

## REAPROVEITANDO CÓDIGOS COM FUNÇÕES

# REAPROVEITANDO CÓDIGOS COM FUNÇÕES

Neste vídeo, o expert [Allan Dieguez](#), Head de Data Science da LuizaLabs, apresenta o processo de reaproveitamento de códigos utilizando funções no Python.



12:32



## POR QUE REAPROVEITAR O CÓDIGO?

- Menos código é escrito e o programa fica mais fácil de ler e manter.
- A lógica é mantida em um só lugar, permitindo adicionar features ou corrigir bugs em todo o programa tendo a mesma quantidade de trabalho.

## ESTRUTURA DE UMA FUNÇÃO

Uma função é composta minimamente de alguns elementos básicos, como mostrado no exemplo abaixo:

```
def nome_da_função(<parâmetros> ou NADA):  
    <código da função>  
    <retorno> ou NADA
```

## REAPROVEITANDO CÓDIGOS COM FUNÇÕES

sequência, seguem os parênteses, que podem estar vazios ou conter argumentos como nomes de parâmetros.

Fecha-se a primeira linha com dois pontos, indicando que, o que vier a seguir, na linha de baixo, será o código da função indentado por 4 espaços ou TAB. Essa indentação de Python indica que todo o código alinhado com a indentação pertence ao escopo definido na linha acima.

O nome da função pode ser escrita de diversas formas. A mais correta, de acordo com a convenção [PEP8](#) para nomes de função, segue as seguintes regras:

- Apenas palavras com letras minúsculas.
- Palavras separadas por underscore (\_).
- O nome deve explicar sucintamente o que a função faz.

Alguns exemplos:

- Exemplo **bom**: **soma\_dois\_numeros**
- Exemplo **ruim**, não explica o que fez: **s\_nums\_xy**
- Exemplo ruim, pouco sucinto:  
**somatorio\_regular\_sem\_pesos\_de\_numeros**
- Outro exemplo **ruim**, que não usa o formato sugerido:  
**SomarDoisNumeros**

**Muito importante:** a partir do momento em que a linha **return** for executada, o interpretador do Python encerra a execução e volta para as linhas de código imediatamente após a chamada da função.

A seguir, um exemplo de função sem parâmetros e que não retorna nada.

## REAPROVEITANDO CÓDIGOS COM FUNÇÕES

A **chamada da função** sempre necessita dos parêntesis para executar. Se não, o interpretador entende que é o tipo da função que está sendo retornado.

```
dizer_bom_dia()  
Bom Dia
```

## ARGUMENTOS DA FUNÇÃO

Os **argumentos** (ou parâmetros de entrada) da função são declarados dentro dos parênteses na primeira linha. São efetivamente **variáveis criadas dentro do escopo da função**, que só podem ser acessadas pelo nome enquanto o interpretador estiver executando a função.

Vamos criar uma função que soma dois números:

```
def soma_dois_numeros(n1, n2):  
    return n1 + n2
```

Agora vamos chamar essa função:

```
soma_dois_numeros(23, 44)  
67
```

## ARGUMENTOS DEFAULT

Se a pessoa desenvolvedora colocar uma **atribuição** de valor em um argumento de entrada, esse argumento **deixa de ser obrigatório** na chamada da função. Se for esse o caso, o argumento **assume o valor colocado como padrão** e toda vez que a função for chamada sem o argumento o parâmetro terá o valor definido na função.

Vamos reaproveitar um código que escrevemos na aula de estruturas condicionais e criar uma função que chama dizer\_cumprimento:

## REAPROVEITANDO CÓDIGOS COM FUNÇÕES

```
print( 'Bom dia' )  
elif 13 <= hora <= 18:  
    print("Boa tarde")  
elif 19 <= hora <= 24:  
    print("Boa noite")  
else:  
    print("Ainda é de madrugada")
```

Podemos chamar a função passando o argumento de hora:

```
dizer_cumprimento(20)  
Boa noite
```

Ou podemos chamar a função sem nenhum parâmetro, mas como definimos 12 como o valor padrão para hora no momento que criamos a função o Python irá assumir este como o valor para hora.

```
dizer_cumprimento()  
Bom dia
```

## (BÔNUS) FUNÇÃO ANÔNIMA: *LAMBDA*

A função anônima, ou lambda, é um recurso da linguagem **Python** que permite que seja criado um código que funciona como **uma função simples**, sem precisar criar um espaço na memória especialmente para isso, como é feito com as funções definidas por def.

É uma das maneiras ideais para se passar um comportamento de função muito simples para dentro de alguma outra função ou trecho de código que será executado.

### Estrutura geral de um Lambda

A composição do Lambda é muito simples:

```
lambda <parâmetros> : <operação>
```

## REAPROVEITANDO CÓDIGOS COM FUNÇÕES

sequência, são declarados os parâmetros que serão usados na operação que segue os dois pontos (:).

Essa operação precisa ser muito simples, não pode passar de uma linha de código que trabalha exclusivamente os parâmetros declarados na entrada, e não há necessidade de aplicar um **return**: o que resultar da operação será automaticamente retornado.

Os parâmetros são declarados da mesma forma que nas funções tradicionais, com a diferença de que não são declarados dentro de parênteses.

Vamos criar uma função lambda

```
lambda x: x ** 2
```

Criando assim, como ela é uma função anônima, não temos nenhum meio de executá-la, vamos atribuí-la a uma variável então para poder testar.

```
fn = lambda x: x ** 2  
fn(12)  
144
```

## EXEMPLOS DE USO

É mais fácil entender a utilidade do Lambda quando ele é usado em funções que esperam um Lambda para funcionar bem. O exemplo a seguir ilustra bem o seu uso.

### Função sorted()

Função que ordena uma lista de elementos ordenáveis. Usado normalmente não necessita de Lambda, como no exemplo a seguir:

## REAPROVEITANDO CÓDIGOS COM FUNÇÕES

```
[1, 2, 2, 4, 4, 5, 7, 8, 9]
```

É possível modificar o `sorted` para ordenar primeiro os números pares e seguir com os ímpares ordenados.

```
sorted(numeros, key=lambda x: x * 10 if (x % 2 == 1) else x)  
[2, 2, 4, 4, 8, 1, 5, 7, 9]
```

---

**AVANÇAR**

## REAPROVEITANDO CÓDIGOS COM FUNÇÕES

### CONCLUSÃO



Parabéns! Que bom que você chegou até aqui.

**Você agora consegue reaproveitar códigos através das funções no Python.** Revise os aprendizados desta aula e avalie se você já tem segurança para realizar essas ações.

### CHECKLIST

Ao final desta aula, você deve sentir segurança para:

- Conhecer a estrutura de uma função (**def, nome, argumento, return**)
- Entender que pode ter argumentos **opcionais** e **obrigatórios**
- Entender o que é **docstring**
- Aplicar as funções **dir()** e **help()**
- **Escrever uma função** simples
- Criar uma função que recebe **argumentos**
- Conhecer a estrutura de uma função anônima (**lamda: argumentos**)

## REAPROVEITANDO CÓDIGOS COM FUNÇÕES

Caso ainda tenha dúvidas, não hesite em voltar aos conteúdos e buscar ajuda, esses fundamentos são super importantes para você construir uma base de trabalho sólida. Esta aula ficará disponível para você rever quando quiser.

[Notebook usado no vídeo](#)

Até breve!

---

CONCLUIR