Criando múltiplas variáveis

```
n1, n2, n3, n4, n5 = 'disciplina ', 'algoritmos lógica programação', 2023, '1º período', 'ads'
print(f'\nImprimindo o valor das variáveis \n')
print(n1)
print(n2)
print(n3)
print(n4)
print(n5)
```

Transformando variáveis em lista – **empacotamento de listas**

```
n1, n2, n3, n4, n5 = 'disciplina ', 'algoritmos lógica programação', 2023, '1º período', 'ads'

numeros =[n1, n2, n3, n4, n5]

print(f'\nA nova lista: {numeros}')

print(f'\nO tipo de dados da nova lista é {type(numeros)} \n')
```

Observe que as iniciais dos elementos da lista estão todas em maiúsculas, existe duas formas de resolver esse problema:

Transformando o início de cada string em maiúsculas e inserindo em uma lista - função title() e upper() = transforma toda string em maiúscula – **empacotamento de listas**

```
n1, n2, n3, n4, n5 = 'disciplina ', 'algoritmos lógica programação', 2023, '1º período', 'ads'

lista =[n1.title(), n2.title(), n3, n4.title(), n5.upper()]

print(f'\nLista com iniciais em maiúsculas: {lista} \n')
```

Transformando uma string em lista - split() - **empacotamento de listas**

```
curso = 'Curso de Análise e Desenvolvimento de Sistemas - IFRO Campus Ariquemes'

print(f'\nO valor da string é: {curso} ')

print(f'\nO tipo da string é: {type(curso)}')

lista = curso.split() # convertendo em lista

print(f'\nA lista criada a partir da string é: {lista}')

print(f'\nO tipo da variável lista é: {type(lista)}')
```

Atenção: Por padrão o comando split() separa os elementos da lista pelo espaço entre as palavras quando forem *strings*. Caso as palavras da lista sejam separadas por um caractere como vírgula (,) ou ponto e vírgula, insira o separador desejado dentro dos parênteses do comando split(';')

Transformando uma lista em string - comando join() – **desempacotamento de listas**

```
lista = ['Algoritmos', 'e', 'Lógica', 'de', 'Programação', 'Curso', 'ADS']

print(f'\nLista original: {lista}')

disciplina_curso = ' '.join(lista)

print(f'\nA string possui o valor: {disciplina_curso}')

print(f'\nO tipo da variável criada é: {type(disciplina_curso)} \n')
```

Desempacotando listas para múltiplas variáveis

```
lista = ['IFRO', 'CAMPUS', 'ARIQUEMES', 2023]

var1, var2, var3, var4 = lista

print(f'\nO valor da variável var1 é: {var1}')
print(f'O tipo da variável var1 é: {type(var1)}')

print(f'\nO valor da variável var2 é: {var2}')
print(f'O tipo da variável var2 é: {type(var2)}')

print(f'\nO valor da variável var3 é: {var3}')
print(f'O tipo da variável var3 é: {type(var3)}')

print(f'\nO valor da variável var4 é: {var4}')
print(f'O tipo da variável var4 é: {type(var4)}')
```

Atenção: se tivermos mais elementos para desempacotar do que variáveis para receber os valores teremos ValueError e versa...

inserindo valores de elementos em qualquer parte da lista - comando insert

Este comando precisa de dois argumentos:

1º O índice onde pretende inserir o novo elemento;

2º E qual é o valor desse novo elemento

Como exemplo vamos inserir a string ‘IFRO – Campus Ariquemes’ entre as strings ‘Genoveva e ‘Gerundina’, ou seja, entre o índice 1 e 2.

```
# indice       0            1            2            3
lista  =  ['Gertrudez', 'Genoveva', 'Gerundina', 'Gesmeralda']

print(f'\nLista original: {lista}')

lista.insert(2, 'IFRO – Campus Ariquemes')

print(f'\nLista alterada: {lista}')
```

inserindo valores de elementos no final da lista com o comando insert

```
# indice        0            1                      2                  3            4
lista   =   ['Gertrudez', 'Genoveva', 'IFRO – Campus Ariquemes', 'Gerundina', 'Gesmeralda']

print(f'\nLista original: {lista}')

lista.insert(5, 'Curso  de ADS')

print(f'\nLista alterada: {lista} \n')
```

Se inserir um índice maior como por exemplo 15 numa lista que só 5 elementos o Python irá entender que esse novo elemento vai para o índice 6

Alterando valores dos elementos de uma lista - pelo número do índice

```
# indice        0             1              2              3
lista   =   ['Gertrudez', 'Genoveva', 'Gerundina', 'Gesmeralda']

print(f'\nLista original: {lista}')

# Altere o nome que está no índice 2 pela string abaixo
lista[2] = 'Amaranto'

print(f'\nLista alterada: {lista}')

print(f'\nO tipo da variável lista é: {type(lista)} \n')
```

Exercício: usando um editor de códigos altere o valor do elemento Genoveva para 152:

```
# indice        0             1              2              3
lista   =   ['Gertrudez', 'Genoveva', 'Gerundina', 'Gesmeralda']

print(f'\nLista original: {lista}')

# Altere o nome que está no índice 2 pela string abaixo
lista[1] = 152
print(f'\nLista alterada: {lista}')
```

lista com tipos de dados diferentes

```
lista = [ 12.15, 4, False, 'Tkinter', [3, 2, 1], 1950]

print(f'\nA lista original é: {lista}')
```

Repetindo os elementos de uma lista

```
lista = ['IFRO']

print(f'\nLista original: {lista} \n')

lista *= 5

print(f'\nlista com elementos repetidos: {lista} \n')
```

Removendo o último elemento de uma lista comando pop()

```
# indice       0           1                       2                    3            4
lista  =  ['Gertrudez', 'Genoveva', 'IFRO – Campus Ariquemes', 'Gerundina', 'Gesmeralda']

print(f'\nLista original: {lista}')

lista.pop()

print(f'\nLista com o último elemento removido: {lista} \n')
```

A função pop() sem argumento sempre irá remover o último elemento de uma lista. Se o desenvolvedor informar o índice que pretende remover a função pop() irá realizar a remoção.

Removendo o elemento de uma determinada posição da lista - pop(índice)

```
# indice       0           1                       2                    3            4
lista  =  ['Gertrudez', 'Genoveva', 'IFRO – Campus Ariquemes', 'Gerundina', 'Gesmeralda']

print(f'\nLista original: {lista}')

lista.pop(3)

print(f'\nLista com o elemento do indice 3 removido: {lista} \n')
```

Se informar um índice que não existe na lista o pop() irá retornar o erro:

Removendo o último elemento de uma lista - pop() - passando o elemento removido para uma variável

```
# indice       0  1   2   3  4   5      6  7       8         9
lista   =    [ 1, 22, 13,  54, 5, 'IFRO', 6, 77, 'ARIQUEMES', 8 ]

print(f'\nLista original: {lista}')

variavel = lista.pop()

print(f'\nO valor capturado da lista é: {variavel}')

print(f'\nO tipo da variável capturada é: {type(variavel)}')

print(f'\nLista foi alterada para: {lista}')
```

Essa função pop() pode ser combinada com um argumento para capturar um elemento que está numa determinada posição da lista.

Removendo um elemento da lista pelo seu valor – comando remove()

- Remova o valor ‘Genoveva’ de sua lista.

```
# indice          0           1                   2                  3            4
lista   =   ['Gertrudez', 'Genoveva', 'IFRO – Campus Ariquemes', 'Gerundina', 'Gesmeralda']

print(f'\nLista original: {lista}')

lista.remove('Genoveva')

print(f'\nLista com o elemento Genoveva foi removido: {lista} \n')
```

Lista com valores duplicados – qual dos elementos o remove() irá retirar?
- Remova ‘Genoveva’ de sua lista. Porém agora esse nome está duplicado

```
# indice          0           1                   2                  3            4
lista   =   ['Gertrudez', 'Genoveva', 'IFRO – Campus Ariquemes', 'Gerundina', 'Gesmeralda', 'Genoveva']

print(f'\nLista original: {lista}')

lista.remove('Genoveva')

print(f'\nUm dos elementos Genoveva foi removido: {lista} \n')
```

O remove irá retirar da lista a primeira referência que ele encontrar. Se informar um valor que não existe na lista o remove() irá retornar o erro:

Removendo um elemento da lista pelo seu valor – comando remove() e tentando passar o elemento removido para uma variável

```
# indice     0  1   2   3  4   5     6   7     8          9   10
lista   =   [ 1, 22, 13,  54, 5, 'IFRO', 6, 77, 'ARIQUEMES', 8, 'IFRO' ]

print(f'\nLista original: {lista}')

variavel = lista.remove('IFRO')

print(f'\nO valor capturado da lista é: {variavel}')

print(f'\nO tipo da variável capturada é: {type(variavel)}')

print(f'\nLista foi alterada para: {lista} \n')
```

O remove() removeu inclusive da memória o primeiro elemento com valor ‘IFRO’, no entanto ele não inseriu o valor dentro da variável. Essa é a diferença entre usar o pop().

Excluindo todos os elementos da lista - **clear()**:

```
# indice       0            1                        2                  3              4              5
lista  =  ['Gertrudez', 'Genoveva', 'IFRO – Campus Ariquemes', 'Gerundina', 'Gesmeralda', 'Genoveva']

print(f'\nLista original: {lista}')

print(f'\nA quantidade de elementos da lista original é: {len(lista)}')

lista.clear()

print(f'\nLista com todos os elementos excluídos: {lista}')

print(f'\nA quantidade de elementos da lista original agora é: {len(lista)}')
```

Descobrindo o índice de um elemento na lista - comando index()

```
lista = ['Gertrudez', 'Genoveva', 'IFRO – Campus Ariquemes', 'Gerundina', 'Gesmeralda']

print(f'\nLista original: {lista}')

print('\nEm qual indice da lista está o elemento cujo valor é: Gerundina')

print(f'\nO indice do elemento Gerundina é: {lista.index("Gerundina")} \n')
```

Procurando um elemento na lista a partir de posição inicial tendo como base o índice

```
lista = ['IFRO', 'Campus', 'Ariquemes', 'ASD', 123, 32, 45, 99, 100, 101]

print(f'\nLista original: {lista}')

print('\nQual indice inicial para o final da lista devemos começar a procurar elemento 100?')

print(f'\nO elemento 100 da lista está no indice: {lista.index(100, 2)} \n')  # busca o número 100 a partir do indice 1
```

Percorrendo os elementos da lista com *slice*

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 'IFRO', 'Campus', 'Ariquemes', 'Rondônia']

print(f'\nLista original: {lista}')

print(f'\nPercorrendo todos os elementos da lista: {lista[::]} \n', end = " ")

print(f'\nPercorrendo os elementos a partir do índice 1 lista: {lista[1::]} \n', end = " ")

print(f'\nPercorrendo os elementos a partir do índice 0  até o indice 4 da lista: {lista[:4:]} \n', end = " ")

print(f'\nPercorrendo os elementos a partir do índice 4  até o indice 10 da lista: {lista[4:10:]} \n', end = " ")

print(f'\nPercorrendo os elementos a partir do índice 1 até o final da lista passando de 2 em 2: {lista[1::2]} \n', end = " ")

print(f'\nPercorrendo os elementos a partir do índice 5 até o indice 9 da lista passando de 3 em 3: {lista[5:11:3]} \n', end = " ")
```

Percorrendo os elementos a partir de seus índices - while

```
# indice:     0          1             2            3       4     5          6           7         8
lista   =   ['IFRO', 'Campus', 'Ariquemes', 'ASD', 123, 32, 'Rondônia', 'Brasil', 2023]

print(f'\nLista original: {lista}')

print(f'\nPercorrendo e Imprimindo os elementos a partir de seu indice com while')
i = 0
while i < len(lista):
   print(lista[i])
   i +=1
```

Iterando sobre listas - comando for()

```
lista = list(range(10))

print(f'\nLista original: {lista} \n')

soma = 0
for i in lista:
    print(i, end=', ')
    soma += i

print(f'\n\nTotal: {soma}')
```

Gerando índice para um elemento da lista com for()

```
#              0          1             2            3                     4
lista = ['IFRO', 'Campus', 'Ariquemes', 'ASD', 'Algoritmo e Lógica de Programação']

print(f'\nLista original: {lista} \n')


print(f'\nImprimindo o indice e o valor de cada elemento da lista')

for indice, lista in enumerate(lista):
    print(indice, lista)
```

Apresentando lista com índice negativo - funciona como um circulo onde o primeiro elemento está ligado ao final da lista

```
#índice      0           1               2             3
lista = ['IFRO', 'Campus', 'Ariquemes', 'ASD']

print(f'\nLista original: {lista} \n')

print(f'\nLista a partir do indice -1: {lista[-1]}')

print(f'\nLista a partir do indice -2: {lista[-2]}')

print(f'\nLista a partir do indice -3: {lista[-3]}')

print(f'\nLista a partir do indice  -4: {lista[-4]}')
```

Invertendo os elementos de uma lista - slice

```
lista = [55, 37, 0, 90, 5, 20, 1,118, 'Anísio', 'Zenaide', 'Ximenes', 'Luiz', 'Beatriz']

print(f'\nLista original: {lista}')

lista = lista[: :-1]

print(f'\nLista em ordem inversa: {lista} \n')
```

Invertendo os elementos de uma lista - reverse

```
lista = [55, 37, 0, 90, 5, 20, 1,118, 'Anísio', 'Zenaide', 'Ximenes', 'Luiz', 'Beatriz']

print(f'\nLista original: {lista}')

lista.reverse()

print(f'\nLista em ordem inversa: {lista} \n')
```

Ordenando listas - em ordem crescente - sort()

```
lista = ['Anísio', 'Zenaide', 'Ximenes', 'Luiz', 'Beatriz']
print(f'\nLista original: {lista}')

lista.sort()
print(f'\nLista em ordem crescente: {lista} \n')
```

Ordenando listas - em ordem crescente - sort()

```
lista = [55, 37, 0, 90, 5, 20, 1,118 ]

print(f'\nLista original: {lista}')

lista.sort()
print(f'\nLista em ordem crescente: {lista} \n')
```

Ordenando listas - em ordem decrescente - sort(reverse)

```
lista = ['Anísio', 'Zenaide', 'Ximenes', 'Luiz', 'Beatriz']

print(f'\nLista original: {lista}')

lista.sort(reverse = True)
print(f'\nLista em ordem decrescente: {lista} \n')
```

Ordenando listas - em ordem decrescente - sort(reverse)

```
lista = [55, 37, 0, 90, 5, 20, 1,118 ]

print(f'\nLista original: {lista}')

lista.sort(reverse = True)
print(f'\nLista em ordem decrescente: {lista} \n')
```

As funções sort(reverse = True) não permitem o uso de variáveis com tipos diferentes exemplo: inteira e caracteres gerando o seguinte erro.

Listas: soma, valor máximo, mínimo e tamanho

```
lista = [55, 37, 0, 90, 5, 20, 1,118, -5, 74, -20, 63]

print(f'\nLista original: {lista}')

print(f'\nSoma dos elementos: {sum(lista)}')

print(f'\nMaior valor na lista: {max(lista)}')

print(f'\nMenor valor na lista: {min(lista)}')

print(f'\nQuantidade de elementos na lista: {len(lista)} \n')
```

Copia de lista - *Shalow*

```
lista = [1, 2, 3, 4, 5, 6, 'IFRO', 'ADS' ]

print(f'\nLista original: {lista}')

nova = lista

nova.append(4)
nova.append('Ariquemes')

print(f'\nLista original: {lista}')

print(f'\nCópia da lista original com os novos elementos: {nova} \n')
```

Utilizamos a cópia via atribuição copiando os dados da lista original para nova lista, porém, mas após realizar modificação em uma das listas elas se refletiram em ambas as listas. Em Python essa ação é chamada de *Shalow copy*, ou seja, nesse tipo de cópia uma lista fica conectada a outra (dependente da outra).

Copia de lista - *Deep*

```
lista = [1, 2, 3, 4, 5, 6, 'IFRO', 'ADS' ]

print(f'\nLista original: {lista}')

nova = lista.copy()

print(f'\nCópia da lista original: {nova}')

nova.append(4)
nova.append('Ariquemes')

print(f'\nLista original: {lista}')
print(f'\nCópia da lista original com os novos elementos: {nova} \n')
```

Utilizarmos o *copy* para copiar dados da lista original para uma nova lista, no entanto, eles ficaram totalmente independentes, mesmo quando houve alterações em uma das listas. Em Python essa ação é chamada de *Deep Copy*.

Exercício:

a) Faça um *script* onde o operador digite 10 números em uma estrutura de repetição.

- Armazene os números pares em uma lista;
- Armazene os números impares em outra lista;
- Após sair da estrutura, apresente na tela a lista que contém os números pares e a lista que contém os números ímpares.

REFERÊNCIA:

Mueller, John Paul. **Programação Funcional Para Leigos**. Alta Books. Edição do Kindle, 2020.

F. V. C. Santos, Rafael. **Python: Guia prático do básico ao avançado** (Cientista de dados Livro 2) (p. 1). rafaelfvcs. 2ª Edição - Kindle, 2020.

MENEZES, Nilo Ney Coutinho. **Introdução à programação com Python: Algoritmos e lógica de programação para iniciantes**. Editora: Novatec - 3ª edição, 2019 - São Paulo - SP.

Piva Junior, Dilermando. Algoritmos e programação de computadores / Piva Jr ... [et al.]. - 2. ed. - Rio de Janeiro: Elsevier, 2019. 528 - ISBN 9788535292480