**Texto base**

**9**

**Funções - Passando parâmetros por Valor e Referência**

Gilberto Alves Pereira

***Resumo***

*Sabemos que os parâmetros de uma função tem o importante papel de fornecer os insumos (matéria-prima) da função. Utilizando corretamente os parâmetros de uma função podemos deixá-la muito mais reutilizável. Sabemos também que as funções conseguem alterar apenas as variáveis do seu escopo, as variáveis locais a função. Vamos aprender neste capítulo as duas formas que a função tem de passar parâmetros para uma função: por valor e por referência. Vamos ver que na passagem por valor a variável utilizada como parâmetro não consegue ser alterada pela função mas na passagem por referência a função consegue alterar essa variável.*

# 1.1. Introdução

A utilização de parâmetros em funções é um mecanismo muito eficiente para permitir a reutilização da função. Esses parâmetros são considerados as matérias-primas das funções, seus insumos. Representam os dados necessários para que a função consiga cumprir seu objetivo. Sabemos também que os parâmetros e as variáveis criadas na função são variáveis locais e portanto seus valores não conseguem ser acessados externamente às funções. Seria possível então criarmos uma função que teria como objetivo o preenchimento de uma variável como por exemplo um vetor? A seguir vamos entender mais detalhadamente como isso é possível em algumas situações.

# 1.2. Passagem de Parâmetros para funções

Os parâmetros carregam as informações para a função. Existem duas formas de passarmos essas informações para as funções:

**por Valor** : O valor passado como parâmetro é copiado para a função

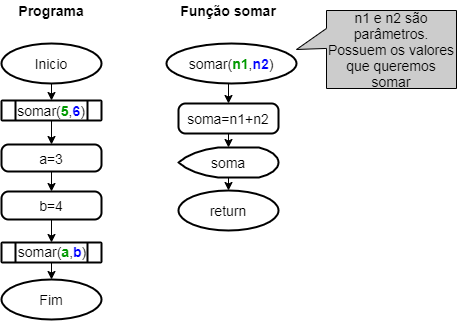
**por Referência**: Não é criada uma cópia. Usa-se a própria variável do parâmetro

# 

# 1.2.1 Passagem por valor

Esse é o tipo mais comum utilizado. Nesse caso os a função recebe os valores passados na chamada da função. Observemos o exemplo abaixo:

**Exemplo 1** - Criar uma função que recebe como parâmetro 2 números e exibe a soma desses números.



**Figura 9.1. Fluxograma e Fluxograma de uma função que soma e exibe dois valores.**

**Fonte Autor.**

No Fluxograma acima temos a direita o fluxograma da função, Perceba que temos dois parâmetros **n1** e **n2** utilizados para receber os valores a serem somados. À esquerda temos um fluxograma que invoca a função **soma** duas vezes. Na primeira vez usamos o comando **soma(5,6)** . Quando executamos esse comando, a função **soma** é chamada e usamos como parâmetros os valores **5** e **6**. Perceba que na invocação da função não estamos usando variáveis e sim números. Nesse caso o que ocorre quando a função é chamada é que o número **5** é copiado para a variável local **n1** e o número **6** é copiado para a variável local e também parâmetro **n2**. O primeiro valor **5** é copiado para o primeiro parâmetro **n1** e o segundo valor **6** é copiado para o segundo parâmetro **n2**. Chamamos isso de passagem posicional pois dependendo da posição do valor passado ele será copiado para um parâmetro. O primeiro valor para o primeiro parâmetro e assim por diante. A partir daí os comandos da função são executados considerando-se esses valores para as variáveis **n1** e **n2** até a exibição e o comando return.

Devemos lembrar que a execução do comando **return** numa função finaliza a função naquele ponto. Caso existam outros comandos após o comando **return** eles não serão executados. Outra coisa que acontece após a execução do comando **return** é que todas as variáveis locais criadas na função são desalocadas da memória, ou seja, elas não existem mais. Portanto quando saímos da função todas aquelas informações usadas na função, os parâmetros e os cálculos são perdidos. Dizemos que essas variáveis são efêmeras e seu tempo de vida se resume ao tempo que a função demora para ser executada. A única coisa que podemos aproveitar de uma função é o valor que ela retorna, o valor usado juntamente com o comando return. Mesmo assim, esse valor que pode ser retornado pelo comando **return** não é obrigatório. Que é o caso da nossa função **soma**. Ela não retorna nada. Ela exibe o valor da soma calculado na função. Perceba que nessa função usamos apenas o comando return, sem nenhum valor na frente do comando.

Seguindo o fluxograma, na segunda chamada da função **soma** perceba que usamos o comando **soma(a,b)**. Nesse caso, os valores que serão passados como parâmetro serão **3** e **4** que são os valores das variáveis **a** e **b** respectivamente. Assim, na invocação da função o valor da variável **a (3)** será copiado para a variável **n1** da função e o valor da variável **b (4)** será copiado para a variável **n2** da função soma. Observe o Quadro 9.1 na linha segunda chamada. A partir daí ocorre a execução da função de maneira semelhante a primeira invocação da função, a única diferença ocorre em relação aos valores dos parâmetros, 5 e 6 na primeira execução e 3 e 4 na segunda. Note que apesar de termos usado números como parâmetro na primeira chamada e variáveis na segunda chamada, o funcionamento da função foi semelhante. Isso ocorre porque, sob o ponto de vista da função, o que ela recebe são valores e não variáveis. O que é passado para a função como parâmetro são os valores das variáveis ou valores que passamos diretamente nos parâmetros como na primeira chamada desse exemplo. Esse é o motivo pelo qual esse tipo de passagem de parâmetro é chamado de passagem por valor. É o tipo mais comum de passagem de parâmetros.

| **Programa** | **Função somar** |
| --- | --- |

|  | **Tela** | **a** | **b** | **soma** | **n1** | **n2** |
| --- | --- | --- | --- | --- | --- | --- |
| **1a Chamada** | 11 |  |  | **11** | **5** | **6** |
| **2a Chamada** | **7** | **3** | **4** | **7** | **3** | **4** |

**Quadro 9.1. Simulação do Fluxograma da função soma.**

**Fonte Autor**

# 1.2.2 Passagem por referência

Nesse tipo de passagem de parâmetro conseguimos alterar o conteúdo da variável usada na invocação da função. Nesse caso o que é passado para a função é o endereço de memória da variável, não o valor da variável. Observemos o exemplo 2 abaixo:

**Exemplo 2 -** Criar uma função que recebe um vetor como parâmetro e multiplica seus elementos por 5.

Primeiramente perceba que o objetivo dessa função é alterar o conteúdo de um vetor passado como parâmetro. Assim criamos o vetor **a** no fluxograma do programa **a=[1,2,3]** e passamos esse vetor juntamente com 2 outros parâmetros: **n** - o tamanho do vetor e **m** o valor que vamos multiplicar cada elemento do vetor. Assim se invocarmos a função passando **m** sendo **5** esperamos que ao final da execução o vetor **a** seja alterado para **[5,10,15]**.

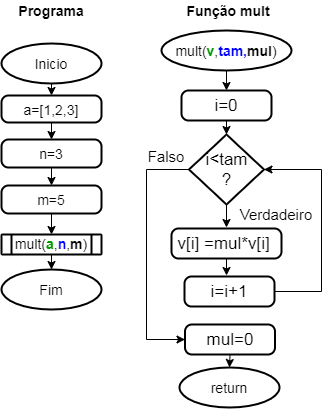
Podemos observar na função **mult** que temos três parâmetros (variáveis locais): **v,tam** e **mult**, onde **v** é o vetor que queremos alterar, **tam** é o tamanho do vetor e **mult** é o valor que iremos usar para multiplicar o conteúdo do vetor **v**. Perceba que a multiplicação do vetor é feita através de um loop que vai alterar o conteúdo de cada uma das posições do vetor através do comando **v[i]=mult\*v[i]**. Quando **i** vale **0** o conteúdo da posição **0** do vetor **v** é alterada, depois o mesmo ocorre para 1 e assim por diante.

Vimos na passagem por valor que os parâmetros de uma função são variáveis locais que são destruídas logo após o término da execução da função. Se é assim, o vetor v também é destruído quando saímos da função. Então como ocorre a alteração do vetor a?

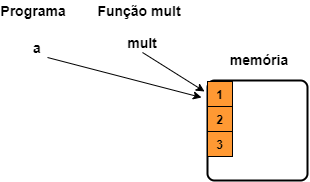
Em Python assim como em várias outras linguagens quando passamos vetores e alguns outros tipos de variáveis como parâmetro a passagem ocorre de uma forma um pouco diferente.

Vamos entender melhor como ocorre. No nosso exemplo, quando passamos o vetor **a** como parâmetro o que ocorre é o parâmetro **v** da função não recebe os valores do vetor **a**. Ele funciona como um apelido do vetor **a**. Ou seja, o vetor **v** recebe a referência de **a** e não seu conteúdo como ocorre na passagem por valor. Dessa forma, tanto **v** quanto **a** estarão acessando o mesmo local na memória onde estão as informações do vetor **a (**Figura 9.3). Então qualquer alteração em **v** na verdade estará alterando o vetor **a** (ambas acessam as mesmas posições na memória). Qualquer alteração feita no parâmetro correspondente afetará o vetor original

Outra observação que podemos fazer é que numa mesma chamada podemos ter tipos de passagem diferentes. Assim, no nosso exemplo, a variável **a** é passada por referência e as outras variáveis **n** e **m** são passadas por valor.



**Figura 9.2. Fluxograma de um função que altera os valores de um vetor passado como parâmetro. Fonte Autor.**



**Figura 9.3 As variáveis a e mult apontam para a mesma posição de memória. Elas possuem a mesma referência (o mesmo endereço). Fonte Autor.**

No quadro 9.2 podemos observar 2 coisas:

1. Perceba que as alterações na variável v da função mult refletem na variável a do programa.
2. As alterações da variável **mul** da função **mult** não refletem na variável **m** do programa

| **Programa** | **Função mult** |
| --- | --- |

| **a[0]** | **a[1]** | **a[2]** | **n** | **m** | **v[0]** | **v[1]** | **v[2]** | **tam** | **mul** | **i<tam** | **i** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **3** | **5** | **1** | **2** | **3** | **3** | **5** | **0<3V** | **0** |
| **5** | **10** | **15** |  |  | **5** | **10** | **15** |  | **0** | **1<3V** | **1** |
|  |  |  |  |  |  |  |  |  |  | **2<3V** | **2** |
|  |  |  |  |  |  |  |  |  |  | **3<3F** | **3** |

**Quadro 9.2. Simulação do Fluxograma da função mult.**

**Fonte Autor**

Em resumo, em Python uma variável dos tipos primitivos (int, float, str) sempre é passada por valor. Não conseguimos em Python passar essas variáveis por referência.

Existem alguns tipos de variáveis como listas, tuplas, dicionários ou objetos (Classes) que são passados sempre por referência mesmo sem nenhuma indicação. Em Python, variáveis desses tipos não conseguimos passar por valor.

Ou seja, em Python o tipo de variável define como será a sua passagem por valor ou referência.

# 1.3. Exemplos de outras linguagens

Os exemplos que vimos acima são de códigos em Python. O tipo da variável define o tipo de passagem. O mesmo acontece com algumas linguagens como Java. Em outras linguagens podemos escolher como será o tipo da passagem. O tipo de passagem fica explícito.

Seguem alguns exemplos:

**Visual Basic -** Veja abaixo que os parâmetros passados por valor são precedidos por Byval e por referência por Byref. Conseguimos passar tipos primitivos (inteiro, real) por referência ou valor.Vetores são passados sempre por referência.



**Figura 9.4. Exemplo em Visual Basic de uso de passagem por valor e referência. Fonte Autor.**

**Linguagem C** - Veja abaixo que os parâmetros passados por valor não possuem nenhuma indicação e por referência indicamos por **\***. Conseguimos passar tipos primitivos por referência ou valor. Vetores são passados sempre por referência.



**Figura 9.5. Exemplo em Linguagem C de uso de passagem por valor e referência. Fonte Autor.**

**Pascal** - Veja abaixo que os parâmetros passados por valor não possuem nenhuma indicação e por referência indicamos por **var**. Conseguimos passar tipos primitivos por referência ou valor. Vetores são passados sempre por referência.

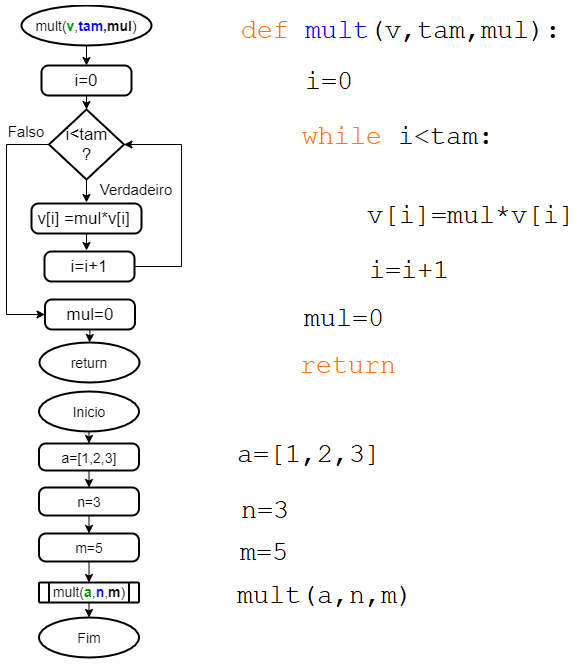


**Figura 9.6. Exemplo em Pascal de uso de passagem por valor e referência. Fonte Autor.**

**1.7. Comparativo entre fluxograma e Python**

Abaixo podemos identificar um fluxograma do Exemplo 2 e sua conversão para Python.

| **Fluxograma** | **Python** |
| --- | --- |

****

**Figura 9.7. Fluxograma e seu respectivo código em Python do exemplo 2. 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>