Aula 6 - Tuplas, Dicionários e Conjuntos

June 10, 2021

1 Tuplas, Dicionários, Conjuntos e Listas

1.0.1 Listas

- Pode-se aumentar e diminuir o tamanho
- Pode-se modificar o valor de um elemento
- É a mais popular dos 4
- É inicializável com []

[15]:
```
lista = []
lista = [0,1,2]
lista = [i for i in range(3)]
```

1.0.2 Tuplas

- É totalmente imutável (não permite alterações)
- Útil para dados fixos
- Muito mais rápida do que uma lista
- É inicializável com ()

[52]:
```
tupla = ()
tupla = (0,1,2)
tupla = tuple([i for i in range(5)])
```

1.0.3 Conjuntos

- Não é ordenado
- Não aceita valores duplicados
- Possui operação dos Conjuntos Matemáticos
- É inicializável com {}

[72]:
```
conjunto = {'item'}
conjunto = {'a',1,2,'b'}
```

1.0.4 Dicionários

- Possuem chave e valor
- São mutáveis
- Não aceitam chaves duplicadas
- É inicializável com {‘chave’:‘valor’}

[146]:
```
dicionario = {'chave':'valor','chave2':'valor2'}
```

1.1 Revisão sobre Listas

[3]:
```
lista = []
print(lista)
print(type(lista))
print(type([]))
```

```
[]
<class 'list'>
<class 'list'>
```

[5]:
```
lista = [0,1,2,3,4,5,6,7,8,9,10]
print(lista)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[6]:
```
lista = []
for i in range(11):
    lista.append(i)
print(lista)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[7]:
```
lista = [i for i in range(11)]
print(lista)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[10]:
```
lista[6] = 10
lista[5] = 15
```

[13]:
```
lista.remove(4)
```

1.2 Aula de Tuplas

[18]:
```
tupla = ()
print(tupla)
print(type(tupla))
print(type(()))
```

```
()
<class 'tuple'>
<class 'tuple'>
```

[28]:
```
tupla = (1,2,3) # Ela é inicializado e fica fixa dessa maneira
print(tupla)
```

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

[20]:
```
print(tupla[0])
print(tupla[2])
```

```
1
3
```

[22]:
```
tupla.append(4) # Tupla não permite adicionar novos elementos
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-22-45b4d5b54522> in <module>
----> 1 tupla.append(4) # Tupla não permite adicionar novos elementos

AttributeError: 'tuple' object has no attribute 'append'
```

[25]:
```
tupla[0] = 4 # Tupla não permite modificar os elementos existentes
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-25-e2db72385e60> in <module>
----> 1 tupla[0] = 4 # Tupla não permite modificar os elementos existentes

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

[27]:
```
tupla.remove(1) # Tupla não permite remover elementos
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-27-acce71a0cf9c> in <module>
----> 1 tupla.remove(1) # Tupla não permite remover elementos

AttributeError: 'tuple' object has no attribute 'remove'
```

Conclusão: Tuplas são imutáveis

Para criarmos ou convertermos um objeto para uma tupla podemos usar a função tuple()

[46]:
```
tupla = tuple([i for i in range(11)])
print(tupla)
```

```
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

[48]:
```
lista = [1,2,3]
tupla = tuple(lista)
print(type(lista))
print(type(tupla))
```

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

Podemos tranformar uma lista em tupla e vice-versa, dessa forma, podemos editar seus valores

[50]:
```
tupla = (1,2,3,4,5)
lista = list(tupla)
lista.append(6)
lista[2] = 'Teste'
lista.remove(4)
tupla = tuple(lista)
print(tupla)
```

```
(1, 2, 'Teste', 5, 6)
```

Operações Gerais para Estruturas de Dados Tuplas, Dicionários, Conjuntos e Listas:

[51]:
```
tupla = (1,2,3,4,5)
print(tupla[0])
print(tupla[1:4])
print(tupla[-1])
print(len(tupla))
print(3 in tupla) # True
print(4 not in tupla) #False

for i in tupla:
    print(i)
```

```
1
(2, 3, 4)
5
5
True
False
1
2
3
4
5
```

1.3 Aula sobre Conjuntos (sets)

Vem da teoria dos conjuntos matemáticos, tendo todas as operações eles possuem

[58]:
```
conjunto = {'teste'}
print(conjunto)
print(type(conjunto))
print(type({1}))
```

```
{'teste'}
<class 'set'>
```

```
<class 'set'>
```

[65]:
```
conjunto = {1,2,3,1,4,3}
print(conjunto) # Não aceitam duplicatas
```

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

[74]:
```
conjunto1 = {'a','b','c','d'}
conjunto2 = {'j', 'm', 'n', 'c', 'd'}
conjunto1.add('f') # Add => Adição => Operação dos Conjuntos Matemáticos
conjunto2.add('g')
print(conjunto1) # Não são ordenados
print(conjunto2) # Não são ordenados
```

```
{'b', 'd', 'f', 'a', 'c'}
{'n', 'm', 'g', 'd', 'j', 'c'}
```

Operações com um só conjunto por vez
ADD: Adiciona apenas um item

[77]:
```
frutas = {'maçã','uva'}
print(frutas)
frutas.add('laranja')
print(frutas)
```

```
{'maçã', 'uva'}
{'morango', 'pera', 'uva', 'melancia', 'maçã', 'laranja', 'abacaxi'}
```

Update: Adiciona mais de um item por vez

[83]:
```
frutas = {'maçã','uva'}
frutas.update(['pera','abacaxi','melancia','morango'])
print(frutas)
```

```
{'morango', 'pera', 'uva', 'melancia', 'maçã', 'abacaxi'}
```

Remove: Remove, mas dá erro se o valor não existir
Discard: Remove e não da erro se o valor não existir

[82]:
```
frutas.remove('maçã') # Remove dá erro se o valor já não existe
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-82-f2f4e077e059> in <module>
----> 1 frutas.remove('maçã')

KeyError: 'maçã'
```

[91]:
```
frutas.discard("maçã") # Não dá erro se o valor não existir
print(frutas)
```

```
{'morango', 'pera', 'uva', 'melancia', 'abacaxi'}
```

Clear: Remove todos os itens

[94]:
```
frutas.clear()
print(frutas)
```

```
set()
```

Operações com um mais de um conjunto por vez
Union: Faz a união (soma) de dois conjuntos, retornando um novo conjunto
Update: Também une dois conjuntos, porém, modifica direto no próprio conjunto que chamar esse método

[96]:
```
conjunto1 = {'a','b','c','d'}
conjunto2 = {'j', 'm', 'n', 'c', 'd'}



    # set       =>       # get
conjunto_total = conjunto1.union(conjunto2) # Union me retorna um valor␣
 ↪(conjunto)
print(conjunto_total)
```

```
{'n', 'b', 'm', 'd', 'j', 'a', 'c'}
```

[98]:
```
conjunto1 = {'a','b','c','d'}
conjunto2 = {'j', 'm', 'n', 'c', 'd'}

conjunto1.update(conjunto2) # Update modifica o meu conjunto
print(conjunto1)
```

```
{'n', 'b', 'm', 'd', 'j', 'a', 'c'}
```

Intersection: Me mostra a intersecção entre um e outro
Difference: Me mostra a diferença entre o primeiro e segundo

[99]:
```
conjunto1 = {'a','b','c','d'}
conjunto2 = {'j', 'm', 'n', 'c', 'd'}

interseccao = conjunto1.intersection(conjunto2)

print(interseccao)
```

```
{'d', 'c'}
```

[102]:
```
conjunto1 = {'a','b','c','d'}
conjunto2 = {'j', 'm', 'n', 'c', 'd'}

diferenca = conjunto1.difference(conjunto2)
diferenca2 = conjunto2.difference(conjunto1)
```

```
print(diferenca)
print(diferenca2)
```

```
{'b', 'a'}
{'n', 'j', 'm'}
```

Issubset: Se o primeiro é um subconjunto do segundo

[111]:
```
conjuntinho = {1,2,3}
conjuntao = {1,2,3,4,5,6}

subconjunto = conjuntinho.issubset(conjuntao)
print(subconjunto)
```

```
True
```

Set(): Converte para Conjunto
Método Construtor: É o método que cria uma instancia (objeto) daquela classe

[116]:
```
set() # Construtor da classe set
tuple() # Construtor da classe tupla
list() # Consrtutor da classe lista
print()
```

[117]:
```
lista = [1,2,3,4,5]
tupla = ('a','b','c','d','e','f')
conjunto1 = set(lista)
conjunto2 = set(tupla)

print(conjunto1)
print(conjunto2)
```

```
{1, 2, 3, 4, 5}
{'b', 'e', 'd', 'f', 'a', 'c'}
```

1.4 Aula Dicionário

É uma estrutura de dados composta por chaves e valores.

[122]:
```
# listas = [] => Mutavel, adiciona e remove itens...
# tuplas = () => Imutavel, mais rapida...
# conjuntos = {} => Conjuntos numéricos, operações de conjuntos, não ordenados..
 ↪.

# Procura => Resposta
# chave => valor
# cat => gato
# dog => cachorro
```

```
# apple => maça
# while => enquanto

# É composto por chaves e valores
```

[126]:
```
dict = {'dog':'cachorro', # Não aceita repetições de chaves
        'cat':'gato',
        'dog':'ave' # Sobreescreve o valor atual da chave
       }

print(dict)
print(type(dict))
print(type({'dog':'cachorro'}))
```

```
{'dog': 'ave', 'cat': 'gato'}
<class 'dict'>
<class 'dict'>
```

[127]:
```
# Dicionario suporta varios itens
# Chaves não podem ser repetidas
# É mutável (alterável)
```

[128]:
```
dict = {'dog':'cachorro', # Não aceita repetições de chaves
        'cat':'gato',
        'bird':'ave' # Sobreescreve o valor atual da chave
       }
print(dict)
```

```
{'dog': 'cachorro', 'cat': 'gato', 'bird': 'ave'}
```

dicionario['chave']: Procura pela chave e retorna o valor. Dá erro se não existir a chave
dicionario.get('chave'): Procura pela chave e retorna o valor. Não dá erro se não existir a chave

[131]:
```
dict['dog'] # Não é acessível por índices, mas sim, por chaves
```

[131]:
```
'cachorro'
```

[132]:
```
dict.get('dog')
```

[132]:
```
'cachorro'
```

[137]:
```
dict['cachorro'] # Retorna erro se não existir
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-137-42f448f8a46f> in <module>
----> 1 dict['cachorro'] # Retorna erro se não existir
```

```
KeyError: 'cachorro'
```

[138]:
```
dict.get('cachorro') # Não da erro e não existir
```

Dicionários são mutáveis (alteráveis)

[139]:
```
dict = {'dog':'cachoro', # Não aceita repetições de chaves
        'cat':'gato',
        'bird':'ave' # Sobreescreve o valor atual da chave
       }
print(dict)
```

```
{'dog': 'cachoro', 'cat': 'gato', 'bird': 'ave'}
```

[141]:
```
dict['dog'] = 'cachorro'
print(dict)
```

```
{'dog': 'cachorro', 'cat': 'gato', 'bird': 'ave'}
```

Embora recomendável, não necessariamente uma chave deva ser uma string

[151]:
```
dict = {'string_chave':1,'string_chave2':2.4,'string_chave3':'string'}
print(dict)
```

```
{'string_chave': 1, 'string_chave2': 2.4, 'string_chave3': 'string'}
```

[153]:
```
dict_num = {1:'um',2:'dois',3:'tres',4:'quatro',5:'cinco'}
print(dict_num)
```

```
{1: 'um', 2: 'dois', 3: 'tres', 4: 'quatro', 5: 'cinco'}
```

[154]:
```
dict_num[1] # Embora se pareça com um indice, é uma CHAVE
```

[154]:
```
'um'
```

[155]:
```
for i in range(1,6):
    print(dict_num[i])
```

```
um
dois
tres
quatro
cinco
```

1.5 Realizando loop nos dicionários

[161]:
```
dict = {'dog':'cachoro', # Não aceita repetições de chaves
        'cat':'gato',
        'bird':'ave' # Sobreescreve o valor atual da chave
       }
```

```
for chaves in dict: # Me retorna as chaves
    print(chaves)

print('-'*25)

for chave in dict:
    print(dict[chave]) # Me retorna os valores das chaves
```

```
dog
cat
bird
-------------------------
cachoro
gato
ave
```

[168]:
```
# chaves => keys()
# valores => values()
# itens => items()

print("Todas as chaves:",dict.keys())
print("Todos os valores:",dict.values())
print("Todos os itens:",dict.items())
```

```
Todas as chaves: dict_keys(['dog', 'cat', 'bird'])
Todos os valores: dict_values(['cachoro', 'gato', 'ave'])
Todos os itens: dict_items([('dog', 'cachoro'), ('cat', 'gato'), ('bird',
'ave')])
```

[166]:
```
for valor in dict.values():
    print(valor)
```

```
cachoro
gato
ave
```

[167]:
```
for chave in dict.keys():
    print(chave)
```

```
dog
cat
bird
```

[173]:
```
for chave, valor in dict.items():
    print(f"A chave é {chave} e o valor é {valor}")
    print(f'O tipo da chave é {type(chave)} e o tip do valor é {type(valor)}')
```

```
A chave é dog e o valor é cachoro
```

```
O tipo da chave é <class 'str'> e o tip do valor é <class 'str'>
A chave é cat e o valor é gato
O tipo da chave é <class 'str'> e o tip do valor é <class 'str'>
A chave é bird e o valor é ave
O tipo da chave é <class 'str'> e o tip do valor é <class 'str'>
```

[172]:
```python
for chave, valor in dict.items():
   print(f"Em inglês é {chave} e em português é {valor}")
```

```
Em inglês é dog e em português é cachoro
Em inglês é cat e em português é gato
Em inglês é bird e em português é ave
```

Checando existência

[178]:
```python
if 'dog' in dict:
    print(f"O {dict['dog']} está no dicionário")
else:
    print('Eu não sei a tradução dessa palavra')
```

```
O cachoro está no dicionário
```

Adicionando itens no dicionário

[182]:
```python
dict['monkey'] = 'macaco'

if 'monkey' in dict:
    print(f"O {dict['monkey']} está no dicionário")
else:
    print('Eu não sei a tradução dessa palavra')
```

```
O macaco está no dicionário
```

[183]:
```python
dict['banana'] = 'banana'
print(dict)
```

```
{'dog': 'cachoro', 'cat': 'gato', 'bird': 'ave', 'monkey': 'macaco', 'banana':
'banana'}
```

Removento Itens:
Pop: Remove apenas um item
Clear: Limpa a lista (remove todos os itens)

[186]:
```python
if 'banana' in dict:
    dict.pop('banana')
print(dict)
```

```
{'dog': 'cachoro', 'cat': 'gato', 'bird': 'ave', 'monkey': 'macaco'}
```

[187]:
```python
dict.clear()
print(dict)
```

```
{}
```

1.6 Aninhando Estruturas

Estruturas uma dentro da outra

1.6.1 Dicionários Aninhados

[190]:
```
# Cada curso => Nome, Categoria, Instrutor, Alunos
cursos = {
         'solid':{
            'nome': 'Aprenda Design Patterns com SOLID',
            'categoria': 'Design Patterns',
            'instrutor': 'Felipe Cabrera',
            'alunos': 5000
             },
         'android9':{
             'nome': 'Android 9.0 Avançado: APIs Nativas e Banco de Dados',
             'categoria': 'Android',
             'instrutor': 'Alisson Bolsoni',
             'alunos': 5000
         },
         'vscode':{
             'nome': 'VS CODE: Produtividade Infinita',
             'categoria': 'Programação',
             'instrutor': 'Felipe Cabrera',
             'aluns': 5000
         }
         }
```

[191]:
```
print(cursos)
```

```
{'solid': {'nome': 'Aprenda Design Patterns com SOLID', 'categoria': 'Design
Patterns', 'instrutor': 'Felipe Cabrera', 'alunos': 5000}, 'android9': {'nome':
'Android 9.0 Avançado: APIs Nativas e Banco de Dados', 'categoria': 'Android',
'instrutor': 'Alisson Bolsoni', 'alunos': 5000}, 'vscode': {'nome': 'VS CODE:
Produtividade Infinita', 'categoria': 'Programação', 'instrutor': 'Felipe
Cabrera', 'aluns': 5000}}
```

[193]:
```
for chave in cursos.keys():
    print(chave)
```

```
solid
android9
vscode
```

[198]:
```
print(cursos['solid']['nome'])
print(cursos['solid']['instrutor'])
print(cursos['solid']['categoria'])
```

```
print(cursos['solid']['alunos'])
```

```
Aprenda Design Patterns com SOLID
Felipe Cabrera
Design Patterns
5000
```

1.6.2 Listas Aninhadas

[201]:
```
lista = [[0,1,2,3],[1.2,3.4,5.6],['abc','def','ghi']]
print(lista)
print(lista[2])
print(lista[2][1])
```

```
[[0, 1, 2, 3], [1.2, 3.4, 5.6], ['abc', 'def', 'ghi']]
['abc', 'def', 'ghi']
def
```

1.6.3 Tuplas Aninhadas

[207]:
```
tuplas = ((0,1),(2.3,4.5),('a','b'),('bylearn','felipe'))
print(tuplas)
print(tuplas[1])
print(tuplas[1][0])
print(tuplas[3][0][2:])
```

```
((0, 1), (2.3, 4.5), ('a', 'b'), ('bylearn', 'felipe'))
(2.3, 4.5)
2.3
learn
```

1.6.4 'Conjuntos Aninhados'

Conjuntos matemáticos não são aninhados, não há conjunto dentro de outro, e sim subconjuntos

[214]:
```
conjunto = {1,2,{3,4},5} # Não dá para se fazer, de fato, um conjunto aninhado.␣
↪Isso não está na teoria dos conjuntos
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-214-f6f5a78e1fe4> in <module>
----> 1 conjunto = {1,2,{3,4},5} # Não dá para se fazer, de fato, um conjunto␣
↪aninhado. Isso não está na teoria dos conjuntos

TypeError: unhashable type: 'set'
```

[218]:
```
conj = set()
conj.add(frozenset((1,2)))
```

```
conj.add(frozenset([3,4]))
conj.add(frozenset((5,6)))
print(conj)

for i in conj:
    print(type(i))
```

```
{frozenset({3, 4}), frozenset({5, 6}), frozenset({1, 2})}
<class 'frozenset'>
<class 'frozenset'>
<class 'frozenset'>
```

1.7 Estruturas Aninhadas entre si

[221]:
```
z = ([1,2,3],{'chave':'valor'},{2,3,4},[1,(2,3),{'chave':(2,3)}])
print(super_estrutura)
```

```
([1, 2, 3], {'chave': 'valor'}, {2, 3, 4}, [1, (2, 3), {'chave': (2, 3)}])
```

[ ]:
```

```