Aula 05 – Funções

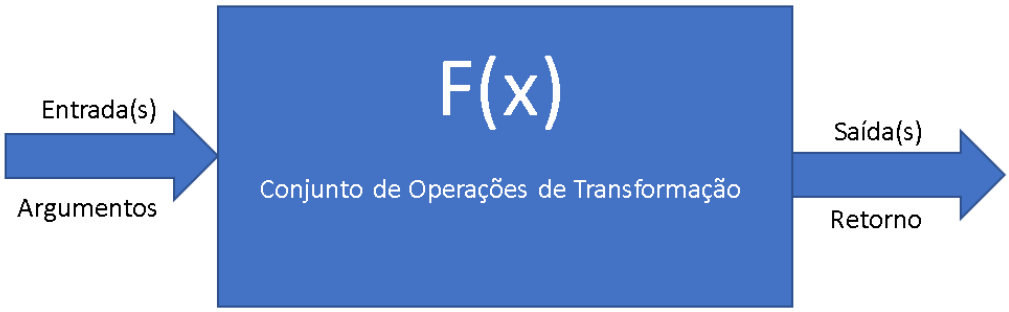
Funções: Noções Gerais

Até este momento já utilizamos várias funções diferentes. Alguns exemplos:

- Input() → Função para receber valores de entrada digitados pelo usuário.
- Print() → Função para imprimir valores de saída na tela.
- Type() → Função de determinação do tipo de uma variável.
- Len() → Função que dá o comprimento de uma lista.
- Int(), Float(), Str() → Funções de conversão.

Mas, afinal de contas, o que é uma função?

Uma **função** é uma sequência nomeada de instruções, um bloco de linhas de comando organizadas e geralmente executadas sequencialmente. Esse bloco possui um propósito específico dentro do nosso código. Podemos pensar em uma função por meio de seus elementos constituintes.



Para que uma função seja considerada útil deverá apresentar pelo menos um desses elementos constituintes. Contudo, não precisará necessariamente apresentar todos os três ao mesmo tempo (entrada, saída, "corpo"). Há funções somente com entradas e saídas, sem grandes transformações em seu "corpo". Há funções sem quaisquer entradas e saídas que somente executam operações e não nos oferecem valores de retorno significativos, entre outras coisas. Precisamos estar preparados para lidar com todos os diversos tipos de função.

Por que usar funções?

Caso ainda não esteja claro porque é importante dividir o nosso código em vários blocos, aqui estão alguns motivos:

- Criar uma função te dá a oportunidade de *nomear um conjunto de instruções*. Isso pode tornar o seu código mais legível e intuitivo.

- Funções *eliminam a necessidade de códigos repetitivos*. Para reaproveitar o código várias vezes, basta chamar a função cada vez que precisar executar aquelas tarefas. Precisou alterar alguma coisa específica? Altere uma vez só no corpo da função.

- A função permite *testar* pequenas porções do seu código individualmente, o que é muito útil principalmente no processo de *depuração ou debugging*. Teste cada pequena parte, depois junte tudo em uma tarefa mais complexa.

Definições e Chamadas de Função

A **declaração** ou **definição** de uma função, é o processo pelo qual especificamos o nome, as variáveis de entrada, as variáveis de saída e o conjunto de operações específicas de uma função. Vamos passar passo-a-passo por cada um desses elementos. Importante lembrar que, como já dito, nem sempre a função apresentará todos esses elementos. O caso analisado será o de uma função "completa".

a) A primeira coisa que definimos em uma função é o seu **nome**. Fazemos isso após usar a palavra reservada **def**. Uma vez atribuído o nome, colocamos os parênteses e os dois pontos na sequência. Todo o corpo da função será indentado em um nível hierárquico após os dois pontos. Veja:

```
def cantar_musica():
  #corpo da função aqui
```

b) Depois do nome, pensamos nas **variáveis de entrada** da função. **Argumento** é o nome genérico alternativo que damos às variáveis de entrada. Os argumentos são sempre declarados dentro dos parênteses iniciais. Veja:

```
def cantar_musica(verso_01, verso_02, ponte, refrao):
  #corpo da função aqui
```

Obs. Não confundir argumento com parâmetro. Argumento é o nome genérico de uma variável que trabalharemos ao longo da função, parâmetro é o valor específico que será atribuído ao argumento da função durante uma chamada. Por exemplo, na função "cantar_musica" a variável "verso_01" é um argumento, enquanto "Someone told me longa go, there is a calm before the storm..." é um valor específico que podemos usar como **parâmetro** da função em uma de suas chamadas.

c) Depois das variáveis de entrada, pensamos no conjunto de operações específicas que faremos com elas ou no **corpo da função**. Veja:

```
def cantar_musica(verso_01, verso_02, ponte, refrao):
    parte_01 = verso_01 + ponte + refrao
    parte_02 = verso_02 + ponte + refrao
    parte_03 = refrao + refrao
    musica_completa = parte_01 + parte_02 + parte_03
```

d) Finalmente, pensamos na **saída da função** ou no seu **valor de retorno**. Tal valor é colocado após a palavra reservada "return". Veja:

```
def cantar_musica(verso_01, verso_02, ponte, refrao):
    parte_01 = verso_01 + ponte + refrao
    parte_02 = verso_02 + ponte + refrao
    parte_03 = refrao + refrao
    musica_completa = parte_01 + parte_02 + parte_03
    return musica_completa
```

Importante: após declarar uma função, sempre verificar se as entradas e saídas atendem os critérios específicos do seu projeto. Por exemplo, se precisaria retornar uma variável do tipo float, mas por algum motivo estou retornando um int, poderei ter problemas no futuro.

Uma vez que nossa função esteja definida, criamos um objeto de função que podemos usar repetidas vezes. Ao dar um print no nome da função, veremos algo do tipo:

```
<function cantar_musica at 0x7f8a56c9b1f0>
```

Cada vez que a usamos um objeto de função com valores de entradas específicos, estamos realizando uma **chamada** de função. Lembre-se que as instruções da função não executadas até ela ser chamada pela primeira vez.

Vamos criar uma letra divertida fazendo a chamada da função cantar_musica().

```
p1 = '''Você estava tão longe, mas agora está tão perto.
Acho que não consigo mais viver sem você.
Volta pra mim\n\n'''

p2 = '''Mais uma letra sertaneja
Enfadonho sertanejo
Da mais pura sofrência
Que desgraça para a humanidade\n\n'''

m_ponte = """Não desista dos seus sonhos
Sou o coach dos agroboys
Você ainda não percebeu?
Vou lavar a sua mente\n\n"""

m_refrao  = """Oleleihuuuuuuu Oleleihuuuuuuu
Tchaka tchaka, tikbum pá e mais outras onomatopeias\n\n"""

letra = cantar_musica(p1,p2,m_ponte,m_refrao)
print(letra)
```

Veja que como temos uma **função com valor de retorno** ou **funções com resultado**, precisamos armazenar o valor de retorno em uma variável (atribuição) para não perdermos tudo o que fizemos. Para uma **função sem valor de retorno** ou **função nula**, essa atribuição não é necessária, pois uma função desse tipo sempre retorna o valor especial do tipo nulo (None). A função print() é um exemplo de função sem valor de retorno, enquanto input () tem valor de retorno bem estabelecido.

```
retorno_se_houver = print()
print(retorno_se_houver)
retorno_se_houver = input()
print(retorno_se_houver)
```

Fluxo de Execução e Dependências

Devemos pensar sempre no **fluxo de execução** do nosso programa ao declarar nossas funções. Se decidirmos usar algo na nossa função sem ter declarado esse algo previamente, teremos um erro. A ordem de organização das funções, portanto, importa!

Chamamos as coisas as quais a nossa função depende para funcionar de **dependências**, isso inclui outras variáveis, funções e valores. Vejamos um exemplo prático, vamos criar uma nova função chamada “repete_letra” que fica repetindo a letra de música gerada pela função “cantar_musica” várias vezes. Veja:

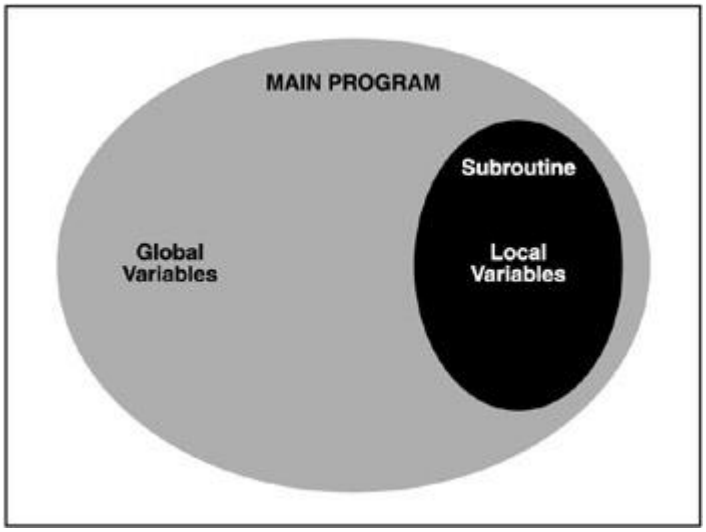
```
def repete_letra(n, v1,v2,p,r):
    for i in range(n):
        cantar_musica(v1,v2,p,r)
```

O que aconte se repete_letra() vier antes de cantar_musica()? E se não atribuirmos os valores v1,v2, p e r antes de fazermos a primeira chamada da função?

Escopo Local e Global

Quando criamos uma variável dentro de uma função, ela tem escopo **local**. Ou seja, só existe dentro da função. Se tentarmos usar essa variável fora, teremos um erro. Faça o teste.

Para ampliar o escopo de uma variável pra **global**, devemos declará-la fora da função e usar a palavra reservada "global" com o nome da variável antes de utilizá-la dentro da função. Teste você mesmo.



Variáveis globais não são muito recomendáveis em programas complexos, pois por não ficarem reservadas a um contexto específico, são mais difíceis de serem monitoradas. É melhor pensarmos em **sequenciamento de funções** em caso de programas complexos.

Módulos

Módulos são coleções de funções, variáveis e outras coisas mais.

Todo o módulo necessita de uma **instrução de importação** antes de ser utilizado (já visto)

Usamos o **operador de ponto** para navegar dentro de um módulo e acessar, por exemplo, objetos do tipo de função. Quanto mais pontos, mais "fundo" estamos na árvore de classificação do módulo. Vamos fazer isso para alguma função da biblioteca "Math". Abra a documentação e tente acessar alguma função da biblioteca.

Ao recuperar uma função da blbioteca, neste caso, a função também assume o papel de um **objeto de módulo**, ou seja, um objeto que foi importado do módulo.

Argumentos Opcionais

Podemos tornar um **argumento opcional** dentro de uma função, em Python, atribuindo um valor a ele quando o declaramos. Por exemplo:

```
#EXEMPLO OPTIONAL ARGS

def calcula_media(nota_1, nota_2, nota_3 = 5):
  media = (nota_1 + nota_2 + nota_3)/5
  return media

print(calcula_media(7,8))
```

Torna o argumento nota_3 opcional. É necessário sempre inserir primeiro os argumentos obrigatórios e deixar os argumentos opcionais no final.

Exercícios

1. **Teórico.** Um objeto de função é um valor que pode ser atribuído a uma variável ou passado como argumento. Por exemplo, do_twice() é uma função que toma o objeto de função como argumento e o chama duas vezes:

```
def do_twice(f):
  f()
  f()
```

Aqui está um exemplo que usa a do_twice() para chamar outra função (a print_spam()):

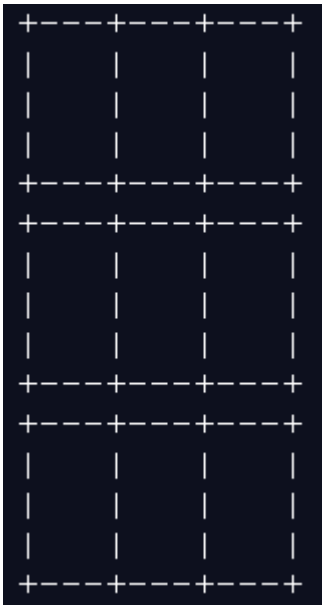
```
def print_spam():
  print('spam')

do_twice(print_spam)
```

   a) Digite o exemplo anterior e teste-o.
   b) Altere do_twice para que receba dois argumentos, um objeto de função e um valor. Em seguida, altere a função print_spam passando o valor como argumento.
   c) Crie a função do_four() que chame a função do_twice() duas vezes. Faça pequenas alterações para entender o encadeamento lógico e o encapsulamento das funções.

2. **Embaralha palavra (anagramas).** Construa uma função que receba uma string como parâmetro e devolva outra string com os carateres embaralhados. Por exemplo: se função receber a palavra python, pode retornar npthyo, ophtyn ou qualquer outra combinação possível, de forma aleatória. Padronize em sua função que todos os

caracteres serão devolvidos em caixa alta ou caixa baixa, independentemente de como foram digitados.

3. **Desenha moldura**. Construa uma função que desenhe um mapa nxn (1x1, 2x2, 3x3, etc.) usando os caracteres '+' , '–' e '| '. Esta função deve receber dois parâmetros, linhas e colunas, sendo que o valor por omissão é o valor mínimo igual a 1. Deverá também receber a dimensão n da matriz. O resultado final deverá ser uma imagem similar à imagem abaixo (para o caso de um grid 3x3:



Dica 1 – Pense em criar uma função para desenhar somente uma linha. Depois, uma função para desenhar somente as colunas e fechar um grid simples.

Dica 2 – Para criar mais blocos na vertical, basta repetir a função de desenhar um quadrado (ou retângulo) várias vezes.

Dica 3 – Para criar mais blocos na horizontal você precisa alterar a variável que contém a forma da linha. Como fazer isso?

Desafio

Pesquise sobre ***as lambda functions***, um tipo função muito utilizado em Python. O que elas são? Como funcionam? Quais são suas vantagens e desvantagens? Você consegue implementar um exemplo didático do seu uso?