FT 07
Curso: UFCD 10793
UFCD/Módulo/Temática: UFCD 10793 **- Fundamentos de Python**
Ação: 10793_05/N
Formador/a: Sandra Liliana Meira de Oliveira
Data:
Nome do Formando/a:

Objetivo: Criação e Manipulação de Tuplas

Reproduz os seguintes exemplos. O objetivo é instanciar tuplas e aplicar as operações específicas deste tipo de dados

1. Criar e imprimir tuplas. Para criarmos tuplas podemos usar () ou o construtor tuple(). Uma tupla pode ter diferentes tipos de dados.

```
# create a tuple using ()
# number tuple
number_tuple = (10, 20, 25.75)
print(number_tuple)
# Output (10, 20, 25.75)

# string tuple
string_tuple = ('Jessa', 'Emma', 'Kelly')
print(string_tuple)
# Output ('Jessa', 'Emma', 'Kelly')

# mixed type tuple
sample_tuple = ('Jessa', 30, 45.75, [25, 78])
print(sample_tuple)
# Output ('Jessa', 30, 45.75, [25, 78])

# create a tuple using tuple() constructor
sample_tuple2 = tuple(('Jessa', 30, 45.75, [23, 78]))
print(sample_tuple2)
# Output ('Jessa', 30, 45.75, [23, 78])
```

2. Criar uma tupla com um único item.

```
# without comma
single_tuple = ('Hello')
print(type(single_tuple))
# Output class 'str'
print(single_tuple)
# Output Hello

# with comma
single_tuple1 = ('Hello',)
# output class 'tuple'
print(type(single_tuple1))
# Output ('Hello',)
print(single_tuple1)
```

3. Embalar e desembalar dados

Em Python, podemos criar uma tupla empacotando um grupo de variáveis. O empacotamento pode ser usado quando queremos agrupar vários valores num única variável. Geralmente, essa operação é chamada de empacotamento de tuplas. Da mesma forma, podemos descompactar os itens apenas atribuindo os itens da tupla ao mesmo número de variáveis. Esse processo é chamado de "Descompactação".

```
# packing variables into tuple
tuple1 = 1, 2, "Hello"
# display tuple
print(tuple1)
# Output (1, 2, 'Hello')

print(type(tuple1))
# Output class 'tuple'

# unpacking tuple into variable
i, j, k = tuple1
# printing the variables
print(i, j, k)
# Output 1 2 Hello
```

4. Tamanho de uma tupla

```
tuple1 = ('P', 'Y', 'T', 'H', 'O', 'N')
# length of a tuple
print(len(tuple1))
# Output 6
```

5. Iterando uma tupla

```
# create a tuple
sample_tuple = tuple((1, 2, 3, "Hello", [4, 8, 16]))
# iterate a tuple
for item in sample_tuple:
    print(item)
```

6. Aceder aos itens de uma tupla:

- Indexação: primeiro elemento tem índice 0
- Se indicarmos o valor do índice maior que o comprimento de uma tupla, dará erro. O mesmo quando indicamos um valor não inteiro para o índice.
- Admite indexação negativa.
- Podemos ainda aceder a diversos elementos indicando um intervalo de índices (slicing) – similar às listas.
- também podemos fatiar a tupla usando indexação negativa

```
tuple1 = ('P', 'Y', 'T', 'H', 'O', 'N')
for i in range(4):
    print(tuple1[i])
```

```
tuple1 = ('P', 'Y', 'T', 'H', 'O', 'N')

# IndexError: tuple index out of range
print(tuple1[7])
```

```
tuple1 = ('P', 'Y', 'T', 'H', 'O', 'N')

# TypeError: tuple indices must be integers or slices, not float
print(tuple1[2.0])
```

```
tuple1 = ('P', 'Y', 'T', 'H', 'O', 'N')
# Negative indexing
# print last item of a tuple
print(tuple1[-1])  # N
# print second last
print(tuple1[-2])  # O

# iterate a tuple using negative indexing
for i in range(-6, 0):
    print(tuple1[i], end=", ")
# Output P, Y, T, H, O, N,
```

```
tuple1 = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

# slice a tuple with start and end index number
print(tuple1[1:5])
# Output (1, 2, 3, 4)
```

```
tuple1 = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

# slice a tuple without start index
print(tuple1[:5])
# Output (0, 1, 2, 3, 4)

# slice a tuple without end index
print(tuple1[6:])
# Output (6, 7, 8, 9, 10)
```

```
tuple1 = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

# slice a tuple using negative indexing
print(tuple1[-5:-1])
# Output (6, 7, 8, 9)
```

7. Encontrar items – método index

Podemos procurar um determinado item na tupla usando o método index(). O mesmo retornará a posição desse item.
O método index() aceita três argumentos:
item – O item que precisa ser pesquisado
start – (Opcional) - O valor inicial do índice a partir do qual a pesquisa será iniciada
end – (Opcional) - O valor final do indíce da pesquisa

```
tuple1 = (10, 20, 30, 40, 50)

# get index of item 30
position = tuple1.index(30)
print(position)
# Output 2
```

```
tuple1 = (10, 20, 30, 40, 50, 60, 70, 80)
# Limit the search locations using start and end
# search only from location 4 to 6
# start = 4 and end = 6
# get index of item 60
position = tuple1.index(60, 4, 6)
print(position)
# Output 5
```

```
tuple1 = (10, 20, 30, 40, 50, 60, 70, 80)
#index out of range
position= tuple1 .index(10)
print(postion)
# Output ValueError: tuple.index(x): x not in tuple
```

8. Verificar se um item existe

```
tuple1 = (10, 20, 30, 40, 50, 60, 70, 80)
# checking whether item 50 exists in tuple
print(50 in tuple1)
# Output True
print(500 in tuple1)
# Output False
```

9. Adicionar e alterar itens numa tupla:

Uma lista é um tipo mutável, o que significa que podemos adicionar ou modificar valores, mas as tuplas são imutáveis, portanto, não podem ser alteradas. Além disso, como uma tupla é imutável, não há métodos internos para adicionar itens à tupla. Se tentarmos modificar algum valor obteremos um erro.

```
tuple1 = (0, 1, 2, 3, 4, 5)
tuple1[1] = 10
# Output TypeError: 'tuple' object does not support item assignment
```

Como solução alternativa, podemos converter a tupla numa lista, adicionar itens e convertê-la novamente como tupla.

```
tuple1 = (0, 1, 2, 3, 4, 5)

# converting tuple into a list
sample_list = list(tuple1)
# add item to list
sample_list.append(6)

# converting list back into a tuple
tuple1 = tuple(sample_list)
print(tuple1)
# Output (0, 1, 2, 3, 4, 5, 6)
```

10. Removendo itens

Como as tuplas são imutáveis não existem os métodos pop() nem remove(). A única forma de removermos um elemento é usarmos a técnica de conversão em lista.

O método del() apaga toda a tupla

```
sampletup1 =(0,1,2,3,4,5,6,7,8,9,10)
del sampletup1

print(sampletup1)
```

```python
tuple1 = (0, 1, 2, 3, 4, 5)

# converting tuple into a list
sample_list = list(tuple1)
# reomve 2nd item
sample_list.remove(2)

# converting list back into a tuple
tuple1 = tuple(sample_list)
print(tuple1)
# Output (0, 1, 3, 4, 5)
```

11. Contar as ocorrências de um item.

```python
tuple1 = (10, 20, 60, 30, 60, 40, 60)
# Count all occurrences of item 60
count = tuple1.count(60)
print(count)
# Output 3

count = tuple1.count(600)
print(count)
# Output 0
```

12. Cópia de uma tupla

```python
tuple1 = (0, 1, 2, 3, 4, 5)

# copy tuple
tuple2 = tuple1
print(tuple2)
# Output (0, 1, 2, 3, 4, 5)

# changing tuple2
# converting it into a list
sample_list = list(tuple2)
sample_list.append(6)

# converting list back into a tuple2
tuple2 = tuple(sample_list)

# printing the two tuples
print(tuple1)
# Output (0, 1, 2, 3, 4, 5)
print(tuple2)
# Output (0, 1, 2, 3, 4, 5, 6)
```

13. Concatenar(juntar) duas tuplas – usar o operador +

```
tuple1 = (1, 2, 3, 4, 5)
tuple2 = (3, 4, 5, 6, 7)
```

```
# concatenate tuples using + operator
tuple3 = tuple1 + tuple2
print(tuple3)
# Output (1, 2, 3, 4, 5, 3, 4, 5, 6, 7)
```

14. Concatenar(juntar) duas tuplas – usar a função soma

```
tuple1 = (1, 2, 3, 4, 5)
tuple2 = (3, 4, 5, 6, 7)

# using sum function
tuple3 = sum((tuple1, tuple2), ())
print(tuple3)
# Output (1, 2, 3, 4, 5, 3, 4, 5, 6, 7)
```

15. Tuplas aninhadas:

```
nested_tuple = ((20, 40, 60), (10, 30, 50), "Python")

# access the first item of the third tuple
print(nested_tuple[2][0])  # P

# iterate a nested tuple
for i in nested_tuple:
    print("tuple", i, "elements")
    for j in i:
        print(j, end=", ")
    print("\n")
```

16. Funções internas: max(), min(). Estas funções não podem ser aplicadas em tuplas com tipos de dados heterógeneos

```
tuple1 = ('xyz', 'zara', 'abc')
# The Maximum value in a string tuple
print(max(tuple1))
# Output zara

# The minimum value in a string tuple
print(min(tuple1))
# Output abc

tuple2 = (11, 22, 10, 4)
# The Maximum value in a integer tuple
print(max(tuple2))
# Output 22
# The minimum value in a integer tuple
print(min(tuple2))
# Output 4
```

```
tuple3 = ('a', 'e', 11, 22, 15)
# max item
print(max(tuple3))
```

17. Função all() – verdadeiro se todos os valores verdadeiros e no caso de tupla vazia. Falso nos restantes casos

```
# all() with All True values
tuple1 = (1, 1, True)
print(all(tuple1))  # True

# all() All True values
tuple1 = (1, 1, True)
print(all(tuple1))  # True

# all() with One false value
tuple2 = (0, 1, True, 1)
print(all(tuple2))  # False

# all() with all false values
tuple3 = (0, 0, False)
print(all(tuple3))  # False

# all() Empty tuple
tuple4 = ()
print(all(tuple4))  # True
```

18. Função any() – Pelo menos um valor verdadeiro. Retorna falso quando todos os valores são falso e a tupla está vazia.

```
# any() with All True values
tuple1 = (1, 1, True)
print(any(tuple1))  # True

# any() with One false value
tuple2 = (0, 1, True, 1)
print(any(tuple2))  # True

# any() with all false values
tuple3 = (0, 0, False)
print(any(tuple3))  # False

# any() with Empty tuple
tuple4 = ()
print(any(tuple4))  # False
```

Quando devemos usar Tuplas?

As tuplas são listas imutáveis e dão utilizadas quando:

- Se pretende apresentar dados de leitura ou fixos;
- Para ser usadas como chave para os dicionários;
- Aumentar a velocidade de pesquisa.