Unidade 3: Estruturas Básicas de Dados ¶

Objetivos de aprendizagem:

1. Reconhecer as características chave de cada estrutura. Utilizar corretamente cada estrutura quando apropriado e acessar os dados correspondentes armazenados na estrutura.

Consulte este Doc (https://docs.python.org/3/tutorial/datastructures.html) para obter informações adicionais sobre cada estrutura de dados.

3.1: Lista

Lista: uma estrutura de dados mutável que armazena elementos em um formato desordenado, como uma matriz.

In [ ]:
```
# Iniciar uma lista vazia
list1 = []
# OU
list1 = list()

# Iniciar uma lista com elementos
list2 = ['hello', 'hola', 'olá']

"""
Os elementos da lista NÃO precisam ser do mesmo tipo, mas isto é incomum.
Neste caso, cada lista poderia representar a série de informações sobre uma pessoa,
mas é preciso lembrar quais informações são armazenadas em cada index. ---> Existe um
a opção melhor para este propósito - dicionário.
"""
list3 = ["John", "male", 20, False]
```

In [ ]:
```
# Acess às informações armazenadas na lista por posição ("index")
# Nota: em CS, a primeira posição é SEMPRE 0
print("First element in list2 : "+ list2[0])
print("Second element in list2 : "+ list2[1])
```

```
First element in list2 : hello
Second element in list2 : hola
```

In [ ]:
```
# Inserir um novo elemento como um local específico, no index 1
list2.insert(1,'hallo')
list2[1]
```

Out[ ]:
```
'hallo'
```

In [ ]:
```
# Anexar um novo elemento no FIM da lista
list2.append('bye')
list2
```

Out[ ]:
```
['hello', 'hallo', 'hola', 'olá', 'bye']
```

In [ ]:
```
# Remover um elemento da lista especificando o elemento que você deseja remover
list2.remove('hello')
```

In [ ]:
```
# list2 após o 'hello' é REMOVIDA
list2
```

Out[ ]:
```
['hallo', 'hola', 'olá', 'bye']
```

In [ ]:
```
"""
Outra maneira de remover um elemento: pop()
pop() permite que você identifique uma posição
"""

list2.append("hello")

list2.pop()

list2
```

Out[ ]:
```
['hallo', 'hola', 'olá', 'bye']
```

In [ ]:
```
"""
As listas também podem ser ordenadas.
O método de ordenação depende de como a interface comparável é implementada para os o
bjetos da lista.

Neste caso da list2, sort() funciona ordenando caracteres individuais na string de ac
ordo com o código ASCII.
"""
list2.sort()
list2
```

Out[ ]:
```
['bye', 'hallo', 'hola', 'olá']
```

In [ ]:
```
"""
Como a lista é dinâmica, significa que o tamanho da lista aumenta ou diminui à medida
que inserimos ou removemos elementos,
poderíamos executar len() para encontrar o tamanho da lista em um determinado moment
o.
"""


# Como len() retorna int, para concatená-la a uma string, precisamos computá-la.
print("size of list1 = " + str(len(list1)))
print("size of list2 = " + str(len(list2)))
```

```
size of list1 = 0
size of list2 = 4
```

In [ ]:
```
# Imprimir itens na lista como string, separados por uma vírgula
",".join(list2)
```

Out[ ]:
```
'bye,hallo,hola,olá'
```

In [ ]:
```
# Você também pode ter uma lista de listas. Por exemplo:

lists = []
lists.append([1,2,3])
lists.append(['a','b','c'])

lists
```

Out[ ]:
```
[[1, 2, 3], ['a', 'b', 'c']]
```

In [ ]:
```
# Da mesma forma, você pode indexar as listas multidimensionais.

lists[1]
```

Out[ ]:
```
['a', 'b', 'c']
```

In [ ]:
```
lists[1][0]
```

Out[ ]:
```
'a'
```

3.2: Tupla

Tupla: "lista" imutável que não pode ser manipulada depois de criada.

Elas suportam todas as listas de operações, exceto aquelas que modificam a lista.

In [ ]:
```
# Inicializar uma tupla vazia
y = tuple()
y

# Criar uma nova tupla de elementos
x = (1,2,3)
```

In [ ]:
```
# ERROR: não pode ser adicionado a uma tupla
x.append(4)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-ce148ff4c21d> in <module>()
      1 # ERROR: não pode ser adicionado a uma tupla
----> 2 x.append(4)

NameError: name 'x' is not defined
```

In [ ]:
```
# Isto é OK porque está criando uma nova tupla com x e (4,5,6) adicionado no final
x + (4,5,6)
```

Out[ ]:
```
(1, 2, 3, 4, 5, 6)
```

In [ ]:
```
# x NÃO é modificado pela linha anterior
x
```

Out[ ]:
```
(1, 2, 3)
```

In [ ]:
```
# Criar uma nova tupla com x aparecendo duas vezes
x * 2
```

Out[ ]:
```
(1, 2, 3, 1, 2, 3)
```

In [ ]:
```
# Index dos elementos na tupla
x.index(3)
```

Out[ ]:
```
2
```

In [ ]:
```
# abreviação para
#  (a,b,c) = (1,2,3)

# Isto também atribui a = 1, b = 2, c = 3

a,b,c = 1,2,3
```

In [ ]:
```
a
```

Out[ ]:
```
1
```

In [ ]:
```
b
```

Out[ ]:
```
2
```

In [ ]:
```
c
```

Out[ ]:
```
3
```

In [ ]:
```
# Converter uma tupla em uma lista
x = (1,2,3,4)
list(x)
```

Out[ ]:
```
[1, 2, 3, 4]
```

In [ ]:
```
# Converter uma lista em uma tupla
x = [1,2,3,4]
tuple(x)
```

Out[ ]:
```
(1, 2, 3, 4)
```

In [ ]:
```
# Declarar uma nova tupla, nome "person"
person  = ('Jane','Doe', 21)

# "Pacote"/associar cada elemento da tupla com uma etiqueta. Observe a ordem das etiq
uetas.
first, last, age = person
```

In [ ]:
```
first
```

Out[ ]:
```
'Jane'
```

In [ ]:
```
last
```

Out[ ]:
```
'Doe'
```

In [ ]:
```
age
```

Out[ ]:
```
21
```

In [ ]:
```
# ERROR: x é uma tupla de 4 valores, mas está tentando descompactar SOMENTE 3 element
os.
# Incompatibilidade no tamanho da tupla
x = (1,2,3,4)

a,b,c = x
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-45-4acb77e9e93d> in <module>()
      3 x = (1,2,3,4)
      4
----> 5 a,b,c = x

ValueError: too many values to unpack (expected 3)
```

In [ ]:
```
# Isto é OK!
x = [1,2,3,4]
a,b,c,d = x
```

In [ ]:
```
a
```

Out[ ]:
```
1
```

In [ ]:
```
b
```

Out[ ]:
```
2
```

In [ ]:
```
c
```

Out[ ]:
```
3
```

In [ ]:
```
d
```

Out[ ]:
```
4
```

3.3: Conjunto

Conjunto: uma estrutura de dados mutável que armazena objetos não duplicados e imutáveis e ordena os elementos em ordem ascendente. Cada elemento do conjunto é único.

In [ ]:
```
# Inicializar um conjunto vazio
newSet = set()
newSet
```

Out[ ]:
```
set()
```

In [ ]:
```
# Um conjunto com elementos
ex1 = {1, 2, 2, 1, 1}

ex1
```

Out[ ]:
```
{1, 2}
```

In [ ]:
```
ex2 = {j for j in range(10)}
ex2
```

Out[ ]:
```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [ ]:
```
# 2 já existe no ex2. O que acontece se quisermos acrescentar o 2 novamente?
  # Nota: A implementação do conjunto NÃO definiu append(), então usaremos add().
    # add() irá inserir o novo elemento na posição correta com a ordenação do conjunt
o
ex2.add(2)
ex2.add(100)
ex2.add(50)
ex2
```

Out[ ]:
```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 50, 100}
```

In [ ]:
```
# objetos mutáveis não podem ser colocados em um conjunto
d_set = {[1,2,3]}

d_set
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-55-6b6994b2706f> in <module>()
      1 # mutable objects can't go in a set
----> 2 d_set = {[1,2,3]}
      3
      4 d_set

TypeError: unhashable type: 'list'
```

In [ ]:
```
# Converter uma lista em um conjunto
ages = [10, 5, 4, 5, 2, 1, 5]

set_of_ages = set(ages)

set_of_ages
```

Out[ ]:
```
{1, 2, 4, 5, 10}
```

In [ ]:
```
# Converter um conjunto em uma lista
list_of_ages = list(set_of_ages)

list_of_ages
```

Out[ ]:
```
[1, 2, 4, 5, 10]
```

In [ ]:
```
# Converter um conjunto em uma tupla
tuple_of_ages = tuple(list_of_ages)

tuple_of_ages
```

Out[ ]:
```
(1, 2, 4, 5, 10)
```

In [ ]:
```
# A ordem é irrelevante na comparação de conjuntos, uma vez que os elementos são orde
nados

{1,2,3} == {2,1,3}
```

Out[ ]:
```
True
```

3.4: Dicionário

Dicionário: uma estrutura de dados que armazena pares de valores chave nos quais as chaves DEVEM ser objetos imutáveis.

In [ ]:
```
# Inicializar um dicionário vazio
# O mesmo do conjunto, mas com:
dict = {}

# Declare um dicionário com pares de chaves/valores
dict2 = {'a': 5, 'b': 10, 'c': 100, 'd': 9.5}
```

In [ ]:
```
# Acessar dados em um dicionário com uma chave
dict2['b']
```

Out[ ]:
```
10
```

In [ ]:
```
# Atualização do valor de uma chave existente
dict2['b'] = 50

dict2['b']
```

Out[ ]:
```
50
```

In [ ]:
```
# O que acontece se quisermos ter acesso ao valor de uma chave inexistente? (por exemplo, 'z')


# Podemos esperar um ERROR porque esta chave não existe, portanto, não tem valor.
# Esta é uma suposição correta.
dict2['z']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-4-132d7852d7b7> in <module>()
      4 # Podemos esperar um ERROR porque esta chave não existe, portanto, não tem valor.
      5 # Esta é uma suposição correta.
----> 6 dict2['z']

KeyError: 'z'
```

In [ ]:
```
# # Mas se fizermos isso, será que isso ainda retornará um ERROR?
dict2['z'] = 999

dict2['z']
```

POR QUÊ?

O bloco de código anterior funciona porque "atualizamos" primeiro o valor da chave, atualizar = inserir (neste caso), e quando queremos acessar o valor desta chave, esta chave já existe no dicionário para o mapeamento atribuído.

In [ ]:
```
# Os valores no dicionário podem ser misturados
# Vejamos um exemplo com dict{}, um dicionário vazio iniciado acima.
# Primeiro, vamos inserir alguns pares de chaves/valores no programa.

dict["greeting"] = "hello message"
dict["alphabet"] = ['a', 'b', 'c', 'd', 'e']
dict["check-in"] = False
dict["phoneNumber"] = 8007782346

dict
```

In [ ]:
```
# Nota IMPORTANTE: a chave deve ser um objeto imutável (algo que não pode ser modificado)
# A string é imutável, porque você não poderia simplesmente apagar um caractere em uma string. Uma string é uma string, do jeito que ela é.

# Vemos acima que uma lista pode ser um valor no dicionário.
# O que acontece quando tentamos fazer dela uma chave?

# ERROR: tipo inalcançável de lista

# Porque poderíamos modificar a lista inserindo novos elementos, ordenando elementos, apagando elementos ou outras formas de modificá-la, mas ela NÃO PODE ser uma chave


dict[['a','b', 'c']] = [False, True, False]
```

In [ ]:
```
# Mas como a tupla é imutável, podemos substituir a lista por uma tupla
dict[('a','b', 'c')] = [False, True, False]

dict
```

In [ ]:
```
# Podemos também buscar todas as chaves
dict.keys()
```

In [ ]:
```
# Ou todos os valores
dict.values()
```

In [ ]:
```
# Os elementos de um dicionário também podem retornar como um par.
dict.items()
```

In [ ]:
```

```