PYTHON

ARRAY

arrays são implementados por meio do **módulo *array*** que se trata de uma estrutura de dados usada para armazenar uma coleção ordenada de elementos do **mesmo tipo**. Detalhando melhor essa definição, o *array* é uma coleção de **elementos homogêneos**, ou seja, todos os elementos no *array* devem ser do mesmo tipo de dado. Outra característica importante, o *array* usa menos memória do que a lista, é mais rápido nas operações de leitura e possibilita executar operações matemáticas sobre todos os itens de forma otimizada. Além disso, os *arrays* em Python **são imutáveis**, o que significa que não é permitido adicionar ou remover elementos em um determinada posição após a criação do array.

para criar um *array* em Python, é necessário importar o **módulo *array*** e definir qual é o tipo de dado, conforme exemplos de código abaixo.

```
import array

meu_array = array.array('i', [1, 2, 3, 4, 5])

print(meu_array)
```

Os seguintes itens e métodos também são suportados:
array.**typecode**: O caractere typecode usado para criar o array.

array.**itemsize**: O tamanho em bytes de um item do array em representação interna.

array.**append**(*x*): Adiciona um novo item com valor *x* ao final do array.

array.**count**(*x*): Retorna a quantidade de ocorrências de *x* no array.

array.**extend**(*iterable*): Acrescenta os itens em *iterable* ao final do array.
Se *iterable* for outro array, ele deve ter *exatamente* o mesmo type code; senão, ocorrerá um TypeError. Se *iterable* não for um array, ele deve ser iterável e seus elementos devem ser do tipo correto para ser acrescentado ao array,

array.**fromlist**(*list*): Adiciona itens de *list*. Isso é equivalente
a `for x in list: a.append(x)` exceto que se ocorrer um errro de tipo, o vetor não é alterado.

array.**fromlist**(*list*): Adiciona itens de *list*. Isso é equivalente
a `for x in list: a.append(x)` exceto que se ocorrer um erro de tipo, o vetor não é alterado.

VETORES [ ] + Biblioteca

os vetores e as matrizes são estruturas de dados estáticas que suportam apenas um tipo de dado. Essas estruturas são muito utilizadas para manipularção de de dados que envolvem calculo. Quando criamos um vetor numpy vazio, por padrão, é criado do tipo float. Uma vez definido seu tamanho, não é mais possivel alterar. São acessados também por indices que iniciam no zero(0), ocupam menos espaço na memória que as listas e são mais rápidos também. Para criá-lo é necessario importar a biblioteca numpy.

(PODER SER SÒ DE UM TIPO) (VETOR NÂO É UMA LISTA(EM PYTHON)) Array e Vetor é a mesma coisa.
Para criar um vetor em python precisar importar biblioteca:
OBS.: Vetor só aceita nomes com até 8 caracteres

```
import numpy as np # Para usar é necessário importar a biblioteca
vetor= np.array([1,2,3,4,5])
print(vetor)

vetor.dtype #retornar o tipo do vetor(server somente no vetor)


vetor[0]= 100 # altera a posição 0 do vetor para 100


vetor.max() #retornar o valor maior do vetor


vetor.mean() # retornar a média aritmética dos elementos do vetor

vetor.sum() #retornar a soma dos elementos do vetor

print(vetor*3) # multiplica cada elemnto do vetor por 3


vetor.sort() # coloca o vetor em ordem crescente(coloca fisicamnete na memoria(na ordem do indice))
del vetor #apaga o vetor


vetor1 = np.arange(10) #cria um vetor com números sequenciais de 0 a 9


vetor3 = np.empty(5, ) #cria um vetor vazio de 5 posições do tipo float(informar a localização da posição)
print(vetor3)
```

```
#CARREGANDO UM VETOR PELO TECLADO
vetor = np.empty(5)
for i in range(5):
   vetor[i] = input('valor')
print(vetor)
```

MATRIZES(vetor de dimensão)

```
matriz = np.array([[1,2,3], [4,5,6]]) #cria uma matriz de 2 x 3

I = np.eye(5) #MATRIZ IDENTIDADE

print(I)

D = np.diag(np.arange(5)) #MATRIZ DIAGONAl

print(D)
```

LISTA[ ]

Lista é uma estrutura de dados usada para armazenar uma coleção ordenada de elementos de tipos variados, ou seja, nesse tipo de estrutura de dados em Python os **elementos são heterogêneos**, é permitido armazenar elementos de diferentes tipos, além de ser flexível em termos de tamanho, ou seja, é permitido adicionar e remover elementos a qualquer momento. Listas em Python são implementadas internamente como *arrays* dinâmicos, o que significa que o tamanho da lista pode ser aumentado ou reduzido conforme necessário. Porém, devido as listas em Python serem heterogêneas, o consumo de memória é muito maior do que o consumo de memória de um *array*.

As operações básicas em uma lista em Python incluem adicionar e remover elementos, acessar elementos por índice e modificar elementos. As listas em Python são ideais para armazenar coleções de dados que precisam ser alterados com frequência ou cujo tamanho não é conhecido.

As listas são criadas usando colchetes, exemplo:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

```
valores= [] #cria lista vazia

lista.append(5) #insere um elemento no final da lista

lista.insert(0,50) #insre o valor na posição desejada
del lista[4] #apaga o elemento da posição desejada

lista.remove(6) #elimina o elemento 6 da lista

lista.pop() #sem parâmento remove e retornar o ultimo elemento

lista.pop(2) #com parâmetro remove o elemnto na posição desejada

valores = list(range(1,10)) #PREENCHENDO UMA LISTA COM RANGE
valores[-1] # retornar o ultimo elemento da lista

valores[:] #mostra a lista inteira

valores[:3] # do incio até o 3

valores[3:] # a partir do 3 até o final

valores.sort() #coloca a lista em ordem crescente
max(valores) #retorna o maior valor da lista
min(valores) #retorna o menor valor da lista
len(valores) #retornar o número de elemntos da lista
sum(valores) #retornar a soma dos elemntos da lista
valores.count(50) #conta quantas vezes o elemnto 50 aparece na lista
nomes.sort() #coloca a lista em ordem alfabética
numeros.clear() #apaga todo o conteudo da lista
(sum(notas)/len(notas)) #média de nota em lista

EXEMPLOS:
for i in nomes: #não pula linha
 print(i, end=' ')

# INSERINDO ELEMNTOS NA LISTA PELO TECLADO
for i in range(5):
  numeros.append(int(input('digite um valor: ')))
print(numeros)
```

DICIONARIO { }

É uma forma de coleção de dados em que se guarda uma chave e um valor correspondente. É similar a um dicionário mesmo, em que há sempre um termo e uma tradução.

Para iniciar um dicionário em Python, você precisa definir a variável e colocar os valores entre chaves {}.

De um lado, com as aspas, pode-se definir a chave específica; do outro, o valor associado. Cada par é separado por vírgulas.

Fica mais ou menos assim:

dic = {“a”: 20, “b”: 40}

```
pessoas = {'joão':18, 'Claudio':52, 'Mariana':16, 'Francisco': 62}

pessoas['Mariana'] #retornar o valor da chave 'Mariana'
pessoas['Mariana'] = 26 #alaterar o valor da chave 'Mariana'
pessoas ['Fernando'] = 40 #insere o 'Fernando' no discionario
del pessoas['Claudio'] #apaga a chave mencionada
pessoas.keys() #retornar chaves
pessoas.values() #retorna os valores
pessoas.items() # retorna as chaves  e os valores
```

```
 #CARREGANDO UM DICIONARIO COM O INPUT
 dados = {}  #cria um discionario vazio

 dados['nome'] = input('nome: ')
 dados['idade'] = int(input('idade: ')) #fazendo casting de int(mudar/acrescentar o tipo(ex:int, float) do dado)
 dados['salario'] = float(input('salario: ')) #fazendo casting de float
 print ('+='*30)
 print(dados)
```

```
# CAREGANDO DICIONARIOS EM UMA LISTA
estado1 = {'uf':'são paulo', 'sigla':'SP'}
estado2= {'uf': 'minas gerais', 'sigla': 'MG'}

brasil = []
brasil.append(estado1) #acrescenta o dicionario 'estado' a lista
brasil.append(estado2) #acrescenta o dicionario 'estado' a lista
```

```
brasil[1]['uf'] #para imprimir o valor de uma chave especifica. O 1 significa a cada que ta a sigla na lista(a lista sempre incia no 0)
```

TUPLAS( )

São imutáveis. Isso significa que não podemos alterar um mesmo objeto tupla, ou seja, mudar uma de suas referências internas (seus valores), nem adicionar ou remover elemento algum. São criadas com parênteses ou pela função: tuple(). Para uma tupla de valor único, devemos sempre colocar uma vírgula (,) no final do valor, mesmo com os parênteses, ou o Python não interpretará como tupla. Tuplas se comportam como **listas estáticas**. Assim, ainda podemos deduzir (e confirmar) que, por conta disso, tuplas ocupam menos espaço na memória comparadas a listas

```
tupla.append(50) #não é possivel  acrescentar valores em uma tupla

print(sorted(nomes)) #mostra as tuplas em ordem alfabética

print(nomes.index('amanda')) #retornar o indice do valor atribuido

numeros[-1] #retorna a ultima posição

numeros[-3:]# retornar as 3 ultimas posições

numeros[:-3] # retornar do inicio as as 3 ultimas
```

```
um=(5,) #só funciona como tupla pra um elemento quando colocar uma vircula no final.
type(num)

num = () #cria uma tupla vazia
for i in range(5):
  x= int(input('digite um número: '))
  num = num + (x,)
print(num)
```

PILHAS=( ) + classe

Pilhas são estruturas de dados em que só é possível inserir um novo elemento no final da pilha e só é possível remover um elemento do final da pilha. Dizemos que pilhas seguem um protocolo em que **o último a entrar é o primeiro a sair**. Pilhas são geralmente implementadas com arranjos.

```
# definir a função push para adicionar elementos
 def push(self, x):
     self.data.append(x)
# definir a função pop para remover o último elemento da lista
 def pop(self):
     if len(self.data) > 0:
         return self.data.pop(-1)
```

```
def empty(pilha): #verifica se pilha está vazia
    return len(pilha) == 0
def push(pilha,item): #adiciona itens na pilha
    pilha.append(item)
def pop(pilha): #remove itens da pilha
    if empty(pilha):
        print("ATENÇÃO PILHA VAZIA")
    else:
        item=pilha.pop()
        return item
def top(pilha): #retorna item do topo sem remove-lo
    if empty(pilha):
        print("ATENÇÃO PILHA VAZIA")
    else:
        return pilha[len(pilha)-1]
def size(pilha): #retorna tamanho da pilha
    return len(pilha)
```

```
class Pilha:
    def __init__(self):
        self.lista = [] # cria
uma lista vazia para armazenar
os elementos da pilha

    def push(self, elemento):

self.lista.append(elemento) #
insere o elemento no final da
lista

    def pop(self):
        if not self.is_empty():
# verifica se a pilha não está
vazia
            return
self.lista.pop() # remove e
retorna o elemento do final da
lista
```

- Para que possa utilizar o tipo(Type) Pilha, deverá criar uma classe (obs.: caso contrário será ima lista(list)) segue o exemplo:

```
class Pilha:
    def __init__(self):
        self.itens = []

    def esta_vazia(self):
        return not
bool(self.itens)

    def empilhar(self, dado):
        self.itens.append(dado)

    def desempilhar(self):
        if not
self.esta_vazia():
            return
self.itens.pop()

    def topo(self):
        if not
self.esta_vazia():
            return
self.itens[-1]
```

PANDAS () + biblioteca

É uma biblioteca de código aberto usada para trabalhar com dados relacionais ou rotulados de forma fácil e intuitiva. Ele fornece várias estruturas de dados e operações para manipular dados numéricos e séries temporais. Oferece uma ferramenta para limpar e processar seus dados. É a biblioteca Python mais popular usada para análise de dados.

Necessário importar o nome da biblioteca.

```
import pandas as pd
dados = pd.read_excel('/content/idades.xlsx') #importa uma planilha externa,
print(dados)


dados.head() #mostra os primeiros 5 elementos

dados['NOME'] #mostra somente a coluna de Nome

dados[dados['NOME']=='BEATRIZ'] #retornar uma informação especifico

dados[dados['IDADE']>30] #retorna as pessoas com mais de 30 anos

dados.sort_values(by='NOME') #coloca o dataFrame em ordem alfabetica

dados.describe() #retorna um resumo estatisco dos dados

nomes = list(dados['NOME']) #Tranformando uma coluna do Dataframe em lista
print(nomes)


import matplotlib.pyplot as plt #biblioteca para criar grafico
plt.plot(nomes, idade)
plt.title('Grafico de idades') #cria um titulo
plt.xlabel('nomes') #coloca titulo no eixo do x (caso não queira de apareça basta
utilizar o : no final)
plt.ylabel('idade') # #coloca titulo no eixo do y
plt.xticks(fontsize = 7 )# muda o tamanho da fonte
plt.barh(nomes, idade, color='r') #gera um grafico de linhas
plt.title('Grafico de idades') #cria um titulo
```

```
plt.xlabel('nomes') #coloca titulo no eixo do x (caso não queira de apareça basta utilizar o : no final)
plt.ylabel('idade') # #coloca titulo no eixo do y
plt.xticks(fontsize = 7 )# muda o tamanho da fonte
```

CONJUNTOS{ }

São uma estrutura de dados que armazena elementos únicos e não ordenados. Oferecem uma maneira eficiente de lidar com operações de conjunto, como união, interseção e diferença. É definido usando chaves={} ou a função built-in set(). Oferecem uma ampla gama de métodos e operações que podemos utilizar para manipular os elementos armazenados neles

1. **Adicionar elementos:** Podemos adicionar elementos a um conjunto usando o método add(). Por exemplo:

   ```
   meu_conjunto.add(6)
   ```

2. **Remover elementos:** Podemos remover elementos de um conjunto usando o método remove() ou discard(). A diferença é que remove() gera um erro se o elemento não existir no conjunto, enquanto discard() não gera nenhum erro. Por exemplo:

   ```
   meu_conjunto.remove(5)
   ```

3. **União de conjuntos:** Podemos unir dois conjuntos usando o método union() ou o operador |. Por exemplo:

   ```
   conjunto1 = {1, 2, 3}
   conjunto2 = {3, 4, 5}
   uniao = conjunto1.union(conjunto2)
   ```

4. **Interseção de conjuntos:** Podemos encontrar os elementos comuns entre dois conjuntos usando o método intersection() ou o operador &. Por exemplo:

   ```
   intersecao = conjunto1.intersection(conjunto2)
   ```

```
len(nomes)#Retornar o tamanho do conjunto

#OPERAÇÕES COM CONJUNTO
a= {1,2,5,6,9}
b= {0,6,2,12, 7}
a == b #a é igual a b?


# UNIÃO DE CONJUNTOS
a= {1,2,5,6,9}
b= {0,6,2,12, 7}
a != b #a diferença de b?




a= {1,2,5,6,9}
b= {0,6,2,12, 7}
a & b # MOSTRA OS ELEMENTOS QUE TEM EM COMUM


nomes.clear() #limpa conjunto

conjunto = set() #criando conjunto vazio pelo metodo set()
type(conjunto)
```

REFERÊNCIA

PILHAS: https://panda.ime.usp.br/panda/static/pythonds_pt/03-EDBasicos/03-Pilhas.html

CONJUNTO: https://docs.python.org/pt-br/3/tutorial/datastructures.html

DICIONARIO: https://www.hashtagtreinamentos.com/dicionarios-em-python?gad_source=1&gclid=Cj0KCQiAyKurBhD5ARIsALamXaGRcmWfoVQn2ve3C12UPs2xizCyljRhcCLd4Dgra0WkJmvI4KET7oMaAuihEALw_wcB

TUPLAS: https://www.alura.com.br/artigos/conhecendo-as-tuplas-no-python

PANDAS: https://www.alura.com.br/artigos/pandas-o-que-e-para-que-serve-como-instalar?utm_term=&utm_campaign=%5BSearch%5D+%5BPerformance%5D+-+Dynamic+Search+Ads+-+Artigos+e+Conteúdos&utm_source=adwords&utm_medium=ppc&hsa_acc=7964138385&hsa_cam=11384329873&hsa_grp=111087461203&hsa_ad=682526577071&hsa_src=g&hsa_tgt=aud-456779235754:dsa-843358956400&hsa_kw=&hsa_mt=&hsa_net=adwords&hsa_ver=3&gad_source=1&gclid=Cj0KCQiAyKurBhD5ARIsALamXaEmgJD1OubhPduLdAKXNYK2HAifLZQHeXQP-XCSWzLKEUCQz-Pq3n4aAlqMEALw_wcB