Sumário

[1 Propriedades 2](#_Toc151660137)

[2 Strings 2](#_Toc151660138)

[3 Lista [ ] 2](#_Toc151660139)

[4 Tupla ( ) 2](#_Toc151660140)

[5 IF - ELIF – ELSE 2](#_Toc151660141)

[6 While 2](#_Toc151660142)

[7 For 2](#_Toc151660143)

[8 Funções 2](#_Toc151660144)

[8.1 Definir uma Função 2](#_Toc151660145)

[8.2 Escopo da Função 2](#_Toc151660146)

[8.3 return 3](#_Toc151660147)

[8.4 \*args (Argumentos não nomeados) 4](#_Toc151660148)

[8.5 Closure 4](#_Toc151660149)

[9 dict { } (dicionários) 4](#_Toc151660150)

[10 Set ( ) 6](#_Toc151660151)

# 1 Propriedades

Não existe **constantes** em python, mas por convenção, quando se deseja criar uma variável que o valor não será alterado, letras maiúsculas na variável para indicar que aquele valor é constante.

# 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 ( )

Sets - Conjuntos em Python (tipo set).

Sets em Python são mutáveis, porém aceitam apenas tipos imutáveis como valor interno.

Criando um set.

#set(iterável) ou {1, 2, 3}

s1 = set('Luiz')

s1 = set()  # vazio

s1 = {'Luiz', 1, 2, 3}  # com dados

Sets são eficientes para remover valores duplicados de iteráveis.

Seus valores serão sempre únicos;

Não aceitam valores mutáveis;

Não tem índexes;

Não garantem ordem;

São iteráveis (for, in, not in)

l1 = [1, 2, 3, 3, 3, 3, 3, 1]

s1 = set(l1)

l2 = list(s1)

s1 = {1, 2, 3}

print(3 not in s1)

for numero in s1:

    print(numero)

## 10.1 Métodos úteis

add, update, clear, discard

s1 = set()

s1.add('Luiz')

s1.add(1)

s1.update(('Olá mundo', 1, 2, 3, 4))

# s1.clear()

s1.discard('Olá mundo')

s1.discard('Luiz')

print(s1)

# print(s1)

## 10.2 Operadores úteis

união | união (union) - Une

intersecção & (intersection) - Itens presentes em ambos

diferença - Itens presentes apenas no set da esquerda

diferença simétrica ^ Itens que não estão em ambos

s1 = {1, 2, 3}

s2 = {2, 3, 4}

s3 = s1 | s2

s3 = s1 & s2

s3 = s2 - s1

s3 = s1 ^ s2

print(s3)

Exemplo:

letras = set()

while True:

    letra = input('Digite: ')

    letras.add(letra.lower())

    if 'l' in letras:

        print('PARABÉNS')

        break