Variáveis, tipos e estruturas

Sunday, September 13, 2020 11:17 AM

Número

Os números podem ser do tipo **int (5)** ou do tipo **float (5,0)**

```
In [16]: int(6.3)
Out[16]: 6
```

```
In [18]: float(9)
Out[18]: 9.0
```

Abs(x) - Retorna o módulo de **x**

```
In [8]: abs(-2)
Out[8]: 2
```

Pow(x,y) - Retorna **x** elevado a **y**

```
In [9]: pow(5,2)
Out[9]: 25
```

Round(x) - Retorna o valor arredondado de **x**

```
In [14]: round(10/3)
Out[14]: 3
```

Operações com números

Operador	Significado	Exemplo
+	Soma	2 + 2 → 4
-	Subtração	3 – 2→ 1
*	Multiplicação	2 * 3 → 6
/	Divisão	10 / 2 → 5
%	Módulo	5 % 2 → 1
**	Potência	4 ** 2 → 16
int()	Converte para inteiro	int(3.2) → 3
float()	Converte para float	float(2) → 2.0

Operadores relacionais

Operador	Significado
==	Igualdade / equivalência
!=	Desigualdade / Inequivalência
>	Maior que
<	Menor que
>=	Maior que ou igual a
<=	Menor que ou igual a

String

String

Pode ser só uma letra ou uma frase, mas sempre deve estar escrita entre aspas, duplas ou simples.

s[n] - Retorna o caractere na posição **n** da string **s**

```
In [1]: s = 'string em python'

In [2]: s[0]
Out[2]: 's'
```

s[x : y : z] - Me escreve os caracteres da posição **x** a **y** de **z** em **z** ***Espaço também é um caractere**

```
In [9]: s[0:10:2]
Out[9]: 'srn m'
```

s.lower() - Escreve a string **s** em minúscula

s.upper() - Escreve a string **s** em maiúscula

```
In [11]: s.upper()
Out[11]: 'STRING EM PYTHON'
```

s.capitalize() - Escreve a primeira letra da string **s** em maiúsculo

```
In [12]: s.capitalize()
Out[12]: 'String em python'
```

s.split() - Separa as palavras da frase **s**, colocando-as em uma lista, cada palavra é um termo da lista

```
In [13]: s.split()
Out[13]: ['string', 'em', 'python']
```

s.count('x') - Conta quantas vezes o caractere **x** aparece

```
In [21]: s.count('y')
Out[21]: 1
```

s.find('x') - Retorna a posição do caractere **x**

```
In [22]: s.find('y')
Out[22]: 11
```

Lista [ ]

Listas são estruturas mutáveis, que guardam uma lista de elementos.

lista[ ] - Posso criar uma **lista** vazia e ir adicionando itens depois

lista[n] - Posso acessar o elemento na posição **n** da **lista**

```
In [1]: lista = ['arroz','suco', 'batatinha']

In [2]: lista[2]
Out[2]: 'batatinha'
```

del lista[n] - Posso deletar um elemento na posição **n** da **lista**

```
In [3]: del lista[0]

In [4]: lista
Out[4]: ['suco', 'batatinha']
```

lista[n][m] - Posso acessar assim um elemento na posição **n** da **lista**, e na posição **m** da **lista** dentro da **lista**

```
In [5]: lista = [[0,1], [2,3]]

In [7]: lista[1][0]
Out[7]: 2
```

len(lista) - Me mostra o comprimento da **lista**

```
In [8]: len(lista)
Out[8]: 2
```

max(lista) - Me mostra o maior valor da **lista**

```
In [9]: max(lista)
Out[9]: [2, 3]
```

min(lista) - Me mostra o menor valor da **lista**

```
In [10]: min(lista)
Out[10]: [0, 1]
```

lista.append(x) - Adiciona o número **x** ao final da **lista**, se for adicionar uma string preciso colocar entre aspas

lista.index('item') - Me retorna a posição do termo **item** da **lista**

lista.insert(x, 'item') - Insere na posição **x** o termo **item** na **lista**

```
In [16]: lista.insert(1,'batatinha')

In [18]: lista
Out[18]: [[0, 1], 'batatinha', [2, 3]]

In [19]: lista.index('batatinha')
Out[19]: 1
```

lista.remove('item') - Remove o termo item da **lista**

lista.sort() - Ordena a lista

Tupla ( )

As tuplas não são mutáveis como as listas, então eu posso usar os mesmos comandos usados nas listas, com exceção daqueles que alteram, como append, insert, remove.

```
In [2]: tupla = ('a', 10, 'batatinha')

In [3]: tupla.index('batatinha')
Out[3]: 2

In [4]: len(tupla)
Out[4]: 3
```

Para modificar um valor de uma tupla eu preciso transformá-la em uma lista (comando **list**), modificar a lista e depois transformar essa lista de volta numa tupla (comando **tuple**).

```
In [30]: tupla = ('a', 10, 'batatinha')

In [31]: lista = list(tupla)

In [32]: del lista[0]

In [33]: tupla = tuple(lista)

In [34]: tupla
Out[34]: (10, 'batatinha')
```

Dicionário { }

Dicionários apresentam a seguinte estrutura: **dic = {'chave' : 'valor'}**

dic.keys() - Me retorna as chaves do dicionario **dic**

dic.values() - Me retorna os valores do dicionario **dic**

dic.items() - Me retorna as chaves e os valores do dicionario **dic**

```
In [5]: dic = {'chave1' : 'valor1', 'chave2' : 'valor2'}

In [6]: dic.keys()
Out[6]: dict_keys(['chave1', 'chave2'])

In [7]: dic.values()
Out[7]: dict_values(['valor1', 'valor2'])

In [8]: dic.items()
Out[8]: dict_items([('chave1', 'valor1'), ('chave2', 'valor2')])
```

dic.update(dic2) - Adiciona os itens do dicionário **dic2** no **dic**

```
In [9]: dic2 = {'chave_add' : 'valor_add'}

In [10]: dic.update(dic2)

In [11]: dic
Out[11]: {'chave1': 'valor1', 'chave2': 'valor2', 'chave_add': 'valor_add'}
```

dic[chave_nova] = valor novo - Adiciona ao final do dicionário **dic** uma **chave_nova** e um **valor novo**

```
In [14]: dic
Out[14]: {'chave1': 'valor1',
          'chave2': 'valor2',
          'chave_add': 'valor_add',
          'ch_nova': 'v_novo'}
```

Variáveis

Uma variável pode ser do tipo: **número, string, lista, tupla ou dicionário.**

Tipo de Objeto	Categoria	Mutável?
Números	Numérico	Não
Strings	Sequência	Não
Listas	Sequência	Sim
Dicionários	Mapeamento	Sim
Tuplas	Sequência	Não

Operadores de atribuição

Operador	Significado	Exemplo
=	Atribuição	z = 10
+=	Soma	z += 10 (equivalente a z = z + 10)
-=	Subtração	z -= 10 (equivalente a z = z – 10)
*=	Multiplicação	z *= 10 (equivalente a z = z * 10)
/=	Divisão	z /= 10 (equivalente a z = z / 10)
%=	Módulo	z %= 10 (equivalente a z = z % 10)
**=	Potência	z **= 10 (equivalente a z = z ** 10)
//=	Divisão inteira	z //= 10 (equivalente a z = z // 10)

Operadores lógicos

Operador	Significado	Exemplo
and	Se ambos operadores forem True, retorna True	(x and y) é True
or	Se um dos operadores for True, retorna True	(x or y) é True
Not	Usado para reverter o estado da lógica	Not (x and y) é False