Programação Python

Aula 04: Tuplas, Conjuntos e Dicionários

Prof. Eduardo Corrêa Gonçalves

24/03/2021

Sumário

Tupla (1/6)

- **Tupla (*Tuple*) - Definição**
  - ED idêntica à lista (coleção ordenada de elementos), porém **imutável**.
    - Isso significa que:
      - Você **não** pode adicionar, remover ou alterar elementos.
      - Mas pode recuperar elementos (pois essa operação não altera a tupla).

  - Importante para:
    - Definir **constantes**.
    - Trabalhar com **dados protegidos**.
    - Trabalhar com a técnica de **atribuição múltipla**.

Tupla (2/6)

- **Criando Tuplas**
  - Devemos especificar uma sequência de valores entre parênteses, onde os valores devem estar separados por vírgula:

```
t1 = (10, 20, 30, 40, 50)
t2 = (100,)              # tupla com um único elemento: usa-se vírgula no final
t3 = 10, 20, 30, 40, 50  # se tupla tem mais de 1 elemento,
                         # posso omitir os parênteses
tupla_vazia = tuple()
tupla_mista = ('Pen Drive', 25.90, 'Laptop', 2690)
m1 = ( (1, 2, 3), (4, 5, 6) )    #tupla de tuplas
m2 = ( [1, 2, 3], [4, 5, 6] )    #tupla de listas
```

Tupla (3/6)

- **Operações – qualquer coisa que não modifique a tupla**

```
t = ("John", "Yoko", "Julian", "Sean")
t[1]                    # ‘Yoko‘ –  veja que utilizamos [ ] para acessar item.
t[1:3]                  # ('Yoko', 'Julian’)
"Paul" in t             # False
type(t)                 # <class ‘tuple’>
t.index("Julian")   # 2


# não pode modificar ...
t[3]='Paul’
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Tupla (4/6)

- **E se eu quiser modificar uma tupla?**
  - A tupla é imutável. Depois de criada, nenhum elemento pode ser alterado.
    - Para alterar uma tupla, só atribuindo um novo valor para a mesma.

```
>>> t = (5, 9, 1, 7, 3)    # como ordenar essa tupla?? Faremos em 3 passos
>>>
>>> lst = list(t)          # passo 1: cria uma lista a partir da tupla, usando list()
>>> lst
[5, 9, 1, 7, 3]
>>> lst.sort()             # passo 2: ordenamos a lista criada.
>>>
>>> t = tuple(lst)         # passo 3: transformo a lista ordenada em tupla,
                           # usando tuple() e atribuindo o resultado a “t”.
>>> t
(1, 3, 5, 7, 9)
```

Tupla (5/6)

- **E se eu quiser modificar uma tupla?**
  - Caso um dos elementos da tupla for mutável (ex: uma lista), esse elemento específico pode ser alterado (mas não removido da tupla).

```
>>> t = (5, [10, 20], 100)
>>> t[1].append(30)              # Ok, posso mexer no segundo elemento...
>>> t
(5, [10, 20, 30], 100)

>>> t[2] = 1000                  # Mas não posso mexer nos outros!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> t[0] = 1000
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Tupla (6/6)

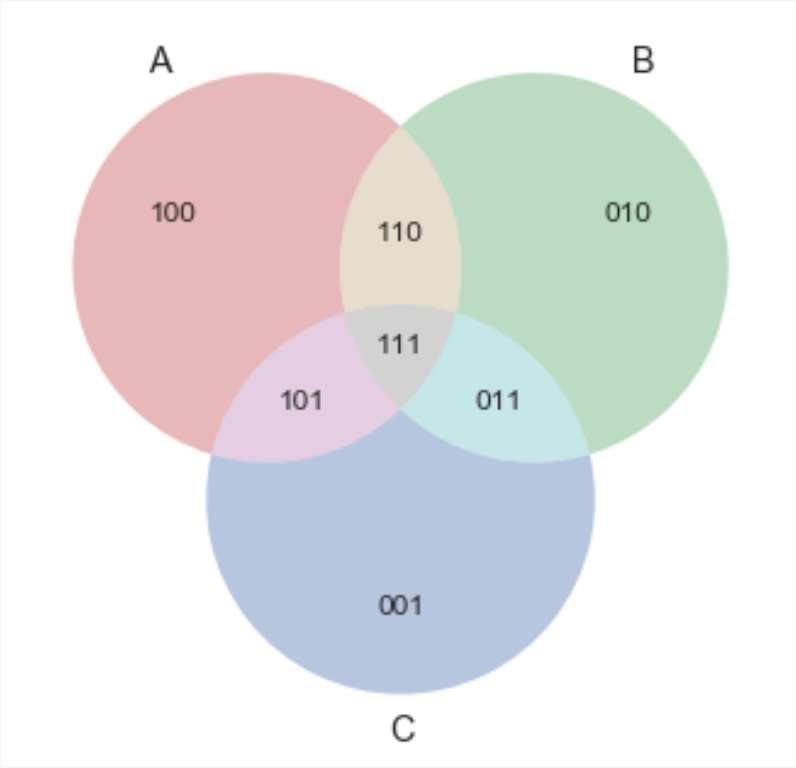
- **Atribuição Simultânea**
  - Também chamada de **tuple assignment**
  - **Exemplo 1:**
    - x, y, z = 6, 2, 5
      - Como o Python processa?
        - Primeiro, o lado direito da atribuição é automaticamente empacotado em uma tupla.
        - Depois, é automaticamente desempacotado e seus elementos são atribuídos às variáveis “x”, “y” e “z”
        - No final, temos x=6, y=2, z=5
  - **Exemplo 2**: código para trocar o conteúdo de duas variáveis

```
#sem atribuição simultânea
temp = a
a = b
b = temp
```

```
#com atribuição simultânea
a, b = b, a
```

Conjunto (1/7)

- **Conjunto (*Set*) - Definição**
  - Esta ED implementa o mesmo conceito de **conjunto matemático**:
    - coleção não-ordenada de elementos distintos.



  - Características:
    - **Não** permite elementos **duplicados**.
    - **Mutável** – pode ser alterado.
    - **Iterável** – capaz de retornar seus elementos um por vez em um laço.
    - **Não é sequência** – elementos não possuem ordem determinada (**não** possuem índice).

Conjunto (2/7)

- **Criando Conjuntos**
  - Devemos especificar uma sequência de valores entre chaves, onde os valores devem estar separados por vírgula:
    - generos = {'Drama', 'Romance', 'Ação', 'Aventura'}
    - numeros = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
    - conjunto_vazio = **set**()
  - Se tentarmos criar um conjunto com elementos repetidos, o Python removerá as repetições automaticamente:

```
>>> x = {1, 1, 2, 1, 2, 2}
>>> x
{1, 2}
```

Conjunto (3/7)

- **Operações básicas**:
  - Pertence / Não Pertence
  - Iterar (percorrer todos os elementos - *não há como garantir a ordem*)
  - União, Interseção, Diferença, Diferença Simétrica
  - Contém / Está Contido
  - Inserir / Remover elementos

  - **Observações**:
    - Note que a maior parte das operações consiste nas tradicionais operações de conjunto da matemática.

    - **Não temos indexação** e **nem fatiamento**, pois os elementos dos conjuntos não possuem índice.

Conjunto (4/7)

- **Pertence (in) e Não Pertence (not in)**

```
familia = {"John", "Yoko", "Julian", "Sean"}

"Julian" in familia        # True      (in = Pertence)
"Ringo" in familia         # False
"Ringo" not in familia     # True      (not in = Não Pertence)
```

- **Obtendo as propriedades de um conjunto**

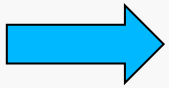
```
# tipo do objeto conjunto
type(familia)                    # <class 'set'>

# cardinalidade do conjunto
# (número de elementos)
len(familia)                     # 4
```

Conjunto (5/7)

- **Iterando**
  - Resultado podem vir em **qualquer ordem**, pois não existe o conceito de ordenação em conjuntos.
  - Não existe iteração por índice, pois os elementos não possuem índice.

```
familia = {"John", "Yoko", "Julian", "Sean"}
for pessoa in familia:
    print(pessoa)
```



```
Julian
Yoko
John
Sean
```

Conjunto (6/7)

- **Inserindo e Removendo Elementos**

```
c = {"John", "Yoko"}
d = {"Julian", “Sean"}
lst = ["Ringo", "Stu", "Pete"]

c.add("Paul")          # insere “Paul”: {'Paul', 'Yoko', 'John’}
c.add("George")        # insere “George”: {'Paul', 'Yoko', 'George', 'John'}
c.remove("Yoko")       # remove “Yoko”: {'Paul', 'George', 'John’}
c.update(d)            # insere os elementos de d em c:
                       # {'Sean', 'Julian', 'John', 'George', 'Paul’}
c.pop()                # remove um elemento de forma aleatória:
                       # {'Julian', 'John', 'George', 'Paul’}
c.update(lst)          # insere os elementos da lista lst em c:
                       # {'Pete', 'Julian', 'Ringo', 'John', 'Stu', 'George', 'Paul'}

c.clear()              # esvazia o conjunto
```

Conjunto (7/7)

- **União (|), Interseção (&), Diferença (-) e Diferença Simétrica (^)**

```
a = {0,1,2,3}
b = {2,3,4,5}

a | b        # União: {0,1,2,3,4,5}
a & b        # Interseção: {2,3}
a - b        # Diferença: {0,1}
a ^ b        # Diferença Simétrica: {0,1,4,5}
```

- **Contém (>=) e Está Contido (<=)**

```
c = {1,2}
a >= c       # True
c <= a       # True
c <= b       # False
```

Resumo – Conjunto

- **Pontos fortes**:
  - Extremamente simples de usar
  - Suporta as operações básicas com conjuntos.
  - Busca por elemento tem tempo constante e não depende da cardinalidade do conjunto. A busca rápida garante velocidade para outras operações (interseção, diferença, etc.).
- **Pontos fracos**:
  - Não é tão flexível e nem tem tantas operações como a lista.
- **Quando usar?**:
  - Quando você tiver um grupo não ordenado de valores únicos.
  - Quando as operações que fazem sentido sobre os seus dados são as operações clássicas com conjuntos: união, interseção, pertence/não pertence, etc.

Tarefa

(1) - Sejam 2 conjuntos:

exame = {'Bella', 'Edward', 'Renesmee'} (*estudantes que fizeram o exame*)

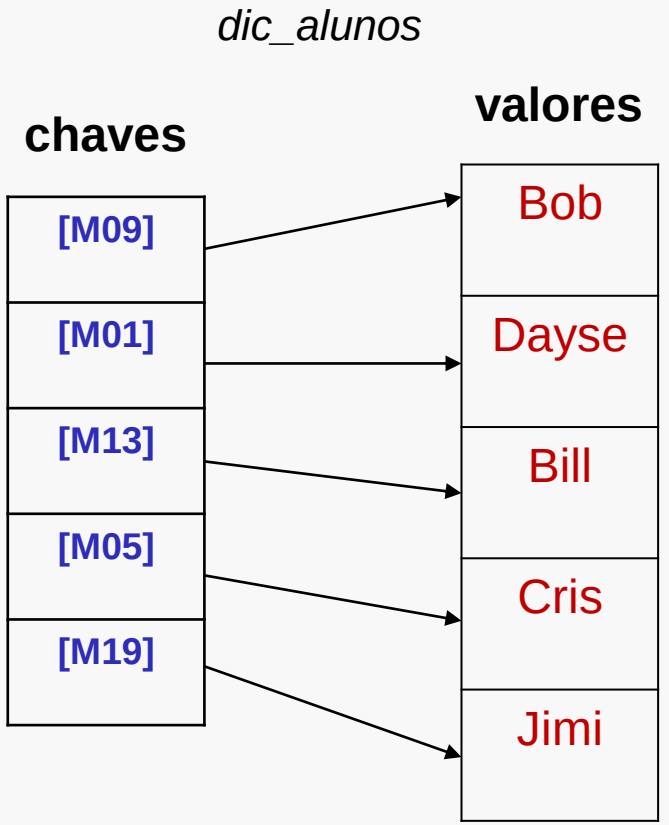
projeto = {'Jacob', 'Carlisle', 'Alice', 'Bella', 'Esme', 'Edward'} (*estudantes que submeteram um projeto*)

Utilize as operações de conjunto para resolver as questões abaixo:

- Quais estudantes realizaram o exame e submeteram um projeto?
- Que estudantes apenas realizaram o exame?
- Que estudantes apenas submeteram o projeto?
- Listar os nomes de todos os estudantes.
- Listar todos os estudantes que apenas realizaram o exame ou apenas submeteram o projeto (mas não as duas coisas juntas)

Dicionário (1/11)

- **Dicionário (*Dictionary*) – Definição:**
  - ED em que elementos são **pares chave:valor**.
    - A chave (*key*) identifica um item e o valor armazena o conteúdo do mesmo.
    - Qualquer valor pode ser recuperado de forma **extremamente rápida** por sua chave
      - A *chave é hasheada...*



  - Características:
    - As **chaves** devem ser **únicas**, mas podem existir **valores duplicados**.
    - **Mutável** – pode ser alterado.
    - **Iterável** – capaz de retornar seus elementos um por vez em um laço.
    - **Não é sequência** – elemento não pode ser indexado por posição.

Dicionário (2/11)

- **Criando Dicionários**
  - Devemos especificar, entre chaves, uma relação de elementos do tipo chave:valor separados por vírgula:
    - dic_alunos = {"M09":"Bob",
      "M01":"Dayse",
      "M13":"Bill",
      "M05":"Cris",
      "M19":"Jimi"}
    - dic_titulos = {'Portela':22,'Mangueira':19,'Beija-Flor':14}
    - dic_vazio = **dict**() # Também pode ser dic_vazio = { }
    -

Dicionário (3/11)

- **Criando Dicionários**
  - Também é possível criar dicionários aninhados. Neste exemplo, cada chave (f1, f11 e f8) está associada a uma tupla com 5 elementos.

```
>>> filmes= {
"f1": ( "O Filho da Nova", 2001, "AR", 123, 7.9),
"f11": ( "Orgulho e Preconceito", 2005, "UK", 129, 7.8),
"f8": ( "Um Conto Chinês", 2011, "AR", 93, 7.3)
}
```

  - Chaves devem ser únicas, mas valores podem ser duplicados.

```
>>> food = {"bacon" : "yes", "egg" : "yes", "spam" : "no" }
```

Dicionário (4/11)

- **Operações básicas**:
  - Recuperar o valor de uma chave
  - Adicionar **entrada** (*par chave:valor*)
  - Modificar o valor de uma chave.
  - Pesquisar se chave existe
  - Remover **entrada** (*par chave:valor*)
  - Recuperar todas as chaves ou todos os valores.
  - Iterar pelas chaves, valores ou pares chave:valor

- **Obs1**: **Não temos indexação** e **nem fatiamento**, pois os elementos dos dicionários são indexados por chaves e não por posição.

- **Obs2**: O termo **entrada** (*entry*) também é utilizado para designar um **par chave:valor**.
  - Esse termo é originário dos dicionários tradicionais. https://www.macmillandictionary.com/learn/dictionary-entry.html

Dicionário (5/11)

- **Recuperando o valor de uma chave**

```
dic = {"mae":"Yoko", "pai":"John", "filho":"Sean"}

dic["mae"]          # 'Yoko'
dic['filho']        #'Sean'

>>> dic["primo"]    # Ocorre erro se especifico chave inexistente.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'primo'
```

- **Adicionando uma nova entrada (elemento chave:valor)**

```
dic["enteado"] = "Julian"  #{'mae': 'Yoko', 'pai': 'John', 'filho': 'Sean', 'enteado': 'Julian'}
```

- **Modificando o valor de uma chave**

```
dic["pai"] = "John Lennon"   #{'mae': 'Yoko', 'pai': 'John Lennon', 'filho': 'Sean',
                               'enteado': 'Julian'}
```

Dicionário (6/11)

- **Inserindo e removendo entradas**

```
dic1 = {"mae":"Yoko", "pai":"John", "filho":"Sean"}
dic2 = {"bateria": "Ringo", "baixo": "Paul", "guitarra1": "George"}

dic1.pop("mae")        # remove “mae”: {'pai': 'John', 'filho': 'Sean'}
del dic1["filho"]      # com del posso também apagar 1 entrada : {'pai': 'John'}

dic1.update(dic2)      # insere os elementos de dic2 em dic1:
                       # {'pai': 'John', 'bateria': 'Ringo', 'baixo': 'Paul', 'guitarra1': 'George’}

dic1["guitarra2"] = "John"  # insere nova entrada:  {'pai': 'John', 'bateria': 'Ringo’,
                            # 'baixo': 'Paul', 'guitarra1': 'George', 'guitarra2': 'John'}

dic1.pop("pai")        # remove a entrada “pai": { 'bateria': 'Ringo’, ‘baixo': 'Paul’,
                       #  'guitarra1': 'George', 'guitarra2': 'John’}

dic1.clear()           # esvazia o dicionário
```

Dicionário (7/11)

- **Checando se uma chave existe**

```
dic = {"mae":"Yoko", "pai":"John", "filho":"Sean"}

"mae" in dic          # True      (in = a chave pertence ao dicionário?)
"sogra" in dic        # False
```

- **Checando se um valor existe (não havíamos mostrado...)**

```
>>> "Yoko" in dic.values()
True
>>> "Yoki" in dic.values()
False
```

- **Obtendo as propriedades de um dicionário**

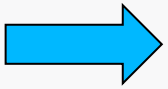
```
type(dic)             # <class 'dict'>  - tipo do objeto dicionário
len(dic)              # 3 - número de elementos do dicionário
```

Dicionário (8/11)

- **Iterando apenas sobre as chaves**
  - Para iterar sobre as chaves, há duas formas possíveis:
  1. Não usar método nenhum
  2. Usar o método **keys**(), que retorna uma visão contendo a lista de chaves.

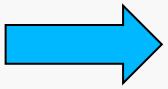
```
dic = {"mae":"Yoko", "pai":"John", "filho":"Sean"}

for chave in dic:
    print(chave)
```



```
mae
pai
filho
```

```
for chave in dic.keys():
    print(chave)
```



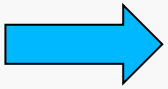
```
mae
pai
filho
```

Dicionário (9/11)

- **Iterando apenas sobre os valores**
  - Para iterar sobre os valores, usa-se o método **values()**, que retorna uma visão contendo a lista de valores.

```
dic = {"mae":"Yoko", "pai":"John", "filho":"Sean"}

for valor in dic.values():
    print(valor)
```



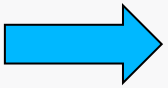
```
Yoko
John
Sean
```

Dicionário (10/11)

- **Iterando sobre chaves e valores ao mesmo tempo**
  - Mais interessante, as chaves e seus respectivos valores podem ser recuperados ao mesmo tempo usando o método **items()**
  - Com a técnica de **desempacotamento**, colocamos a chave e o valor em duas variáveis distintas.

```
dic = {"mae":"Yoko", "pai":"John", "filho":"Sean"}

for chave, valor in dic.items():
    print(chave, valor)
```



```
mae Yoko
pai John
filho Sean
```

  - **Obs.**: Até o Python 3.5, resultados poderiam vir em qualquer ordem. A partir do Python 3.6, eles vêm na ordem em que foram inseridos.

Dicionário (11/11)

- **Transformações**
  - É possível gerar listas, tuplas e conjuntos a partir de dicionários. Basta usar os construtores dessas EDs.

```
dic = {"mae":"Yoko", "pai":"John", "filho":"Sean"}

>>> list(dic.values())       # gera lista com os valores do dicionário
['Yoko', 'John', 'Sean']

>>> set(dic) ())        # gera conjunto com as chaves do dicionário
{'mae', 'filho', 'pai'}

>>> tuple(dic.items()) # gera tupla 2d com os pares chave:valor
(('mae', 'Yoko'), ('pai', 'John'), ('filho', 'Sean'))
```

Resumo – Dicionário

- **Pontos fortes**:
  - Muitas aplicações práticas importantes.
  - Busca por chave tem tempo constante, independente do número de entradas do dicionário.
- **Pontos fracos**:
  - Não é tão simples de usar como as listas e conjuntos.
- **Quando usar?**:
  - Sempre que você tiver um conjunto não ordenado de **chaves exclusivas** que mapeie para valores.
  - Quando a maioria das consultas de seu programa for por chave.

Tarefa

- DOJO: Imprimindo as *k* palavras mais frequentes de um texto.

Referências

- Corrêa, E. (2020). “Meu Primeiro Livro de Python”. V 2.0.0, edubd, 2020. (*capítulo 3*).
  - Disponível em: https://github.com/edubd/meu_primeiro_livro_de_python

- Python 3 Tutorial. Dictionaries https://www.python-course.eu/python3_dictionaries.php