

N_ALG PROG_A7 - Texto de apoio

Site: [EAD Mackenzie](#)

Tema: ALGORITMOS E PROGRAMAÇÃO I {TURMA 01D} 2023/1

Livro: N_ALG PROG_A7 - Texto de apoio

Impresso por: CAIO FRESSATTI PINHEIRO .

Data: domingo, 5 fev 2023, 02:04

Índice

1. FUNÇÕES EM PYTHON

- 1.1. Rotina e Funções
- 1.2. Solicitando a Execução de uma Rotina: Chamando uma função
- 1.3. Retorno de Função
- 1.4. Passagem de Parâmetros
- 1.5. Escopo de Variáveis

2. REFERÊNCIAS

1. FUNÇÕES EM PYTHON

Até agora, vimos os conceitos fundamentais do Python, tais como variáveis, expressões, estruturas de controle, entrada e saída. Do ponto de vista prático, contudo, essas estruturas sozinhas ainda não são suficientes.

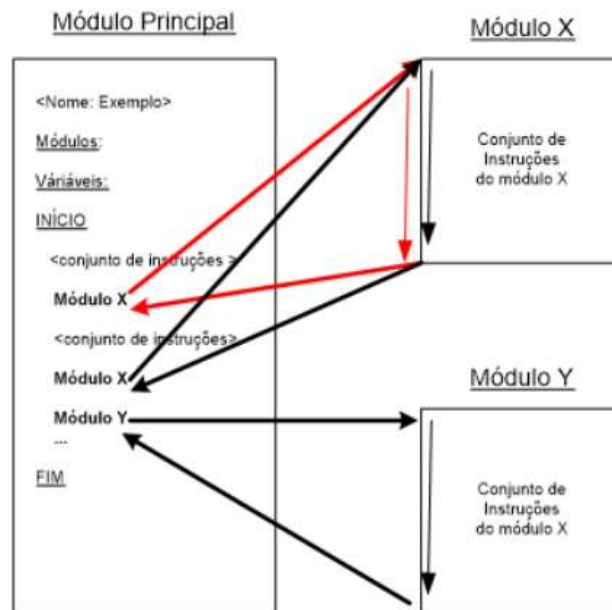
Pensemos em problemas mais complexos. Por exemplo, um smartphone contém cerca de 10 milhões de linhas de código, imagine o esforço para desenvolver e depurar um software dessa dimensão.

A fim de gerenciar a complexidade de um grande problema, é preciso quebrá-lo em subproblemas menores. Então, cada subproblema pode ser analisado e resolvido separadamente, permitindo reuso e não tendo que construir uma solução a partir do zero.

Nesse sentido, módulos ou rotinas são blocos de construção fundamentais no desenvolvimento de software.

1.1. Rotina e Funções

Uma **rotina** é definida como um grupo de instruções que executa alguma tarefa definida, que constitui um trecho de algoritmo com uma função bem definida e o mais independente possível das demais partes do algoritmo.



Fonte: Elaborada pela autora.

Uma rotina pode ser **invocada ou chamada** quantas vezes for necessário em um dado programa.

Quando uma rotina termina de ser executada, automaticamente retorna para o ponto a partir de onde foi chamada. Tais módulos podem ser pré-definidos pela linguagem de programação ou projetados e implementados por um programador.

Uma função em Python é uma rotina escrita na linguagem de programação.

Pode ser definida da seguinte forma:

def <nome da função> (lista de parâmetros):

<corpo da função>

Exemplo 1:

```
#Funcao media
def media (n1, n2, n3):
    soma=n1+n2+n3
    return soma/3.0
```

A primeira linha de definição de uma função é o **cabeçalho** e define a **assinatura da função**. O cabeçalho da função inicia com a palavra-chave def seguida por um identificador (media), que é o nome da função.

O nome da função é seguido por uma lista de identificadores (n1,n2,n3) chamada de **parâmetros formais** ou simplesmente parâmetros.

Os parâmetros são variáveis locais do módulo inicializadas na chamada do módulo. Eles são especificados como uma lista de declaração de variáveis. Se o módulo não recebe parâmetros, a lista pode ser vazia.

Exemplo 2:

```
#Funcao exibe mensagem ao usuario  
def exibeMsg():  
    print("Este programa calcula a media de 3 números")
```

Após a definição da função, tem-se o **corpo**, dentro do qual são declaradas variáveis locais e são codificadas as instruções.

Funções são, geralmente, definidas no início do programa. Contudo, toda função deve ser definida antes de ser chamada.

1.2. Solicitando a Execução de uma Rotina: Chamando uma função

Um módulo envolve dois momentos bem diferentes:

- a criação ou declaração; e
- a execução da rotina.

Uma rotina é executada quando alguma **outra** rotina o invocar, isto é, quando **outra** rotina chamá-lo (muitas vezes, essa outra rotina é o programa principal).

Quando queremos utilizar as rotinas que criamos para resolver uma tarefa, ou seja, solicitar sua execução, devemos fazer chamada ao módulo.

A **chamada** da rotina é a forma de solicitar a execução da rotina em determinado passo do algoritmo.

Para fazer a chamada de um módulo, devemos especificar qual é o nome da rotina e passar argumentos (valores iniciais ou variáveis) para seus parâmetros.

Exemplo 1: considerando a função média definida anteriormente, sua chamada pode ser:

```
#Chamada da função media
res=media(num1,num2,num3)
```

Exemplo 2: considerando a função exibeMsg definida anteriormente, sua chamada é:

```
#Chamada da função exibeMsg
exibeMsg()
```

Exemplo 3: O programa completo que calcula a média aritmética entre três números inteiros.

```
##Objetivo: Encontrar media aritmetica de 3 números
#Entrada: 3 números inteiros
#Saída: a media dos números lidos

#Funcao exibe mensagem ao usuario
def exibeMsg():
    print("Este programa calcula a media de 3 números")

#Funcao media
def media (n1, n2, n3):
    soma=n1+n2+n3
    return soma/3.0

#Principal
num1=3
num2=7
num3=1

#Chamada da função exibeMsg
exibeMsg()

#Chamada da função media
res=media(num1,num2,num3)

print("A media entre %d,%d e %d = %.2f" %(num1,num2,num3,res))
```

1.3. Retorno de Função

Verificamos no exemplo do cálculo da média que a chamada de cada uma das funções `media()` e `exibeMsgs()` foi feita de forma diferente. Isso se deve ao fato de que uma função pode ser classificada em:

- Função que não retorna valor; ou
- Função que retorna valor.

Uma função que não retorna um valor possui um efeito lateral ou uma ação, tal como exibir os dados de saída na tela.

Tecnicamente, toda função em Python é uma função que retorna um valor, uma vez que qualquer função que não retorna, explicitamente, um valor retorna automaticamente um valor especial `None`. Contudo, consideraremos como função que não retorna valor.

Exemplo 1: No exemplo anterior, temos a função `exibeMsg()` que apenas mostra ao usuário o que o programa faz.

Em sua criação, não há referência à instrução `return`.

```
#Funcao exibe mensagem ao usuario
def exibeMsg():
    print("Este programa calcula a media de 3 números")
```

Em sua chamada, não há uma variável que receba seu retorno nem podemos chamá-la dentro da função `print()`.

```
#Principal
num1=3
num2=7
num3=1

#Chamada da função exibeMsg
exibeMsg()
```

Em uma função que retorna um valor, a saída da função é feita quando a instrução `return` é encontrada e o valor da variável ou expressão é retornado ao programa ou à função que a chamou.

No exemplo, tem-se uma chamada da função média com três argumentos `media(3,7,1)`, a execução é desviada para dentro da rotina, executa a operação, quando encontra o `return` soma, a execução volta para o ponto do programa onde foi chamada e o valor é atribuído à variável `res`.

Observe isso no código abaixo:

```
#Funcao media
def media (n1, n2, n3):
    soma=n1+n2+n3
    return soma/3.0

#Principal
num1=3
num2=7
num3=1

#Chamada da função exibeMsg
exibeMsg()

#Chamada da função media
res=media(num1,num2,num3)
```

1.4. Passagem de Parâmetros


A correspondência entre os argumentos atuais da chamada e os parâmetros formais da função é determinada pela ordem em que os argumentos são passados, e não pelos seus nomes.

Considere o exemplo seguinte. Os argumentos atuais num1 e num2 são copiados para os parâmetros formais n1 e n2, mas posso também passar o argumento num1 para o parâmetro n2. Nesse exemplo, a função VerificaMenor2Numeros é chamada uma vez, passando os argumentos num1, num2 e, outra vez, passando num2, num1. Cada chamada é independente e cada um é passado com o que é logicamente necessário naquele momento.

Observe, contudo, que os valores correntes dos argumentos são copiados para os parâmetros na ordem em que são chamados, ou seja, primeiro argumento para o primeiro parâmetro, segundo argumento para o segundo parâmetro e assim sucessivamente. Ou seja, um argumento é associado a um parâmetro particular baseado em sua posição na lista de argumentos.

```
#Funcao que verifica se um numero e menor que outro
def VerificaMenor2Numeros (n1,n2):

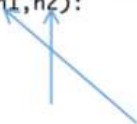
elif VerificaMenor2Numeros (num2,num1):
```



A linguagem Python permite uma outra maneira de chamar uma função, por meio do uso de argumento palavra-chave ou argumento keyword. Nesse caso, um argumento é especificado pelo nome do parâmetro.

```
#Funcao que verifica se um numero e menor que outro
def VerificaMenor2Numeros (n1,n2):

elif VerificaMenor2Numeros (n2=30, n1=23):
    print("Ele e seu irmao mais novo!")
```



1.5. Escopo de Variáveis

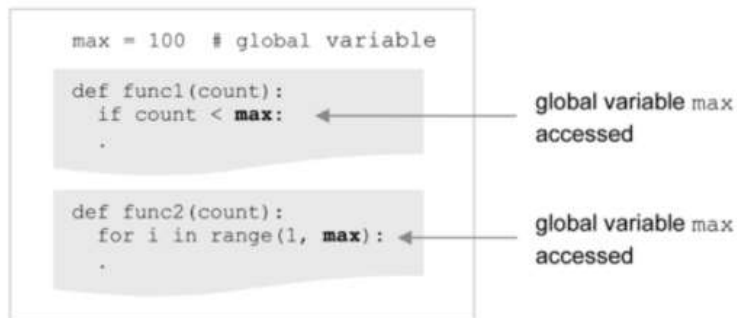
Uma variável é chamada de variável local se somente é acessível a partir de dentro de uma dada função. Seu ciclo de vida, ou seja, o período de tempo que a variável existirá, é igual à duração da execução de sua função.

Ou ainda, as variáveis locais são automaticamente criadas e a memória alocada quando uma função é chamada e destruída, memória desalocada, quando a função termina sua execução.

Uma variável global é aquela definida fora de qualquer definição de função. São ditas terem um escopo global e estarão disponíveis na memória ao longo de toda a execução do programa.

Todas as funções dentro do escopo de uma variável global podem acessá-la e alterá-la. Por essa razão, o uso de variáveis globais não é considerado uma boa prática de programação.

Veja, no exemplo seguinte, que a variável `max` é definida fora das funções `func1` e `func2` e, portanto, é considerada global e pode ser acessível pelas duas funções.



Fonte: Dierbach (2012).

2. REFERÊNCIAS

DIERBACH, C. *Introduction to Computer Science Using Python: A Computational Problem Solving Focus*. New York: Wiley, 2012.