Programação e Análise de Dados com Python

Programa de Pós-Graduação em Economia - PPGE

Python - Tuplas e aplicações

Prof. Hilton Ramalho

Prof. Aléssio Almeida

Objetivo

Apresentar operações com tuplas no `Python`.

Conteúdo

Objetos mais usados em Python

Tuplas

- Uma tupla é uma coleção de objetos separados por vírgula. Ela também pode ser delimitada por parênteses `( )`.
- As tuplas podem ser heterogêneas e formadas por diferentes tipos de objetos (texto, número, lista, dicionários,...).
- As tuplas também podem ser aninhadas (agrupadas), ou seja, tuplas cujos elementos também são tuplas ou outros objetos.
- uma tupla é um **objeto imutável**, isto é, não pode ser alterada após ser criada. Essa é a principal diferença de uma **lista**.

Métodos aplicáveis a objetos tupla (tuple)

Método	Descrição
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Vejamos alguns exemplos:

In [ ]:
```
help(tuple)
```

In [ ]:
```
a = 1, 2, 3
print(a, type(a))
```

```
(1, 2, 3) <class 'tuple'>
```

In [ ]:
```
# Criando uma tupla
v = (1, 3, 4)
type(v)
```

Out[ ]:
```
tuple
```

In [ ]:
```
# Criando uma tupla
v = 1, 3, 4
type(v)
```

Out[ ]:
```
tuple
```

In [ ]:
```
# Criando uma tupla
v = ('a', 'b', 'c')
type(v)
```

Out[ ]:
```
tuple
```

In [ ]:
```
# Criando uma tupla
v = ('a', 'b', 'c', 1, 2, 3)
print(v, type(v))
```

```
('a', 'b', 'c', 1, 2, 3) <class 'tuple'>
```

In [ ]:
```
# Criando um tupla cujos elementos são listas
v = ([1,2], [2, 4, 5], [])
print(f"Objeto {v}. O tipo é: {type(v)}")
```

```
Objeto ([1, 2], [2, 4, 5], []). O tipo é: <class 'tuple'>
```

In [ ]:
```
# Criando um tupla heterogênea
v = ([1,2], 1, 'a', 'b', [0, 4, 5], [], 8, 10)
print(v, type(v))
```

```
([1, 2], 1, 'a', 'b', [0, 4, 5], [], 8, 10) <class 'tuple'>
```

In [ ]:
```
# Tuplas agrupadas - cujos elementos são outras tuplas
produto = (('banana', 'maça', 'milho'), ('limão',), ('uva', 'arroz', 'feijã
print(produto, type(produto))
```

```
(('banana', 'maça', 'milho'), ('limão',), ('uva', 'arroz', 'feijão')) <clas
s 'tuple'>
```

Tupla com um único elemento

In [ ]:
```
v = 'a',
print(v, type(v), sep = "\n")
```

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

In [ ]:
```
# Observe a diferença
v = ('a')
print(v, type(v), sep = "\n")
```

```
a
<class 'str'>
```

In [ ]:
```
# Observe a diferença
v = ('a',)
print(v, type(v), sep = "\n")
```

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

Acessando elementos de uma Tupla

- A consulta de elementos de uma tupla segue o **mesmo padrão do caso das listas**, ou seja, usando o indexador de posição de cada elemento.

In [ ]:
```
produto = ('banana', 'maçã', 'milho')

# Elemento na posição 0
produto[0]
```

Out[ ]:
```
'banana'
```

In [ ]:
```
produto = ('banana', 'maçã', 'milho')
# Elemento na posição 1
produto[1]
```

Out[ ]:
```
'maçã'
```

In [ ]:
```
produto = ('banana', 'maçã', 'milho')
# Elemento na posição 2
produto[2]
```

Out[ ]:
```
'milho'
```

In [ ]:
```
produto = ('banana', 'maçã', 'milho')
# Elemento da última posição
produto[-1]
```

Out[ ]:
```
'milho'
```

In [ ]:
```
v = ( 1, [2,34,54], ('a',), ('b', 'c', 'd'), 'w', 'z', 3 )
# Elemento na posição 1
v[1]
```

Out[ ]:
```
[2, 34, 54]
```

In [ ]:
```
v = (1, [2,34,54], ('a',), ('b', 'c', 'd'), 'w', 'z', 3)
# Elemento na posição 2 é uma tupla
v[2]
```

Out[ ]:
```
('a',)
```

In [ ]:
```
v = (1, [2,34,54], ('a',), ('b', 'c', 'd'), 'w', 'z', 3)
# Elemento na posição 4 é um texto
v[4]
```

Out[ ]:
```
'w'
```

In [ ]:
```
v = (1, [2,34,54], ('a',), ('b', 'c', 'd'), 'w', 'z', 3)
# Elemento na última posição é um número inteiro
v[-1]
```

Out[ ]:
```
3
```

Empacotar e Desempacotar elementos de uma Tupla

- Essa é uma das características interessantes de uma tupla. Podemos acessar elementos por meio de uma desestruturação (desempacotamento).
- A desestruturação permite criar novos objetos a partir de elementos da tupla.

In [ ]:
```
# Empacotar uma sequência de textos para uma tupla
produto = 'banana', 'maçã', 'milho', [1,3,5], (3,4)
produto
```

Out[ ]:
```
('banana', 'maçã', 'milho', [1, 3, 5], (3, 4))
```

In [ ]:
```
# Empacotar uma sequência de textos para uma tupla
produto = 'banana', 'maçã', 'milho'
produto
```

Out[ ]:
```
('banana', 'maçã', 'milho')
```

In [ ]:
```
# Descontrução por indexador
a = produto[0]
b = produto[1]
c = produto[2]

print(f'Objeto a={a}, Objeto b={b}, Objeto c={c}')
```

```
Objeto a=banana, Objeto b=maçã, Objeto c=milho
```

In [ ]:
```
# Desempacotar cada elemento de uma tupla e passá-los para variáveis distir
a, b, c = produto
print(f'variável a={a}, variável b={b}, variável c={c}')
```

In [ ]:
```
# Desempacotar uma tupla aninhada criando objetos a partir de seus elemento
t = (('a','b'), (1,2,3), [2,3], 4, 6 )
# Os 2 elementos de t são tuplas
x, y, z, w, h = t
print(f'variável x={x}, variável y={y}, variável z={z}, variável w={w}, var
```

Usando o operador `_` para ignorar elementos

In [ ]:
```
# Desestruturar apenas o primeiro elemento (ignora os dois últimos)
produto = 'banana', 'maçã', 'milho'
el, _ , _ = produto
print(el)
```

```
banana
```

In [ ]:
```
# Desestruturar apenas os últimos elementos (ignora o primeiro)
_ , b , c = produto
print(b,c)
```

```
maçã milho
```

In [ ]:
```
# Desempacotar/descontruir os elementos da lista para variáveis
a, b, c = produto
print(f'variável a={a}, variável b={b}, variável c={c}')
```

```
variável a=banana, variável b=maçã, variável c=milho
```

Uma tupla é mutável?

- Será que podemos alterar os elementos de uma Tupla ?

In [ ]:
```
t = (1,3,2)

# Vamos tentar alterar o elemento da posição 0
t[0] = 4
```

In [ ]:
```
# Observe a diferença em relação a uma lista
v = [1,3,2]
v[0] = 4
print(v)
```

```
[4, 3, 2]
```

E se uma tupla incluir uma lista, é possível alterar esse elemento?

In [ ]:
```
w = ('a', ['b', 'c'])
len(w)
```

Out[ ]:
```
2
```

In [ ]:
```
w[1] = 6
```

In [ ]:
```
w = ('a', ['b', 'c'])
w[1][0] = 'u'
print(w)
```

```
('a', ['u', 'c'])
```

In [ ]:
```
w = ('a', ['b', 'c'])
w[1] = ["w","z"]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-41-5473e3aec606> in <module>()
      1 w = ('a', ['b', 'c'])
----> 2 w[1] = ["w","z"]

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

In [ ]:
```
# Note a diferença - Acessamos o elemento da posição 1 (que é uma lista).
# Contudo, uma lista é mutável e podemos adicionar um novo elemento na list
tupla[1].append('d')

# Observe que a tupla não foi alterada e sim o seu elemento de posição 1 (l
tupla
```

In [ ]:
```
# Contando elementos ou a dimensão de uma tupla. Para isso, também usamos a
t = 3,4,'a'
print(f" A dimensão da tupla {t} é {len(t)}.")
```

In [ ]:
```
# Contando elementos ou a dimensão de uma tupla. Para isso, também usamos a
t = 'a', 1, 4, [4, 5], []
print(f" A dimensão da tupla {t} é {len(t)}.")
```

O Método .count()

- O método `.count()` permite contar o número de vezes que determinado elemento aparece na tupla.

In [ ]:
```
# Usando o método count
w = (1,3,4,4,5,6,7,8,9,4,3)
print(w)

# Total de vezes que o elemento 3 aparece
w.count(3)
```

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

Out[ ]:
```
2
```

In [ ]:
```
w.count(8)
```

Out[ ]:
```
1
```

In [ ]:
```
w.count(4)
```

Out[ ]:
```
3
```

In [ ]:
```
# Tupla mista
w = (1, 3, [3,4], (5,6), [3,4], (5,6), 0)
print(w)
```

```
(1, 3, [3, 4], (5, 6), [3, 4], (5, 6), 0)
```

In [ ]:
```
w.count([3,4])
```

Out[ ]:
```
2
```

In [ ]:
```
w.count((5,6))
```

Out[ ]:
```
2
```

In [ ]:
```
w.count(3)
```

Out[ ]:
```
1
```

O Método .index()

- Permite encontrar a primeira posição de um elemento específico quando ele aparece na tupla.

In [ ]:
```
w = (1,3,4,4,5,6,7,8,9,4,3)

# Posição do elemento 4 (primeira aparição)
w.index(4)
```

Out[ ]:
```
2
```

In [ ]:
```
w = (1,3,4,4,5,6,7,8,9,4,3)

# Posição do elemento 5 (primeira aparição)
w.index(5)
```

Out[ ]:
```
4
```

In [ ]:
```
w = (1, 3, [3,4], (5,6), [3,4], (5,6), 0)

w.count((5,6))
w.index((5,6))
```

Out[ ]:
```
2
```

Concatenação de Tuplas

- Podemos usar o operador `+`

In [ ]:
```
w = ("a", "b")
z = ("d", "g")

w+z
```

Out[ ]:
```
('a', 'b', 'd', 'g')
```

In [ ]:
```
z + w
```

Out[ ]:
```
('d', 'g', 'a', 'b')
```