

# AULA 06

LINGUAGEM DE PROGRAMAÇÃO II - LP212

# ESTRUTURAS DE DADOS TUPLAS E DICIONÁRIOS

*PROFA. ANA PAULA MÜLLER GIANCOLI*

Instituto Federal de São Paulo - IFSP

Campus Bragança Paulista

# AGENDA

Tuplas

Dicionários

Exemplos



```
self.file.  
self.fingerprints.  
  
def request_seen(self, request)  
    fp = self.request_fingerprint  
    if fp in self.fingerprint  
        return True
```

**PARA APROFUNDAMENTO -  
CONSULTE O PEP308:**

[HTTPS://WWW.PYTHON.ORG/DEV/PEPS/PEP-0308/](https://www.python.org/dev/peps/pep-0308/)



# TUPLAS

# TUPLAS

PADRÃO

- Uma **tupla** é uma sequência ordenada de **zero** ou mais referências de objetos.
- **Tuplas** suportam o mesmo fatiamento e mesmo acesso da sintaxe de **strings**.

# TUPLAS

PADRÃO

- Assim como as strings, as **tuplas** são **imutáveis**, logo não é permitido substituir ou deletar nenhum de seus elementos.
- Caso seja necessário modificar alguma sequência ordenada, devemos utilizar uma **lista** ao invés de uma **tupla**.

# TUPLAS

PADRÃO

- A  **tupla**  poderá ser convertida para uma  **lista**  por meio da  **função list()** , e depois aplicado as mudanças necessárias na lista.
- É uma estrutura de dado, representada por uma sequência, fechada por  **parênteses (... )** , sendo que o primeiro elemento está na posição zero,  **índice zero** .

# TUPLAS

PADRÃO

- As **tuplas** fornecem apenas dois métodos:
  - **t.count(x)**: retorna o número de vezes que o objeto x ocorre na tupla.
  - **t.index(x)**: retorna a posição de índice da primeira ocorrência x a partir da esquerda para a direita.

Gerará um erro **ValueError** caso não exista o x.



# TUPLAS

PADRÃO

```
>>> x = (1, 2, 3)
```

```
>>> type(x)
```

```
<class 'tuple'>
```

```
>>> y = list(x)
```

```
>>> x
```

```
(1, 2, 3)
```

```
>>> y
```

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

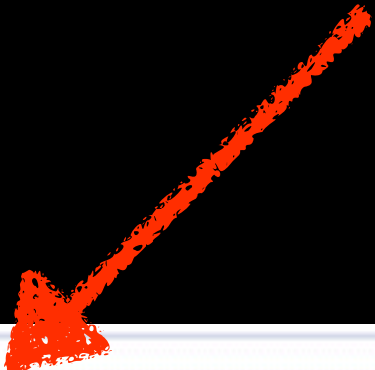
```
> |
```

# TUPLAS

PADRÃO

- *É possível alterar um objeto dentro de uma tupla, desde que este objeto seja mutável, modificável.*

- Exemplo:



```
>>> t = ('Ana', ['Muller', 'Giancoli'], 10, 'Paula', 30)
>>> t[1][1] = 'Muller Giancoli'
>>> t
('Ana', ['Muller', 'Muller Giancoli'], 10, 'Paula', 30)
```

The code demonstrates tuple immutability by attempting to modify a mutable object (a list) inside a tuple. The initial tuple is `('Ana', ['Muller', 'Giancoli'], 10, 'Paula', 30)`. The list element at index 1 is modified from `['Muller', 'Giancoli']` to `['Muller', 'Muller Giancoli']`. The final tuple is `('Ana', ['Muller', 'Muller Giancoli'], 10, 'Paula', 30)`. The indices 0 and 1 are circled in the output to show the positions of the modified list and its first element.

# TUPLAS

PADRÃO

- Declarações de Tuplas:
  - Sem uso de parênteses: **tz = 1, 2, 3**
  - Com uso de parênteses: **tz = ( 1, 2, 3 )**
  - **\* Importante:** Com um único elemento: **tz = 2,**
  - Vazia: **tz = ( )**

# TUPLAS

PADRÃO

- Por que usar Tuplas?
- São mais rápidas que as listas.
- São usadas na formatação de strings.
- Para uso de elementos constantes.

# DICIONÁRIOS



# DICIONÁRIOS

PADRÃO

- Um **Dict** é uma coleção desordenada de **zero** ou mais pares de chave-valor, no qual as **chaves** são referência de objetos referidos a objetos de qualquer tipo.
- **Dicionários** são **mutáveis**.
- Podemos adicionar ou remover itens, elementos.
- Como são desordenados, não temos a informação da posição, do índice, logo, **não podem ser fatiados**.

# DICIONÁRIOS

PADRÃO

- Podem ser criados utilizando **chaves vazias** `{ ... }`.
- **Chaves não vazias** podem conter um ou mais itens separados por vírgulas, em que cada qual consiste numa **chave**, o literal **dois pontos** e um **valor**.

```
k = { chave1 : valor1, chave2: valor2, ..., chaveN : valorN }
```

# DICIONÁRIOS

PADRÃO

```
>>> g = {}

>>> type(g)

<class 'dict'>

>>> k = {'nome': 'Ana', 'sobrenome': 'Giancoli'}

>>> k['sobrenome']

'Giancoli'

>>> 'Giancoli' in k

False

>>> 'sobrenome' in k

True

> |
```

# DICIONÁRIOS

PADRÃO

- Podem ser criados utilizando **construtor dict()**.
- Pode receber 2 tipos de parâmetros:
  - **Listas de tuplas**, sendo que cada tupla contém uma chave e conteúdo.
  - **Sequências de itens** no formato chave=valor.

# DICIONÁRIOS - LISTA DE TUPLAS

## PADRÃO

- Uma **lista de tuplas**, onde cada tupla da lista contém uma chave e conteúdo, separados por vírgula.

```
>>> dict([('Nina', 1223), ('Pedro', 3423), ('Bill', 4565)])  
{'Nina': 1223, 'Pedro': 3423, 'Bill': 4565}  
> |
```



# DICIONÁRIOS - LISTA DE TUPLAS

PADRÃO

- Dado um dicionário composto por uma lista de tuplas.
- Qual é o valor da chave **40.9** ? **Como acessar?**

```
>>> itens = dict([(40.9, 10), ('leite', 2), ('abacaxi', 8), (12, 4.88)])  
>>> num = itens[40.9]  
>>> num  
10  
> |
```

# DICIONÁRIOS - LISTA DE TUPLAS

## PADRÃO

- Dado um dicionário composto por uma lista de tuplas.
- Qual é o valor da chave **abacaxi** ? **Como acessar?**

```
>>> itens = dict([(40.9, 10), ('leite', 2), ('abacaxi', 8), (12, 4.88)])  
>>> fruta = itens['abacaxi']  
>>> fruta  
8  
> |
```

# DICIONÁRIOS - LISTA DE TUPLAS

## PADRÃO

- Dado um dicionário composto por uma lista de tuplas.
- Qual é o valor da chave **leite** ? **Como acessar?**

```
>>> itens = dict([(40.9, 10), ('leite', 2), ('abacaxi', 8), (12, 4.88)])
>>> laticinio = itens['leite']
>>> laticinio
2
> |
```

# DICIONÁRIOS - SEQUÊNCIA DE PARES

## PADRÃO

- A partir de **sequências de pares de chave-valor**.
- Nesse caso as chaves precisam ser strings, mas são escritas sem aspas.

```
>>> dict(ana = 4139, giancoli = 4127, paula = 4098)
{'ana': 4139, 'giancoli': 4127, 'paula': 4098}
> |
```

# DICIONÁRIOS - SEQUÊNCIA DE PARES

## PADRÃO

- A partir de **sequências de pares de chave-valor**. Nesse caso as chaves precisam ser strings, mas são escritas sem aspas. Caso contrário, ocorrerá erro:

**SyntaxError: keyword can't be an expression**

```
>>> prod = dict(10 = 23.4, 20 = 20)
```

```
File "<stdin>", line 1
```

```
SyntaxError: keyword can't be an expression
```

```
> |
```



# DICIONÁRIOS

## PADRÃO

- Criar um dicionário, onde a chave é o número a ser elevado ao quadrado, e o valor, é o seu quadrado.
- Dada a tupla:  $t=(2,4,6,8)$

```
>>> {x: x**2 for x in (2, 4, 6, 8)}
```

```
{2: 4, 4: 16, 6: 36, 8: 64}
```

```
> |
```

# MÉTODOS COM DICIONÁRIOS



# MÉTODOS DE DICIONÁRIOS

FROMKEYS(LISTA, VALOR)

Serve para:

Retornar um novo dicionário cujas **chaves** são os *elementos de lista* e cujos **valores** são *todos iguais a valor*.

Se **valor** não for especificado, o **default** é **none**.

```
>>> nomes = {}  
  
>>> nomes.fromkeys([4, 2])  
  
{4: None, 2: None}  
  
>>> nomes.fromkeys(["Ana", "Paula"], 30)  
  
{'Ana': 30, 'Paula': 30}  
  
> |
```

# MÉTODOS DE DICIONÁRIOS

GET(CHAVE, VALOR)

```
>>> dias = {"Paula": [10, 30], "Ana": [15], "Muller": [2]}
```

```
>>> dias
```

```
{'Paula': [10, 30], 'Ana': [15], 'Muller': [2]}
```

```
>>> dias.get("Paula")
```

```
[10, 30]
```

```
>>> dias.get("Ana")
```

```
[15]
```

```
>>> dias.get("Muller")
```

```
[2]
```

```
>>> dias.get("Giancoli", "Dias não encontrados.")
```

```
'Dias não encontrados.'
```

```
> |
```

## Serve para:

Obtém o conteúdo associado à chave. Podemos passar dois parâmetros (chave e um valor padrão para retornar caso essa chave não seja encontrada).

- Se a **chave** existe, retorna **valor**.
- Se a **chave** não for especificada, as chamadas de **get** para **chaves inexistentes** retornam **none** ou **valor padrão**.

# MÉTODOS DE DICIONÁRIOS

IN

Serve para:

Retornar **True** se **chave** pertence ao dicionário e **False** caso contrário.

```
>>> dias = {"Paula": [10, 30], "Ana": [15], "Muller": [2]}
```

```
>>> "Giancoli" in dias
```

```
False
```

```
>>> "Muller" in dias
```

```
True
```

```
> |
```



# MÉTODOS DE DICIONÁRIOS

D.ITEMS( )

Serve para:

Retornar uma lista com todos os pares **chave-valor** do dicionário no formato de **tupla**.

```
>>> dias = {"Paula": [10, 30], "Ana": [15], "Muller": [2]}  
>>> dias.items()  
dict_items([('Paula', [10, 30]), ('Ana', [15]), ('Muller', [2])])  
> |
```

# MÉTODOS DE DICIONÁRIOS

D.KEYS( )

Serve para:

Retornar uma **lista** com todas as **chaves** do dicionário.

```
>>> dias = {"Paula": [10, 30], "Ana": [15], "Muller": [2]}  
>>> dias.keys()  
dict_keys(['Paula', 'Ana', 'Muller'])  
> |
```

# MÉTODOS DE DICIONÁRIOS

D.VALUES( )

Serve para:

Retornar uma **lista** com todos os **valores** do dicionário.

```
>>> dias = {"Paula":[10, 30], "Ana":[15], "Muller": [2]}  
>>> dias.values()  
dict_values([[10, 30], [15], [2]])  
> |
```

# MÉTODOS DE DICIONÁRIOS

D.POP(CHAVE)

Serve para:

Remover o par **chave-valor** do elemento correspondente à **chave** do dicionário.

Caso não exista, ocorrerá erro **KeyError**.

```
>>> dias = {"Paula": [10, 30], "Ana": [15], "Muller": [2]}
```

```
>>> dias.pop("Muller")
```

```
[2]
```

```
>>> dias
```

```
{'Paula': [10, 30], 'Ana': [15]}
```

```
>>> dias.pop("Muller")
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
KeyError: 'Muller'
```

```
> |
```

# MÉTODOS DE DICIONÁRIOS

D.POPITEM( )

Serve para:

Remover o par **chave-valor** de um elemento *aleatório* do dicionário.

```
>>> dias = {"Paula": [10, 30], "Ana": [15]}
>>> dias.popitem()
('Ana', [15])
> |
```

# MÉTODOS DE DICIONÁRIOS

## Serve para:

Uma forma de adicionar um elemento ao dicionário é utilizando a sua chave e atribuindo o conteúdo a ela.

Exemplo:

```
>>> dias = {'Paula': [10, 30], 'Ana': [15]}
>>> dias
{'Paula': [10, 30], 'Ana': [15]}
>>> dias['Muller'] = 10, 30, 40
>>> dias
{'Paula': [10, 30], 'Ana': [15], 'Muller': (10, 30, 40)}
>>>
> |
```

# MÉTODOS DE DICIONÁRIOS

D.UPDATE(X)

## Serve para:

Atualizar um dicionário com elementos de outro. Ou adicionar mais elementos.

Os itens em **X** (dicionário) são adicionados um a um ao dicionário original (**D** - que foi usado para chamar a função update).

```
>>> d = {"a": 1, "b": 2, "c": 3}
>>> x = {"z": 10, "y": 20}
>>> d.update(x)
>>> d
{'a': 1, 'b': 2, 'c': 3, 'z': 10, 'y': 20}
> |
```



# MÉTODOS DE DICIONÁRIOS

## ITERANDO COM DICIONÁRIOS

Serve para:

A iteração em elementos de um dicionário é feita a partir da chave. *Lembre-se de que com dicionários não temos ordem pré-definida.*

```
>>> notas = {"Pedro": [9.0, 5.0], "Eduardo": [2.0, 7.5], "Kelly": [3.5, 6.0]}
>>> for n in notas:
...     media = sum(notas[n])/len(notas[n])
...     print(f'A média de {n} é {media}.')
...
A média de Pedro é 7.0.
A média de Eduardo é 4.75.
A média de Kelly é 4.75.
> |
```

The background of the slide is a dark grey or black color, overlaid with a blurred image of Python code. The code is written in a monospaced font with syntax highlighting: keywords like 'self', '@classmethod', 'def', 'if', and 'return' are in a light blue or cyan color, while other parts of the code are in a light green or yellow color. The code is out of focus, creating a bokeh effect. In the top right corner, there is a partial view of a circular logo with a blue outer ring and a yellow inner shape. In the bottom left corner, there is another partial view of a similar circular logo, also with a blue outer ring and a yellow inner shape.

IMPORTANTE

# IMPORTANTE

## ITERANDO COM DICIONÁRIOS

Para recuperar a chave-valor ao mesmo tempo na iteração de um dicionário, poderemos executar da seguinte forma:

```
>>> notas = {"Pedro": [9.0, 5.0], "Eduardo": [2.0, 7.5], "Kelly": [3.5, 6.0]}
>>> for k, v in notas.items():
...     print(k, v)
...
Pedro [9.0, 5.0]
Eduardo [2.0, 7.5]
Kelly [3.5, 6.0]
> |
```

# IMPORTANTE

## ITERANDO COM DICIONÁRIOS

Ao percorrer uma sequência, o índice de posição e o valor correspondente podem ser recuperados ao mesmo tempo usando a função `enumerate()`.

```
>>> for i, v in enumerate(['Batman', 'Coringa', 'Bill']):  
...     print(i,v)  
...  
0 Batman  
1 Coringa  
2 Bill  
> |
```



# EXERCÍCIOS-EXEMPLOS

# AULA06\_EXEMPLO01

- Escreva uma programa que conta a quantidade de vogais em um texto e armazena tal quantidade em um dicionário, onde a chave é a vogal considerada.



## AULA06\_EXEMPLO02

- Escreva um programa que lê duas notas de vários alunos e armazena tais notas em um dicionário, onde a chave é o nome do aluno. A entrada de dados deve terminar quando for lida uma string vazia como nome. Exiba a média do aluno, dado seu nome (solicite o nome a ser verificado).



## AULA06\_EXEMPLO03

- Uma pista de Kart permite 5 voltas para cada um de 3 corredores. Escreva um programa que leia todos os tempos em segundos e os guarde em um dicionário, onde a chave é o nome do corredor.
- Ao final diga de quem foi a melhor volta da prova e em que volta; e ainda a classificação final em ordem (1o o campeão). O campeão é o que tem a menor média de tempos.

# OUTROS



# EXEMPLO I

```
>>> agenda = {"Ana": [99887766, 99887755], "Paula": [92345678],  
               "Maria": [99887711, 99665533]}  
  
>>> celAna = agenda["Ana"]  
  
>>> celAna  
[99887766, 99887755]  
  
> |
```

- E o celular da Maria?
- E da Paula?

# EXEMPLO I

```
>>> agenda = {"Ana": [99887766, 99887755], "Paula": [92345678],  
               "Maria": [99887711, 99665533]}  
  
>>> celMaria = agenda["Maria"][0]  
  
>>> celMaria  
  
99887711  
  
> |
```



# EXEMPLO I

```
>>> agenda = {"Ana": [99887766, 99887755], "Paula": [92345678],  
              "Maria": [99887711, 99665533]}  
  
>>> celPaula = agenda["Paula"]  
  
>>> celPaula  
[92345678]  
  
> |
```

PERGUNTAS ?



# REFERÊNCIAS

<http://python.org.br>

[https://www.programiz.com/python-programming/methods/  
dictionary](https://www.programiz.com/python-programming/methods/dictionary)