FUNÇÕES

Uma função é um bloco de código que só é executado quando é chamado. Você pode passar dados, que chamamos de parâmetros, para uma função e ela pode ou não retornar dados como resultado.

Criação de uma função

Em Python, uma função é definida usando a palavra-chave `def` :

Exemplo:

```
def funcao():
 print("Olá, Mundo!")
```

Chamando uma função

Para chamar uma função, use o nome da função seguido por parênteses.

Exemplo:

```
def funcao():
 print("Olá, Mundo!")
funcao()

#Resultado:
Olá, Mundo!
```

Argumentos

As informações podem ser passadas para funções como argumentos.

Os argumentos são especificados após o nome da função, entre parênteses. Você pode adicionar quantos argumentos quiser, apenas separe-os com uma vírgula.

O exemplo a seguir tem uma função com um argumento (fname). Quando a função é chamada, passamos um primeiro nome, que é usado dentro da função para imprimir o nome completo:

Exemplo:

```
def funcao(fname):
 print(fname + " Correia Pessoa")
funcao("João Vítor")
#Os argumentos costumam ser abreviados para args nas documentações do Python.

#Resultado:
João Vítor Correia Pessoa
```

Parâmetros ou argumentos?

Os termos *parâmetro* e *argumento* podem ser usados para a mesma coisa: informações que são passadas para uma função.

Da perspectiva de uma função:

- Um parâmetro é a variável listada entre parênteses na definição da função.
- Um argumento é o valor enviado para a função quando ela é chamada.

Número de Argumentos

Por padrão, uma função deve ser chamada com o número correto de argumentos. O que significa que se sua função espera 2 argumentos, você deve chamar a função com 2 argumentos, nem mais, nem menos.

Exemplo:

```
def funcao(fname, lname):
 print(fname + " " + lname) #note o espaço para manter os nomes separados
funcao("João Vítor", "Pessoa")

#Resultado:
João Vítor Correia Pessoa
```

Se você tentar chamar a função com 1 ou 3 argumentos, receberá um erro.

Exemplo:

```
def funcao(fname, lname):
 print(fname + " " + lname)
funcao("João")

#Resultado:
TypeError: funcao() missing 1 required positional argument: 'lname'
#Indicando exatamente a ausência de um argumento na chamada da função.
```

Argumentos arbitrários

Se você não sabe quantos argumentos serão passados para sua função, adicione um `*` antes do nome do parâmetro na definição da função. Dessa forma, a função receberá uma *tupla* de argumentos e poderá acessar os itens de acordo.

Exemplo:

```
def funcao(*nomes):
 print("O nome escolhido foi " + nomes[2])
funcao("João", "Maria", "Kamylla")

#Resultado:
O nome escolhido foi Kamylla
#Note que ao utilizar o * não é necessário declarar a quantidade de argumentos na abertura da função, mas na chamada será necessário.
```

Os argumentos arbitrários costumam ser abreviados para ***args** nas documentações do Python.

Argumentos de Palavras-Chave

Você também pode enviar argumentos com a *chave* = sintaxe de valor . Dessa forma, a ordem dos argumentos não importa.

Exemplo:

```
def funcao(nome3, nome2, nome1):
print("O nome escolhido foi " + nome3)
funcao(nome1 = "João", nome2 = "Maria", nome3 = "Kamylla")

#Resultado:
O nome escolhido foi Kamylla
```

Os argumentos de palavra-chave costumam ser abreviados para **kwargs** nas documentações do Python.

Argumentos de palavras-chave arbitrárias

Se você não souber quantos argumentos de palavra-chave serão passados para sua função, adicione dois asteriscos: `**` antes do nome do parâmetro na definição da função. Dessa forma,

a função receberá um *dicionário* de argumentos e poderá acessar os itens de acordo.

Exemplo:

```
def funcao(**nomes):
print("O nome escolhido foi" + nome1["João"])
my_function(nome1 = "João", nome2 = "Maria")

#Resultado:
O nome escolhido foi João
```

Os argumentos de palavra-chave arbitrária costumam ser abreviados para **`**kwargs`** nas documentações do Python.

Valor do parâmetro padrão

O exemplo a seguir mostra como usar um valor de parâmetro padrão. Se chamarmos a função sem argumento, ela usará o valor padrão.

Exemplo:

```
def funcao(nacionalidade = "brasileiro"):
 print("Eu sou do" + nacionalidade)
funcao("japonês")
funcao("argentino")
funcao()

#Resultado:
Eu sou do japonês
Eu sou do argentino
Eu sou do brasileiro
Eu sou do brasileiro
```

Passando uma lista como um argumento

Você pode enviar qualquer tipo de dado de argumento para uma função (string, número, lista, dicionário, etc.), e será tratado como o mesmo tipo de dado dentro da função.

Exemplo:

```
def funcao(frutas):
 for x in frutas:
  print(x)
frutas = ["maçã", "banana", "cereja"]
funcao(frutas)

#Resultado:
maçã
banana
cereja
```

Valores Retornados

Para permitir que uma função retorne um valor, use a instrução `return`:

Exemplo:

```
def funcao(x):
 return 5 * x #Porduto de x com 5
print(funcao(3))
print(funcao(5))
print(funcao(9))

#Resultado:
15
25
45
```

A declaração de passagem

As funções não podem estar vazias na definição, mas se por algum motivo você tiver uma função sem conteúdo, coloque a instrução `pass` para evitar um erro.

Exemplo:

```
def funcao():
 pass
```

Recursão

Python também aceita recursão de função, o que significa que uma função definida pode chamar a si mesma.

A recursão é um conceito matemático e de programação comum. Isso significa que uma função chama a si mesma. Isso tem a vantagem de significar que você pode percorrer os dados para chegar a um resultado.

O desenvolvedor deve ter muito cuidado com a recursão, pois pode ser muito fácil escorregar e escrever uma função que nunca termina, ou que usa uma quantidade excessiva de memória ou potência do processador. No entanto, quando escrito corretamente, a recursão pode ser uma abordagem de programação muito eficiente e matematicamente elegante.

💡 Nota: Em ciência da computação, a recursividade é a definição de uma sub-rotina que pode invocar a si mesma. Um exemplo de aplicação da recursividade pode ser encontrado nos analisadores sintáticos recursivos para linguagens de programação.

Exemplo:

```
def test_recursion(k):
 if (k > 0):
  resultado = k + test_recursion(k - 1)
  print(resultado)
 else:
  resultado = 0
 return resultado
print("\n\nResultados de exemplo de recursão")
test_recursion(6)

#Resultado:


Resultados de exemplo de recursão
1
3
6
10
15
```

Neste exemplo, test_recursion() é uma função que definimos para chamar a si mesma. Usamos a variável k como os dados, que diminui (-1) toda vez que recursamos. A recursão termina quando a condição não for maior que 0 (ou seja, quando for 0).

Para um novo desenvolvedor, pode levar algum tempo para descobrir como isso funciona exatamente; a melhor maneira de descobrir é testando e modificando.