Aula 8: Funções

Em programação, temos um princípio fundamental chamado **DRY**: *Don't Repeat Yourself* (Não se Repita).

Imagine que em 10 lugares diferentes do seu código, você precise imprimir um cabeçalho bonito:

```
print("-"*40)
print("  Bem-vindo ao nosso aplicativo! 😀")
print("-"*40)
```

Se você copiar e colar isso 10 vezes, e depois decidir mudar o caractere de `-` para `*`, você teria que corrigir em 10 lugares! Trabalhoso e manualmente nada prático.

Podemos guardar essa "rotina" em uma caixa e apenas "chamar a caixa" quando precisarmos! Essa caixa é o que chamamos de **Função**.

Conceito

Pense em uma função como uma **receita de bolo** ou um **manual de instruções** para uma tarefa específica.

1. **Você define a receita uma única vez:** Você escreve os passos (o código) e dá um nome a ela (ex: `fazer_bolo_de_chocolate`).
2. **Você usa a receita quantas vezes quiser:** Toda vez que você quiser um bolo de chocolate, você não reescreve a receita inteira. Você apenas "chama" a receita (ex: `fazer_bolo_de_chocolate()`).

Em Python, uma função é um bloco de código que só é executado quando é chamado. Você pode passar dados/variáveis para dentro dela (chamamos isso de **parâmetros** ou **argumentos**) e ela pode devolver informações trabalhadas para você (chamamos isso de **retorno**). Isso se tornar uma boa prática em programação por vários motivos:

- **Reutilização de Código:** Evita o "copia e cola". Escreva uma vez, use várias.
- **Organização:** Seu código fica mais limpo, mais fácil de ler e de entender.
- **Facilidade de Manutenção:** Se você precisar corrigir um bug ou melhorar uma tarefa, você mexe em um só lugar (dentro da função).

Criando e Chamando Funções

Vamos ver como isso funciona na prática. A sintaxe básica é:

```
def nome_da_funcao(parametro1, parametro2..., parametroN):
    # Bloco de código (indentado)
    # O que a função faz
    return valor_de_retorno # Opcional
```

Exemplo 1: Sem parâmetros, Sem retorno

```
# 1. DEFINIÇÃO da função
def dar_boas_vindas():
    print("-"*40)
    print("  Bem-vindo ao nosso aplicativo! 😀")
    print("-"*40)

# 2. CHAMADA da função
# O código abaixo só será executado se você "chamar" a função pelo nome:

print("Início do programa.")
dar_boas_vindas()  # <-- Isso executa o código dentro da função
print("Meio do programa.")
dar_boas_vindas()  # <-- Podemos chamar de novo!
print("Fim do programa.")
```

Exemplo 2: Com Parâmetros

E se quisermos dar boas-vindas a uma pessoa específica? Temos:

```
# 'nome_da_pessoa' é um PARÂMETRO.
# É uma variável que só existe dentro da função.
def boas_vindas_personalizado(nome_da_pessoa):
    print("-"*40)
    print(f"Olá, {nome_da_pessoa}! Seja bem-vindo(a)! 😀")
    print("-"*40)

# Ao chamar a função, passamos o ARGUMENTO (o valor)
boas_vindas_personalizado("Maria")
boas_vindas_personalizado("João")
```

Exemplo 3: Com Retorno

Funções são muito mais interessantes quando elas processam dados e nos devolvem um resultado. Usamos, para isso, a palavra-chave `return`:

```
# Esta função recebe dois números e DEVOLVE a soma deles
def somar(a, b):
    resultado = a + b
    return resultado

# Para usar o valor, precisamos guardá-lo em uma variável
soma1 = somar(5, 10)
soma2 = somar(100, 50)

print(f"O primeiro resultado é: {soma1}")
print(f"O segundo resultado é: {soma2}")
```

```
print(f"Você pode usar direto no print: {somar(3, 3)}")
```

Atividade Assistida

Faça um código que possa ler 3 pares de números inteiros, calcular e imprimir a soma de cada par separadamente. Obrigatoriamente, devemos usar a função somar que acabamos de criar.

```
# 1. Definimos nossa ferramenta: a função de somar
def somar(a, b):
    """
    Esta função recebe dois números (a e b) e retorna a soma deles.
    (Isso é uma 'docstring', uma boa prática para documentar o que a função faz)
    """
    resultado = a + b
    return resultado

# 2. Parte principal do nosso programa
print("Calculadora de Somas")

# 3. Vamos usar um loop 'for' para tratar dos 3 pares
for i in range(3):
    print(f"\n--- Calculando {i+1}º par ---")

    # Pedimos os números ao usuário
    num1 = int(input("Digite o primeiro número: "))
    num2 = int(input("Digite o segundo número: "))

    # Chamamos a função com os números que o usuário digitou
    # e guardamos o valor que ela 'retornou'
    resultado_da_soma = somar(num1, num2)

    # Imprimimos o resultado
    print(f"A soma de {num1} + {num2} é = {resultado_da_soma}")

print("\nPrograma finalizado!")
```

Atividades Práticas

1. Controle de Pesca

Crie um programa que ajude um pescador a controlar sua produtividade. Toda vez que ele traz um peso de peixes maior que o estabelecido pelo regulamento (100 quilos), ele deve pagar uma multa de R$ 4,00 por quilo excedente.

- O programa deve ler o peso de peixes (em quilos) pescado no dia.

- **Você deve criar uma função** (ex: `calcular_multa(peso_total)`) que recebe o peso e **retorna** o valor da multa (que pode ser `0.0` se estiver dentro do limite).
- Se o valor da multa retornado for maior que zero, mostre a multa.
- Caso contrário, mostre a mensagem "Peso dentro do limite. Nenhuma multa a pagar."
- Pergunte o peso de *várias* pescarias feitas ao longo da semana. O loop para quando o usuário digitar 0. Ao final, mostre o **total** de multa acumulado no dia.

2. Calculadora de IMC

Crie um programa que leia a altura e o peso de N pessoas (pergunte ao usuário quantas pessoas são). Para cada pessoa, mostre seu IMC e a classificação.

- Fórmula: `IMC = PESO / (ALTURA * ALTURA)`
- **Obrigatório (Função 1):** Crie uma função `calcular_imc(peso, altura)` que receberá os valores e retornará o IMC calculado.
- **Obrigatório (Função 2):** Crie *outra* função `obter_classificacao(imc)` que recebe o valor do IMC (calculado pela função 1) e **retorna** uma string com a classificação.
    - *Valores de Referência:*
        - Menor que 18.5: "Abaixo do peso"
        - 18.5 a 24.9: "Peso normal"
        - 25.0 a 29.9: "Sobrepeso"
        - 30.0 ou mais: "Obesidade"
- O programa principal deve pedir N, fazer um loop N vezes, pedir peso e altura, chamar as duas funções e imprimir o resultado formatado.

3. Conversor de Temperatura

Crie um programa que permita ao usuário converter temperaturas entre Celsius e Fahrenheit.

- **Função 1:** Crie uma função `celsius_para_fahrenheit(temp_c)` que recebe a temperatura em Celsius e retorna o valor em Fahrenheit.
    - Fórmula: `F = (C * 9/5) + 32`
- **Função 2:** Crie uma função `fahrenheit_para_celsius(temp_f)` que recebe a temperatura em Fahrenheit e retorna o valor em Celsius.
    - Fórmula: `C = (F - 32) * 5/9`
- O programa principal deve perguntar ao usuário qual conversão ele quer fazer (ex: "1 para C->F" ou "2 para F->C"), pedir o valor, chamar a função correta e mostrar o resultado.

  Desafio: Criar uma única função que faça qualquer uma das conversões, sempre perguntando ao usuário qual é desejada.

Funções e Módulos

Vimos como criar funções e de que maneira elas podem "conversar" entre si. Podemos utilizar bibliotecas de código (módulos) que já vêm com o Python. São chamadas **bibliotecas nativas**.

Abaixo temos o módulo `random`. Ele nos permite gerar números aleatórios, embaralhar listas e muito mais. Para usá-lo, precisamos **importá-lo** no início do nosso código:

```
import random
```

Com isso, acessamos várias funções prontas, como `random.randint(min, max)`, que nos dá um número inteiro aleatório entre um intervalo `min` e `max`.

Atividade Assistida

Vamos criar um programa que faz duas tarefas:

1. Tem uma função para gerar dados aleatórios.
2. Tem funções para realizar os 4 cálculos matemáticos básicos.
3. No final, vamos integrar tudo: gerar dados e passá-los para as funções de cálculo.

Etapa 1:

Esta função cria uma *lista* de números aleatórios.

```
import random # Sempre no topo do arquivo!

def gerar_dados(qtd, min_val, max_val):
    """
    Gera uma LISTA de números aleatórios.
    - qtd: quantos números queremos na lista
    - min_val: o valor mínimo (inclusivo)
    - max_val: o valor máximo (inclusivo)
    """

    # A estrutura a seguir se chama "List Comprehension".
    # É um jeito rápido de criar uma lista usando um loop.
    lista_de_dados = [random.randint(min_val, max_val) for _ in range(qtd)]

    return lista_de_dados

# Testando a função
dados_aleatorios = gerar_dados(5, 1, 100) # Gera 5 números entre 1 e 100
print(f"Dados gerados: {dados_aleatorios}")
```

Etapa 2:

Vamos criar nossas operações aritméticas. Elas serão simples: recebem dois números e retornam o resultado.

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

def subtrair(a, b):
    return a - b

def multiplicar(a, b):
    return a * b

def dividir(a, b):
    # Precisamos tratar a divisão por zero!
    # Este é um ótimo exemplo de lógica dentro de uma função.
    if b == 0:
        print("Erro: Divisão por zero não é permitida.")
        return 0 # Podemos retornar 0 ou None, ou uma string de erro
    else:
        return a / b
```

Etapa 3:

Agora, usamos a função `gerar_dados` para criar duas listas. Depois, vamos percorrer essas listas e aplicar nossas funções de cálculo em cada par de números.

```
import random

# --- Nossas Definições de Funções (Etapa 1 e 2) ---

def gerar_dados(qtd, min_val, max_val):
    """Gera uma lista de 'qtd' números aleatórios entre 'min_val' e 'max_val'."""
    return [random.randint(min_val, max_val) for _ in range(qtd)]

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

def subtrair(a, b):
    return a - b

def multiplicar(a, b):
    return a * b

def dividir(a, b):
    """Divide a por b, com tratamento para divisão por zero."""
    if b == 0:
        return "Erro (div/0)"
```

```
    else:
        # Arredondando para 2 casas decimais para ficar bonito
        return round(a / b, 2)

# --- Nossa Integração ---

QTD_DE_DADOS = 5 # Quantos pares de números queremos testar

print("Gerando dados...")
# Geramos duas listas de dados independentes
lista1 = gerar_dados(QTD_DE_DADOS, 1, 20)
lista2 = gerar_dados(QTD_DE_DADOS, 0, 10) # Permitindo 0 na lista 2 para testar a divisão

print(f"Lista 1: {lista1}")
print(f"Lista 2: {lista2}")
print("-"*40)
print("Iniciando Cálculos (elemento a elemento):")

# Vamos usar um loop 'for' para "caminhar" pelas listas
# A função 'zip' é usada para parear elementos de duas listas
for num1, num2 in zip(lista1, lista2):

    print(f"\nPar: ({num1}, {num2})")

    # Agora, chamamos nossas funções de cálculo com esses números
    print(f"Soma:       {num1} + {num2} = {somar(num1, num2)}")
    print(f"Subtração:  {num1} - {num2} = {subtrair(num1, num2)}")
    print(f"Multipl.:   {num1} * {num2} = {multiplicar(num1, num2)}")
    print(f"Divisão:    {num1} / {num2} = {dividir(num1, num2)}")
```

Atividades Práticas

1. Verificador de Ano Bissexto

Crie uma função chamada eh_bissexto(ano):

- A função deve receber um ano (inteiro) como parâmetro.
- Ela deve **retornar** `True` (Booleano) se o ano for bissexto, e `False` caso contrário.
- *Regras do ano bissexto:* É divisível por 4, *exceto* para anos divisíveis por 100, *a menos que* sejam também divisíveis por 400. (Ex: 2000 e 2400 são bissextos; 1900 e 2100 não são).
- No programa principal, peça um ano ao usuário e imprima "O ano X É bissexto" ou "O ano X NÃO é bissexto", baseado no retorno da função.

2. Contagem de Caracteres

Crie uma função chamada contar_caractere(texto, caractere_procurado):

- A função deve receber uma string `texto` e uma string `caractere_procurado` (de um só caractere).
- Ela deve **retornar** o número de vezes que o `caractere_procurado` aparece no `texto`. (Não diferencie maiúsculas de minúsculas!)
- *Dica:* Use um loop `for` para percorrer o `texto` e use `.lower()` para tratar os caracteres.
- No programa principal, peça ao usuário uma frase e uma letra, e mostre o resultado da contagem.

3. Simulador de Dado

Usando o módulo random, crie uma função rolar_dado(lados).

- A função deve receber o número de `lados` do dado (ex: 6, 10, 20).
- Ela deve **retornar** um número aleatório entre 1 e o número de `lados` (use `random.randint(1, lados)`).
- No programa principal, crie um "simulador de batalha":
    - Peça ao usuário para "Rolar para o Ataque (d20)". Chame a função `rolar_dado(20)`.
    - Peça ao usuário para "Rolar para o Dano (d8)". Chame a função `rolar_dado(8)`.
    - Imprima os resultados de cada rolagem.