Unidade 8: Modularização de Código

Prof. Daniel Caetano

Objetivo: Modularização de Código com Construção de Funções.

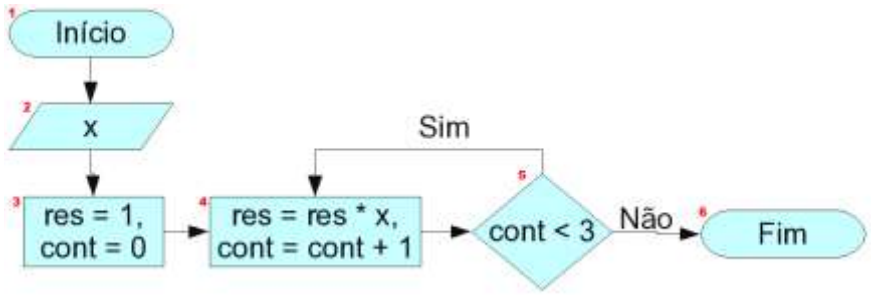
Bibliografia: ASCENCIO, 2007; MEDINA, 2006; SILVA, 2010; SILVA, 2006.

INTRODUÇÃO

Nas aulas anteriores aprendemos a fazer algoritmos simples, com um único bloco de texto. Isso é prático para programas mais simples como os que temos feito, mas para programas mais complexos, a situação muda.

Imaginemos, por exemplo, que um dia tenhamos que fazer um programa que calcule uma infinidade de operações de derivação (derivadas). Se a cada vez que precisarmos programar uma derivação tivermos que escrever todo o código de derivação (que é enorme e complexo)... estaríamos com a nossa vida bem complicada!

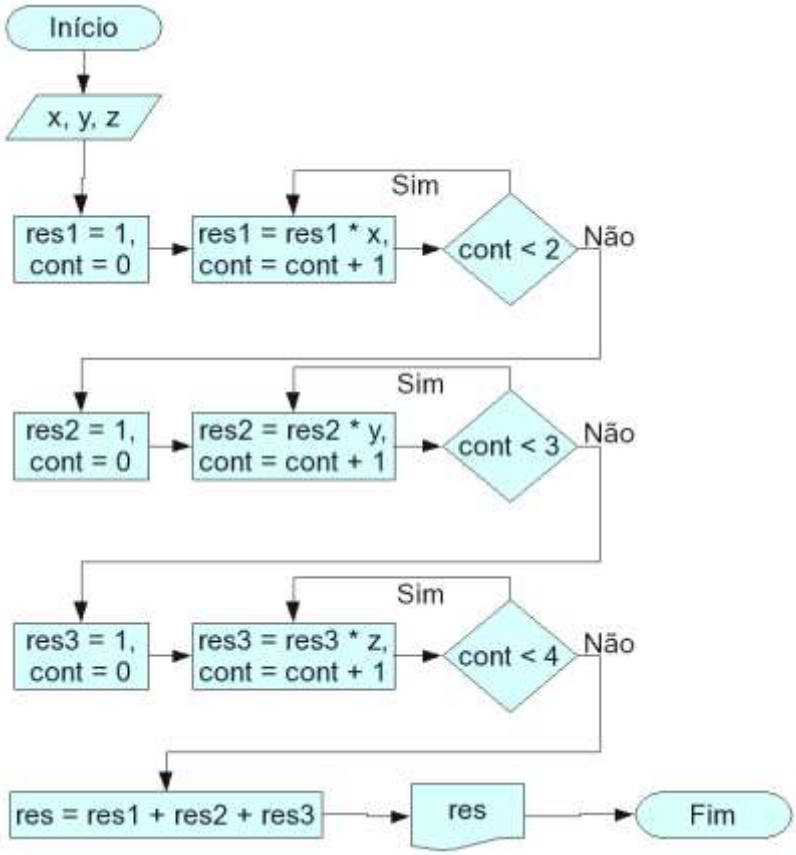
Vamos exemplificar. Considere o algoritmo do fluxograma abaixo, que resolve a potência x^3, usando a lógica de que $x^3 = x * x * x$.



A lógica do algoritmo é simples: iniciamos o resultado res = 1 e, para cada , valor da contagem cont, multiplicamos res por x (res = res * x). A tabela a seguir mostra os valores das variáveis em cada passo, supondo que o usuário tenha digitado o valor 3 para x:

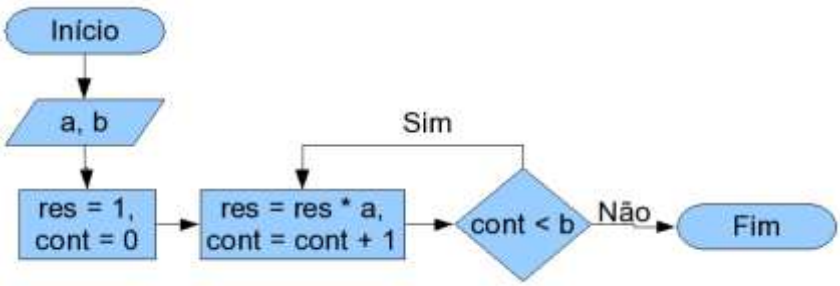
Passo	Bloco	x	res	cont	cont < 3
1	1	-	-	-	-
2	2	3	-	-	-
3	3	3	1	0	-
4	4	3	3	1	-
5	5	3	3	1	SIM
6	4	3	9	2	-
7	5	3	9	2	SIM
8	4	3	27	3	-
9	5	3	27	3	NÃO
10	6	3	**27**	3	-

Observe que, ao final, o resultado res = 27, que é exatamente o valor para 3^3. Agora, veja só como ficaria um fluxograma completo para o cáculo da expressão $x^2+y^3+z^4$:



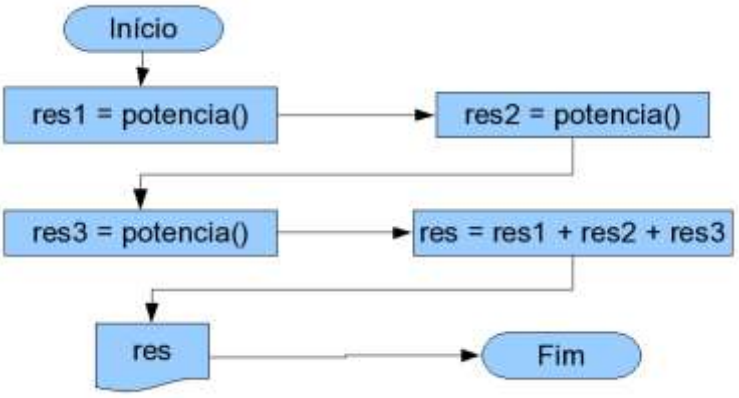
Observe que, neste fluxograma, temos uma porção de blocos repetidos, com poucas variações internas. Isso ocorre porque, obviamente, a forma de calcular a potência de um valor por outro (a^b) é sempre parecido, mudando apenas o valor que é usado na multiplicação (a) e o limite do cálculo lógico (b).

Que tal se pudéssemos dar um nome para esse bloco e, ao invés de repetí-lo, pudéssemos apenas indicar seu uso? Imagine que possamos definir o bloco chamado **potencia()**, definido conforme a seguir:



Esse é um algoritmo que recebe dois números, **a** e **b**, e calcula a potência: $\mathbf{a^b}$.

Vamos reconstruir o programa anterior, que calcula **res = $x^2+y^3+z^4$** apenas com o uso da expressão **potencia()** para que todo o processo apresentado acima seja executado, resultando no valor do cálculo:

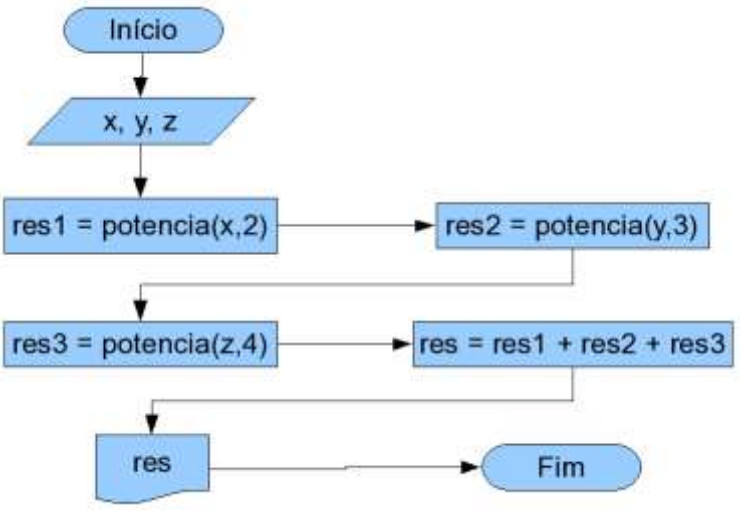


Bem mais simples de entender, não? Entretanto, esse resultado não é tão bom; o programa original pedia apenas **três** números; esse pede **seis**, porque o algoritmo que chamamos de **potencia()** considera que o usuário deve digitar os dois (o valor da base e o do expoente) para poder ser usado... e esse algoritmo, por sua vez, é executado três vezes (2x3 = 6)

Não seria interessante se o programa principal pudesse indicar para o fluxo "potência" que deseja fazer o cálculo com dois valores específicos? Poderíamos usar uma notação do seguinte tipo:

potencia(a,b)

Isto é: dentro dos parênteses, colocaremos as informações necessárias para o cálculo, já no programa principal, de maneira que o algoritmo não tenha que solicitá-las ao usuário. Observe a próxima versão do algoritmo:



No fundo, o que fizemos foi criar **dois** algoritmos: um deles, chamado **potencia()** não é usado diretamente pelo usuário, mas sim pelo nosso algoritmo principal.

Aos algoritmos que são usados por outros algoritmos damos o nome de **função**.

1. USO DE FUNÇÕES

Até o presente momento, o conceito de função foi apresentado de maneira lúdica, com o uso de fluxogramas. Agora veremos, na prática, como definimos uma função. Vamos verificar o código abaixo em Python, que executa uma média entre dois números. Observe as linhas destacadas em verde e vermelho... o que você percebe?

```
# Calcula a media de dois números

print("Calcula a média de dois números")
n1 = float( input ("Digite o 1o. número: ") )
n2 = float( input ("Digite o 2o. número: ") )
m = (n1+n2)/2
print("A média é: ", m)
```

Elas são bem parecidas, não? Vamos criar, então, uma função chamada **lenumero()**, que receberá qual é o número de que deve ser lido (1 para 1o., 2 para 2o., 3 para 3o.... e assim por diante) e nos retornará o valor lido, de maneira que possamos reescrever esse programa da seguinte forma:

```
# Calcula a media de dois números

print("Calcula a média de dois números")
n1 = lenumero(1)
n2 = lenumero(2)
m = (n1+n2)/2
print("A média é: ", m)
```

Se digitarmos o programa acima, entretanto, teremos um erro... o computador irá reclamar que não sabe o que significa **lenumero()**, é claro! Temos, então, de explicar para ele o que é esse tal de **lenumero()**.

Podemos fazer isso assim (não se preocupe, por enquanto, com a forma de escrever, isso será explicado mais adiante):

```
# Calcula a media de dois números

# Programa Principal
print("Calcula a média de dois números")
n1 = lenumero(1)
n2 = lenumero(2)
m = (n1+n2)/2
print("A média é: ", m)

# Função que lê um número
def lenumero(posicao) :
        # Aqui vai o código da função
```

A linguagem ainda não ficará feliz, porque ainda não escrevemos o código da função. Vamos escrevê-lo, então.

```
# Calcula a media de dois números

# Programa Principal
print("Calcula a média de dois números")
n1 = lenumero(1)
n2 = lenumero(2)
m = (n1+n2)/2
print("A média é: ", m)

# Função que lê um número
def lenumero(posicao) :
      # Lê o valor e converte para float
      num = float( input("Digite o " + str(posição) + "o. número: ") )
```

O Python ainda vai reclamar. Isso ocorre porque, no programa, usamos a função **lenumero** da seguinte forma:

n1 = lenumero(1);

Ou seja: eu estou pegando o resultado de **lenumero** e guardando em uma variável. Ocorre que o computador não tem bola de cristal e ele **não sabe** qual valor a função **lenumero** deve responder, isto é, qual é o valor que vai ser fornecido por **lenumero** para que seja armazenado em **n1**!

Para ajudá-lo, precisamos usar a instrução **return**. É a instrução **return** que indica para uma função qual é o valor que deve ser respondido para quem a chamou. Isso é feito da seguinte forma:

```
# Calcula a media de dois números

# Programa Principal
print("Calcula a média de dois números")
n1 = lenumero(1)
n2 = lenumero(2)
m = (n1+n2)/2
print("A média é: ", m)

# Função que lê um número
def lenumero(posicao) :
      # Lê o valor e converte para float
      num = float( input("Digite o " + str(posicao) + "o. número: ") )
      return num
```

Tudo pronto, você executa e... o Python **ainda** reclama! Que coisa chata!

Na verdade, o Python está reclamando porque o programa é executado do início para o fim, ou seja, de cima para baixo. Por essa razão, quando o programa encontra pela primeira vez a instrução:

n1 = lenumero(1);

Ele não sabe o que é isso ainda. Como o computador, ao executar um programa em Python é um bichinho ansioso, isso pode ser resolvido de uma maneira muito simples: colocar a função **lenumero** *antes* do algoritmo principal.

```
# Calcula a media de dois números

# Função que lê um número
def lenumero(posicao) :
      # Lê o valor e converte para float
      num = float( input("Digite o " + str(posicao) + "o. número: ") )
      return num

# Programa Principal
print("Calcula a média de dois números")
n1 = lenumero(1)
n2 = lenumero(2)
m = (n1+n2)/2
print("A média é: ", m)
```

Observe um segundo fato interessante: como o computador sabe qual é o trecho de código que é parte da função e qual não é? A resposta é pelos espaços que antecedem as linhas: observe que as linhas da função estão mais à direita, a partir da segunda linha da função (a primeira linha é a que começa com a palavrinha “def”).

2. EXPLICANDO FUNÇÕES

Bem, já vimos como escrever uma função: basicamente é escrever um outro pequeno programa. Já vimos até como retornar um resultado. Mas como é construída aquela primeira linha "feia" do nome da função?

Observe abaixo:

```
# Função que lê um número
def lenumero(posicao):
        # Aqui vai o código da função
```

A primeira linha, especificamente, é: **def lenumero(posicao):** . O que isso significa?

Essa primeira linha se chama **declaração** da função ou **assinatura** da função. Essa linha diz para o computador (e para qualquer programador que vá usá-la) **como** ela deve ser usada. Essa declaração é composta por três partes principais:

def nome_da_função (parâmetro_da_funcao) :

"Def" indica que estamos definindo uma função (ou um novo comando, se quiser interpretar assim).

O nome da função é o nome que usaremos para executá-la, isto é, sempre que quisermos executar esse trecho de código, usaremos este nome, como já foi visto no programa. As regras para esse nome são as mesmas usadas para variáveis: só letras simples e *underline*, sem códigos especiais, sem espaço. Números são permitidos, **desde que o nome não comece com número!**

E o parâmetro_da_função?

Bem, o parâmetro_da_função descreve a informação que é necessária à função para que ela possa ser executada. No caso da nossa função **lenumero**, o parâmetro é usado simplesmente para imprimir a mensagem:

"Digite o **X**o. número: "

Onde **X** é substituído pelo valor que o programador forneceu ao desenvolver o programa. Assim:

Código:	**Texto impresso:**
lenumero(654);	"Digite o 654o. número: "
lenumero(2);	"Digite o 2o. número: "
lenumero(76);	"Digite o 76o. número: "

O parâmetro funciona, dentro da função, como se fosse uma variável com um valor pré-definido. Assim, precisamos dar um **nome** para o parâmetro, como para qualquer outra variável.

A linha termina com os dois pontos, indicando que a partir da linha de baixo teremos um conjunto de comandos que faz parte dessa função, todos eles devendo estar deslocados de um mesmo número de espaços à direita!

Assim, observe novamente a declaração da função:

def lenumero(posicao):

Essa declaração significa que:

a) Essa função pode ser acessada pelo nome **lenumero**.

b) Essa função recebe um parâmetro que, dentro da função, será acessado por uma variável chamada **posicao**.

Uma função pode ter vários parâmetros. Para indicar isso, usamos a vírgula separando cada um deles. Por exemplo: no caso da função **potencia()** do início da aula, tínhamos dois parâmetros: **a** e **b**, para que ela pudesse executar o cálculo $\mathbf{a^b}$... Assim, aquela função poderia ser declarada da seguinte forma:

def potencia(a, b):

Eventualmente podemos criar uma função que não precise de parâmetros. Nesse caso, indicamos isso deixando vazio o espaço entre os parênteses. Consideremos, por exemplo, uma função chamada **imprimeajuda**, que não recebe parâmetros e nem retorna nenhum valor. Ela seria declarada desta forma:

def imprimeajuda():

2.1. Corpo da Função

Depois da declaração, o código da função deve ser especificado. Observe que, logo após a declaração, é sempre indicado um trecho deslocado à direita! Observe:

```
# Função que lê um número
def lenumero(posicao):
        # Aqui vai o código da função
```

Esse deslocamento indica quais linhas devem ser executadas ao se chamar o nome de função definido logo acima. Tudo que estiver deslocado à direita (sempre pelo MESMO número de espaços) será executado quando a função for chamada.

Caso a função possa retornar um valor, esse valor deve ser retornado ao FINAL da função, com o comando **return**.

3. ESCOPO DAS VARIÁVEIS

Assim como no programa principal, é possível usar variáveis. Além daquelas recebidas como parâmetros, é possível declarar novas variáveis. A declaração se faz como no programa principal: basta guardar um valor dentro dela. Por exemplo:

```
# Função que lê um número
def lenumero(posicao) :
      # Lê o valor e converte para float
      num = float( input("Digite o " + str(posicao) + "o. número: ") )
      return num
```

Os pontos importantes... IMPORTANTES e que não devem ser esquecidos são:

a) A variável declarada em uma função **NÃO EXISTE** em outra função, isto é, a variável só vale dentro da função em que ela foi declarada.

b) Uma variável declarada no programa principal, **EXISTE** dentro de todas as outras funções.

c) Alterar valor das variáveis-parâmetro é possível, mas **A ALTERAÇÃO SERÁ PERDIDA** quando a função terminar sua execução.

Observe os exemplos a seguir:

```
# Função 1
def funcao1():
     b = 1;
     print("B na Funcao1: ",b)
     print("A na Funcao1: ",a)

# Função 2
 def funcao2():
      b = b + 1;
      print("B na Funcao2: ", b)
      print("A na Funcao2: ", a)

# Programa Principal

a = 1;
print("A na principal: ", a)
funcao1()
funcao2()
print("B na principal: ", b)
```

Esse programa irá dar erros em várias das linhas. Na rotina principal**,** ele irá reclamar que não sabe o que é a variável **b**. E ele reclama com razão, a variável **b** não foi declarada na rotina principal.

Na **funcao1** ele não irá reclamar: a variável **b** foi definida lá dentro e, no momento em que a função é executada, a variável **a** já foi criada na função principal. Finalmente, na **funcao2** ele irá reclamar de **b** porque nesse caso não é possível definir a variável **b** com passe em um valor de **b** que ainda não existe.

Assim, reforçando a primeira regra: **as variáveis só têm validade dentro das funções em que foram declaradas!**

Observe, agora, o programa a seguir:

```
# Função 1
def funcao(a):
     print("A no inicio da função: ", a)
     a = a + 1
     print("A no fim da função: ", a)

# Programa Principal
a = 1;
print("A no inicio da função principal: ", a)
funcao(a)
print("A no fim da função principal: ", a)
```

Observe que a **funcao** recebe o valor correto, atualiza este valor, mas ao voltar para a rotina principal, a alteração se perdeu.

Para reforçar: **é possível modificar variáveis-parâmetro, mas a modificação se perde quando a função termina**. Para retornar valores é preciso usar a instrução **return**.

4. BIBLIOGRAFIA

ASCENCIO, A.F.G; CAMPOS, E.A.V. **Fundamentos da Programação de Computadores**. 2ed. Rio de Janeiro, 2007.

MEDINA, M; FERTIG, C. **Algoritmos e Programação: Teoria e Prática**. 2ed. São Paulo: Ed. Novatec, 2006.

SILVA, I.C.S; FALKEMBACH, G.M; SILVEIRA, S.R. **Algoritmos e Programação em Linguagem C**. 1ed. Porto Alegre: Ed. UniRitter, 2010.

_