



Algoritmos e Programação de Computadores

Tuplas e Dicionários

Agenda

— — —

- Tuplas
- Dicionários
 - Operações
 - Métodos
- Exemplo


Tuplas

- Tuplas são uma sequência de elementos separados por vírgulas, representados ou não entre parênteses, isto é, os parênteses não são obrigatórios.
- Pode-se ainda misturar elementos de tipos diferentes.
- Porém, ao contrário de listas, as **tuplas são imutáveis**.
- Exemplo: (18, "abril", 9.5, 1) é uma tupla de 4 elementos.

Tuplas

- Mais exemplos de tuplas.

```
tupla1 = ('abril', 18, 4, 2018)
tupla2 = (1, 2, 3, 4, 5, 6, 7)
tupla3 = "a", "b", "c", "d"
tupla4 = ("MC102", )
tupla5 = ()
```



tupla4 representa uma tupla com um único elemento. A vírgula após o elemento é necessária para diferenciar de uma expressão entre parênteses.

Tuplas

- O que será impresso?

```
t1 = 'A',  
t2 = ('A')  
print(type(t1))  
print(type(t2))
```

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

Tuplas

- Como strings, tuplas são **imutáveis**.

```
a = (3, "maio", 9.5, 1)
```

```
a[2] = 9.0
```

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

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

```
TypeError: 'tuple' object does not support item assignment
```

- A utilidade de uma lista imutável ficará mais clara na aula de hoje, na seção de dicionários.

Tuplas: Acessando Valores

- As operações para acessar os elementos ou sub-sequências de um lista e de uma string, também funcionam em tuplas.

```
a = (3, "maio", 9.5, 1)
```

```
a[2]
```

```
9.5
```

```
a[1:3]
```

```
("maio", 9.5)
```

Tuplas: Empacotamento e Desempacotamento

- Os elementos de uma tupla podem ser acessados de uma forma implícita na atribuição (conhecido como desempacotamento).

```
x, y = (18, 20)
```

```
x
```

```
18
```

```
y
```

```
20
```


Tuplas: Empacotamento e Desempacotamento

- A tupla também pode ser implicitamente criada apenas separando os elementos por vírgula (conhecido como empacotamento).

```
18, 20  
(18, 20)
```

```
"maio", 9.5  
( 'maio', 9.5)
```

Tuplas: Empacotamento e Desempacotamento

```
nome, sobrenome, turma = input().split()
```

```
Sandra Avila MC102KLMN
```

- O que será impresso?

```
nome, sobrenome, turma
```

```
('Sandra', 'Avila', 'MC102KLMN')
```

Dicionários

Dicionários

- Dicionários são estruturas de dados que associam uma chave com um valor.
- Os valores podem ser um dado de qualquer tipo, mas **as chaves só podem ser dados de tipos imutáveis**.
- As chaves precisam ser únicas.
- Um tipo dicionário é escrito da seguinte forma:

Um dicionário é denotado por { }.

```
dicionario = {chave1: valor1, ..., chaveN: valorN}
```

Dicionários

- Dicionários são estruturas de dados que associam uma chave com um valor.
- Os valores podem ser um dado de qualquer tipo, mas **as chaves só podem ser dados de tipos imutáveis**.
- As chaves precisam ser únicas.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}  
print(type(ra))  
<class 'dict'>
```

Dicionários


- O dicionário abaixo pode representar os RAs dxs alunxs, com o nome (uma string, que é imutável) como chave e o valor associado a cada chave é o RA (um inteiro).
- Acessar o valor associado a uma chave é feito como no exemplo:

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}  
ra["Liz"]  
229874  
ra["Sofia"]  
199745
```

Dicionários

- O valor associado a uma chave pode ser modificado, ou uma nova chave (e seu valor) podem ser incluídos no dicionário.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}  
ra  
{'Hugo': 215793, 'Liz': 229874, 'Sofia': 199745}
```



Um dicionário é uma coleção não ordenada de pares chave-valor.

Dicionários

- O valor associado a uma chave pode ser modificado, ou uma nova chave (e seu valor) podem ser incluídos no dicionário.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}
ra
{'Hugo': 215793, 'Liz': 229874, 'Sofia': 199745}
ra['Hugo'] = 215739
ra['Diego'] = 193278
ra
{'Diego': 193278, 'Hugo': 215739, 'Liz': 229874,
'Sofia': 199745}
```


Operações em Dicionários

- O laço **for** aplicado a um dicionário faz a variável do laço passar por todas as **chaves** do dicionário.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}
```

```
for x in ra:  
    print(x)
```

Liz

Hugo

Sofia

Operações em Dicionários

- O operador **in** verifica se uma **chave** está no dicionário.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}
```

```
"Sofia" in ra
```

```
True
```

```
"Aline" in ra
```

```
False
```

Operações em Dicionários

- Acessar uma chave que não existe causa erro de execução.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}  
ra['José']
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
KeyError: 'José'
```

Métodos em Dicionários

- `items()` retorna todos os pares chave/conteúdo do dicionário.
- `keys()` retorna todas as chaves do dicionário.
- `values()` retorna todos os valores do dicionário.

Métodos em Dicionários

- `items()` retorna todos os pares chave/conteúdo do dicionário.
- `keys()` retorna todas as chaves do dicionário.
- `values()` retorna todos os valores do dicionário.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}
ra.items()
dict_items([('Liz', 229874), ('Hugo', 215793), ('Sofia', 199745)])
ra.keys()
dict_keys(['Liz', 'Hugo', 'Sofia'])
ra.values()
dict_values([229874, 215793, 199745])
```

Métodos em Dicionários

- `items()` retorna todos os pares chave/conteúdo do dicionário.
- `keys()` retorna todas as chaves do dicionário.
- `values()` retorna todos os valores do dicionário.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}
list(ra.items())
[('Liz', 229874), ('Hugo', 215793), ('Sofia', 199745)]
list(ra.keys())
['Liz', 'Hugo', 'Sofia']
list(ra.values())
[229874, 215793, 199745]
```

Métodos em Dicionários

- O método `get (chave)` retorna o valor atribuído à chave.

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}
```

```
print(ra.get("Hugo"))
```

```
215793
```

```
print(ra.get("Maria"))
```

```
None
```

```
print(ra.get("Maria", "N/A"))
```

```
N/A
```

Iterando em Dicionários

- Ao fazer uma iteração sobre dicionários, a chave e o valor correspondente podem ser recuperados ao mesmo tempo usando o método `items()`:

```
ra = {"Liz": 229874, "Hugo": 215793, "Sofia": 199745}
```

```
for nome, numero in ra.items():  
    print(nome, numero, sep=' ')
```

```
Liz 229874
```

```
Hugo 215793
```

```
Sofia 199745
```


Exemplo

Contando Letras

- Faça uma função que dada uma string, retorna a letra mais comum nessa string (em caso de empate retorne qualquer uma das mais frequentes).
 - Ideia: usar um dicionário para contar cada letra.
 - A letra é a chave do dicionário, e o valor será quantas vezes a letra foi encontrada.

Contando Letras

```
string = input("Digite uma string: ")
conta = {} # dicionário vazio
for letra in string:
    if letra in conta:
        conta[letra] += 1
    else:
        conta[letra] = 1 # adiciona letra no dicionário
letramais = ''
for chave in conta:
    if letramais == '': # nenhuma mais comum ainda
        letramais = chave
    elif conta[chave] > conta[letramais]:
        letramais = chave
print(letramais)
```

Referências & Exercícios

- Os slides dessa aula foram baseados no material de MC102 do Prof. Eduardo Xavier (IC/Unicamp)
- <https://panda.ime.usp.br/pensepy/static/pensepy/11-Dicionarios/dicionarios.html>