**Funções**

Funções são uma espécie de "subprograma". Elas são como pequenos pedacinhos de programa que podem ser chamadas pelo nome. Para criar uma função usamos o comando "def" e o nome da função. Elas são um bloco de comando assim como if/elif/else, while e for.

def hello():

print("Olá, mundo!")

Quando uma função é chamada pelo nome, o fluxo do programa é interrompido. Em seguida, a função inteira é executada, e ao final dela, o programa volta a ser executado do mesmo ponto que parou.

hello()

# Saída: Olá, mundo!

É possível passarmos informações para uma função. Chamamos essas informações de "parâmetros" ou "argumentos. Na definição da função, daremos nome a esses parâmetros. Dentro da função, eles serão tratados como se fossem variáveis.

def ola(nome):

print("Olá", nome)

Ao chamarmos a nossa funcao, passamos os parâmetros entre parênteses. O nome ou valor passado *não precisam* ser iguais aos declarados na função! O nome do parâmetro interno da função é um "apelido" que a função dará para o valor ou variável que passamos.

ola("Maria")

# Saída: Olá, Maria

aluno = "João"

ola(aluno)

# Saída: Olá, João

A função pode receber vários parâmetros separados por vírgula.

def dadosPessoais(nome, idade, cidade):

print("Seu nome é {}, você tem {} anos e mora em {}.".format(nome, idade, cidade))

dadosPessoais("José", 30, "Maceió")

# Saída: Seu nome é José, você tem 30 anos e mora em Maceió.

Os parâmetros podem ser passados fora de ordem. Porém, para isso precisamos explicitar qual parâmetro estamos passando, para evitar ambiguidade ou erros de interpretação do Python.

dadosPessoais(idade=56, cidade="Florianópolis", nome="Ana")

# Saída: Seu nome é Ana, você tem 56 anos e mora em Florianópolis.

**Funções com resposta**

Uma função pode ter uma "resposta". Por exemplo, nossa função pode ser uma conta, e o resultado da conta pode ser útil em nosso programa. Dizemos que essas funções "retornam" um valor. Utilizamos a palavra "return" para fazer uma função retornar sua resposta. Quando uma função retorna um valor, ela pode ser usada em expressões matemáticas ou lógicas, ter seu valor atribuído a uma variável, etc.

def somatorio(lista):

soma = 0

for item in lista:

soma = soma + item

return soma

numeros = [1, 2, 3, 4, 5]

soma\_dos\_numeros = somatorio(numeros)

print("A soma dos elementos da lista vale: ", soma\_dos\_numeros)

if somatorio(numeros) > 50:

print("Que soma grande!")

else:

print("Que soma pequena!")

'''

Saída:

A soma dos elementos da lista vale: 15

Que soma pequena!

'''

**Funções recursivas**

Algumas funções são chamadas funções recursivas. A recursividade (ou recursão) é uma propriedade na qual uma função faz referência a si mesma. Quando a função encontra uma nova referência, ela irá pausar sua execução atual e iniciar a execução da nova instância, para só então retomar sua execução.

Assim como nos loops, é necessário ter alguma condição para que as chamadas recursivas sejam interrompidas, evitando que executem para sempre.

def funcaoRecursiva(numero):

if (numero != 1):

funcaoRecursiva(numero - 1)

print(numero)

print("Testando a função recursiva:")

funcaoRecursiva(10)

Note que no exemplo acima passamos 10 para a função. Sua execução foi interrompida por uma nova chamada passando 9, depois 8, depois 7... Ao chegar em 1, ele foge da condicional e imprime 1, encerrando a execução. Então a instância que recebeu 2 tambem conclui sua execução, depois a chamada 3, a 4... A 10, que foi a 1a chamada, encerra por último.

Dizemos que é um comportamento de pilha - exatamente como uma pilha de pratos sobre a mesa: O primeiro prato que foi colocado sobre a mesa será o último a sair, pois todos os pratos colocados sobre ele precisam ser retirados antes de você poder retirar o último.

Problemas recursivos normalmente são problemas do tipo "dividir para conquistar": Temos um "caso base", ou seja, um ou mais casos onde há uma resposta direta; E temos um "caso geral", que é uma versão reduzida do problema original.