**Texto base**

**6**

Funções

Gilberto Alves Pereira

***Resumo***

*Você tem ideia de como três programadores conseguem trabalhar em um mesmo programa? Existe uma maneira simples de reaproveitar um código que resolve um problema em um outro programa?*

*Uma função nada mais é do que um trecho de código que damos um nome. Os parâmetros e o retorno potencializam muito o uso das funções. Esses são os assuntos discutidos nesse texto.*

# 1.1. Introdução

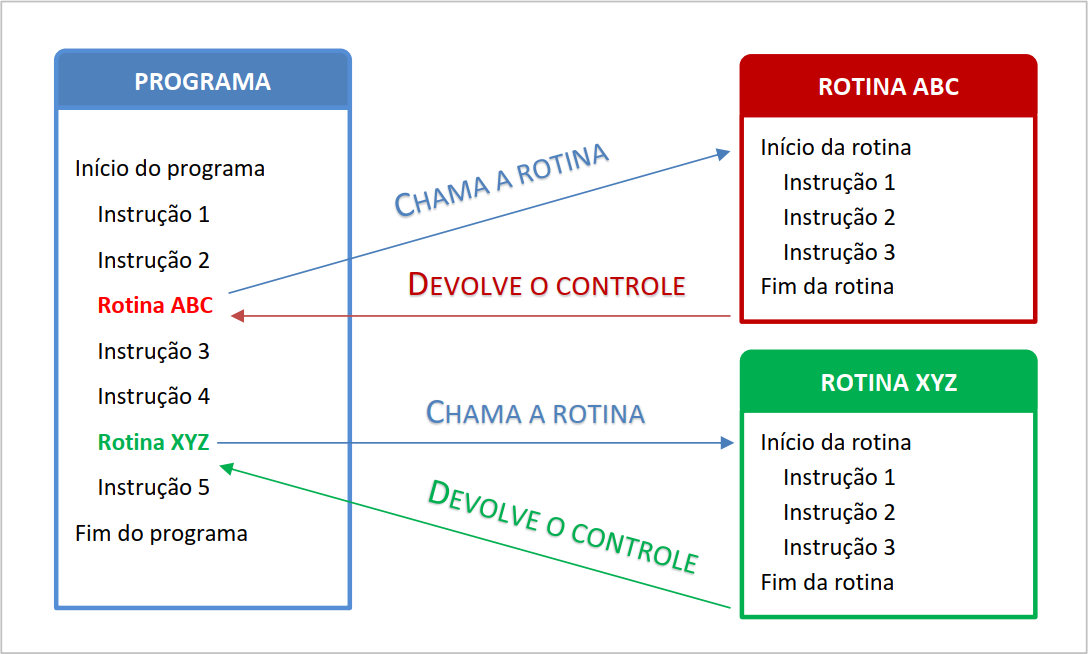
Até agora vimos conceitos fundamentais, tais como entrada e saída, variáveis, operadores, expressões e estruturas de controle de fluxo de execução (DIERBACH, 2012). Contudo, do ponto de vista prático, essas estruturas sozinhas, ainda não são suficientes. Vamos pensar em problemas mais complexos. Por exemplo, um smartphone contém cerca de 10 milhões de linhas de código, imagine o esforço para desenvolver e depurar softwares dessa dimensão (DIERBACH, 2012).

Para gerenciar a complexidade de grandes problemas, é útil quebrá-los em subproblemas menores. Então, cada subproblema pode ser analisado e resolvido separadamente, permitindo soluções mais simples, legíveis e reutilizáveis. Algo importante para não perder tempo reconstruindo códigos (DIERBACH, 2012).

Nesse sentido, funções são blocos de construção fundamentais no desenvolvimento de software e muito usadas no mercado de trabalho.

# 1.2. Rotinas e Funções

Uma **rotina** é definida como um grupo de instruções que executa alguma tarefa bem definida e identificada por um nome (DIERBACH, 2012). Algoritmos maiores, por exemplo programas completos, podem ser compostos por diversas rotinas que são independentes do programa principal (quanto maior a independência melhor).

**Figura 6.1. Funcionamento de programa que utiliza funções. Fonte Autor.**

Uma rotina pode ser **invocada ou chamada** quantas vezes for necessário em um dado programa. No momento em que é chamada (a execução do programa chega na linha do código onde está o nome da rotina), o controle de execução é redirecionado para o código da rotina (DIERBACH, 2012).

Enquanto a rotina está em execução, o código posterior à linha com o nome da rotina fica em **estado de espera**, pois não é possível seguir em frente enquanto a rotina não terminar sua tarefa.

Quando uma rotina termina de ser executada, o controle da **execução retorna** automaticamente para o ponto onde foi chamada e o processamento continua com as instruções sucessoras (DIERBACH, 2012).

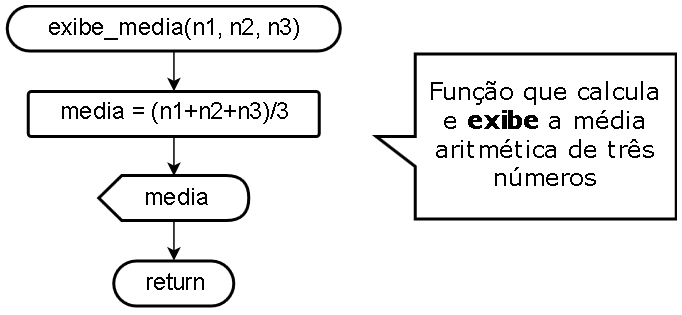
As rotinas podem ser **pré-definidas** da própria linguagem de programação ou **construídas** pelo programador quando necessário. Existem diversas rotinas prontas em Python, por exemplo, para cálculo de raiz quadrada, para gerar números aleatórios e para exibir mensagens para o usuário.

Usaremos o conceito de **função** como um tipo de rotina que pode ser criada da seguinte forma:

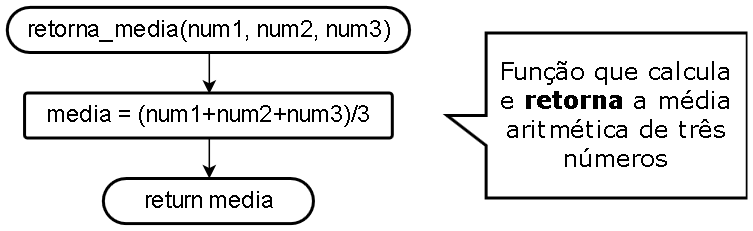
| **Programa principal** | **Definição da função** |
| --- | --- |
|  |  |

**Figura 6.2. Fluxograma e Fluxograma de uma função. Fonte Autor.**

**Exemplo 1:**

**Figura 6.3. Fluxograma de um função que exibe a média. Fonte Autor.**

**Exemplo 2:**



**Figura 6.4. Fluxograma de um função que retorna a média. Fonte Autor.**

A primeira linha de definição de uma função é chamada **cabeçalho** ou **assinatura**. O cabeçalho da função inicia com um identificador, que é o nome da função (DIERBACH, 2012). No exemplo 1 (Figura 6.3) o identificador da função é **exibe\_media**, já no exemplo 2 (Figura 6.4) o identificador é **retorna\_media**.

O nome da função é seguido por um par de parênteses e, entre eles, alguns identificadores chamados de **parâmetros formais** ou simplesmente parâmetros. No exemplo 1 os parâmetros são n1, n2 e n3, já no exemplo 2 os parâmetros são num1, num2, num3.

Os parâmetros são variáveis locais da função e inicializadas no momento em que a função é chamada. Os parâmetros devem ter um nome que, preferencialmente, faça referência ao dado que será armazenado, assim como variáveis comuns. As funções podem ser construídas com **qualquer número** de parâmetros, inclusive sem nenhum parâmetro.

Os parâmetros servem, essencialmente, para **comunicar algum dado externo** à função. Por exemplo, para calcular a média de três números a função precisa saber quais são os três números, por isso os valores são **passados como parâmetros** para ela.

Após o cabeçalho da função, escreve-se o **corpo da função**, onde são inseridas as variáveis locais e as demais instruções que a compõem.

Funções são geralmente definidas no início do programa. Contudo, a regra geral é: toda função deve ser definida **antes** de ser chamada.

# 1.3. Solicitando a Execução de uma Rotina

A concepção de uma rotina envolve dois momentos bem diferentes:

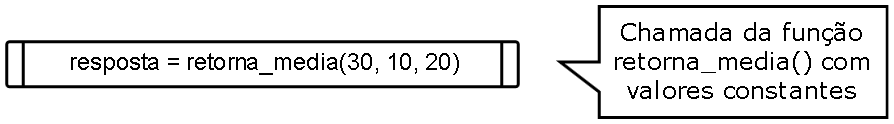
* a criação/declaração e
* a execução da rotina.

Uma rotina é executada quando é chamada, seja no programa principal ou em outra rotina (muitas vezes o programa principal é chamado de rotina-principal).

A **chamada** da rotina é a forma de solicitar sua execução em um determinado passo do algoritmo e, como dito anteriormente, no momento em que é chamada, o código da rotina entra em execução.

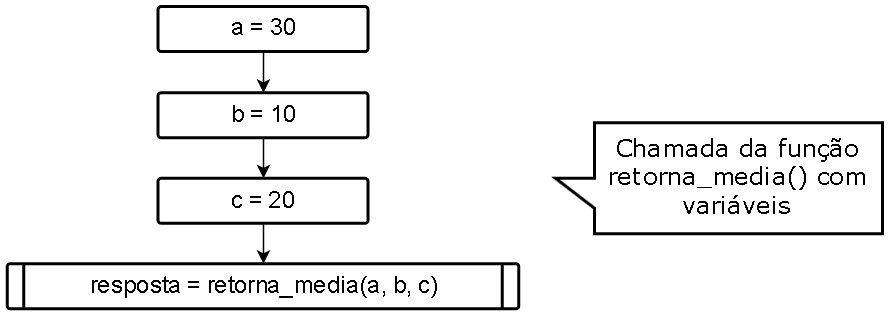
Para fazer a chamada de uma rotina devemos especificar seu nome e passar os **argumentos** necessários (podendo ser valores constantes ou variáveis). Note que os argumentos passados às rotinas são associados aos seus parâmetros.

Considerando a função retorna\_media(num1, num2, num3) definida anteriormente, a sua chamada pode ser feita assim:



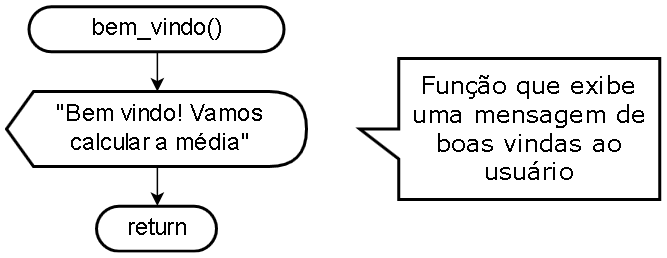
**Figura 6.5. Chamada da função retorna média. Fonte Autor.**

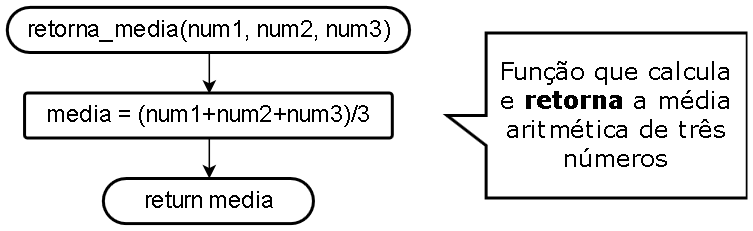
É possível chamar a função retorna\_media(num1, num2, num3) e obter o mesmo resultado da seguinte maneira:



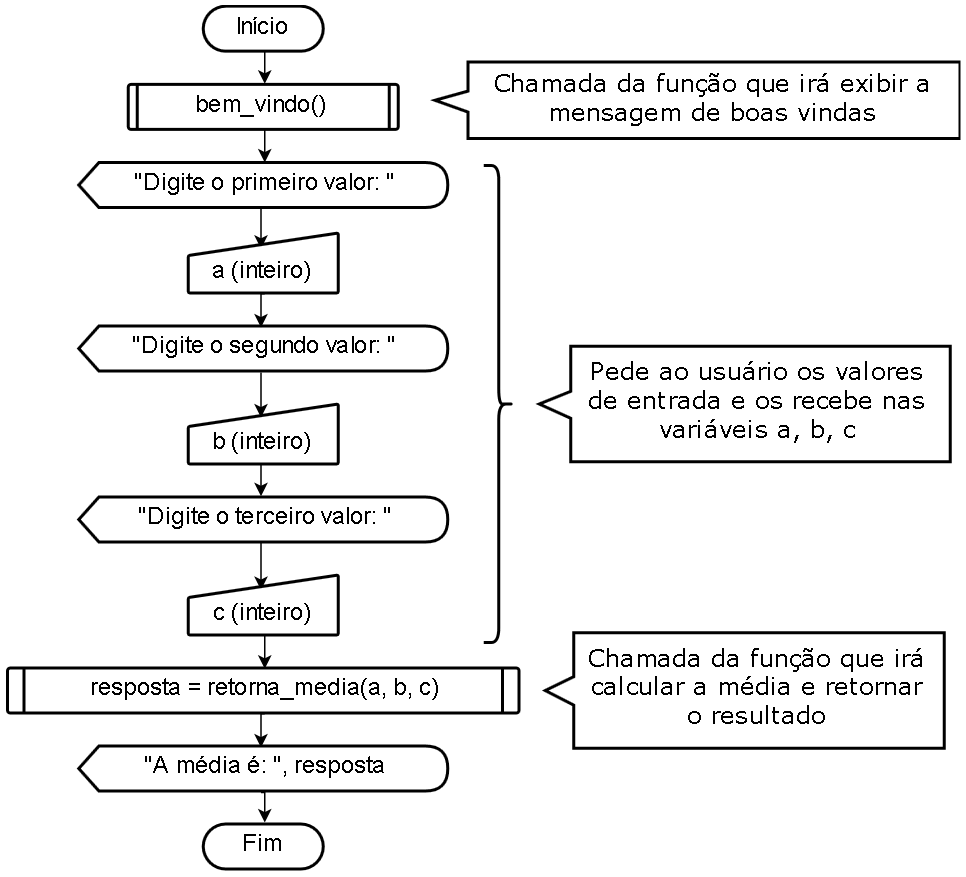
**Figura 6.6. Chamada da função retorna média usando variáveis nos parâmetros. Fonte Autor.**

O fluxograma completo que calcula a média aritmética entre 3 números inteiros usando a função retorna\_media(num1, num2, num3), deve ser dividido em duas partes, uma para a definição das funções e outra para o programa principal:

****

****

**Figura 6.7. Definição das Funções. Fonte Autor.**

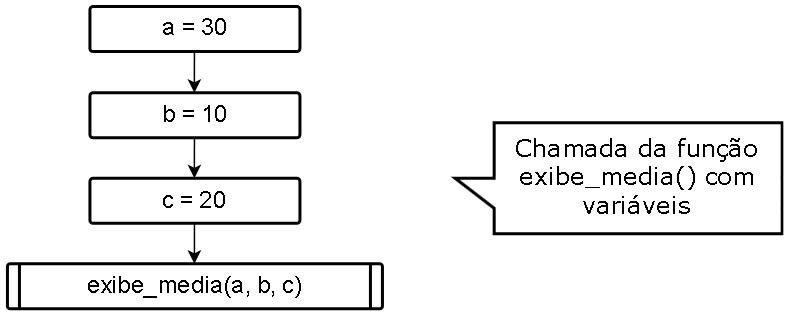
****

**Figura 6.8. Programa Principal. Fonte Autor.**

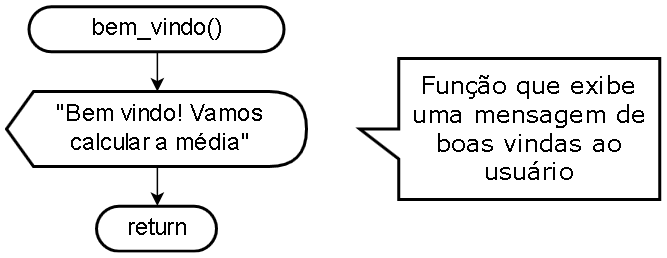
Considerando a função exibe\_media(n1, n2, n3) definida anteriormente, a sua chamada pode ser feita assim:

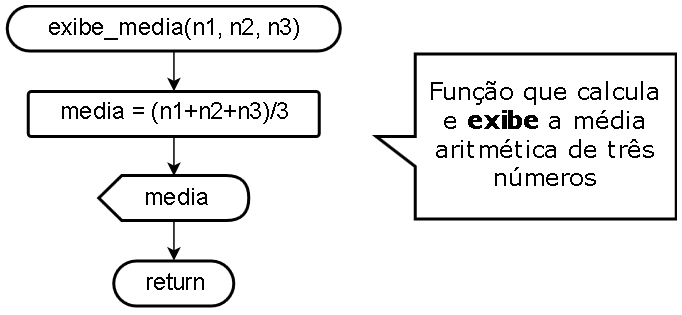
**Figura 6.9. Chamada da função exibe\_media usando três valores constantes. Fonte Autor.**

Também é possível chamar a função exibe\_media(n1, n2, n3), e obter o mesmo resultado, assim:

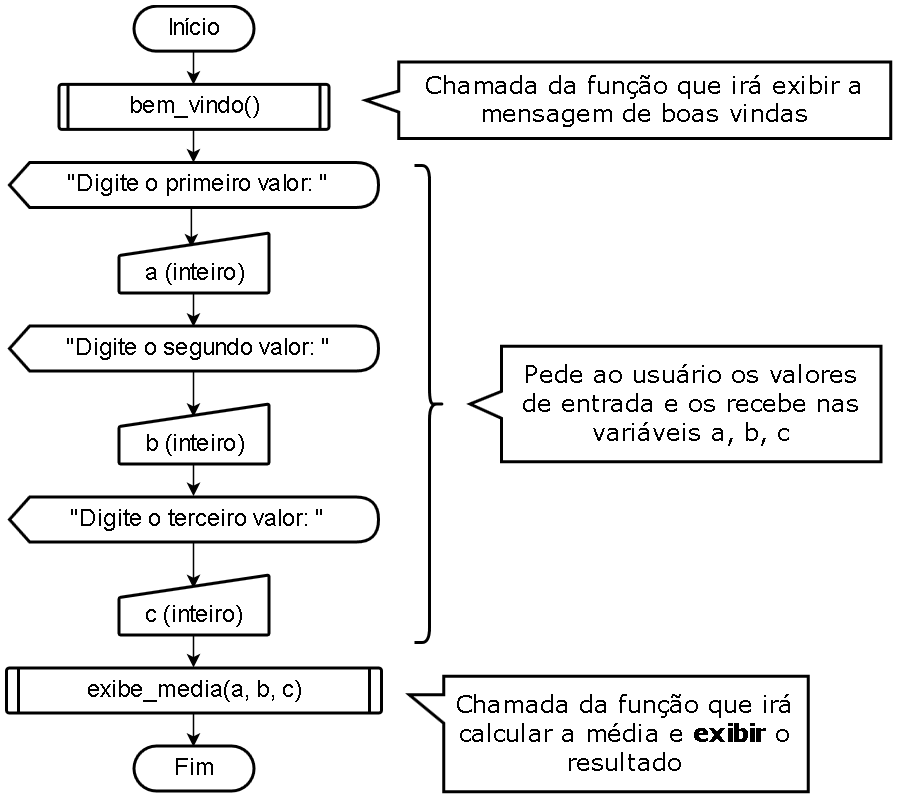
**Figura 6.10. Chamada da função exibe\_media usando três variáveis. Fonte Autor.**

O fluxograma completo que calcula a média aritmética entre 3 números inteiros usando a função exibe\_media(n1, n2, n3), deve igualmente ser dividido em duas partes, uma para a definição das funções e outra para o programa principal:

****

****

**Figura 6.11. Definição das funções bem\_vindo e exibe\_media. Fonte Autor.**

****

**Figura 6.12. Definição do programa principal exibição da média. Fonte Autor.**

# 1.4. Retorno/Devolução de valor de funções

Toda função, ao encerrar, retorna (ou devolve) o controle de execução à instrução seguinte à que foi chamada. Isso nós já vimos, porém, existe outra característica em funções com alta relevância, o **retorno (ou devolução) de valor resposta**.

Como visto nos exemplos anteriores, foram criadas duas funções: (a) uma chamada exibe\_media e (b) outra chamada retorna\_media. É notável que ambas têm objetivos semelhantes, porém não foram implementadas de forma igual, pois a primeira exibe o valor da média calculada e a outra retorna o valor da média calculada como resposta para a instrução que a chamou.

Portanto, existe uma diferença fundamental entre funções que: (I) **retornam um valor** como resposta e que (II) **não retornam um valor** como resposta.

Uma função que não retorna valor, geralmente, se limita às ações ou efeitos colaterais, tal como exibir os dados de saída na tela. **Não confunda** retornar/devolver valor com exibir algo, esse é um erro comum de iniciantes no uso de funções. Quando um valor é retornado, é possível executar operações sobre ele, como somas, subtrações etc. Quando um valor é somente exibido em uma função, por exemplo, não é possível armazená-lo em uma variável.

Uma forma clara de se acostumar com essa diferença é associar funções que devolvem valor com **perguntas**, por exemplo: raiz\_quadrada(n). É fácil perceber que para essa função o que se espera é uma resposta para a pergunta “*qual é a raiz quadrada do parâmetro n?*”. Note, a resposta não é dada com uma exibição dentro da função, mas sim usando um comando específico, **return**.

Já as funções sem retorno/devolução de valor resposta podem ser associadas a **ordens**, por exemplo: exibe\_idade(ano\_nasc, ano\_atual). Note que a própria intuição ao ler o rascunho de cabeçalho da função deixa claro que não se trata de uma pergunta, e sim de uma ordem cuja ação inclusive consta no próprio nome: provavelmente o que se deve fazer é exibir a idade com base nos argumentos passados para a função.

Uma dúvida comum é **quando usar** cada tipo de função. A resposta direta é: **depende**. A dependência em âmbito escolar, é essencialmente o enunciado da questão, que provavelmente indicará o formato da função. Porém, em um ambiente profissional, a **experiência** e/ou a **especificação** do projeto indicará qual o formato mais adequado. Por isso é importante treinar ambos modelos.

**Agora uma revelação**: tecnicamente **toda função** retorna valor. Assim, conceitualmente os pressupostos anteriores continuam válidos (e seguidos rigorosamente por diversas linguagens de programação), porém Python tem funcionamento peculiar.

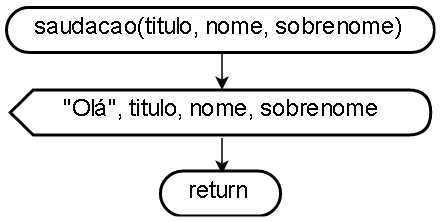
Existem funções que devolvem valores comuns, quando há um **return** explicitamente escrito no código e seguido de um valor, e funções que devolvem um valor especial **None (**em Python**)** que indica exatamente o que o nome sugere, que a função devolve **nada**. Contudo, para fins didáticos, consideramos que funções que não executem uma instrução explícita de retorno de valor são funções sem retorno de valor.

# 1.5. Passagem de Argumentos para Funções

Vimos que funções podem ser definidas contendo parâmetros e que para chamá-las é preciso escrever o nome da função acompanhado dos argumentos correspondentes aos parâmetros.

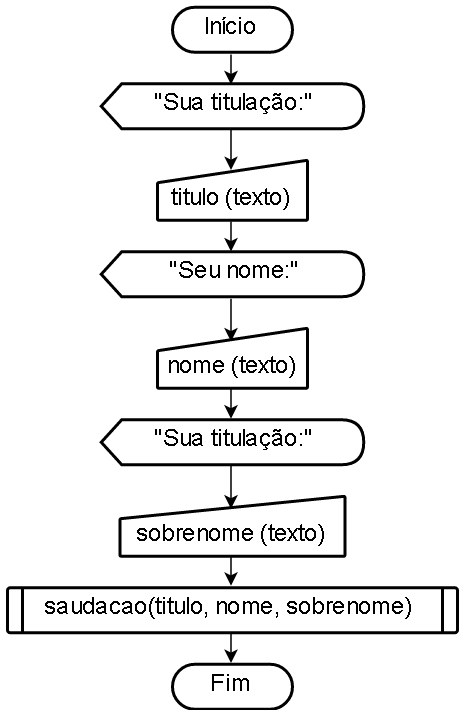
A correspondência entre os argumentos da chamada e os parâmetros da função é determinada pela ordem em que os argumentos são passados e não pelos seus nomes, como pode-se imaginar de início. Ou seja, por padrão, considera os argumentos como **posicionais**.

Para exemplificação, considere a Figura 6.13, uma função que recebe como parâmetros um título, um nome e um sobrenome. O objetivo da função é exibir uma frase no formato 'Olá <titulo> <nome> <sobrenome>':



**Figura 6.13. Fluxograma da Função saudacao. Fonte Autor.**

Note que os parâmetros são usados em uma ordem específica que influenciará no resultado da frase exibida. Considere a Figura 6.14 , um fluxograma completo que usa a função saudacao definida acima e o seu respectivo teste de mesa com as entradas **especialista**, **João** e **Santos**:



**Figura 6.14. Fluxograma que usa a Função saudacao. Fonte Autor.**

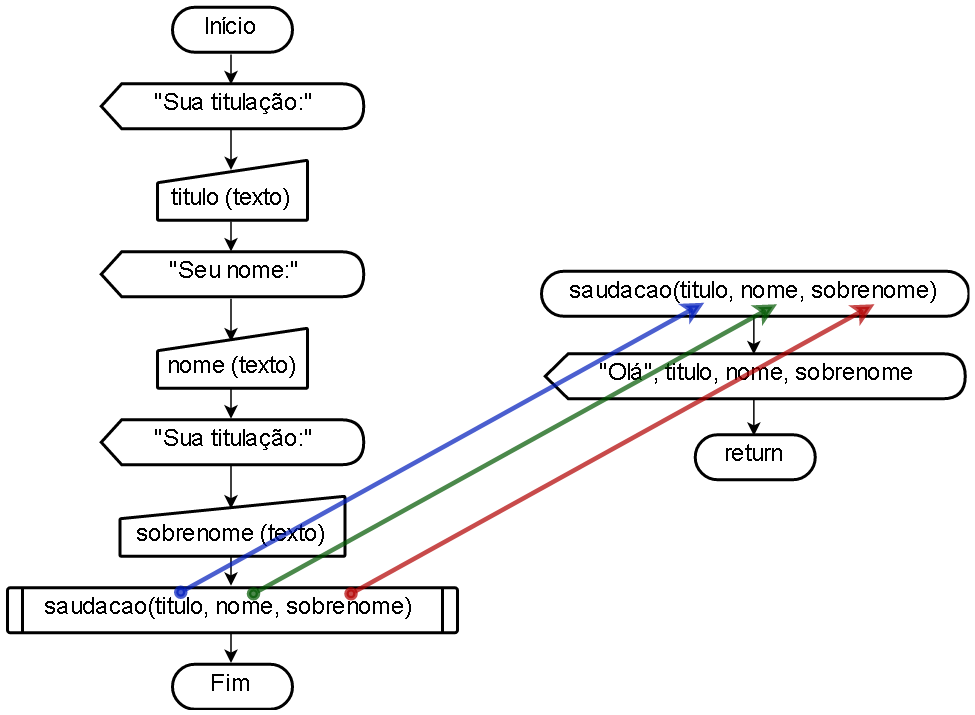
## 

**1.6. Teste de Mesa com Funções**

| | **Escopo global** | | | | --- | --- | --- | | **titulo** | **nome** | **sobrenome** | | especialista | João | Santos |  | **Escopo da função saudacao** | | | | --- | --- | --- | | **titulo** | **nome** | **sobrenome** | | especialista | João | Santos | | | **Tela** | | --- | | Sua titulação: | | Seu nome: | | Seu sobrenome: | | Olá especialista João Santos | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|
|
|

**Figura 6.15. Teste de mesa do programa saudacao. Fonte Autor.**

Observe que os valores dos argumentos são copiados para os parâmetros na ordem em que são escritos, ou seja, o primeiro argumento é associado ao primeiro parâmetro, o segundo argumento é associado ao segundo parâmetro e assim sucessivamente, ou seja, um argumento é associado a um parâmetro particular baseado em sua posição na sequência de argumentos que corresponda à mesma posição do parâmetro na sequência de parâmetros.



**Figura 6.15. Fluxograma que usa a função saudacao usada no teste de mesa. Fonte Autor.**

Portanto não é necessário que os nomes dos argumentos sejam iguais aos dos parâmetros, basta que as posições sejam logicamente equivalentes às dos parâmetros.

# 1.6. Escopo de variáveis - Global e Local

Uma variável é chamada de **variável local** quando é acessível somente dentro de uma única rotina, mais precisamente daquela rotina em que está contida. O seu ciclo de vida, ou seja, o período que a variável existirá, é igual a duração da execução da rotina que a contém.

Quando uma rotina é chamada, ela cria automaticamente as variáveis locais definidas pelos parâmetros formais, atribuindo a elas os valores recebidos através da chamada da função. Conforme a rotina é executada, todo processo em que há a criação de uma nova variável o fará localmente e, ao final da execução da rotina, ou seja, no momento em que ela é encerrada, todas as variáveis locais são automaticamente destruídas.

Uma **variável global** é aquela que é criada no programa principal fora de qualquer função. São ditas terem um escopo global e estarão disponíveis na memória ao longo de toda a execução do programa. Tais variáveis são visíveis a todas as funções a partir do momento que são criadas e só são destruídas quando a execução do programa principal chega ao fim.

A esse espaço de visibilidade das variáveis dá-se o nome de **escopo**. Uma forma de entender o significado de escopo é imaginar que cada função tem uma área de trabalho própria, de onde é possível ver a área de trabalho principal mas não a área de trabalho de outras funções.

Todas as funções definidas no escopo global podem acessar e alterar todas as variáveis globais previamente definidas. Por essa razão o **uso de variáveis globais** é considerado uma **má prática** de programação em grande parte das situações reais e deve ser evitado.

Para evitar a criação de variáveis globais, uma forma comumente utilizada em diversas linguagens de programação, é a criação de uma função chamada main(), sem parâmetros nem retorno, que irá conter todo o código do programa principal. Para que o programa seja executado então, basta chamarmos a função main().

Veja no exemplo a seguir que a variável max é definida no programa principal, fora do escopo das funções test1() e test2() e, portanto, é considerada global, estando acessível a ambas funções.

| F  U  N  Ç  Õ  E  S |  |
| --- | --- |
|  |
| P P  R R  O I  G N  R C  A I  M P  A A  L |  |

**Figura 6.16. Fluxograma com funções mostrando escopo de variáveis. Fonte Autor.**

Dica: Para visualizar a criação das variáveis globais e locais, crie funções e use o pythontutor para visualizar sua execução: [Visualize code in Python, JavaScript, C, C++, and Java](http://www.pythontutor.com/visualize.html).

# 1.7. Comparativo entre fluxograma e Python

Abaixo podemos identificar um fluxograma da função retorna\_media e sua conversão para python.

| **Fluxograma** | **Python**  def bem\_vindo ( ):  print (“Bem Vindo! Vamos calcular a média”)  return  def retorna\_media (num1, num2, num3):  media= (num1+num2+num3) /3  return media  bem\_vindo ( )  print (“Digite o primeiro valor”)  n1=int (input ( ) )  print (“Digite o segundo”)  n2=int (input ( ) )  print (“Digite o terceiro valor”)  n3=int (input ( ) )  resposta=retorna\_media (n1, n2, n3)  print (“A media é”, resposta) |
| --- | --- |

**Figura 6.17. Fluxograma e seu respectivo código em Python. Fonte Autor.**

# 1.8. Referências

DIERBACH, C. “Introduction to Computer Science Using Python: A Computational Problem-Solving Focus”1st Edition, New York: Wiley, 2012.

Python Tutor - <http://www.pythontutor.com/visualize.html>