Python 3 - Mundo 3

Váriaveis compostas (Tuplas)

Podem armazenar vários valores. Diferente das variáveis simples que eram limitadas a um elemento por vez.

No python existem três possibilidades para a criação de variáveis compostas, São elas:

- Através de Tuplas
- Através de Listas
- Através de Dicionários

Exemplo de variável composta:

Laços de repetição em variáveis compostas

Retornar o tamanho de uma variável composta

A função Len( ) pode ser utilizada para retornar o tamanho de uma variável composta.

Tuplas

As tuplas são imutáveis, portanto não é possível alterar qualquer elemento do índice de uma tupla. Ou seja, a única forma de alterar o conteúdo de uma tupla é reiniciando o programa e redefinindo o conteúdo. Diferente dos vetores em outras linguagens, é possível inserir mais de um tipo de variável na mesma tupla.

Declaração de Tuplas

Todo conteúdo de uma Tupla é inserido entre parênteses.

```
lanche = ('Hambúrguer', 'Suco', 'Pizza', 'Pudim')
```

A partir do python 3.5 não é obrigatório inserir os parênteses.

Método para print de tuplas dentro do For()

Segue abaixo os três métodos possíveis para printar o conteúdo de tuplas utilizando o laço de repetição For( )

```
for comida in lanche:
    print(f'Eu vou comer {comida} ')

for cont in range(0, len(lanche)):
    print(f'Eu vou comer {lanche[cont]} na posição {cont}')

for pos, comida in enumerate(lanche):
    print(f'Eu vou comer {comida} na posição {pos}')
```

Inserindo dados em uma tupla

Devido a imutabilidade de uma tupla é necessário inserir os dados no momento de criação da mesma.

Conforme exemplo abaixo:

```
numeros = (int(input('Insira o primeiro número: ')),
           int(input('Insira o segundo número: ')))
```

O mesmo serve para números randômicos usando randint dentro da declaração de cada elemento da tupla.

Manipulando Tuplas

Método Sorted( )

Serve para mostrar a tupla de forma ordenada (ordem alfabética ou numérica).

Exemplo no código:

```
print(sorted(lanche))
```

Saída:

```
lanche = ('Hambúrguer', 'Suco', 'Pizza', 'Pudim')

   ['Hambúrguer', 'Pizza', 'Pudim', 'Suco']
```

Método Count( )

Retorna quantas vezes um elemento aparece na tupla.

Exemplo no código:

```
a = (2, 5, 3, 2, 4, 2)
print(a.count(2))
```

Saída:

```
3

Process finished with exit code 0
```

Método Index( )

Retorna a posição de um elemento na tupla.

Exemplo no código:

```
a = (2, 5, 3, 2, 4, 2)
print(a.index(3))
```

Saída:

```
2

Process finished with exit code 0
```

Caso haja mais de um elemento igual na tupla o index retornado será sempre do primeiro elemento encontrado. Caso queira o index do segundo ou próximo elemento repetido, será necessário o uso de fatiamento.

Exemplo no código:

```
a = (2, 5, 3, 2, 4, 2)
print(a.index(2, 1))    # O index procurará pelo número a partir da posição 1 da tupla
```

Saída:

```
3

Process finished with exit code 0
```

Método Del( )

As tuplas são imutáveis, entretanto é possível fazer uma única mudança que é deletar **completamente** a tupla. Vale lembrar que Del( ) apaga completamente a variável do programa.

Exemplo de código:

```
lanche = ('Hambúrguer', 'Suco', 'Pizza', 'Pudim')
del(lanche)
print(lanche)
```

Saída:

```
Traceback (most recent call last):
  File "C:/Users/eders/PycharmProjects/PythonTeste/aula016a.py", line 3, in <module>
    print(a)
NameError: name 'a' is not defined
```

Método max( )

Retorna o maior elemento de uma tupla ou qualquer outra estrutura.

Exemplo de código:

```
print(f'O menor valor sorteado foi {min(tupla)}')
```

saída:

```
Os valores sorteados foram: 3 1 10 7 10
O maior valor sorteado foi 10
```

Método min( )

Retorna o Menor elemento de uma tupla ou qualquer outra estrutura.

Exemplo de código:

```
print(f'O menor valor sorteado foi {min(tupla)}')
```

Saída:

```
Os valores sorteados foram: 3 1 10 7 10

O menor valor sorteado foi 4
```

Função Len( ) #Length

Retorna o quantos elementos(tamanho) contém uma tupla, string, etc.
OBS: len vem de length (comprimento)

Exemplo em código:

```
tupla = (3, 4, 2, 1)
print(len(tupla))
```

Listas

São semelhantes as tuplas, entretanto mutáveis e declaradas com colchete [ ] ou com o comando list( ). ex: lista = list()

Manipulando Listas

Método append( )

Serve para adicionar elementos no final de uma lista.

Exemplo:

Método insert()

Serve para criar um novo bloco de memória na posição especificada.

Lista antes da execução do insert:

Lista Depois da execução do insert:

A lista foi expandida para 5 elementos e o elemento ZERO e os seguintes foram movidos para direita consecutivamente até a 5ª posição. Assim que a posição 0 ficou vazia, nela foi inserido o elemento declarado no insert.

Método Sort( )

Serve para ordenar a lista do menor para o maior.

```
valores = [8, 2, 5, 4, 9, 3, 0]
valores.sort()
```

Lista após execução do sort( ):

Para ordenar do maior para o menor utilizar o parâmetro reverse = True.

Ex: `valores.sort(reverse=True)`

Método enumerate( )

Serve para pegar o elemento e a posição da lista ao mesmo tempo.
ex:

```
for c, v in enumerate(valores):
    print(f'Na posição {c} encontrei o valor {v}!')
print('Cheguei no final da lista.')
```

saída:

```
Na posição 0 encontrei o valor 5!
Na posição 1 encontrei o valor 9!
Na posição 2 encontrei o valor 4!
Cheguei no final da lista.
```

Deletando elementos de Listas

Comando Del

Deleta a posição 3 de forma que se houverem outros elementos depois do terceiro, eles serão realocados para preencherem o espaço vazio.

Método pop( )

Normalmente usado para deletar o último elemento de uma lista. Também serve para remover elementos de uma posição específica.

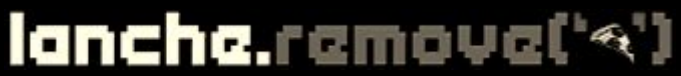
Método clear( )

Serve para limpar o conteúdo de uma lista. Segue abaixo exemplo em código.

```
dados.clear()   # Limpa todo o conteudo da lista
```

Método remove( )

Diferente dos métodos pop e del, deleta pelo conteúdo. Ou seja, deleta a posição em que o elemento passado no parâmetro for encontrado.



Copiando listas

Diferente das variáveis, se uma lista uma lista for atribuída a outra lista como no exemplo abaixo:

```
a = [1, 2, 3, 4]
b = a
```

Será gerada uma ligação entre a lista B e a lista A, de forma que se algum elemento da lista B for alterado, o mesmo elemento da lista A também será.

Para copiar uma lista deve-se utilizar fatiamento, da seguinte forma:

```
a = [1, 2, 3, 4]
b = a[:]
```

Sendo que a lista B está recebendo cada um dos elementos da lista A possibilitando que os elementos de B possam ser alterados sem alterar também os elementos da lista A e vice-versa, visto que não há ligação entre as duas listas desta forma.

Exemplo:

```
a = [1, 2, 3, 4]
b = a[:]
b[2] = 6
print(f'Lista A: {a}')
print(f'Lista B: {b}')
```

Saída:

```
Lista A: [1, 2, 3, 4]
Lista B: [1, 2, 6, 4]
```

Método copy()

Pode assim como o fatiamento, ser usado para copiar uma lista.

Exemplo de uso:

```
a = [1, 2, 3, 4]
b = a.copy()
b[2] = 6
print(f'Lista A: {a}')
print(f'Lista B: {b}')
```

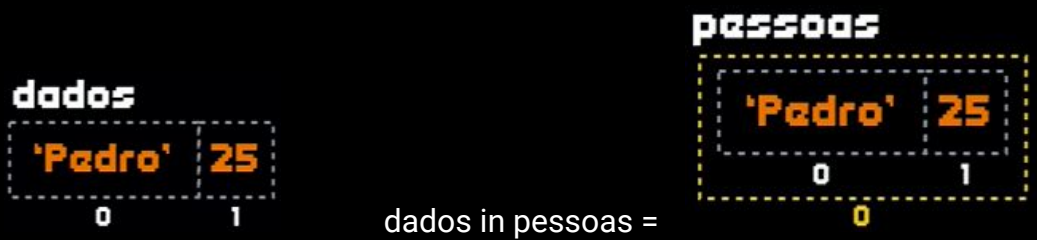
Listas dentro Listas

Para inserir uma lista dentro de uma outra lista, deve-se usar de fatiamento da seguinte forma: " [:]" representando um fatiamento do início ao fim da lista a qual pretende inserir.

ex:

```
pessoas = list()
pessoas.append(dados[:])
```

ou seja:



Foi criado o index 0 em pessoas e o mesmo recebeu o conteúdo de dados.

É possível também inserir listas dentro de uma lista no momento em que a mesma é declarada, conforme imagem abaixo.

```
pessoas = [['Pedro', 25], ['Maria', 19], ['João', 32]]
```

Amostra da lista:

Print de listas compostas

Deve-se referenciar a posição da primeira lista no primeiro colchete e no segundo, a posição da lista que está na posição especificada no primeiro colchete.

```
print(pessoas[0][0])    Pedro
print(pessoas[1][1])    19
print(pessoas[2][0])    João
print(pessoas[1])       ['Maria', 19]
```

Dicionários

Dicionários são identificáveis e declarados com chaves { } ou com o comando dict( )

```
dados = dict()
dados = {                                    }
```

São usados para definir um nome para o índice do elemento. Exemplo:

```
dados = dict()
dados = { 'nome':'Pedro','idade':25 }
```

Dicionário:

O índice de Pedro é nome e o índice de 25 é idade.

Print de dicionários

```
print(dados['nome'])      Pedro
print(dados['idade'])     25
```

Adicionar novos elementos em um Dicionário

O comando append( ) não funciona em dicionários, portanto para criar novos elementos em dicionários pode-se utilizar do comando:

```
dados['sexo'] = 'M'
```

Dicionário:

Remover elementos de dicionários

Para remover elementos de um dicionário deve-se utilizar o comando del.

```
del dados['idade']
```

Dicionário: