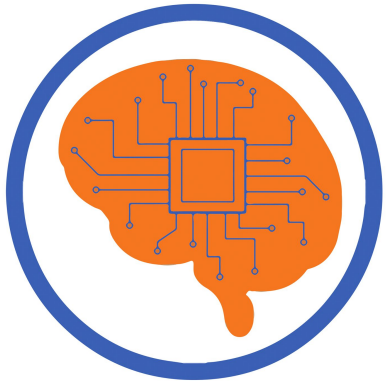
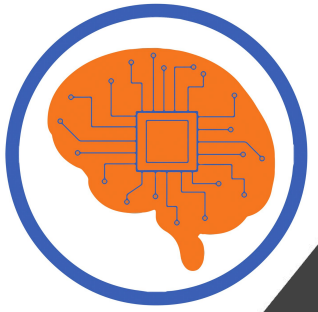
Tutorial de Python e Numpy para Algoritmos Genéticos

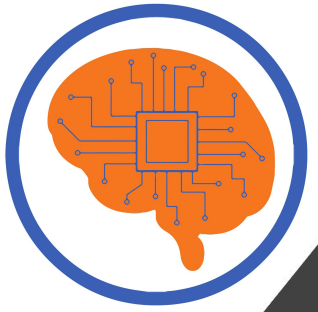
Grupo Turing





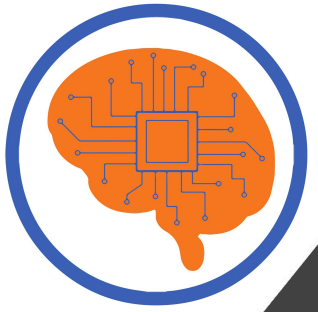
Variáveis

- Uma **variável** é um objeto capaz de armazenar um valor
- Tais variáveis podem reter por exemplo, os seguintes tipos de valores:
  - Números inteiros (..., -2, -1, 0, 1, 2, ...)
  - Pontos flutuantes (-2.102, 1.0, 12.133, ...)
  - Caracteres ('a', 'b', '!', 'A', ...)
  - Conjunto de caracteres ('banana', 'uva12', 'abacaxi')
  - Listas ([1,2,3,4], ['banana', 'uva', 2, 'abacaxi'])
  - Entre outros ...



Declarando Variáveis

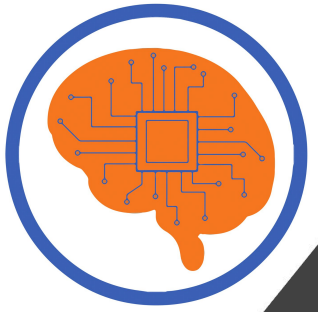
```
# tudo o que for escrito após hashtag é comentário
# comentários não influenciam o funcionamento do código
# sempre deixe seu programa o mais comentado possível

one = 1 # criando variável one, com valor 1.
# a variável pode ter o nome que você quiser
var_float = 1.2
serie = 'Um maluco no pedaço'
```



Listas

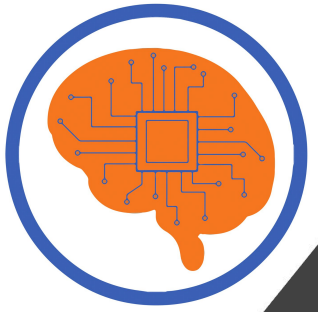
- Uma **lista** é uma variável capaz de armazenar mais de valor

```
minha_lista = [15,25,'uva'] # declaração de uma lista
# podemos acessar seus valores utilizando índices
print(minha_lista[0]) # 15
print(minha_lista[1]) # 25
print(minha_lista[2]) # uva
```



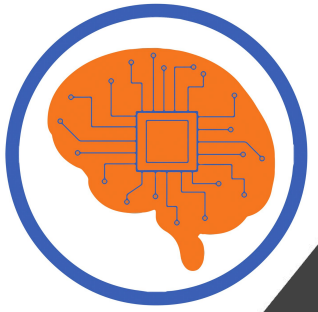
TAMANHO DE UMA LISTA (LEN)

```
lista = [2, 4, girafa]
len(lista) #esse comando retorna 3

lista_composta = [[0,2], [girafa,leao,zebra]]
len(lista_composta) #esse comando retorna 2
len(lista[0]) #esse comando retorna 2
len(lista[1]) #esse comando retorna 3
```



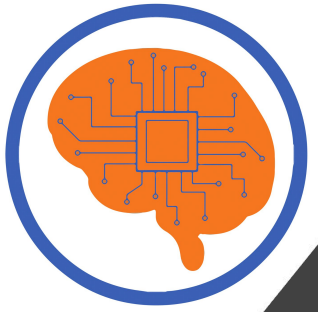
Adicionar elementos a uma lista

```
lista = [1, 3, 2]
lista.append(7)
lista.append(20)
print(lista)
# imprime [1, 3, 2, 7, 20]
```



Escolher parte de uma lista

```
# lista[a:b] retorna uma sublista a partir do elemento a
# até o elemento antes de b

lista = [1, 3, 2, 9, 5, 8]
print(lista[1:4]) # imprime [3, 2, 9] (índices 1 a 3)
print(lista[2:3]) # imprime [2] (índices 2 a 2)
print(lista[3:]) # imprime [9, 5, 8] (a partir do índice 3)
print(lista[:3]) # imprime [1, 3, 2] (índices até 2)
```



Condições

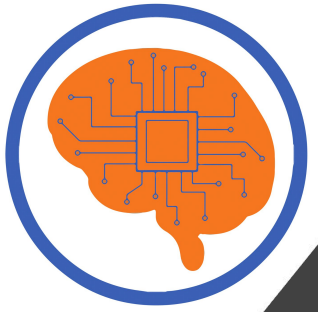
- Uma **condição** é dada por uma expressão que determina a execução de um comando
- Exemplo:

  Se a previsão do tempo indica que vai fazer frio,

  então põe casaco

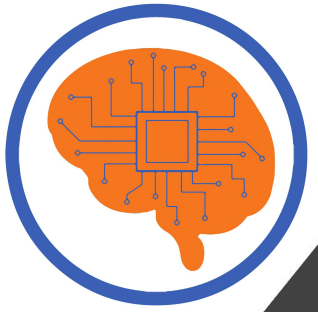
  Senão, não põe casaco
- Em python essas expressões são lógicas, como:

  verificação de igualdade (`a == b`), de desigualdade (`a < b`, `a > b`, `a <= b`, ...) ou diferença (`a != b`)



Adicionando Condições ao Código

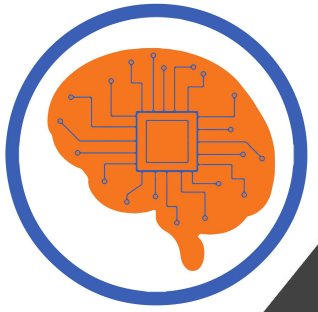
```
temp_do_dia = 10 # variável temperatura
if(temp_do_dia < 17): # se temp menor que 17 graus
    print('Levar casaco') # executa o comando de print
    # o comando print mostra na tela o que está entre aspas
else:
    print('Não levar casaco')
# perceba que nesse caso sempre será executado
# o primeiro comando, pois 10 é menor que 17,
# ou seja, a expressão condicional é verdadeira
```



LOOPS

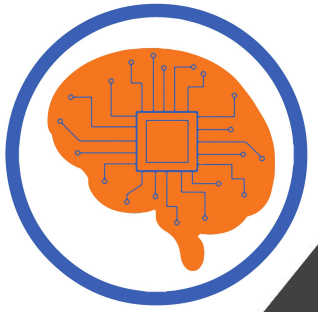
- Um **loop ou laço** é a repetição de uma sequência de comandos até que uma condição seja satisfeita
- Exemplo:

```
num = 1
Enquanto (num < 10) faça
    num = num + 1
    # isso muda o valor de num a cada iteração
Ao final da repetição num tem valor 10
```



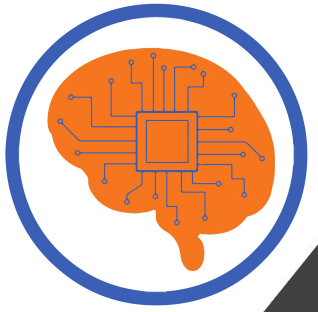
Criando um Loop - While

```
# em python criamos um laço com o operador while

num = 1 # variável int num
while num < 10: # enquanto a condição é verdadeira
    # repita os comandos
    print(num) # mostra o valor do num
    num = num + 1 # soma 1 ao valor de num
# mostra os números de 1 a 9
```



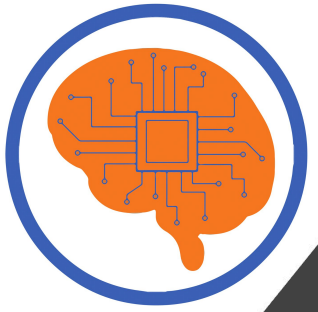
Criando um Loop - For

```
# em python a estrutura for serve para percorrer listas
# e outros elementos de estruturas de dados
lista = ['banana', 'uva', 'abacaxi']
for i in lista:
    print(i)
# banana
# uva
# abacaxi
# perceba que não a necessidade de iterar sobre um inteiro,
# como usando o laço while
```



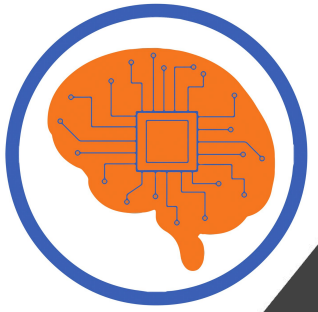
FOR IN RANGE

```
""" Esse comando cria uma lista contendo números inteiros começando do 0 e indo até o número anterior ao especificado no argumento"""
for i in range(5):
    print(i)
    # 0, 1, 2, 3, 4
start = 6 #Você pode adicionar mais argumentos não obrigatórios
stop = -1
step = -2
for i in range(start,stop,step):
    print(i)
    #6, 4, 2, 0
```



BREAK

```
"""O break é uma forma de sair do laço sem que a condição de parada necessariamente tenha sido atingida"""
i=0
while i<10:
    print(i)
    if i == 4:
        break
    i+=1
#0,1,2,3,4
```



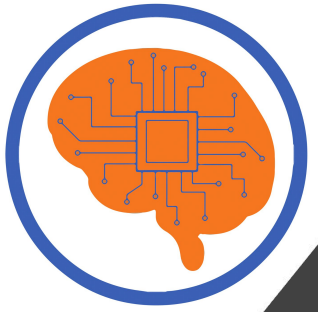
Funções

- Uma **função** é um bloco reutilizável de comandos que executam uma única operação
- Podemos pensar em uma função matemática

  f(x,y,z) = x*y*z

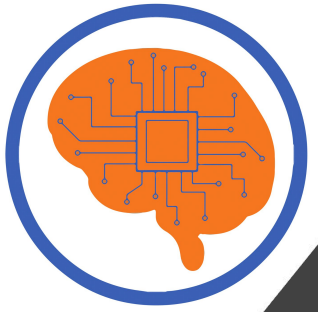
  f é uma função que executa o comando de multiplicar os valores de x, y e z, que são suas entradas

  Assim, podemos utilizar f para vários combinações de x, y e z



Definindo uma Função

```
# em python definimos uma função com o operador def

def minha_funcao(x,y,z):
    return x*y*z
print(minha_funcao(1,1,1)) # 1
print(minha_funcao(1,2,3)) # 6
# note que neste caso suas entradas são inteiros,
# mas podem ser de qualquer tipo, só precisa se atentar
# a forma de operá-los
```



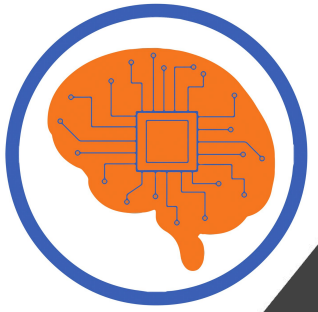
Numpy

Numpy (Numerical + Python) é uma biblioteca do Python que possui funções que auxiliam a manipulação de listas e matrizes, além de funções que produzem números aleatórios. Nesse workshop, usaremos duas funções:

- numpy.random.uniform
- numpy.argmax

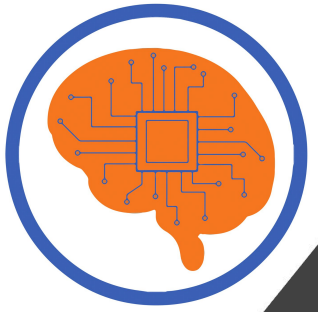
Para poder usar essas funções, no início do seu código deve estar escrito o seguinte comando:

```
import numpy as np
```



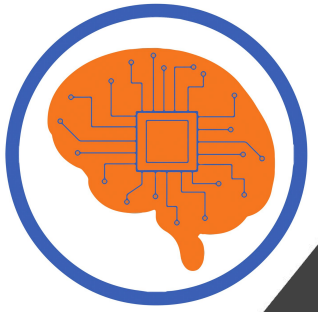
Matrizes

```
# podemos declarar matrizes numpy com o comando np.array()
# assim as matrizes podem ser operadas com certos comandos
>> A = np.array([[2, 0, 0],[0, 2, 0],[0, 0, 2]])
>> A
array([[2, 0, 0],
       [0, 2, 0],
       [0, 0, 2]])
>> A[0, 1] = 6
>> A[2, 2] = 7
>> A
array([[2, 6, 0],
       [0, 2, 0],
       [0, 0, 7]])
```



Multiplicação de Matrizes

```
# podemos declarar matrizes numpy com o comando np.array()
# assim as matrizes podem ser operadas com certos comandos
>> A = np.array([[2, 0, 0],[0, 2, 0],[0, 0, 2]])
>> A
array([[2, 0, 0],
       [0, 2, 0],
       [0, 0, 2]])
>> B = np.array([2, 1, 3])
>> B
array([2, 1, 3])
>> A @ B # com o operador @ multiplicamos matrizes (arrays)
array([4, 2, 6])
```



Numpy.random.uniform()

Essa função recebe 3 parâmetros: a, b, n. Ela gera n números aleatórios no intervalo [a,b).

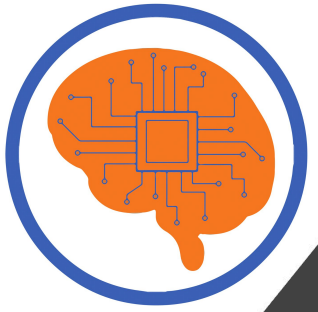
```
>> np.random.uniform(a,b,n)
```

Para criar um número aleatório entre 1 e 2:

```
>> np.random.uniform(1,2,1)
```

Para criar cinco números aleatórios entre -1 e 4:

```
>> np.random.uniform(-1,4,5)
```



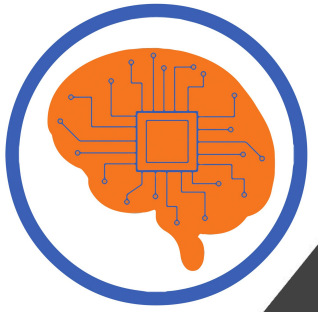
Numpy.random.uniform() com matrizes

Essa função também pode gerar matrizes de números aleatórios.

```
>> np.random.uniform(a,b,(m,n))
```

Para criar uma matriz 3x5 com números entre -1 e 1:

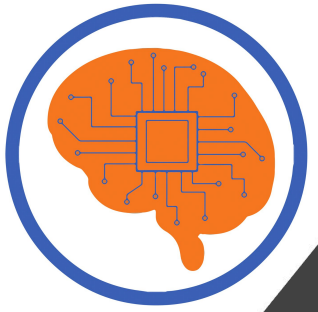
```
>> np.random.uniform(-1,1,(3,5))
array([[ 0.381, -0.101,  0.764,  0.759, -0.871],
       [-0.477,  0.904, -0.529, -0.963,  0.925],
       [-0.477,  0.910,  0.645,  0.636,  0.098]])
```



NUMPY.ARGMAX()

Essa função recebe como entrada uma lista e retorna o índice com o maior valor da lista (lembre que começa a partir do 0). Exemplo:

```
>> np.argmax([3,6,3,5,1])
1

>> np.argmax([1,2,3,4])
3
```



RANDOM.CHOICES()

Essa última função não é do numpy (numpy possui uma função parecida), ela é da biblioteca random. Essa função retorna k elementos aleatórios de uma lista. Para usá-la, deve importá-la no seu código:

```
import random
>> lista = [1,2,3,4,5,6]
>> random.choices(lista, k=4)
[1, 2, 6, 3]
>> a, b = random.choices(lista, k=2)
>> a
2
```