Coleções em Python

Capacitação STEM 2021-2022

Março - 2022

Coleções em Python

- As coleções permitem armazenar múltiplos itens dentro de uma única unidade, que funciona como um container.
- Veremos três dentre as coleções mais utilizadas em Python:
  - Listas (OK)
  - Tuplas
  - Dicionários
  - Sets

Características

Característica	Lista	Tupla	Set	Dicionário
Ordenada	Sim	Sim	Não	Sim
Mutável	Sim	Não	Sim	Sim
Indexada	Sim	Sim	Não	Sim
Permite membros repetidos	Sim	Sim	Não	Não

Tuplas

- Tupla é uma estrutura de dados semelhante a lista. Porém, ela tem a característica de ser imutável, ou seja, após uma tupla ser criada, ela não pode ser alterada.
- Delimitada por parênteses na sua sintaxe e vírgula, ou apenas uma sequência de valores separados por vírgula
  - Apenas um valor entre parênteses não cria uma tupla

```
>>> vogais = ('a', 'e', 'i', 'o', 'u')
>>> vogais
('a', 'e', 'i', 'o', 'u')
>>> t = 10, 20
>>> t
(10, 20)
>>> valor = (123)
>>> valor
123
```

Tuplas

- Assim como é feito nas listas, podemos acessar um determinado valor na tupla pelo seu índice

```
>>> vogais
('a', 'e', 'i', 'o', 'u')
>>> vogais[0]
'a'
>>> vogais[4]
'u'
>>> vogais[5]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
IndexError: tuple index out of rangevogais = ('a', 'e', 'i', 'o', 'u')
```

Tuplas

- O fato da tupla ser imutável faz com que os seus elementos não possam ser alterados depois dela já criada.

```
>>> vogais
('a', 'e', 'i', 'o', 'u')
>>> vogais[0] = 'A'
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Tuplas

- As tuplas devem ser usadas em situações em que não haverá necessidade de adicionar, remover ou alterar elementos de um grupo de itens.
- Exemplos bons seriam os meses do ano, os dias da semana, as estações do ano, etc.
  - Nesses casos, não haverá mudança nesses itens (pelo menos isso é muito improvável).

Tuplas

- Também usadas em situações onde se precisa empacotar e desempacotar dados

```
>>> a, b, c = (10, 20, 30)
>>> a
10
>>> b
20
>>> c
30
```

```
>>> a, b = b, a
>>> a
20
>>> b
10
```

Tuplas - Vantagens

- Como as tuplas são imutáveis, a iteração sobre elas é mais rápida e, consequentemente, possuem um ganho de desempenho com relação às listas;
- Tuplas podem ser utilizadas como chave para um dicionário, já que seus elementos são imutáveis. Já com a lista, isso não é possível;
- Se for necessário armazenar dados que não serão alterados, utilize uma tupla. Isso garantirá que esses sejam protegidos de alterações posteriores.

Dicionários - Definição

- Os dicionários representam coleções de dados que contém na sua estrutura um conjunto de pares chave:valor
  - Cada chave individual tem um valor associado.
  - {key:value}
- Esse objeto representa a ideia de um mapa, que entendemos como uma coleção associativa desordenada.
- A associação nos dicionários é feita por meio de uma chave que faz referência a um valor.

Dicionários - Sintaxe

- A estrutura de um dicionário é delimitada por chaves, entre as quais ficam o conteúdo desse objeto.

```
>>> preco = {"sal":2.1, "açúcar":2.85, "macarrão":4.25}
```

Dicionários – Adicionando elementos

- Para adicionar elementos num dicionário basta associar uma nova chave ao objeto e dar um valor a ser associado a ela.

```
>>> preco = {"sal":2.1, "açúcar":2.85, "macarrão":4.25}
>>> preco
{'sal': 2.1, 'açúcar': 2.85, 'macarrão': 4.25}
>>> preco["arroz"] = 3.81
>>> preco
{'sal': 2.1, 'açúcar': 2.85, 'macarrão': 4.25, 'arroz': 3.81}
```

Dicionários – Acessando elementos

- Uma vez que o dicionário foi populado, o acesso aos itens ou elementos pode ser realizado de modo similar ao método de acesso das listas.
- Entretanto, aqui deve-se passar a chave para qual se deseja acessar o valor

```
>>> preco
{'sal': 2.1, 'açúcar': 2.85, 'macarrão': 4.25, 'arroz': 3.81}
>>> preco["sal"]
2.1
```

- Ou ainda, através da função get()

```
>>> p = preco.get("macarrão")
>>> p
4.25
```

Dicionários – Alterando Elementos

- Os valores podem ser atribuídos diretamente a uma chave específica, isso pode ocorrer quando é necessário alterar atualizar ou inicializar uma chave do dicionário.

```
>>> preco
{'sal': 2.1, 'açúcar': 2.85, 'macarrão': 4.25, 'arroz': 3.81}
>>> preco["açúcar"] = 3.12
>>> preco
{'sal': 2.1, 'açúcar': 3.12, 'macarrão': 4.25, 'arroz': 3.81}
>>> preco["feijão"] = 5.99
>>> preco
{'sal': 2.1, 'açúcar': 3.12, 'macarrão': 4.25, 'arroz': 3.81, 'feijão':
5.99}
```

Dicionários – Removendo elementos

- A operação de deleção pode ser realizada tanto para uma chave específica, como a remoção do dicionário inteiro.

```
>>> preco
{'sal': 2.1, 'açúcar': 3.12, 'macarrão': 4.25, 'arroz': 3.81, 'feijão': 5.99}
>>> del preco["arroz"]
>>> preco
{'sal': 2.1, 'açúcar': 3.12, 'macarrão': 4.25, 'feijão': 5.99}
>>> del preco
>>> preco
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name 'preco' is not defined
```

Dicionários – Limpando o dicionário

- Para deletar todas as chaves e valores de um dicionário, pode ser utilizada a função clear() ou ainda atribuir duas chaves vazias { } ao nome do dicionário.

```
>>> preco = {'sal': 2.1, 'açúcar': 3.12, 'macarrão': 4.25,
'arroz': 3.81, 'feijão': 5.99}
>>> preco
{'sal': 2.1, 'açúcar': 3.12, 'macarrão': 4.25, 'arroz': 3.81,
'feijão': 5.99}
>>> preco.clear()
>>> preco
{}
```

Dicionários – Removendo o último item

- Com a função popitem() também é possível deletar os elementos do dicionário, quando executada, o último item será removido.

```
>>> preco = {'sal': 2.1, 'açúcar': 3.12, 'macarrão': 4.25, 'arroz':
3.81, 'feijão': 5.99}
>>> preco.popitem()
('feijão', 5.99)
>>> preco.popitem()
('arroz', 3.81)
>>> preco.popitem()
('macarrão', 4.25)
>>> preco
{'sal': 2.1, 'açúcar': 3.12}
```

Dicionários - Impressão

- Formas de impressão (o próprio dicionário)

```
>>> preco = {'sal': 2.1, 'açúcar': 3.12, 'macarrão': 4.25,
'arroz': 3.81, 'feijão': 5.99}
>>> print(preco)
{'sal': 2.1, 'açúcar': 3.12, 'macarrão': 4.25, 'arroz': 3.81,
'feijão': 5.99}
```

Dicionários – Impressão (items())

- Formas de impressão (método items: cria uma lista de tuplas)

```
>>> print(preco.items())
dict_items([('sal', 2.1), ('açúcar', 3.12), ('macarrão', 4.25), ('arroz', 3.81), ('feijão', 5.99)])
```

Dicionários – Impressão (for/items())

- Formas de impressão iterando através do for
  - Desempacotando as tuplas criadas pelo método items()

```
>>> for produto,precoProd in preco.items():
...     print(f"O valor de {produto} é {precoProd}")
...
O valor de sal é 2.1
O valor de açúcar é 3.12
O valor de macarrão é 4.25
O valor de arroz é 3.81
O valor de feijão é 5.99
```

Dicionários – Impressão de chaves

- O método keys() também retorna uma lista, entretanto essa lista contém somente as chaves do dicionário, desprezando assim a impressão dos valores associados a cada chave.

```
>>> print(preco.keys())
dict_keys(['sal', 'açúcar', 'macarrão', 'arroz', 'feijão'])
>>> for produto in preco.keys():
...     print(f"Black Friday: {produto}")
...
Black Friday: sal
Black Friday: açúcar
Black Friday: macarrão
Black Friday: arroz
Black Friday: feijão
```

Dicionários – Impressão ordenada

- A impressão dos elementos de um dicionário obedece à ordem que foram inseridos.
- Para realizar a impressão de modo ordenado, utiliza-se a função sorted(). Ordenando a lista em ordem alfabética.

```
>>> sorted(preco.keys())
['arroz', 'açúcar', 'feijão', 'macarrão', 'sal']
```

Dicionários – Impressão com índice

- Ao percorrer um dicionário o índice de posição de cada elemento e o valor correspondente podem ser recuperados, como mostra o exemplo abaixo através da função enumerate().

```
>>> for i, produto in enumerate(preco):
...     print(f"{i+1}o.: {produto}")
...
1o.: sal
2o.: açúcar
3o.: macarrão
4o.: arroz
5o.: feijão
```

Dicionários – Total de itens

- Com o comando len() é possível visualizar o tamanho do dicionário ou a quantidade de itens que ele contém.

```
>>> len(preco)
5
```

Dicionários – Listas de dicionários

- Os dicionários também podem ser armazenados em listas. Através dos métodos já conhecidos como atribuição direta à variável, ou por meio do método append()

```
>>> preco = {'sal': 2.1, 'açúcar': 3.12, 'macarrão': 4.25, 'arroz': 3.81, 'feijão': 5.99}
precoFrutas = {"maçã": 8.99, "pêra": 12.0, "banana": "3 por 10"}
listaPrecos = []
listaPrecos.append(preco)
listaPrecos.append(precoFrutas)
listaPrecos
[{'sal': 2.1, 'açúcar': 3.12, 'macarrão': 4.25, 'arroz': 3.81, 'feijão': 5.99}, {'maçã': 8.99, 'pêra': 12.0, 'banana': '3 por 10'}]
```

Dicionários – Chaves múltiplas (lista)

- É possível passar múltiplos valores a uma chave, isso pode ser feito por meio de uma lista de valores

```
>>> preco = {'sal': ["001", 1.1, 2.1], 'açúcar': ["002", 1.51, 3.12], 'macarrão': ["003", 2.75, 4.25]}
>>> print(f"\n{prod}: ",end="")
...     for dado in dadosProd:
...         print(f"\t{dado}, ", end=" ")
...
sal:  001,  1.1,  2.1,
açúcar:  002,  1.51,    3.12,
macarrão:   003,  2.75,    4.25,
```

Dicionários de dicionários

- É possível criar e inicializar um dicionário que contenha outros dicionários

```
>>> preco1 = {'sal': 2.1, 'açúcar': 3.12, 'macarrão': 4.25, 'arroz':
3.81, 'feijão': 5.99}
>>> preco2 = {'pêra': 9.9, 'maça': 4.99, 'banana': "3 por 10"}
precos = {"Preços das Estivas": preco1, "Preço dos Vegetais":
preco2}
```

```
for p1, p2 in precos.items():
...     print(f"\n{p1}: ")
...     for prod, preco in p2.items():
...         print(f"{prod}: {preco} ")
...     print()
...
Preços das Estivas:
sal: 2.1
açúcar: 3.12
macarrão: 4.25
arroz: 3.81
feijão: 5.99
Preço dos Vegetais:
pêra: 9.9
maça: 4.99
banana: 3 por 10
```

Registros em Python com dicionários

- Em Python, representamos registros através de dicionários
- Podemos criar dicionários de duas maneiras:
  - Já com todos os campos e valores:

```
nome_registro = { 'nome_campo1' : conteudo_campo1,
                  'nome_campo2' : conteudo_campo2,
                   …,
                  'nome_campoN' : conteudo_campoN }
```

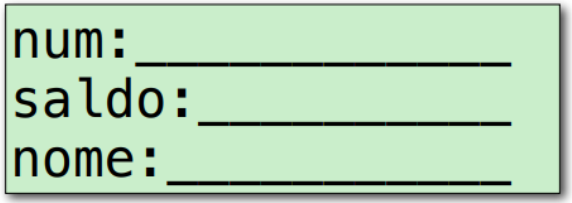
Registros em Python com dicionários

- Inicializa um dicionário vazio e insere os campos e valores depois:

```
nome_registro = {}
nome_registro[‘nome_campo1’] = conteudo_campo1
nome_registro[‘nome_campo2’] = conteudo_campo2
…
nome_registro[‘nome_campoN’] = conteudo_campoN
```

Exemplos

Variável conta




```
conta = {'num' : 123456,
         'saldo' : 1000.00,
         'nome' : 'João Silva'}
```

ou

```
conta = {}
conta['num'] = int(input("Digite o número da conta: "))
conta['nome'] = input("Digite o nome do titular da conta: ")
conta['saldo'] = float(input("Digite o saldo da conta: "))
```

Exemplo: Preenchendo lista de dicionários

```
funcionarios = [] # lista vazia
for i in range(4): # quatro funcionarios
    funcionario = {} # registro funcionario
    funcionario[‘nome’] = input(“Digite o nome do funcionário %d: ”%i)
    funcionario[‘salario’] = float(input(“Digite o salário do \
funcionário %d: ” % i))
    funcionarios.append(funcionario) # adiciona o dicionário à lista
```

Exemplo: Mostrando dicionários em lista

```
for i in range(4):
    print(“Funcionário que ocupa a posição %d no vetor:” % i)
    print(“Nome: %s” % funcionarios[i][‘nome’])
    print(“Salário: R$ %.2f” % funcionarios[i][‘salario’])
```

Sets

- Os sets são uma coleção de itens desordenada, parcialmente imutável e que não podem conter elementos duplicados. Por ser parcialmente imutável, os sets possuem permissão de adição e remoção de elementos.

- Além disso, os sets são utilizados com operações matemáticas
  - união,
  - interseção e
  - diferença simétrica

Declarando Sets

- Seus elementos devem estar entre chaves ({})

```
# Declaração de um set
meu_set = {1, 2, 3, 4, 1}
print(type(meu_set))
```

- Ou utilizando o método set() do próprio Python

```
meu_set_2 = set([1, 2, 8, 9, 10])
print(type(meu_set_2))
```

Adicionando, atualizando e removendo elementos dos Sets

- Os sets permitem as operações de adição, atualização e remoção de elementos.
- Para isso, existe um método para cada funcionalidade.
  - Add
  - Update
  - Discard

Adicionando elementos aos Sets

- o método add() é responsável por adicionar um elemento ao set, desde que ele já não esteja adicionado, pois os sets não permitem elementos duplicados.

```
meu_set = {1, 2, 3, 4, 1}

# Adicionando elementos
meu_set.add(2)
print("Adição", meu_set)
```

Atualizando elementos os Sets

- o método update() permite atualizar os sets e adicionar os elementos (que não estejam duplicados) em sua estrutura.

```
# Atualizando set
meu_set.update([3, 4, 5, 6])
print("Atualição", meu_set)
```

Removendo elementos os Sets

- o método discard() permite a remoção de um elemento do set.

```
# Removendo elemento
meu_set.discard(2)
print("Remoção", meu_set)
```

Operações matemáticas com Sets

- Os sets, além das operações de manipulação vistas anteriormente, permitem operações matemáticas como:
  - União,
  - Interseção,
  - Diferença e
  - Diferença simétrica

União em Sets

- Na união (|), todos os elementos dos dois sets serão “unidos”, formando um único set com todos os elementos, sem repeti-los.

```
meu_set = {1, 2, 3, 4, 1}
meu_set_2 = set([1, 2, 8, 9, 10])

# União
print("União")
print(meu_set | meu_set_2)
print(meu_set.union(meu_set_2))
```

```
União
{1, 2, 3, 4, 8, 9, 10}
{1, 2, 3, 4, 8, 9, 10}
```

Interseção em Sets

- Na Interseção (`&`), apenas os elementos que estiverem nos dois sets restarão.

```
# Interseção
print("Interseção")
print(meu_set & meu_set_2)
print(meu_set.intersection(meu_set_2))
```

```
Interseção
{1, 2}
{1, 2}
```

Diferença de Sets

- Na diferença (-), restarão apenas os elementos que estiverem em um dos set, mas não no segundo

```
# Diferença
print("Diferença")
print(meu_set - meu_set_2)
print(meu_set.difference(meu_set_2))
```

```
Diferença
{3, 4}
{3, 4}
```

Diferença Simétrica em Sets

- na diferença simétrica (^), apenas os elementos que estiverem nos dois sets, porém que não se repitam, serão exibidos

```
# Diferença Simétrica
print("Diferença Simétrica")
print(meu_set ^ meu_set_2)
print(meu_set.symmetric_difference(meu_set_2))
```

```
Diferença Simétrica
{3, 4, 8, 9, 10}
{3, 4, 8, 9, 10}
```

Fontes

- Documentação Oficial: https://docs.python.org/3.8/reference/datamodel.html#the-standard-type-hierarchy
- https://maumneto.github.io/fundamentos-programacao/material/aula16_registros.pdf
- https://www.treinaweb.com.br/blog/manipulando-sets-no-python/
- https://www.treinaweb.com.br/blog/principais-estruturas-de-dados-no-python
- MENEZES, N. N. C. (2014) Introdução à Programação com Python: Algoritmos e Lógica de Programação para Iniciantes. Editora Novatec.

Exercícios

[1] [Instituto AOCP 2020 MJ/SP – Analista de Governança de Dados – Big Data] Um analista do MJSP implementou o
seguinte programa em Python:

```
01 a = [1, 2, 3];
02 b = list(a);
03 print(a);
04 print(b);
05 print(a==b);
06 print(a is b);
```

(A) A resposta do programa é:

```
[1, 2, 3]
[ ]
False
False
```

(B) A resposta do programa é:

```
[1, 2, 3]
[1, 2, 3]
True
True
```

(C) A resposta do programa é:

```
[1, 2, 3]
[1, 2, 3]
True
False
```

(D) A resposta do programa é:

```
[1, 2, 3]
[1, 2, 3]
True
Null
```

(E) O programa não executa por um erro na linha 02, em que é atribuído um tipo de dado diferente daquele definido na
variável b.

[2] [**Instituto AOCP 2020 MJ/SP – Cientista de Dados – Big Data**] In Python, the statement 'for loop' is used for
iterating over a sequence. Considering that, choose the correct alternative that presents a proper example concerning
the use of 'for loop' in Python.

(A)

```
for x = ["John", "Sophie", "Junior"]:
    print(x)
```

(B)

```
array (["John"],["Sophie"],["Junior"])
for each i = 0 to array[i]
    print (array[i])
```

(C)

```
names = ["John", "Sophie", "Junior"]
for x in names:
    print(x)
```

(D)

```
names = {{"John"}, {"Sophie"}, {"Junior"}}
for x in names:
    print(names)
```

(E)

```
array.names("John", "Sophie", "Junior")
for x print(names[x]):
    end for:
```

[3] [**IFB 2017 IFB – Professor – Informática/Desenvolvimento de Sistemas**] Dado o código em Python abaixo, assinale
a alternativa que contém a saída CORRETA gerada pelo “print”:

```
lista = ["cachorro", "hamster", ["pato", "galinha", "porco"], "gato"]
print(lista[3][2])
```

(A) galinha

(B) ga

(C) t

(D) gato

(E) to

[2] [**COMPERVE 2016 UFRN – Técnico de Tecnologia da Informação**] Analise o trecho de código Python a seguir, escrito para a versão 2.6.

```
t=(1,3,4)
t2=(3,4,5)
t3=t+t2
n=0
for e in t3:
    if (e>2):
        n+=e
print(n)
```

Após a execução do código Python, o valor da variável n impresso na tela é

[A] 12

[B] 13

[C] 19

[D] 20

Três tipos de dados fundamentais em Python são as listas ("*lists*"), sequências ou 'tuplas' ("*tuples*") e dicionários ("*dictionaries*"). A respeito dessas estruturas, é correto afirmar:

A) Listas não podem ser modificadas depois de criadas, ao passo que sequências e dicionários podem.

B) Listas podem ser modificadas, mas seu tamanho não pode ser modificado após a criação, ao passo que sequências e dicionários não têm essa limitação.

C) Dicionários não podem ser modificados depois de criados, ao passo que listas e sequências podem.

D) Listas e sequências são indexadas por inteiros, ao passo que dicionários podem ser indexados por "strings".

E) Listas e dicionários são indexados por inteiros, ao passo que sequências podem ser indexadas por "strings".

- Faça um programa em Python para criar um estoque de produtos. Cada produto possui um código, um preço e uma quantidade. O estoque pode armazenar 10 produtos. Depois de receber os produtos do estoque, mostre um relatório com os produtos, a quantidade total de itens e o valor total do estoque.

  - Use uma função cria_produto para criar um registro com código, preço e quantidade e outra função que mostra_produto para mostrar os valores dos campos de um produto

- Faça um programa em Python para criar um estoque de produtos. Cada produto possui um código, um preço e uma quantidade. O estoque pode armazenar 10 produtos. Depois de receber os produtos do estoque, mostre um relatório com os produtos, a quantidade total de itens e o valor total do estoque.

  - Use uma função cria_produto para criar um registro com código, preço e quantidade e outra função que mostra_produto para mostrar os valores dos campos de um produto

  - Incremente o programa anterior, acrescentando uma função que busca um determinado produto por código. A função deve retornar o registro do produto. O programa deve pedir para o usuário digitar um código, buscar o produto, e se encontrá-lo mostrar os dados do produto.