Linguagem de Programação Python

Listas e Tuplas

Professor: Ritomar Torquato

1

07 Listas e Tuplas

Objetivos: Tuplas como Listas Imutáveis; Tuplas como Registros; Manipulação e Aplicações de Tuplas.

2

Tuplas

Em Python, Tupla é uma lista de elementos **imutáveis**.

3

Tuplas

Tuplas em Python são ( ):

```
>>> tupla = ("Mickey Mouse", "Fred Flintstone", "Bart Simpson")
>>> tupla
('Mickey Mouse', 'Fred Flintstone', 'Bart Simpson')
```

Como nas listas, o valor é acessado por seu índice:

```
>>> tupla[0]
'Mickey Mouse'
```

Mas não podem ser seu valor modificado:

```
>>> tupla[0] = 'Frajola'
Traceback (most recent call last):
  File "<pyshell#132>", line 1, in <module>
    tupla[0] = 'Frajola'
TypeError: 'tuple' object does not support item assignment
```

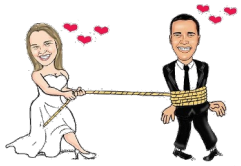
4

Tuplas

Porquê preciso de uma lista que não posso alterar?

Não podemos esquecer que Listas e Tuplas são Estruturadas de Dados

Consistência de Dados

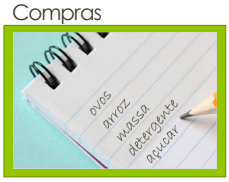


5

Tuplas

Lista: Informação única (homogênea)

- Registro de Lista de Compras



Tupla: Informação única (heterogênea)

- Registro de Agenda de Telefone

NOME	NÚMERO DE TELEFONE
Ana Maria	97130-8955
André	92234-5976
Felipe	5542-1020
Giovana	99123-4001
Sofia	2356-8123
Rui	94562-3459
Vitória	99810-4567

Vários elementos (registros) mas Informação ÚNICA.

6

Tuplas

As listas são destinadas a serem sequências **homogêneas**, enquanto que os Tuplas são estruturas de dados **heterogêneas**.

Phillip J. Eby, contribuidor de diversos projetos do Python

7

Tuplas

- Estrutura de Dados **Homogênea**

Listas são essencialmente homogêneas.

Compras = [Arroz, | Feijão, | Farinha, | Carne, | Tomate]

- Estrutura de Dados **Heterogênea**

Tuplas são essencialmente heterogêneas.

Carro = (FIAT, | Palio, | 2018, | Preto, | 38.000)

Essencialmente mas **não exclusivamente**, já que podem armazenar dados de qualquer tipo em cada posição.

8

Tuplas

	Produto	Quantidade	Preço
Lista	Café	1	2,50
	Biscoito	2	3,49
	Leite em pó	2	3,25
	Açúcar	2	1,99
	Pão	5	0,80
	Queijo	1	9,50

Tupla

Na Lista, faz sentido mexer na **quantidade de itens**.

Na Tupla, não faz sentido mexer na **estruturada da informação**.

9

Tuplas

Como nas listas cada valor é acessado por seu índice.

Lista de Nomes: Estrutura de Dados **Homogênea**

0	1	2	3	4	5	6
Carla	Ana	Bia	Hugo	Dani	Ana	Ísis

Temperatura: Estrutura de Dados **Heterogênea**

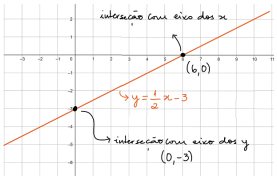
0	1
32	"ºC"

A posição define o significado: (temperatura, escala)

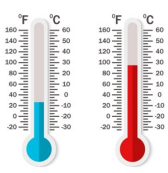
10

exemplo

(6, 0) ou (0, -3)



(28.0, 'C') ou (82.4, 'F')



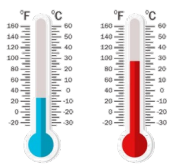
Tuplas são úteis para guardar INFORMAÇÕES com dados MÚLTIPLOS e IMUTÁVEIS. Por exemplo, representar um ponto no espaço cartesiano ou uma temperatura.

11

exemplo

A posição define o significado: (temperatura, escala)

(28.0, 'C') ou (82.4, 'F')



Uma informação tem duas partes e não faz sentido usar métodos de inserção ou alteração, como **append()**.

12

exemplo

```
>>> t = [32, 'C']
>>> t.append('xyz')
>>> t
[32, 'C', 'xyz']
>>> t = (32, 'C')
>>> t.append('xyz')
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    t.append('xyz')
AttributeError: 'tuple' object has no attribute 'append'
>>> t
(32, 'C')
```

Isso é uma temperatura? Não faz sentido!

Se `t` representa uma temperatura, não faz sentido permitir inserir ou alterar os elementos. É melhor gerar um ERRO e manter a consistência da informação.

13

Manipulando Tuplas

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação:

```
>>> tupla = ("a", "b", "c")
>>> tupla
('a', 'b', 'c')
>>> tupla[0]
'a'
>>> tupla[2]
'c'
>>> tupla[ 1: ]
('b', 'c')
>>> tupla * 2
('a', 'b', 'c', 'a', 'b', 'c')
>>> len(tupla)
3
```

Podem ser usadas com **`for`**:

```
>>> for elemento in tupla:
        print(elemento)


a
b
c
```

14

Manipulando Tuplas

Podemos criar tuplas sem o uso dos parênteses explícitos :

```
>>> tupla = 100, 200, 300
>>> tupla
(100, 200, 300)
```

Podem ser utilizadas para o desempacotamento de valores:

```
>>> a, b  = 10, 20
>>> a
10
>>> b
20
```

Também podemos trocar o valor de variáveis:

```
>>> a, b = 10, 20
>>> a, b  = b, a
>>> a
20
>>> b
10
```

15

Manipulando Tuplas

Existe uma sintaxe especial para criação de tuplas **com apenas um elemento**. Usamos (,) vírgula:

```
>>> t2 = (1,)
>>> t2
(1,)
>>> t3 = 1,
>>> t3
(1,)
```

Sem o uso da vírgula, temos:

```
>>> t1 = (1)
>>> t1
1
```

Um número Inteiro.

16

Manipulando Tuplas

Podemos criar uma tupla vazia usando apenas os parenteses:

```
>>> t4 = ()
>>> t4
()
>>> len(t4)
0
```

Tuplas podem ser criadas a partir de Listas com **`tuple`**

```
>>> L = [1, 2, 3]
>>> T = tuple(L)
>>> T
(1, 2, 3)
```

17

Manipulando Tuplas

Não podemos alterar uma tupla depois da criação mas podemos concatená-las, gerando novas tuplas:

```
>>> t1 = 1, 2, 3
>>> t2 = 4, 5, 6
>>> t3 = t1 + t2
>>> t3
(1, 2, 3, 4, 5, 6)
>>> t4 = t1, t2, t3
>>> t4
((1, 2, 3), (4, 5, 6), (1, 2, 3, 4, 5, 6))
```

18

Manipulando Tuplas

Se uma tupla contiver um elemento que pode ser alterado, como uma lista, este continuará com seu funcionamento normal:

```
>>> tupla = ('a', ['b', 'c', 'd'])
>>> tupla
('a', ['b', 'c', 'd'])
>>> len(tupla)
2
>>> len(tupla[1])
3
>>> tupla[1]
['b', 'c', 'd']
>>> tupla[1].append('e')
>>> tupla
('a', ['b', 'c', 'd', 'e'])
```

A tupla não foi alterada.

A lista que ela contém foi alterada.

19

Empacotamento e desempacotamento

Podemos usar Tuplas para passar ou receber valores empacotados de funções.

```
def soma(a, b, c):
    return a + b + c

t = (13, 14, 15)
print(soma(*t)) # Imprime 42
```

Usamos o * (asterisco) para desempacotar a tupla como parâmetros da função soma. Isso evita precisar usar:

```
print(soma(t[0], t[1], t[2])
```

20

Empacotamento e desempacotamento

Podemos criar funções que recebem um número indeterminado de parâmetros fazendo o empacotamento.

```
def soma(*valores):
    soma = 0
    for valor in valores:
        soma += valor
    return soma

print(soma())
print(soma(1))
print(soma(1, 2))
print(soma(1, 2, 3))
```

Usamos o * (asterisco) para empacotar os parâmetros em uma tupla chamada "valores".

É possível chamar a função com qualquer quantidade de argumentos. Inclusive nenhum.

21

Empacotamento e desempacotamento

É comum acessar elementos de Listas iterando sobre a lista. Com Tuplas é comum o acesso via o desempacotamento.

Agenda é uma lista de tuplas com nome e telefone

```
agenda = [
            ('Ana Maria', '97130-8955'),
            ('André', '92234-5976'),
            ('Felipe', '5542-1020'),
            ('Giovana', '99123-4001'),
            ('Sofia', '2356-8123'),
            ('Rui', '94562-3459'),
            ('Vitória', '9810-4567')
         ]

for nome, telefone in agenda:
    print(f'Nome: {nome:30s} Telefone: {telefone}')
```

22

Tuplas como registros

Se considerarmos a tupla apenas como uma lista imutável, a quantidade e a ordem dos itens podem não ser importantes.

Se considerar como colunas de uma tabela de dados, a quantidade será fixa e é a ordem que define o significado.

```
# Coordenadas do aeroporto de Los Angeles
coordenadas_lax = (33.9425, -118.408056)

# Dados sobre Tokyo: nome, ano, população em milhões,
# variação da população, área em km quadrados
cidade, ano, pop, var_pop, area = (
        'Tokyo', 2003, 32450, 0.66, 8014
    )
```

23

Tuplas como registros

```
# Pais e passaporte dos passageiros
passageiros = [('USA', '31195855'), ('BRA', 'CE342567'),
               ('ESP', 'XDA205856'), ('BRA','PI856554')]

for passageiro in sorted(passageiros):
    print(f'%s/%s' % passageiro)

for pais, _ in passageiros:
    print(pais)
```

O operador % entente as Tuplas.

Um valor que não nos interessa é frequentemente atribuído à uma variável chamada _ (sublinhado).

O mecanismo de **empacotar** e **desempacotar** funciona bem ao tratar as tuplas como registros.

24