Tuples e Arrays

Trabalhando com Tuples

Já vimos algumas vezes, mas afinal o que é um **Tuple**?

Assim como vimos no **List**, o **Tuple** também é uma **lista de valores**, porém com uma diferença importante, pois se trata de uma lista **IMUTAVEL**! Ou seja, uma lista onde não poderá ser adicionado, removido ou alterado novos itens!!

E para criar uma **Tuple**, basta declararmos com **()**.

```
1   # Trabalhando com Tuples
2   cores = ('cor a', 'cor b', 'cor c')
3   print(type(cores))
```

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

E veja que ao tentar usar a mesma função que normalmente utilizaríamos no **List** para criar itens, que seria o **append()**, com o **Tuple** não existe essas funções!

```
5   cores.append('cor d')

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  PORTS

PS C:\Projetos\pocs\python-study> & C:/Users/fernando.spelling/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Proj
ys/codigo-fonte.py"
<class 'tuple'>
Traceback (most recent call last):
  File "c:\Projetos\pocs\python-study\04 - Estrutura de dados\02 - Tuples e Arrays\codigo-fonte.py", line 5, in <module>
    cores.append('cor d')
    ^^^^^^^^^^^^
AttributeError: 'tuple' object has no attribute 'append'
PS C:\Projetos\pocs\python-study>
```

Quando utilizar Lists ou Tuples

Como já foi mencionado anteriormente, a principal diferença deles é que uma **Tuple** se trata de uma lista **IMUTAVEL** já o **List** seria uma lista **MUTAVEL**!

Dito isso os **Tuples** seriam lista mais performáticas em relação aos **Lists**!!

Logo...

Utilize List:

- Quando a lista precise ser atualizada, adicionando novos itens, removido itens, etc.
- Quando teremos lista pequenas e não muito grandes!

Utilize Tuple:

- Quando a lista não precisar ser atualizada.
- Quando as listas forem muito grandes, geralmente acima de **1000 itens**!!

Trabalhando com Array

Quando temos listas muito grandes e que elas precisam ser mutáveis, ou seja, precisaram ser **atualizadas(adicionando/removendo itens)**, uma opção muito boa que podemos utilizar seria o **Array**, pois teremos uma performance melhor em relação aos **Lists**!!

E para podermos criar um objeto **Array**, precisamos **importar** o **modulo Array**, pois não vem nativamente no **Python**!

```
from array import array
```

Após em realizar a importação do **modulo**, vamos então criar um objeto **array**, mas para isso precisamos definir 2 argumentos para a criação do array!! Um deles obviamente é a própria lista com as coleções dos itens que queremos no array, já o outro argumento de trata do **typecode**!!

E esse **typecode** se trata de um código referente há o tipo de dado que será a nosso **array**, que no nosso caso será **'u'**, pois se trata de uma lista de **string**.

```
array_texto = array('u', ['a', 'b', 'c'])
print(array_texto.tolist())
```

```
['a', 'b', 'c']
```

E para identificar qual é o tipo de cada **typecode** de um **array** que quiser criar, é só vermos em sua documentação!

array— Arrays eficientes de valores numéricos

Esse módulo define um tipo de objeto que pode representar compactamente um array de valores básicos: caracteres, inteiros, números de ponto flutuante. Arrays são tipos de sequência e funcionam bem parecidamente com listas, porém o tipo dos objetos armazenados é restringido. O tipo é especificado na criação do objeto usando um type code, que é um único caractere. Os seguintes type codes são definidos:

Type code	Tipo em C	Tipo em Python	Tamanho mínimo em bytes	Notas
`'b'`	signed char	int	1	
`'B'`	unsigned char	int	1	
`'u'`	Py_UNICODE	Caractere unicode	2	(1)
`'h'`	signed short	int	2	
`'H'`	unsigned short	int	2	

https://docs.python.org/pt-br/3.7/library/array.html