Sumário

[1 Propriedades 2](#_Toc151402593)

[2 Strings 2](#_Toc151402594)

[3 Lista [ ] 2](#_Toc151402595)

[4 Tupla ( ) 2](#_Toc151402596)

[5 IF - ELIF – ELSE 2](#_Toc151402597)

[6 While 2](#_Toc151402598)

[7 For 2](#_Toc151402599)

[8 Funções 2](#_Toc151402600)

[8.1 Definir uma Função 2](#_Toc151402601)

[8.2 Escopo da Função 2](#_Toc151402602)

[8.3 return 3](#_Toc151402603)

[8.4 \*args (Argumentos não nomeados) 4](#_Toc151402604)

# 1 Propriedades

# 2 Strings

# 3 Lista [ ]

# 4 Tupla ( )

# 5 IF - ELIF – ELSE

# 6 While

# 7 For

# 8 Funções

## 8.1 Definir uma Função

## 8.2 Escopo da Função

Escopo significa o local onde aquele código pode atingir.

Existe o escopo global e local.

O escopo global é o escopo onde todo o código é alcançável.

O escopo local é o escopo onde apenas nomes do mesmo local

podem ser alcançados.

x = 1

def escopo1():

    x = 2

    def escopo2():

        x = 3

        print(x)

    print(x)

    escopo2()

print(x)

escopo1()

print(x)

Saída

1

2

3

1

O valor da variável fora da função não altera o valor da variável dentro da função, e vice-versa. É possível alterar variáveis de escopo externo em escopo interno usando a propriedade global.

def escopo1():

    global x

    x = 2

    def escopo2():

        global x

        x = 3

        print(x)

    print(x)

    escopo2()

print(x)

escopo1()

print(x)

Saída

1

2

3

3

## 8.3 return

Retorna os valores das funções

def soma(*x*, *y*):

    if *x* > 10:

        return [10, 20]

    return *x* + *y*

soma1 = soma(2, 2)

soma2 = soma(3, 3)

print(soma1)

print(soma2)

print(soma(11, 55))

saída

4

6

[10, 20]

## 8.4 \*args (Argumentos não nomeados)

args - Argumentos não nomeados

\* - \*args (empacotamento e desempacotamento)

Lembre-te de desempacotamento:

x, y, \*resto = 1, 2, 3, 4

print(x, y, resto)

def soma(\**args*):

    total = 0

    for numero in *args*:

        total += numero

    return total

print(soma(1, 2, 3))

saída

6

## 8.5 Closure

def criar\_multiplicador(*multiplicador*):

    def multiplicar(*numero*):

        return *numero* \* *multiplicador*

    return multiplicar

duplicar = criar\_multiplicador(2)

triplicar = criar\_multiplicador(3)

quadruplicar = criar\_multiplicador(4)

print(duplicar(2))

print(triplicar(2))

print(quadruplicar(2))

# 9 dict { } (dicionários)

Métodos úteis dos dicionários em Python

**len** - quantas chaves

**keys** - iterável com as chaves

**values** - iterável com os valores

**items** - iterável com chaves e valores

**setdefault** - adiciona valor se a chave não existe

**copy** - retorna uma cópia rasa (shallow copy)

**get** - obtém uma chave

**pop** - Apaga um item com a chave especificada (del)

**popitem** - Apaga o último item adicionado

**update** - Atualiza um dicionário com outro

pessoa = {

    'nome': 'Luiz Otávio',

    'sobrenome': 'Miranda',

    'idade': 900,

}

pessoa.setdefault('idade', 0)

print(pessoa['idade'])

# print(len(pessoa))

# print(list(pessoa.keys()))

# print(list(pessoa.values()))

# print(list(pessoa.items()))

# for valor in pessoa.values():

#     print(valor)

# for chave, valor in pessoa.items():

#     print(chave, valor)

# print(p1['nome'])

# print(p1.get('nome', 'Não existe'))

# nome = p1.pop('nome')

# print(nome)

# print(p1)

# ultima\_chave = p1.popitem()

# print(ultima\_chave)

# print(p1)

# p1.update({

#     'nome': 'novo valor',

#     'idade': 30,

# })

# p1.update(nome='novo valor', idade=30)

# tupla = (('nome', 'novo valor'), ('idade', 30))

#lista = [['nome', 'novo valor'], ['idade', 30]]

#p1.update(lista)

#print(p1)

# 10 Set ( )