Raciocínio Computacional

UNIDADE 05

Estrutura de dados: tuplas e dicionários

Esta unidade tem por objetivo apresentar o conceito de tuplas e dicionários em Python, atuando como estruturas de dados heterogêneas. A partir dessa estrutura, poderão ser tratados conceitos como imutabilidade de dados e acesso a dados a partir de pares chave-valor. Ao aprender esses conceitos, você terá o conhecimento-base necessário para o trato de cadastro e manipulação de registros, fundamental para criação de sistemas de gerenciamento de dados.

Estruturas de dados: tuplas e dicionários

Na unidade anterior, foram apresentados os conceitos de estruturas de dados homogêneas e heterogêneas. Embora as listas sejam estruturas heterogêneas, como todas em Python, elas normalmente são utilizadas como estruturas homogêneas (trabalhando apenas com um tipo de dado). Nesse contexto, existe outra estrutura de dados, chamada tupla, que basicamente pode ser considerada uma lista imutável, mas com características heterogêneas, ou seja, armazena tipos diferentes de dado.

Outra estrutura de dados existente no Python é o dicionário (***dictionary***), que permite o acesso a dados a partir de acesso chave-valor, em que uma chave permite acessar determinado dado. Devido a essa característica, um dicionário pode armazenar diferentes dados na forma de registro, o que o torna uma estrutura ideal para trabalho com informações de cadastros.

Tuplas

Tuplas possuem estrutura muito parecida com listas, sendo a principal diferença o fato de as tuplas serem estruturas imutáveis, ou seja, uma vez definido o seu conteúdo, ele não pode ser modificado. Sua criação é feita por meio do uso de parênteses:

```
endereco_puc = (“Rua Imaculada Conceição”, 1555, “Prado
Velho”, “Curitiba”, “PR”)
documentos_joao = (“5321456-9”, “123456789-00”)
tupla_vazia = tuple()
```

Como pode ser observado, as tuplas são estruturas heterogêneas, podendo armazenar tipos diferentes de dado. Entretanto, os mesmos dados poderiam ser armazenados fazendo uso de uma lista. **Mas onde está a diferença?**

A diferença basicamente é conceitual. Da forma como os dados foram apresentados, como, por exemplo na variável endereco_puc, não seria correto alterar apenas a rua ou o número, pois os dados fazem sentido somente no conjunto completo. Como uma tupla é imutável, deveria ser criada uma nova tupla contendo um novo endereço completo para que fizesse sentido. Temos outros exemplos, como armazenar as informações de expedição de um documento – no caso do RG, quando é necessário adicionar o número, data de emissão e local de expedição; o número da carteira de motorista; o cartão de crédito com a data de validade. A tupla permite que esses dados não sejam alterados. Assim sendo, conceitualmente, as listas são usadas com o sentido de ordem ou encadeamento de informações, enquanto as tuplas são usadas com o sentido de estrutura.

Exemplo de aplicação 1: Elabore um programa que solicite ao usuário o cadastro de endereços para entrega de produtos de uma loja.

```
enderecos = []
print("Cadastro de Endereços de Entrega")
while True:
    logradouro = input("Digite o logradouro: ")
    numero = int(input("Digite o número: "))
    bairro = input("Digite o bairro: ")
    cidade = input("Digite a cidade: ")
    estado = input("Digite o estado: ")
    novo_endereco = (logradouro, numero, bairro, cidade, estado)
    enderecos.append(novo_endereco)
    if input("Deseja cadastrar um novo endereço (s/n): ") == "n":
        break
print("Os endereços cadastrados são:")
for i in range(0, len(enderecos)):
    endereco = enderecos[i]
    print(f"{i}. {endereco[0]}, {endereco[1]}, {endereco[2]} - {endereco[3]}/{endereco[4]}")
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic01.py
Cadastro de Endereços de Entrega
Digite o logradouro: Rua Imaculada Conceição
Digite o número: 1555
Digite o bairro: Prado Velho
Digite a cidade: Curitiba
Digite o estado: PR
Deseja cadastrar um novo endereço (s/n): s
Digite o logradouro: Rua Professor Eurico Rabelo
Digite o número: 0
Digite o bairro: Norte
Digite a cidade: Rio de Janeiro
Digite o estado: RJ
Deseja cadastrar um novo endereço (s/n): n
Os endereços cadastrados são:
0. Rua Imaculada Conceição, 1555, Prado Velho - Curitiba/PR
1. Rua Professor Eurico Rabelo, 0, Norte - Rio de Janeiro/RJ


Process finished with exit code 0
```

Neste exemplo, caso seja efetuada a tentativa de alteração de algum dado da tupla, ocorre um erro do tipo:

```
14.     for i in range(0, len(enderecos)):
15.         endereco = enderecos[i]
16.         print(f"{i}. {endereco[0]}, {endereco[1]}, {endereco[2]} - {endereco[3]}/{endereco[4]}")
17.         endereco[0] = "Novo endereço"
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic01.py

Traceback (most recent call last):
  File "/Users/user/projects/untitled1/ExAplic01.py", line
18, in <module>
    endereco[0] = "Novo endereço"
TypeError: 'tuple' object does not support item assignment


Process finished with exit code 1
```

Embora os dados de uma tupla não possam ser alterados, se eles forem objetos mutáveis, poderão ser alterados sem problemas, uma vez que o que não pode ser mudado é a referência dentro da tupla para esse dado.

Exemplo de aplicação 2: Elabore um programa que crie uma tupla contendo duas listas com dados, altere os dados da primeira lista e verifique se ocorre mudança de dados da tupla.

```
1. tupla_com_lista = ([1, 2, 3], [4, 5, 6])
2. print(id(tupla_com_lista[0]))
3. tupla_com_lista[0].append(9)
4. print(tupla_com_lista)
5. print(id(tupla_com_lista[0]))
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic02.py
4380125824
([1, 2, 3, 9], [4, 5, 6])
4380125824


Process finished with exit code 0
```

Como pode ser visto no exemplo, o fato de alterar os dados dos objetos mutáveis da tupla não altera especificamente os dados da tupla, uma vez que ela armazena os endereços das listas, que não mudam.

Concatenação de tuplas

É possível concatenar tuplas, o que gerar uma nova tupla, uma vez que elas são imutáveis.

Exemplo de aplicação 3: Elabore um programa que concatene tuplas.

```
1. endereco_puc = ("Rua Imaculada Conceição", 1555, "Prado
   Velho", "Curitiba", "PR")
2. print(id(endereco_puc))
3. endereco_puc += ("Brazil",)
4. print(endereco_puc)
5. print(id(endereco_puc))
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic03.py
4430577968
('Rua Imaculada Conceição', 1555, 'Prado Velho',
'Curitiba', 'PR', 'Brazil')
4431654976


Process finished with exit code 0
```

No exemplo, é possível verificar que, embora seja usado o operador “+=”, pela análise dos endereços de memória, foi criada uma tupla, atribuída à variável **endereco_puc**. Outro ponto a destacar está na linha 03, em que se pode ver que uma tupla com um único dado deve ser declarada com uma vírgula no fim; caso contrário, não será considerada uma tupla.

Exemplo de aplicação 4: Elabore um aplicativo que demonstre a necessidade de finalizar uma tupla de um único elemento com uma vírgula no fim.

```
1. tupla1 = ("Brasil")
```

```
2. print(type(tupla1))
3. tupla2 = ("Brasil",)
4. print(type(tupla2))
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic04.py
<class 'str'>
<class 'tuple'>


Process finished with exit code 0
```

Como é possível ver, o simples fato de usar parênteses ao redor de um único dado não cria uma tupla.

Tuplas nomeadas usando namedtuple()

A função ***namedtuple()*** permite nomear tanto tuplas quanto seus dados, o que resulta em um código muito mais legível. Segue sua sintaxe:

```
collections.namedtuple(typename, field_names)
```

Função *namedtuple*

Parâmetro	**Descrição**
typename	Nome do tipo da tupla criada que será usado para referenciá-la.
field_names	Nome dos dados a ser incluídos na tupla.

Para o uso da função ***namedtuples***, é necessário importar a função, fazendo uso de:

```
from collections import namedtuple
```

Exemplo de aplicação 5: Elabore um programa que faça uso de uma tupla chamada Endereco, contendo dados nomeados.

```
1. from collections import namedtuple
2.
3. Endereco = namedtuple("Endereco", ["logradouro", "numero",
   "bairro", "cidade", "estado"])
4. endereco_puc = Endereco(logradouro="Rua Imaculada
   Conceição", numero=1555, bairro="Prado Velho",
   cidade="Curitiba", estado="PR")
5. print(f"Endereço: {endereco_puc[0]}")
6. print(f"  Número: {endereco_puc.numero}")
7. print(f"  Bairro: {endereco_puc.bairro}")
8. print(f"  Cidade: {endereco_puc.cidade}")
9. print(f"  Estado: {endereco_puc.estado}")
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic05.py
Endereço: Rua Imaculada Conceição
  Número: 1555
  Bairro: Prado Velho
  Cidade: Curitiba
  Estado: PR


Process finished with exit code 0
```

Como pode ser visto na linha 05, é possível acessar os dados da tupla a partir do índice, como se fosse uma lista, ou do dado nomeado, como indicado nas linhas 06 a 09. Essa maior legibilidade trazida ao código é importante, especialmente ao trabalhar em um projeto grande.

> **Esta é a diferença maior entre listas e dicionários:** as listas, por serem ordenadas, permitem acessar seus dados a partir de índices numéricos, enquanto os dicionários permitem acessar seus dados por qualquer tipo de índice (*strings*, inteiros etc.), chamado chave.

Para criar um dicionário, é preciso colocar seus dados entre chaves, como segue:

```
faturamento = {“janeiro”: 2000, “fevereiro”: 3500, “março”: 2800}
```

O primeiro dado antes do símbolo de dois-pontos é a chave, seguida do valor. Para acessar o valor cadastrado, usa-se a chave. Por exemplo:

```
faturamento_janeiro = faturamento[“janeiro”]
```

De forma análoga, a chave é usada para alterar um valor:

```
faturamento[“janeiro”] = 5000
```

Exemplo de aplicação 6: Elabore um programa que simule o cadastro de telefones com dicionário como uma agenda, exibindo, ao final, o dicionário.

```
1. agenda = {}
2. print("*** Cadastro de telefones ***")
3. while True:
4.     contato = input("Digite o nome do contato: ")
5.     telefone = input("Digite o telefone do contato: ")
6.     agenda[contato] = telefone
7.     if input("Deseja cadastrar um novo contato (s/n): ") == "n":
8.         break
9. print(agenda)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic06.py
*** Cadastro de telefones ***
Digite o nome do contato: Maria
Digite o telefone do contato: (41) 98765-4321
Deseja cadastrar um novo contato (s/n): s
Digite o nome do contato: João
Digite o telefone do contato: (11) 12345-6789
Deseja cadastrar um novo contato (s/n): s
Digite o nome do contato: Rosana
Digite o telefone do contato: (21) 91827-3645
Deseja cadastrar um novo contato (s/n): n
{'Maria': '(41) 98765-4321', 'João': '(11) 12345-6789',
'Rosana': '(21) 91827-3645'}

Process finished with exit code 0
```

Na linha 01, é possível ver a sintaxe da criação de um dicionário vazio, fazendo uso de chaves. Na linha 06, o contato é criado diretamente pelo uso do dicionário acessando a chave desejada, no caso, o nome do contato. Cabe destacar que, caso seja digitado um nome já utilizado, o valor referenciado por esse nome como chave será alterado. Assim sendo, é importante que primeiramente seja verificado se determinada chave já existe antes de ser utilizada.

Verificando a existência de uma chave ou valor

Para verificar se uma chave já está cadastrada em um dicionário, é utilizada a palavra-chave ***in***, como segue:

```
if “Maria” in agenda:
```

De forma análoga, é possível verificar se um valor já está cadastrado no dicionário. Para tanto, é utilizado o método ***values()***:

```
if “(21) 91827-3645” in agenda.values():
```

Dessa forma, é possível antecipar problemas de lógica, deixando o algoritmo em um estado consistente.

Exemplo de aplicação 7: Elabore um programa que simule o cadastro de telefones com dicionário como uma agenda. Caso seja informado um nome já existente, deve perguntar se deseja alterar os dados existentes. Caso seja um telefone já existente, deve informar que esse telefone já está cadastrado em outro contato, não podendo ser efetuada a inclusão. Ao final, deve exibir o dicionário.

```
1. agenda = {}
2. print("*** Cadastro de telefones ***")
3. while True:
4.     contato = input("Digite o nome do contato: ")
5.     telefone = input("Digite o telefone do contato: ")
6.     if contato in agenda:
7.         if input(f"Contato já cadastrado com o número
   {agenda[contato]}. Deseja alterar? (s/n) ") == "n":
8.             continue
9.     if telefone in agenda.values():
10.        print("Telefone já cadastrado para outro contato")
11.        continue
12.    agenda[contato] = telefone
13.    if input("Deseja cadastrar um novo contato (s/n): ") ==
   "n":
```

```
14.             break
15. print(agenda)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic07.py
*** Cadastro de telefones ***
Digite o nome do contato: Maria
Digite o telefone do contato: (41) 98765-4321
Deseja cadastrar um novo contato (s/n): s
Digite o nome do contato: João
Digite o telefone do contato: (11) 12345-6789
Deseja cadastrar um novo contato (s/n): s
Digite o nome do contato: Maria
Digite o telefone do contato: (11) 12345-6789
Contato já cadastrado com o número (41) 98765-4321. Deseja
alterar? (s/n) s
Telefone já cadastrado para outro contato
Digite o nome do contato: Rosana
Digite o telefone do contato: (21) 91827-3645
Deseja cadastrar um novo contato (s/n): n
{'Maria': '(41) 98765-4321', 'João': '(11) 12345-6789',
'Rosana': '(21) 91827-3645'}


Process finished with exit code 0
```

Na linha 07, há um aspecto importante de comunicação do sistema com o usuário: além de informar que o contato já está cadastrado na agenda, apresenta o seu telefone, fundamentando a decisão do usuário de continuar com a alteração ou não. Nas linhas 06 a 11, são testadas as condições para o cadastro do telefone. Caso alguma seja conflitante, o comando ***continue*** faz com que o código retorne seu fluxo para o início do laço. Caso contrário, o telefone é cadastrado na linha 12.

Remoção de itens de um dicionário

Existem duas formas de remover um item de um dicionário: usando a palavra reservada ***del*** ou o método ***pop*** do dicionário. A remoção com ***del*** pode ser feita da seguinte forma:

```
del agenda[“Maria”]
```

Embora funcional, caso a chave “Maria” não exista, isso gera um erro de execução, como segue:

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic07.py

Traceback (most recent call last):
  File "/Users/user/projects/untitled1/ExAplic07.py", line
3, in <module>
    del agenda["maria"]
KeyError: 'maria'


Process finished with exit code 1
```

Assim, para usar ***del***, é necessário primeiramente verificar se a chave existe.

A segunda forma usa o método ***pop*** do dicionário, retornando o dado removido em caso de sucesso ou um valor-padrão caso tenha falhado a remoção, porém sem gerar erro no sistema.

Exemplo de aplicação 8: Elabore um programa que exemplifique a remoção de itens de um dicionário usando o método *pop()*.

```
1. agenda = {"Maria": "(41) 98765-4321", "João": "(11) 12345-
   6789"}
2. print(agenda.pop("Maria", "Contato com nome Maria
   localizado"))
3. print(agenda.pop("José", "Contato com nome José não
   localizado"))
4. print(agenda)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic08.py
(41) 98765-4321
Contato com nome José não localizado
{'João': '(11) 12345-6789'}


Process finished with exit code 0
```

Dicionário – métodos

Um dicionário é um objeto em Python, ou seja, não faz parte dos tipos primitivos. Assim, possui métodos, que são ações que pode realizar. Segue a lista de métodos que podem ser aplicados sobre dicionários em Python:

Dicionário – métodos

Método	Descrição
clear()	Remove todos os elementos de um dicionário.
copy()	Retorna a cópia de um dicionário.
fromkeys()	Retorna um dicionário com chaves e valores específicos.
get()	Retorna o valor de uma chave específica.
items()	Retorna uma lista contendo uma tupla para cada par chave-valor.
keys()	Retorna uma lista contendo as chaves de um dicionário.
pop()	Remove um elemento de uma chave específica.

popitem()	Remove o último elemento inserido no dicionário.
setdefault()	Retorna o valor de uma chave específica. Caso não exista, insere esse valor.
update()	Adiciona um conjunto de pares chave-valor a um dicionário.
values()	Retorna a lista de valores de um dicionário.

Seguem exemplos de aplicação de métodos sobre dicionários:

```
1. agenda = {"Maria": "(41) 98765-4321", "João": "(11) 12345-
   6789", "Rosana": "(21) 91827-3645"}
2. print(agenda)
3. # adiciona um novo elemento no vetor
4. agenda["José"] = "(19) 98877-1122"
5. print(agenda)
6. # cria um dicionário com chaves e valores específicos
7. chaves = ["chave1", "chave2", "chave3"]
8. valores = "valor"
9. novo_dicionario = dict.fromkeys(chaves, valores)
10. print(novo_dicionario)
11. # mostra a lista de itens do dicionário
12. print(agenda.items())
13. # mostra a lista de chaves do dicionário
14. print(agenda.keys())
15. # mostra a lista de valores do dicionário
16. print(agenda.values())
17. # remove um item de chave específica
18. agenda.pop("Maria")
19. print(agenda)
20. # remove o último item inserido
21. agenda.popitem()
22. print(agenda)
23. # retorna ou insere o valor de uma chave em específico
24. print(agenda.setdefault("Rosana", "(21) 91827-3645"))
25. print(agenda.setdefault("Maria", "(41) 98765-4321"))
26. print(agenda)
27. # adiciona o conteúdo de um dicionário a outro
28. agenda.update(novo_dicionario)
29. print(agenda)
30. # limpa o conteúdo de um dicionário
```

```
31. agenda.clear()
32. print(agenda)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/listas_metodos.py
{'Maria': '(41) 98765-4321', 'João': '(11) 12345-6789',
'Rosana': '(21) 91827-3645'}
{'Maria': '(41) 98765-4321', 'João': '(11) 12345-6789',
'Rosana': '(21) 91827-3645', 'José': '(19) 98877-1122'}
{'chave1': 'valor', 'chave2': 'valor', 'chave3': 'valor'}
dict_items([('Maria', '(41) 98765-4321'), ('João', '(11)
12345-6789'), ('Rosana', '(21) 91827-3645'), ('José',
'(19) 98877-1122')])
dict_keys(['Maria', 'João', 'Rosana', 'José'])
dict_values(['(41) 98765-4321', '(11) 12345-6789', '(21)
91827-3645', '(19) 98877-1122'])
{'João': '(11) 12345-6789', 'Rosana': '(21) 91827-3645',
'José': '(19) 98877-1122'}
{'João': '(11) 12345-6789', 'Rosana': '(21) 91827-3645'}
(21) 91827-3645
(41) 98765-4321
{'João': '(11) 12345-6789', 'Rosana': '(21) 91827-3645',
'Maria': '(41) 98765-4321'}
{'João': '(11) 12345-6789', 'Rosana': '(21) 91827-3645',
'Maria': '(41) 98765-4321', 'chave1': 'valor', 'chave2':
'valor', 'chave3': 'valor'}
{}


Process finished with exit code 0
```

EXERCÍCIO

Tuplas

Exercício de fixação 1: Crie um programa que efetue o cadastro de pessoas com nome, RG e CPF por meio de tuplas, adicionando-as a uma lista e imprimindo essa lista no fim do programa.

Resolução do exercício

Resolução exercício 1

```
pessoas = []
while True:
    nome = input("Digite o nome da pessoa: ")
    rg = input("Digite o número do RG: ")
    cpf = input("Digite o número do CPF: ")
    pessoa = (nome, rg, cpf)
    pessoas.append(pessoa)
    if input("Gostaria de cadastrar mais uma pessoa (s/n): ") == "n":
        break
print(pessoas)
```

C:\

```
/Users/user/untitled1/bin/python /Users/user/projects/untitled1/ExFix01.py
Digite o nome da pessoa: Maria
Digite o número do RG: 1.234.567-8
Digite o número do CPF: 123.456.789-00
Gostaria de cadastrar mais uma pessoa (s/n): s
Digite o nome da pessoa: João
Digite o número do RG: 2.413.645-9
Digite o número do CPF: 432.765.098-11
Gostaria de cadastrar mais uma pessoa (s/n): n
[('Maria', '1.234.567-8', '123.456.789-00'), ('João', '2.413.645-9', '432.765.098-11')]

Process finished with exit code 0
```

Exercício de fixação 2: Crie um programa que cadastre os funcionários de uma empresa e seus dependentes. O funcionário deve ser cadastrado com matrícula, nome e dependentes. Os dependentes devem ser inseridos dinamicamente em uma tupla. Dica: use o operador “+=”.

Resolução exercício 2

```
1. funcionarios = []
2. while True:
3.     nome = input("Digite o nome do
   funcionário: ")
4.     matricula = input("Digite a matrícula do
   funcionário: ")
5.     dependentes = tuple()
6.     while True:
7.         dependente = input("Digite o nome do
   dependente (0 para sair): ")
8.         if dependente == "0":
9.             break
10.        dependentes += (dependente,)
11.    funcionario = (nome, matricula,
   dependentes)
12.    funcionarios.append(funcionario)
13.    if input("Gostaria de cadastrar mais um
   funcionário (s/n): ") == "n":
14.        break
15. print(funcionarios)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix02.py
Digite o nome do funcionário: Marcos
Digite a matrícula do funcionário: 10234
Digite o nome do dependente (0 para sair):
Joaquim
Digite o nome do dependente (0 para sair):
Maria
Digite o nome do dependente (0 para sair):
0
Gostaria de cadastrar mais um funcionário
(s/n): s
Digite o nome do funcionário: Joana
Digite a matrícula do funcionário: 8134
Digite o nome do dependente (0 para sair):
Leandro
Digite o nome do dependente (0 para sair):
0
Gostaria de cadastrar mais um funcionário
(s/n): n
[('Marcos',      '10234',      ('Joaquim',
'Maria')),        ('Joana',        '8134',
('Leandro',))]


Process finished with exit code 0
```

Exercício de fixação 3: Crie um programa que cadastre locais históricos do mundo com suas coordenadas, fazendo uso de tuplas com parâmetros nomeados. Dica: use a função *namedtuple()*.

 Resolução do exercício 

Resolução exercício 3

```
1. from collections import namedtuple
2.
3. monumentos = []
```

```
4. Monumento = namedtuple("Monumento", ["nome",
   "latitude", "longitude"])
5. while True:
6.     nome = input("Nome do monumento: ")
7.     latitude = float(input("Latitude do
   monumento: "))
8.     longitude = float(input("Longitude do
   monumento: "))
9.     monumento = Monumento(nome=nome,
   latitude=latitude, longitude=longitude)
10.    monumentos.append(monumento)
11.    if input("Gostaria de cadastrar mais um
   monumento? (s/n) ") == "n":
12.        break
13. for monumento in monumentos:
14.    print(f"Monumento: {monumento.nome}")
15.    print(f"Coordenadas:
   ({monumento.latitude},
   {monumento.longitude})")
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix03.py
Nome do monumento: Torre Eiffel
Latitude do monumento: 48.8584
Longitude do monumento: 2.2945
Gostaria de cadastrar mais um monumento?
(s/n) s
Nome do monumento: Cataratas do Iguaçu
Latitude do monumento: -25.5469
Longitude do monumento: -54.5882
Gostaria de cadastrar mais um monumento?
(s/n) s
Nome do monumento: Coliseu
Latitude do monumento: 41.8902
Longitude do monumento: 12.4922
Gostaria de cadastrar mais um monumento?
(s/n) n
Monumento: Torre Eiffel
Coordenadas: (48.8584, 2.2945)
Monumento: Cataratas do Iguaçu
Coordenadas: (-25.5469, -54.5882)
Monumento: Coliseu
Coordenadas: (41.8902, 12.4922)


Process finished with exit code 0
```

Dicionários

Exercício de fixação 4: Crie um programa que efetue o cadastro de produtos e preços. Caso o produto já exista, deve perguntar se o usuário pretende atualizar o valor. Imprima o dicionário no fim do programa em formato de lista.

 Resolução do exercício 

Resolução exercício 4

```
produtos = {}
while True:
    nome = input("Digite o nome do produto: ")
    valor = float(input("Digite o valor do produto: "))
    if nome in produtos:
        if input(f"Produto já cadastrado com o valor {produtos[nome]}. Deseja alterar o valor? (s/n)") == "n":
            continue
    produtos[nome] = valor
    if input("Cadastrar um novo produto? (s/n)") == "n":
        break
for produto in produtos:
    print(f"{produto:>20}: R${produtos[produto]:.2f}")
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix04.py
Digite o nome do produto: Caneta
Digite o valor do produto: 2
Cadastrar um novo produto? (s/n)s
Digite o nome do produto: Caderno de
desenho
Digite o valor do produto: 8.5
Cadastrar um novo produto? (s/n)s
Digite o nome do produto: Estojo
Digite o valor do produto: 12.33
Cadastrar um novo produto? (s/n)n
              Caneta: R$2.00
  Caderno de desenho: R$8.50
              Estojo: R$12.33


Process finished with exit code 0
```

Desafio

Exercício de fixação 5: Crie um programa que solicite o valor das vendas e o mês em que cada venda ocorreu. Independentemente da repetição de meses, o aplicativo deve totalizar por mês todas as vendas cadastradas. Ao final, deve informar o valor de vendas de todos os meses do ano. Observação: se for digitado errado o nome do mês, deve informar que o mês é inválido.

Resolução do exercício

Resolução exercício 5

```
1. vendas = {}
2. meses = [
3.     "janeiro",
4.     "fevereiro",
5.     "março",
6.     "abril",
```

```
    "maio",
    "junho",
    "julho",
    "agosto",
    "setembro",
    "outubro",
    "novembro",
    "dezembro"
]
while True:
    mes = input("Informe o mês da venda: ")
    valor = float(input("Informe o valor da venda: "))
    if not mes in meses:
        print("Mês inválido. Tente novamente!")
        continue
    if mes in vendas:
        vendas[mes] += valor
    else:
        vendas[mes] = valor
    if input("Continuar? (s/n) ") == "n":
        break
for mes in meses:
    if mes in vendas:
        print(f"{mes:>12}: R${vendas[mes]:.2f}")
    else:
        print(f"{mes:>12}: R$0.00")
```

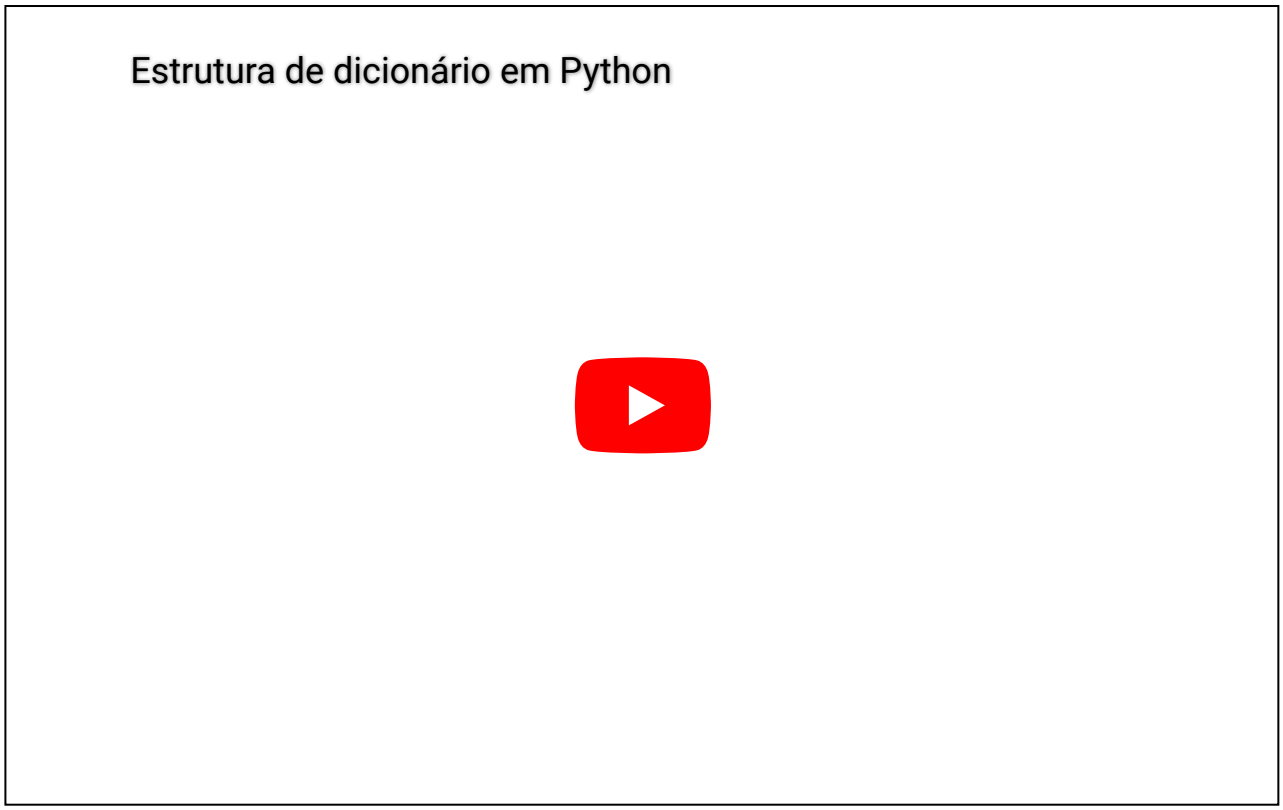
C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix05.py
Informe o mês da venda: janeiro
Informe o valor da venda: 1000
Continuar? (s/n) s
Informe o mês da venda: março
Informe o valor da venda: 1000
Continuar? (s/n) s
Informe o mês da venda: maio
Informe o valor da venda: 1000
Continuar? (s/n) s
Informe o mês da venda: janeiro
Informe o valor da venda: 1500
Continuar? (s/n) s
Informe o mês da venda: março
Informe o valor da venda: 5000
Continuar? (s/n) n
     janeiro: R$2500.00
   fevereiro: R$0.00
       março: R$6000.00
       abril: R$0.00
        maio: R$1000.00
       junho: R$0.00
       julho: R$0.00
      agosto: R$0.00
    setembro: R$0.00
     outubro: R$0.00
    novembro: R$0.00
    dezembro: R$0.00


Process finished with exit code 0
```

Estrutura de dicionário em Python

Neste vídeo, veremos os conceitos de manipulação de dicionários, em conjunto com suas funções *built in* ***items()***, ***values()*** e ***keys()***.



Conclusão

A respeito do uso de tuplas e dicionários na programação em Python, podemos concluir que:

- As tuplas são estruturas de dados imutáveis e heterogêneos.
- Seu uso difere de listas, especialmente em contexto.
- Uma tupla pode ser criada com seus valores entre parênteses ou pelo construtor ***tuple()***.
- Tuplas podem ser criadas com dados nomeados pelo uso da função ***namedtuple()***.
- Dicionários são estruturas de dados não ordenados mutáveis.
- Dicionários são compostos por conjuntos de pares chave-valor.
- Um valor do dicionário pode ser obtido pelo nome do dicionário seguido da chave entre colchetes.
- É possível verificar se uma chave existe ou não em um dicionário usando o comando ***in***.
- É possível verificar um valor em um dicionário usando o comando ***in*** sobre seu método ***values()***.
- Dicionários podem ser modificados a partir de métodos próprios.

Referências

BANIN, S. L. **Python 3**: conceitos e aplicações uma abordagem didática. São Paulo: Érica, 2018.

PYTHON SOFTWARE FOUNDATION. **Documentação Python 3.9.0**. Disponível em: https://docs.python.org/pt-br/3/. Acesso em: 27 out. 2020.