Funções

Como a própria palavra diz, uma função é a capacidade de realizar uma determinada tarefa, utilizando algumas ferramentas a disposição. Em python, as funções seguem, de maneira geral, a definição anterior. Uma função nada mais é do que um trecho de código, composto por uma ou mais instruções, que é executada apenas quando solicitada.

Uma função em python possui duas entidades, a declaração, ou seja, a implementação, as operações que a função deve executar, e a execução que é de fato a execução das capacidades da função. Parece coisa de outro mundo, mas vale lembrar que já vimos algumas funções dentro deste curso, os comandos print() e input() são ambos funções, no entanto acessamos apenas a entidade de execução.

Vamos começar a entender as funções, para isso, vamos declarar (criar) uma nova função:

```
def escrever_nome():
        print("José")
```

Essa é provavelmente um dos exemplos mais simples de declaração de função, `escrever_nome()` é a identificação, ou nome, da função, e quando invocamos essa função ela apresenta na linha de comando o nome "José".

```
escrever_nome()
> "José"
```

Bom, seria muito simples se as funções fossem somente isso, mas a verdade é que as funções possuem muito mais características que as tornam peça fundamental na construção de programas mais complexos. A função de exemplo acima, é muito simples, mas ao mesmo tempo não é muito excitante porque ela sempre vai escrever o nome José, seria muito mais legal se fosse possível dizer a função, qual nome queremos que ela escreva.

A boa notícia é que isso é perfeitamente possível, as funções podem receber algumas "ferramentas" para trabalhar em cima, produzindo resultados diferentes de acordo com o que está disponível, o nome mais correto para essas ferramentas é `argumentos` e cada função pode receber um ou mais argumentos, e os utilizar para gerar resultados impressionantes.

Veja como podemos utilizar um argumento chamado nome para tornar a nossa função escrever_nome(), mais dinâmica.

```
# Definição
def escrever_nome(nome):
        print(nome)

# Execução
escrever_nome("Ana")
> "Ana"
```

Simples não? A verdade é que os argumentos são como se fossem variáveis que outro pedaço do nosso código vai emprestar para a nossa função, e ela por sua vez pode utilizar essa variável para trabalhar e gerar resultados diferentes.

Cada função pode receber teóricamente um número infinito de argumentos, mas é considerada boa prática manter o número total de argumentos o menor possível.

Um ponto importante: Ao invocar (executar) uma função, é imprescindível fornecer a ela a quantidade exata de argumentos necessários, caso contrário, teremos um erro.

Existem situações onde não é possível saber com precisão a quantidade exata de argumentos que uma função vai precisar, para isso, podemos utilizar a técnica de **argumentos arbitrários ou *args**, que na verdade se parecem muito com uma lista. Para utilizar os *args basta adicionar um asterisco antes do nome do argumento:

```
# Declaração
def escrever_nome_completo(*nomes):
        for nome in nomes:
                print(nome)

# Invocação (Execução)
escrever_nome_completo("José", "Alves", "dos", "Santos")
```

No geral, os argumentos estão ordenados pela sequência na qual são passados para a função, mas podemos utilizar os nomes que informamos na declaração para dizer qual o valor de cada argumento ignorando a ordem de chamada.

No exemplo abaixo, temos uma função que faz o print do nome e sobrenome, normalmente a função esperaria que o primeiro argumento fosse o nome e que o segundo seja o sobrenome, mas podemos inverter as coisas se explicitamente informarmos o valor do nome e do sobrenome.

```
def escrever_nome_sobrenome(nome, sobrenome):
        print(nome + " " + sobrenome)

escrever_nome_sobrenome(sobrenome = "Alves", nome = "José")
```

Finalizando o assunto argumentos, podemos definir um valor padrão para os argumentos de uma função, se por algum motivo você precisar de uma função na qual não é necessário informar um dos argumentos todas as vezes, isso pode ser bastante útil. No exemplo abaixo, a função acelerar aumenta a aceleração de um veículo em 1 unidade, mas podemos acelerar mais rápido se informarmos o valor da aceleração.

```
def acelerar(valor = 1):
        print("Acelerando em: " + str(valor))

acelerar() # Acelerando em: 1
acelerar(10) # Acelerando em: 10
```

Ufa, quanta coisa! Funções são bastante complexas, mas ao mesmo tempo muito, mas muito úteis. No entanto, ainda falta uma coisa, e talvez seja a mais importante sobre funções, elas podem receber argumentos, trabalhar esses argumentos, e ainda por cima, **retornar valores**, essa funcionalidade é muito útil, podemos escrever uma função que realiza um cálculo complexo, e reutilizar essa função em nosso código, ou até em programas diferentes como veremos mais para frente.

A função abaixo realiza o cálculo da área de um triangulo a partir do tamanho de cada um dos três lados em um plano bidimensional, utilizando a formula de Heron:

```
def calc_area_triangulo(a, b, c):
        semi_perimetro = (a + b + c) / 2
        area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
        return area
```

No exemplo acima, a palavra return indica que a função vai retornar um valor, que neste caso é a área do triangulo, podemos atribuir o valor de retorno desta função a uma variável por exemplo:

```
lado_a = input("Digite o valor do primeiro lado: ")
lado_b = input("Digite o valor do segundo lado: ")
lado_c = input("Digite o valor do terceiro lado: ")
area_do_triangulo = calc_area_triangulo(lado_a, lado_b, lado_c)
print(area_do_triangulo) # 14.70
```

Bom, com isso abordamos a maioria dos conceitos em torno das funções, ainda existe muito a aprender, mas com este conhecimento básico já podemos criar algumas funções mais interessantes.