51),
 ('to', 50),
 ('and', 43),
 ('was', 42),
```

Exercício

Codificando

- Crie um conjunto para armazenar stop_words;
- Adicione a este conjunto as três palavras mais frequentes e as três menos frequentes;
- Crie um conjunto com todas as palavras do texto menos as presentes em stop_words;
- Crie uma com todas as palavras do texto menos as presentes em stop_words;

Obrigado

altinobasilio@inf.ufg.br

Dúvidas ou sugestões?