ESCOLA INFINITA DE INTELIGÊNCIA ARTIFICIAL

PYTHON - DADOS E OPERADORES

VARIÁVEIS

VARIÁVEIS:

Criar variável
nome_variavel = 5

IMPRIMIR VALOR DA VARIÁVEL
print(nome_variavel)

ENTRADA DE DADOS
input( )

FORMATAR TIPO DE DADO
format( )

OPERADORES RELACIONAIS

Operador de relação	Operação
a == b	igual
a != b	não igual
a < b	menor que
a <= b	menor ou igual a
a > b	maior que
a >= b	maior ou igual a

TRANSFORMAÇÃO DE DADOS

Comando	Para que serve
type( )	Descobrir o **tipo de dado**
int( )	Transforma um dado em **int**
float( )	Transforma um dado em **float**
srt( )	Transforma um dado em **string**
bool( )	Transforma um dado em **booleano**

OPERADORES LÓGICOS

Combinam duas ou mais condições, temos dois operadores lógicos:

OPERADOR LÓGICO: and

- Se a condição um **E** a condição dois forem verdadeiras.

Exemplo:
a = 5
b = 200
c = 300

se a for menor que b **E** b menor que c

if a<b and b<c:
    print("A primeira e a segunda condições são verdadeiras")
>>> A primeira e a segunda condições são verdadeiras

OPERADOR LÓGICO : or

- Se a condição um **OU** a condição dois for verdadeira:

Exemplo:
a = 5
b = 200
c = 300

se a for menor que b **OU** b maior que c

if a<b and b>c:
    print("A primeira ou segunda condição é verdadeira")
>>> A primeira ou segunda condição é verdadeira

OPERADORES DE ATRIBUIÇÃO

Operador de atribuição

Atribui o valor que está à esquerda a sua direita.
O símbolo usado para essa operação é o sinal = .

Exemplo:
nome = "Pedro"
print(nome)
>>> Pedro

OPERADORES ARITMÉTICOS

Operador	Operação
+	adição
-	subtração
*	multiplicação
/	divisão
//	divisão (com resultado inteiro)
%	resto ou módulo
**	exponenciação ou potenciação

TIPOS DE DADOS

Tipos de Dados	Nome
número inteiro	**int**
texto	**string**
número decimal	**float**
dado booleano	**boolean**

PYTHON - ESTRUTURA DE DADOS

ESTRUTURA DE DADOS

Diferente de uma variável, as estrutura de dados servem para armazenar mais do que um valor (dado). Principais estrutura de dados do Python:

- **listas**
- **tuplas**
- **dicionários.**

Nas estruturas de dados cada valor corresponde a um número chamado índex. O computador começa a contar pelo 0.

Olhe o exemplo abaixo:

nome_lista = [“maça”, “pera”, 23, 9.75, "manga"]

ÍNDEX 0 1 2 3 4

TUPLAS

Tuplas são ordenadas, imutáveis e permitem itens duplicados (Como as tuplas são indexadas, elas podem ter itens com o mesmo valor).

CRIAR TUPLA

Criar tupla com itens

```
nome_tupla = (“maça”, “pera”, 23, 9.75, "manga")
```

Selecionar um item pelo *índex*

```
nome_tupla[1]
>>> ("pera")
```

Selecionar uma fatia pelo índex

```
nome_tupla[1:3]
>>> ('pera", 23)
```

Contar a quantidade de itens

```
len(nome_tupla)
>>> 5
```

Contar a quantidade de um item específico

```
nome_tupla.count("manga")
>>> manga
```

LISTAS

Listas são ordenadas, mutáveis e permitem itens duplicados (Como as listas são indexadas, elas podem ter itens com o mesmo valor).

CRIAR LISTA

Criar lista vazia

```
nome_lista = [ ]
```

Criar lista com itens

```
nome_lista = [“maça”, “pera”, 23, 9.75, "manga"]
```

ACESSAR ITEM DA LISTA

Selecionar um item

```
nome_lista[1]
>>> ["pera"]
```

Selecionar uma fatia de itens

```
nome_lista = [ incluso : não incluso ]
nome_lista = [1:3]
>>> ["pera" , 23]
```

MÉTODOS DE LISTA

Contar quantidade de itens

```
len(nome_lista)
>>> 5
```

Contar quantidade de itens específico

```
nome_lista.count("manga")
>>> 1
```

Alterar um item

```
nome_lista[4] = "limão"
>>> [“maça”, “pera”, 23, 9.75, "limão"]
```

Remover um item

```
nome_lista.remove("limão")
>>> [“maça”, “pera”, 23, 9.75]
```

Remover um item usando seu índex

```
nome_lista.pop(4)
```

Adicionar um item no final

```
nome_lista.append("42")
>>> [“maça”, “pera”, 23, 9.75, "42"]
```

Adicionar um item no índex específico

```
nome_lista.insert(3, "uva")
>>> [“maça”, “pera”, 23, "uva" 9.75, "42"]
```

DICIONÁRIOS

Dicionários são ordenados*, mutáveis e não permitem itens duplicados (Como dicionário possui item no formato chave:valor, dois itens podem usar chaves de mesmo nome.).

```
nome_dic = { "chave" : valor }
```

CRIAR DICIONÁRIO

```
nome_dic = { "artista"  : "Jorge Ben Jor",
      "álbum" :  "Samba Esquema Novo’"
                  "ano" :  1963}
```

ACESSAR ITEM DO DICIONÁRIO

```
nome_dic = ["artista"]
>>> Jorge Ben Jor
```

MÉTODOS DO DICIONÁRIO

Contar quantidade de itens

```
len(nome_dic)
>>> 3
```

Alterar um item

```
nome_dic["ano"] = 2021
>>> nome_dic = { "artista" : "Jorge Ben Jor",
      "álbum" :  "Samba Esquema Novo",
                  "ano": 2021}
```

Remover um item usando a chave

```
nome_dic.pop("álbum")
>>> nome_dic = { "artista" : "Jorge Ben Jor",
                  "ano": 2021}
```

Adicionar um item

```
nome_dic["país"] = "Brasil"
>>>  nome_dic = { "artista" : "Jorge Ben Jor",
                  "ano": 2021,
                 "país" : "Brasil"}
```

PYTHON - ESTRUTURA DE CONDIÇÃO

ESTRUTURA DE CONDIÇÃO

As estruturas de condição são usadas para estabelecer que se uma determinada condição for verdadeira, então execute um bloco de código.

Caso contrario, pule o bloco de código e continue a seguir.

Elas são construídas com as palavras reservadas:

- **if**
- **elif**
- **else**

Sua sintaxe é:

if condição for verdadeira:
execute esse bloco de código.

elif outra condição for verdadeira:
execute esse bloco de código

else nenhuma condição for verdadeira:
execute esse bloco de código.

Repare que o **bloco de código está identado**, ou seja, há um espaço / recuo entre o início da linha e o começo no bloco de código

ESTRUTURA DE CONDIÇÃO - ELIF

ESTRUTURA DE CONDIÇÃO ELIF

Podemos ler uma estrutura de condição **ELIF** como a seguir:

- **E SE (elif)** a condição anterior não for verdadeira, então tente está condição.

Exemplo:

```
a = 200
b = 5

if a < b:
    print("a é menor que b")

# E SE a for maior que b
elif a > b:
    print("a é maior que b")
>>>a é maior que b
```

ESTRUTURA DE CONDIÇÃO - IF

ESTRUTURA DE CONDIÇÃO IF

Podemos ler uma estrutura de condição **IF** como a seguir:

- **SE (if)** uma determinada condição for verdadeira, então execute um bloco de código.

Exemplo:

```
a = 200
b = 5

# SE a for maior que b
if a > b:
    print("a é maior que b")
>>>a é maior que b
```

ESTRUTURA DE CONDIÇÃO - ELSE

ESTRUTURA DE CONDIÇÃO ELSE

Podemos ler uma estrutura de condição **ELSE** como a seguir:

- **ENTÃO (else)** nenhuma condição anterior for verdadeira, execute o bloco de código a seguir

Exemplo:

```
a = 200
b = 5

if a < b:
    print("a é menor que b")


elif a == b:
    print("a é igual b")


# ENTÃO a é maior que b
else:
    print("a é maior que b")
>>>a é maior que b
```

Repare que não escrevemos nenhuma condição como a > b , pois usamos else quando NENHUMA outra condição for verdadeira, então só nos resta essa opção.

PYTHON - ESTRUTURA DE REPETIÇÃO

ESTRUTURA DE REPETIÇÃO

As estruturas de repetição são usadas para estabelecer que se uma determinada condição for verdadeira, é para continuar executando um bloco de código.

Caso contrario, pule o bloco de código e continue a seguir.

Elas são construídas com as palavras reservadas:

- **while**
- **for**

ESTRUTURA DE REPETIÇÃO - WHILE

A estrutura de repetição **while** repete a execução de um bloco de código **enquanto** a condição for verdadeira.

Exemplo:

- Incrementando uma unidade:
  variável = variável + 1

```
a = 1
# ENQUANTO a for menor que 5
while a<5:
    print(a)
    a = a + 1
>>> 1
>>> 2
>>> 3
>>> 4
```

Caso não haja incremento ou decremento no valor da variável, a condição se repetira infinitamente.

COMANDOS ESTRUTURA DE REPETIÇÃO

- **break**
  Com o comando break podemos **parar a repetição** antes de percorrer todos os itens da lista.

```
lista_compra = ['laranja', 'limão', 'maracujá', 'goiaba']

for variavel_do_for in lista_compra:
  print(variavel_do_for)
  if variavel_do_for == 'limão':
    break
>>> laranja
>>> limão
```

- **continue**
  Com o comando continue podemos **parar a repetição atual** e continuar com a próxima repetição.

```
lista_compra = ['laranja', 'limão', 'maracujá', 'goiaba']

for variavel_do_for in lista_compra:
    if variavel_do_for == 'limão':
    continue
    print(variavel_do_for)
>>> laranja
>>> maracujá
>>> goiaba
```

- **range()**
  Cria itens entre um determinado intervalo de tempo, sempre incrementando 1.

```
for i in range(5):
print(i)
>>> 0
>>> 1
>>> 2
>>> 3
>>> 4
>>> 5
```

ESTRUTURA DE REPETIÇÃO - FOR

A estrutura de repetição for executa um bloco de código uma vez para cada item de uma estrutura de dados.

Ele atribui este item à uma variável que funciona somente dentro do for e em seguida executa um bloco de código.

```
lista_compra = ['laranja', 'limão', 'maracujá', 'goiaba']

for variavel_do_for in lista_compra:
    print(variavel_do_for)
>>> laranja
>>> limao
>>> maracujá
>>> goiaba
```

USANDO FOR COM DATAFRAME

Podemos usar o comando for para atribuir à uma lista todos os valores de uma ou mais coluna de um DataFrame.

Usando o for para uma coluna de um DataFrame:

```
lista_vazia = []

for variavel_01 in df.COLUNA_ESCOLHIDA:
  lista_vazia.append(variavel_01)

print(lista_pedidos)
```

Usando o for para duas colunas de um DataFrame:

Para reunir o valor de duas colunas usamos o método zip( ) e então usamos a estrutura do for com duas variáveis

```
lista_vazia = []

for variavel_01,variavel_02 in zip(COLUNA_01, COLUNA_02):
    lista_vazia.append([variavel_01, variavel_02])
```

PYTHON - FUNÇÕES

FUNÇÕES

Função é um bloco de código que só é executado quanto é chamado.

- **Toda função é uma ação**
- Nos ajudam a diminuir repetições tornando o código mais organizado

Sua sintaxe é:

```
# Definindo a função
def  nome_funcao ( ):
     bloco de código


# Chamando a função
nome_funcao
```

Exemplo:

```
def parabens( ) :
     print("Parabéns, você criou uma função.")

parabens()
>>> Parabens, você criou uma função.
```

FUNÇÕES COM PARÂMETROS

Quando precisamos especificar detalhes de como fazer esta ação usamos parâmetro e argumento.

- **Parâmetro: variável que recebe um valor (argumento) para ser usado na função**
- **Argumento: valor atribuído ao parâmetro.**

Sua sintaxe é:

```
def nome_funcao( parametro_01 = argumento_01, parametro_02 = argumento_02):
     bloco de código
```

Exemplo:

```
def barra_01(quantidade=40,caracter='@'):
  print(quantidade*caracter)

barra_01(quantidade=10, caracter="!"
)
>>> !!!!!!!!!!
```

VARIÁVEIS GLOBAIS E LOCAIS

Em geral temos dois tipos de variáveis:

- **VARIÁVEL GLOBAL:**
  - É criada fora de uma função e podemos acessar o seu valor de qualquer parte do código.
- **VARIÁVEL LOCAL:**
  - É criada dentro de uma função.
  - Sendo inicializada a cada vez que chamamos a função.
  - Não podemos acessar o valor fora da função.

Exemplo:

- **ano_nascimento é uma variável global.**
- **idade é uma variável local.**

```
ano_nascimento = 1987

def descobrir_idade():
     Idade = 2021 - ano_nascimento
     print("Minha idade é: ", idade)

descobrir_idade( )
>>> Minha idade é 34
```

FUNÇÕES - COMANDO RETURN

Em muitos casos, usamos uma função para executar um bloco de código que nos retorne algo, por exemplo o resultado de uma conta.

- **O comando return dentro de uma função, nos retorna algum conteúdo.**

Exemplo:

```
def area_triangulo(base,altura):
     area = base*altura/2
     return area

area_triangulo(8,10)
>>> 40.0
```

FUNÇÕES - COMANDO RETURN

TRABALHANDO COM O VALOR DO RETURN

Para trabalhar com o valor retornado pelo return atribuímos este valor a uma variável.

Exemplo:

```
resultado = area_triangulo(3,2)
print(resultado)
>>> 3.0
```