Funções

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

Entender como funcionam as funções em Python.

Pré-requisitos

- Python 3
- VSCode

Percurso

Etapa 1 **Estudo aprofundado sobre funções**

Etapa 1

Estudo aprofundado sobre funções

O que são funções?

Função é um bloco de código identificado por um nome e pode receber uma lista de parâmetros, esses parâmetros podem ou não ter valores padrões. Usar funções torna o código mais legível e possibilita o reaproveitamento de código. Programar baseado em funções, é o mesmo que dizer que estamos programando de maneira estruturada.

Exemplo

```
def exibir_mensagem():
    print("Olá mundo!")

def exibir_mensagem_2(nome):
    print(f"Seja bem vindo {nome}!")

def exibir_mensagem_3(nome="Anônimo"):
    print(f"Seja bem vindo {nome}!")

exibir_mensagem()
exibir_mensagem_2(nome="Guilherme")
exibir_mensagem_3()
exibir_mensagem_3(nome="Chappie")
```

Retornando valores

Para retornar um valor, utilizamos a palavra reservada **return**. Toda função Python retorna **None** por padrão. Diferente de outras linguagens de programação, em Python uma função pode retornar mais de um valor.

Exemplo

```python
def calcular_total(numeros):
    return sum(numeros)

def retorna_antecessor_e_sucessor(numero):
    antecessor = numero - 1
    sucessor = numero + 1

    return antecessor, sucessor

calcular_total([10, 20, 34])  # 64
retorna_antecessor_e_sucessor(10)  # (9, 11)
```

Argumentos nomeados

Funções também podem ser chamadas usando argumentos nomeados da forma chave=valor.

Exemplo

```python
def salvar_carro(marca, modelo, ano, placa):
    # salva carro no banco de dados...
    print(f"Carro inserido com sucesso! {marca}/{modelo}/{ano}/{placa}")

salvar_carro("Fiat", "Palio", 1999, "ABC-1234")
salvar_carro(marca="Fiat", modelo="Palio", ano=1999, placa="ABC-1234")
salvar_carro(**{"marca": "Fiat", "modelo": "Palio", "ano": 1999, "placa": "ABC-1234"})

# Carro inserido com sucesso! Fiat/Palio/1999/ABC-1234
```

Args e kwargs

Podemos combinar parâmetros obrigatórios com args e kwargs. Quando esses são definidos (*args e **kwargs), o método recebe os valores como tupla e dicionário respectivamente.

Exemplo

```
def exibir_poema(data_extenso, *args, **kwargs):
    texto = "\n".join(args)
    meta_dados = "\n".join([f"{chave.title()}: {valor}" for chave, valor in
kwargs.items()])
    mensagem = f"{data_extenso}\n\n{texto}\n\n{meta_dados}"
    print(mensagem)

exibir_poema("Zen of Python", "Beautiful is better than ugly.", autor="Tim
Peters", ano=1999)
```

Parâmetros especiais

Por padrão, argumentos podem ser passados para uma função Python tanto por posição quanto explicitamente pelo nome. Para uma melhor legibilidade e desempenho, faz sentido restringir a maneira pelo qual argumentos possam ser passados, assim um desenvolvedor precisa apenas olhar para a definição da função para determinar se os itens são passados **por posição, por posição e nome, ou por nome.**

```
def f(pos1, pos2, /, pos_or_kwd, *, kwd1, kwd2):
      -----------    ----------     ----------
        |             |                   |
        |        Positional or keyword    |
        |                                 - Keyword only
         -- Positional only
```

Positional only

```
def criar_carro(modelo, ano, placa, /, marca, motor, combustivel):
    print(modelo, ano, placa, marca, motor, combustivel)


criar_carro("Palio", 1999, "ABC-1234", marca="Fiat", motor="1.0",
combustivel="Gasolina")  # válido

criar_carro(modelo="Palio", ano=1999, placa="ABC-1234", marca="Fiat",
motor="1.0", combustivel="Gasolina")  # inválido
```

Keyword only

```python
def criar_carro(*, modelo, ano, placa, marca, motor, combustivel):
    print(modelo, ano, placa, marca, motor, combustivel)


criar_carro(modelo="Palio", ano=1999, placa="ABC-1234", marca="Fiat",
motor="1.0", combustivel="Gasolina")  # válido

criar_carro("Palio", 1999, "ABC-1234", marca="Fiat", motor="1.0",
combustivel="Gasolina")  # inválido
```

Keyword and positional only

```
def criar_carro(modelo, ano, placa, /, *, marca, motor, combustivel):
    print(modelo, ano, placa, marca, motor, combustivel)


criar_carro("Palio", 1999, "ABC-1234", marca="Fiat", motor="1.0",
combustivel="Gasolina")  # válido

criar_carro(modelo="Palio", ano=1999, placa="ABC-1234", marca="Fiat",
motor="1.0", combustivel="Gasolina")  # inválido
```

Objetos de primeira classe

Em Python tudo é objeto, dessa forma **funções também são objetos** o que as tornam objetos de primeira classe. Com isso podemos **atribuir funções a variáveis, passá-las como parâmetro para funções, usá-las como valores em estruturas de dados** (listas, tuplas, dicionários, etc) e usar como valor de retorno para uma função (closures).

Exemplo

```python
def somar(a, b):
    return a + b

def exibir_resultado(a, b, funcao):
    resultado = funcao(a, b)
    print(f"O resultado da operação {a} + {b} = {resultado}")

exibir_resultado(10, 10, somar)  # O resultado da operação 10 + 10 = 20
```

Escopo local e escopo global

Python trabalha com escopo local e global, dentro do bloco da função o escopo é local. Portanto alterações ali feitas em objetos imutáveis serão perdidas quando o método terminar de ser executado. Para usar objetos globais utilizamos a palavra-chave **global**, que informa ao interpretador que a variável que está sendo manipulada no escopo local é global. Essa **NÃO é uma boa prática e deve ser evitada.**

Exemplo

```
salario = 2000

def salario_bonus(bonus):
    global salario
    salario += bonus
    return salario

salario_bonus(500)  # 2500
```

Percurso

~~Etapa 1~~ ~~Estudo aprofundado sobre funções~~

Links Úteis

- https://github.com/digitalinnovationone/trilha-python-dio

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)