Variáveis

Variáveis

- ▼ Variáveis: **int**, **float** e **str**
  - int = 3
  - float = 3.14 (utilizar ponto para separar a vírgula)
  - str = "String em Python" (É declarada ao escrever uma frase entre aspas)
- ▼ Variável **bool**
  - bool = True ou " " ou 1
  - bool = True ou "" ou 0
- ▼ Mais de uma variável na mesma linha

```
a, b, c = 1, 2, 3
```

- ▼ Constantes
  - Não existe uma maneira de criar uma constante em Python, que seja semelhante a forma como o DEFINE funciona na linguagem C, por exemplo.
  - Assim, ao criar uma variável com valor que não deve ser alterado, seu nome deve ser escrito com letras maiúsculas, sendo uma boa prática de programação.
- ▼ Descobrir tipo (**type**)

- Exemplo: `print ( type ( 3.14 ) )`
    - Exibe: <class 'float'>

▼ Verificar se caracteres são dígitos (**isdigit( )**)

- isdigit( ) é a função que verifica se todos os caracteres de uma string são dígitos.
    - `print ( "123" . isdigit ()) # True`
- Isso é útil ao solicitar um número utilizando input( ), pois como essa entrada será uma string, a função possibilitará identificar dígitos numéricos nela. Exemplo de uso:

```
numero = input(”Digite um número”)
if numero.isdigit():  #Entra no if caso a string seja um número
```

▼ Arredondar número decimal (**round( )**)

- Serve para arredondar um valor **float**, permitindo escolher a quantidade de casas decimais. Pode ser usado como alternativa do format (**f**). Exemplo:

```
print(round(3.141592))  # Exibe 3
```

- Escolher casas decimais:

```
print(round(3.141592, 3))  # Exibe 3.142
```

▼ Conversão de tipos

- Exemplo: `print ( "1" + 1 )` - Exibe um erro
    - `print ( int ( "1" ) + 1 )` - Exibe 2
- Exemplo: `print ( bool ( "" ))` = False | `print ( bool ( " " ))` = True (Qualquer valor dentro da string)
- `print ( "A" + str ( 1 ))` - Exibe A1

▼ Empacotamento e desempacotamento (***variavel**)

São recursos que permitem trabalhar com múltiplos valores de forma compacta e eficiente. O empacotamento ocorre quando diversos valores são agrupados em uma única variável, como ao criar uma tupla com `a = 1, 2, 3`. Já o desempacotamento permite distribuir os valores de uma coleção em variáveis individuais, como em `x, y, z = a`.

- Exemplo de empacotamento:

```
a = 1, 2, 3
print(a)  # (1, 2, 3)
```

- Exemplo de desempacotamento:

```
x, y, z = [1, 2, 3]
print(x, y, z)  # 1 2 3
```

- O operador `*` pode ser utilizado em situações onde não há variáveis suficientes para corresponder a todos os valores de uma coleção, permitindo agrupar os valores excedentes em uma única variável. Exemplo:

```
a, b, c = [1, 2, 3, 4, 5]
print(a, b, c)  # Exibirá um erro
ValueError: too many values to unpack (expected 3)
```

```
a, b, *resto = [1, 2, 3, 4, 5]
print(a, b, resto)  # Exibe: 1 2 [3, 4, 5]
```

> ❗ Em Python, é comumente utilizado o nome de variável "_ "para variáveis que não serão utilizadas novamente pelo código. Nesse caso, para o exemplo de código anterior, a variável `* resto` poderia ser substituída pelo nome `* _` , em situações nas quais ela representasse um valores que não seriam utilizados futuramente pelo código.

▼ Valores mutáveis e imutáveis

- Valores imutáveis
  - int, float, str, None, bool, range(0, 10) e tuple
- Valores mutáveis
  - list, dict e set

❗ Algum desses estão em outras páginas

▼ Valores Truthy e Falsy

- Valores Truthy :
  - Qualquer número diferente de `0` (`int` ou `float`)
    - Ex: `1`, `3`, `0.5`
  - Strings não vazias
    - Ex: `"Python"`, `" "`
  - Listas não vazias
    - Ex: `[ 1, 2, 3 ]`
  - Tuplas não vazias
    - Ex: `( 0, )`
  - Dicionários não vazios
    - Ex: `{ "chave" : "valor" }`
  - Conjuntos não vazios
- Valores Falsy:
  - `0` (zero inteiro)
  - `0.0` (zero float)
  - `""` (string vazia)
  - `[]` (lista vazia)
  - `()` (tupla vazia)
  - `{}` ou `dict ()` (dicionário vazio)
  - `set ()` (conjunto vazio)
  - `None`
  - `False`

        - Ex: `{ 1, 2, 3 }`
    - `True`