**FACULDADE DE COMPUTAÇÃO E INFORMÁTICA**

**CIÊNCIA DA COMPUTAÇÃO E SISTEMAS DE INFORMAÇÃO**

**Linguagem de Programação I**

**AULA 02: Funções em Python**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | http://images.flatworldknowledge.com/ketchen/ketchen-fig05_x001.jpg | Nossos **objetivos** nesta aula são:   * Explicar o conceito de funções, passagem de parâmetros e funções com e sem retorno; * Explicar a noção do efeito da chamada de uma função; * Diferenciar entre escopo local e global; * Definir e usar funções em Python; * Explicar o conceito de argumentos de uma função em Python; * Escrever programas em Python usando funções definidas pelo programador. | | Macintosh HD:Users:anacris:Desktop:Captura de Tela 2017-02-19 às 18.57.35.png | A referência para esta aula está no **Capítulo 5 (Functions)** do livro:  DIERBACH, C. *Introduction to Computer Science Using Python: A Computational Problem Solving Focus.* 1st Edition, New York: Wiley, 2012. | |
|  |

**Introdução**

* 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.
* Vamos pensar em um problemas mais complexo. 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 quebrar 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.

**Rotina e Funções**

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

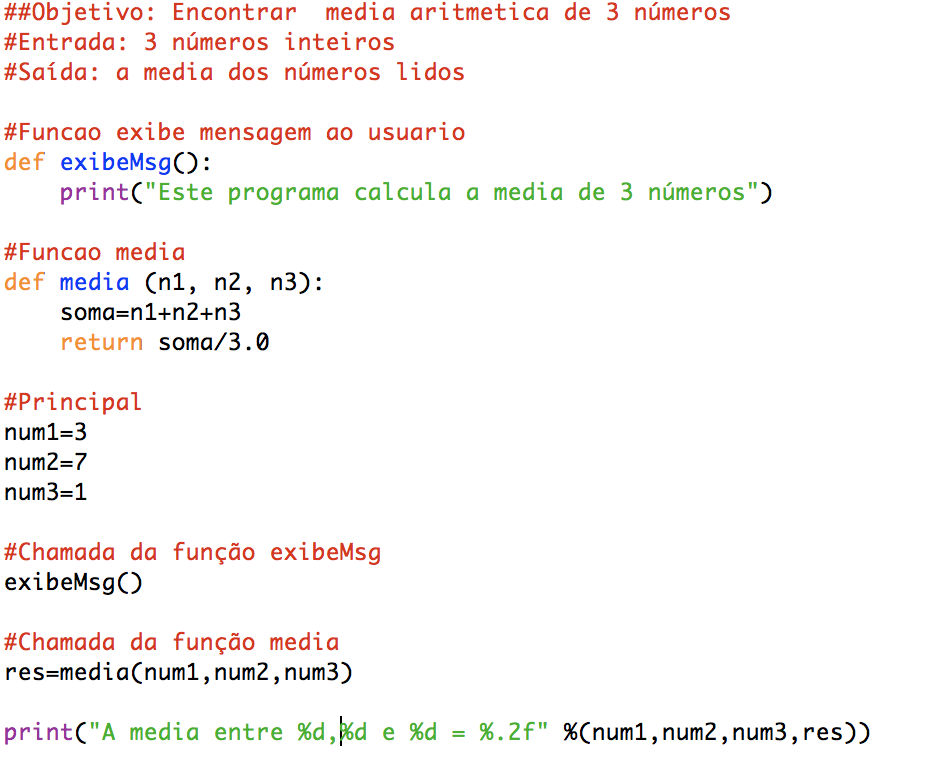


* Uma rotina pode ser **invocada ou chamada** quantas vezes for necessário em um dado programa.
* Quando uma rotina termina de ser executada, automaticamente retorna para o ponto a partir de onde ela foi chamada. Tais módulos podem ser pré-definidas pela linguagem de programação ou projetadas e implementadas por um programador.
* Uma **função** em Python é uma rotina escrito na linguagem de programação. Pode ser definida da seguinte forma:

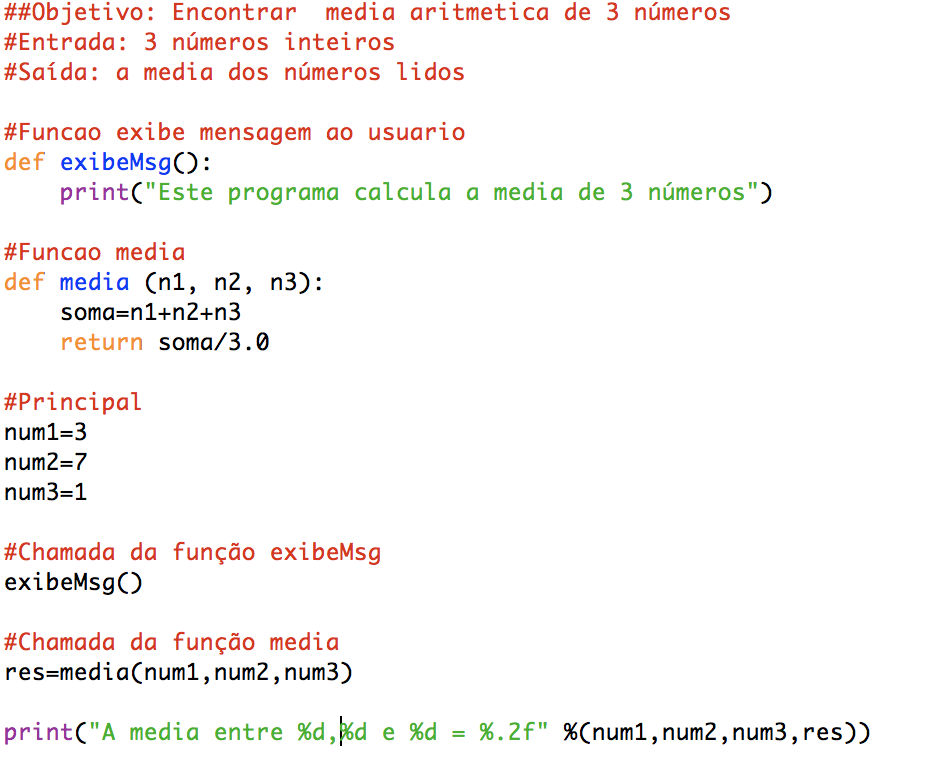
**def <NomeFunção> (lista de parâmetros):**

**<corpo da função>**

* Exemplo 1:



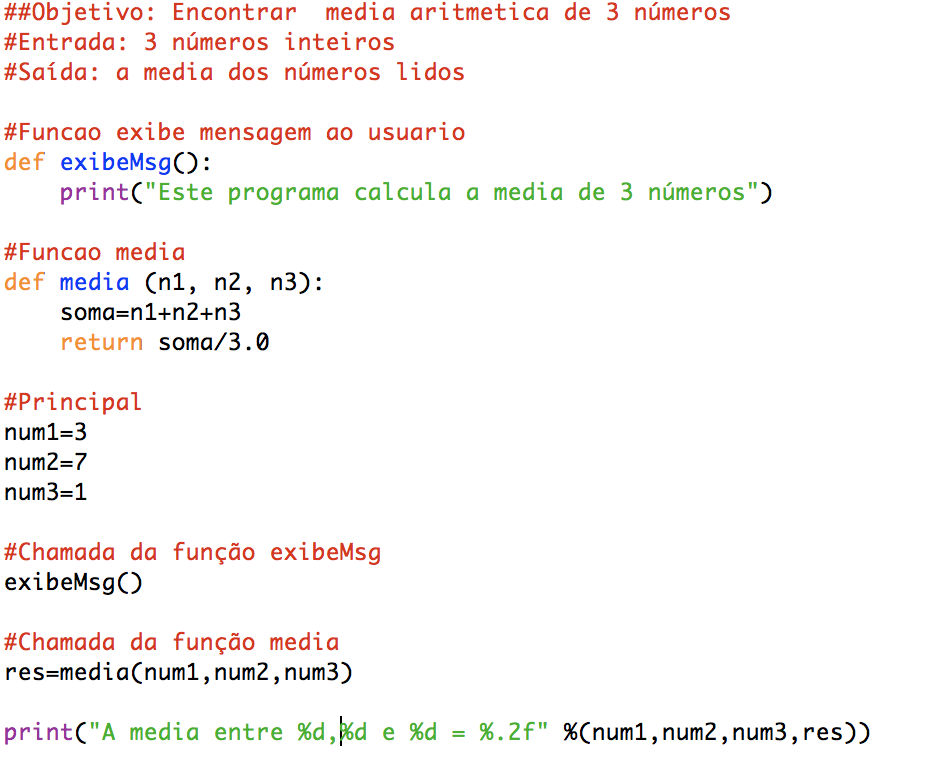
* Exemplo 2:



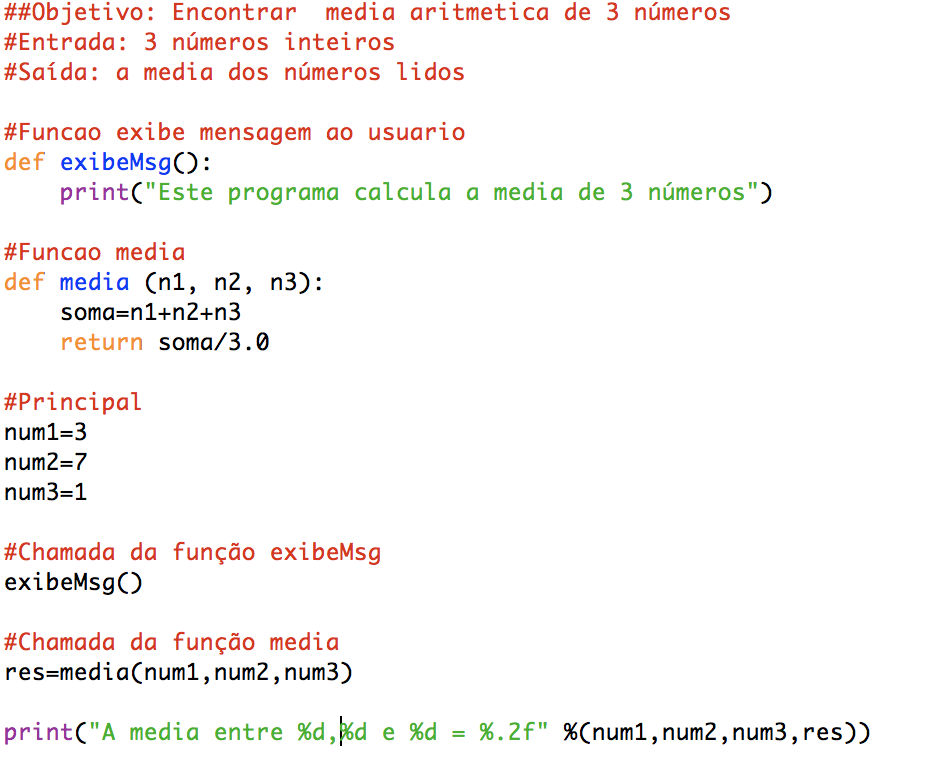
* A primeira linha de definição de uma função é o **cabeçalho**. O cabeçalho da função inicia com a palavra-chave **def** seguido por um identificador (media), que é o nome da função.
* O nome da função é seguido por uma lista de identificadores (n1,n2,n3) chamada de **parâmetros formais** ou simplesmente 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.
* 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.

**Solicitando a Execução de uma Rotina**

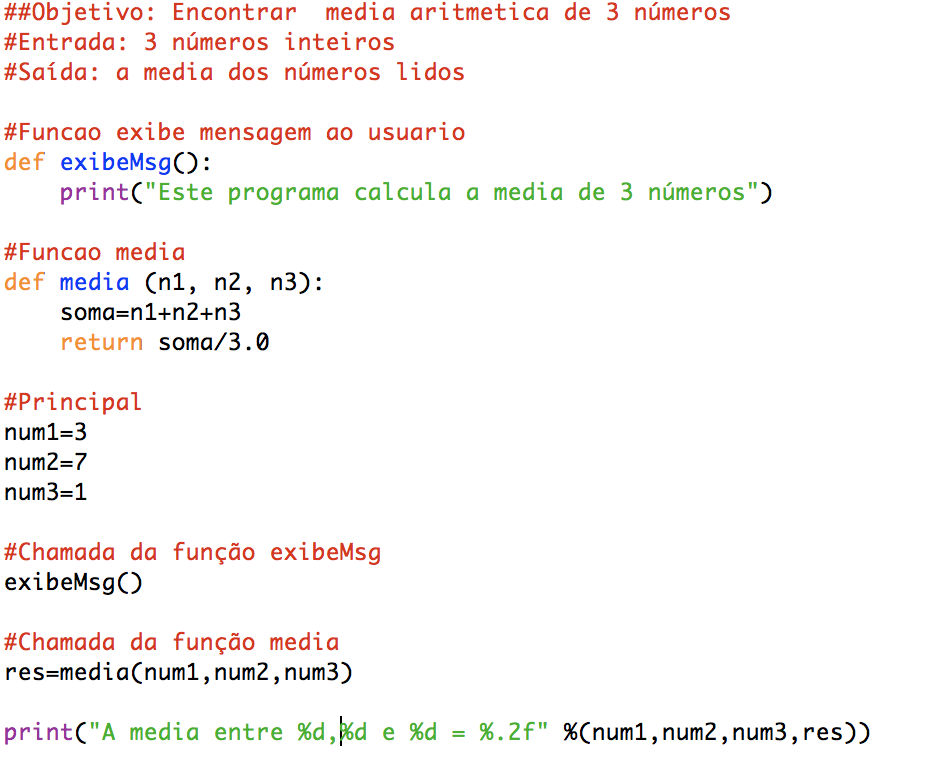
* Um módulo envolve dois momentos bem diferentes:
  + a criação ou declaração e
  + a execução da rotina.
* Uma rotina é executada quando algum **outra** rotina o invocar, isto é, quando **outra** rotina chamá-lo (muitas vezes esse outra rotina é o principal).
* Quando queremos utilizar as rotinas que 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 um determinado passo do algoritmo.
* Para fazer a chamada de um módulo devemos especificar qual o nome da rotina e passar argumentos (valores iniciais ou variáveis) para seus parâmetros.
* Exemplo 1: considerando a função média definida anteriormente, a sua chamada pode ser:



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

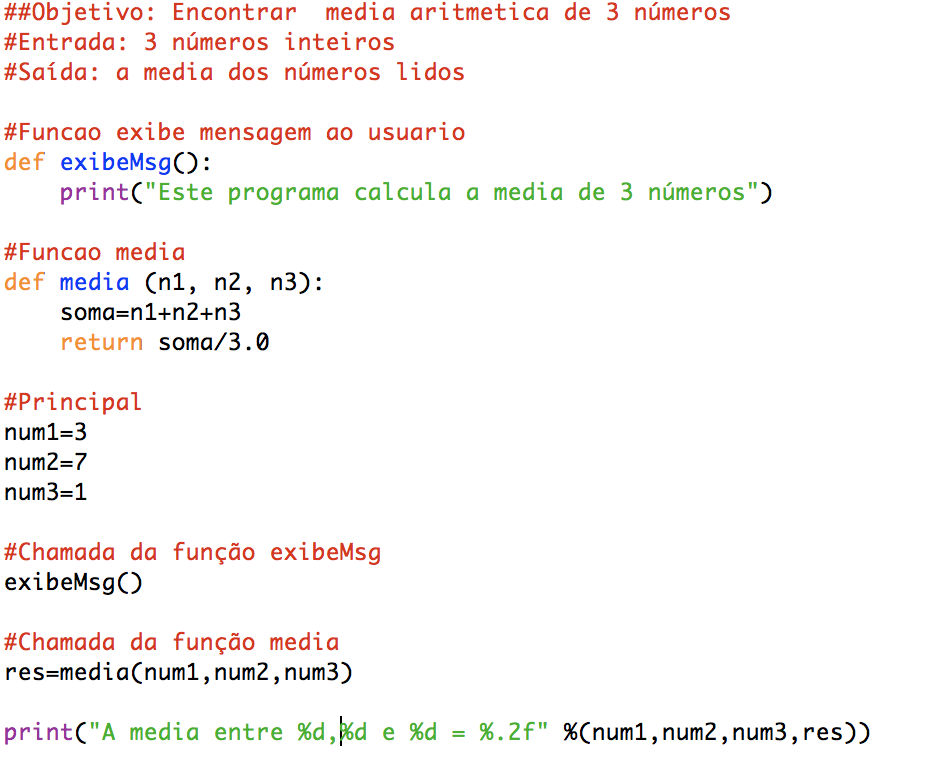


* Exemplo 3: O programa completo que calcula a média aritmética entre 3 números inteiros.

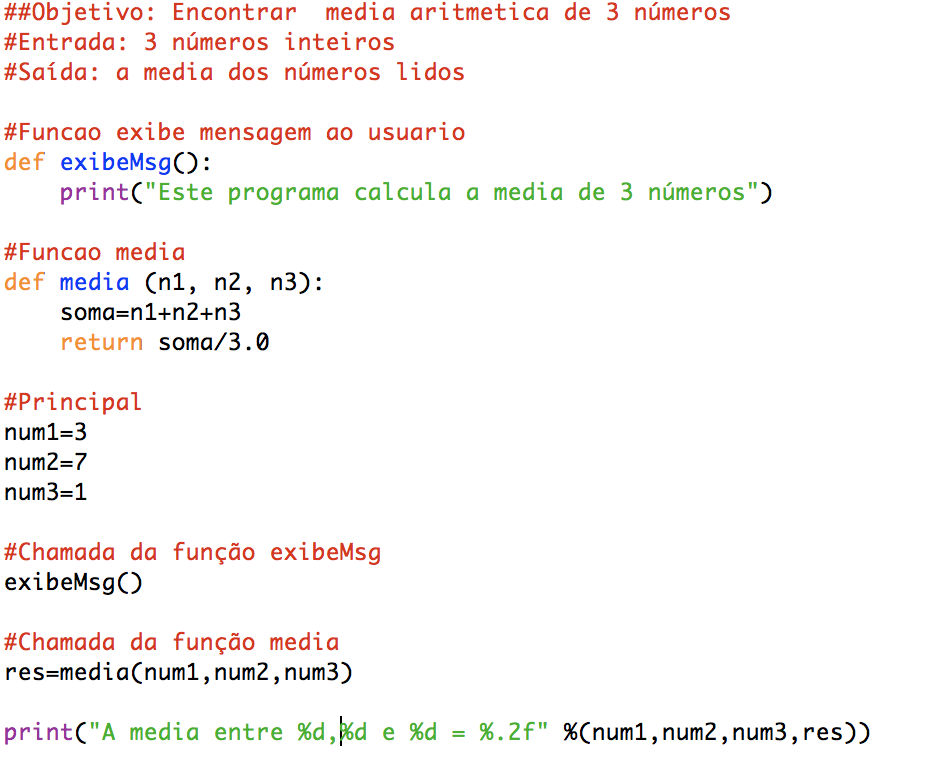


**Retorno de Função**

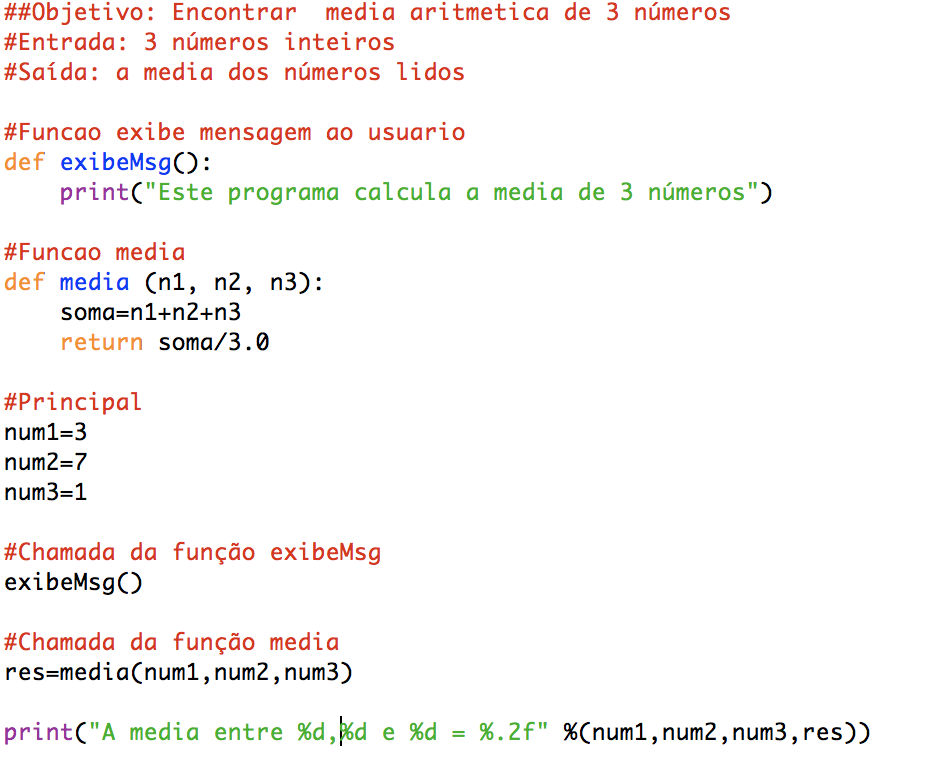
* Verificamos no exemplo do cálculo da média que, a chamada das funções media() e exibeMsgs() ficou diferente. Isso deve-se ao fato que, uma função pode ser classificada em:
  + Função que não retorna valor
  + Função que retorna valor
* Uma função que não retorna um valor, possui um efeito lateral ou 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.
* Na sua criação não há referência a instrução return



* Na sua chamada não há uma variável que receba seu retorno e nem podemos chamá-la dentro da função print().



* 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 função que a chamou.
* No exemplo, tem-se uma chamada da função média com 3 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 a variável res.



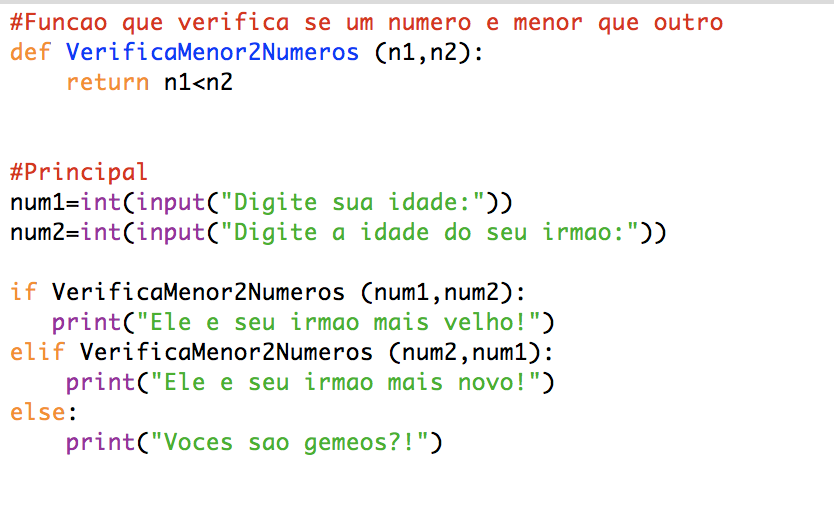
**EXERCÍCIO TUTORIADO 1**

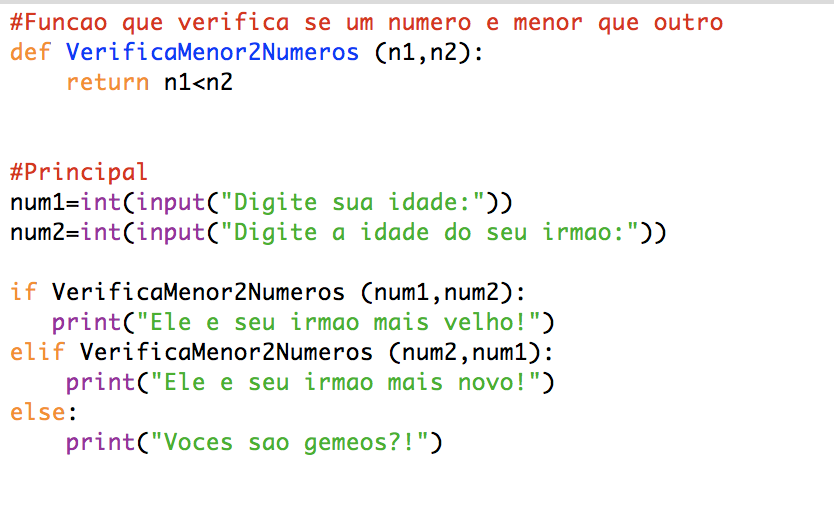
O que será exibido ao digitarmos as seguintes linhas de comando?

|  |  |
| --- | --- |
| (a)  >>> def sayHello():  print(“Hello”)  >>> sayHello() | (e)  >>> def buildHello(name):  return “Hello”+ name + “!  >>> greeting = buildHello(“Charles”)  >>> print(greeting) |
| (b)  >>> t=sayHello() | (f)  >>> buildHello(“Charles”) |
| (c)  >>> t |  |
| (d)  >>> t==None | (g)  >>>buildHello() |
|  |  |

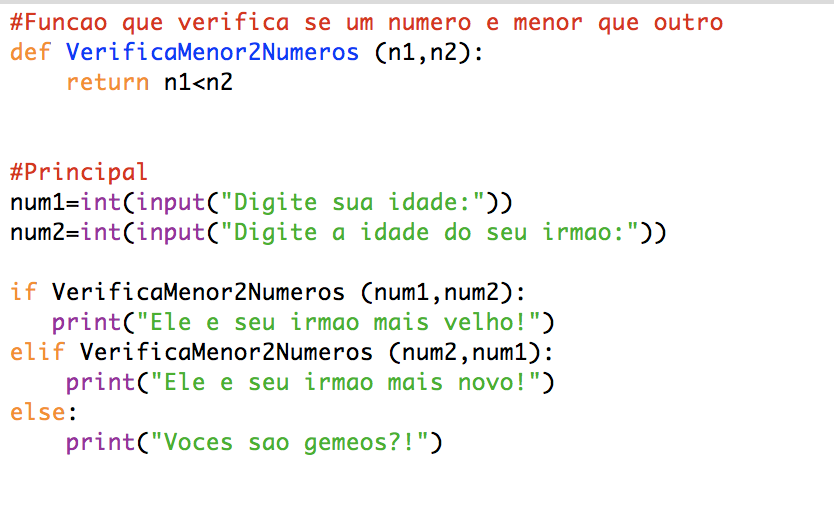
**Passagem de Parâmetros**

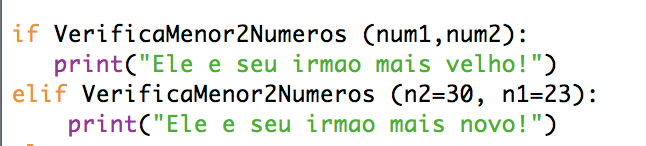
* A correspondência entre os argumentos atuais da chamada e os parâmetros formais da função é determinada pela ordem em que os argumentos são passados e não pelos seus nomes.
* Considere o exemplo seguinte. Os argumentos atuais num1 e num2 são copiados para os parâmetros formais n1 e n2, mas posso também passar o argumento num1 para o parâmetro n2. Nesse exemplo a função VerificaMenor2Numeros e 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.

****

****

* 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.

****

****

**EXERCÍCIO TUTORIADO 2**

O que será exibido ao digitarmos as seguintes linhas de comando?

|  |  |
| --- | --- |
| >>> def addup(first, last):  if first > last:  sum = -1  else:  sum=0  for i in range(first,last+1):  sum=sum+i  return sum | (a)  >>> print(addup(1,10))  (b)  >>> print(addup(first=1,last=10))  (c)  >>> print(addup(last=10, first=1)) |

* A linguagem Python permite ainda um **argumento default**. Se o argumento for omitido na chamada, ele assume um valor default no parâmetro da função.

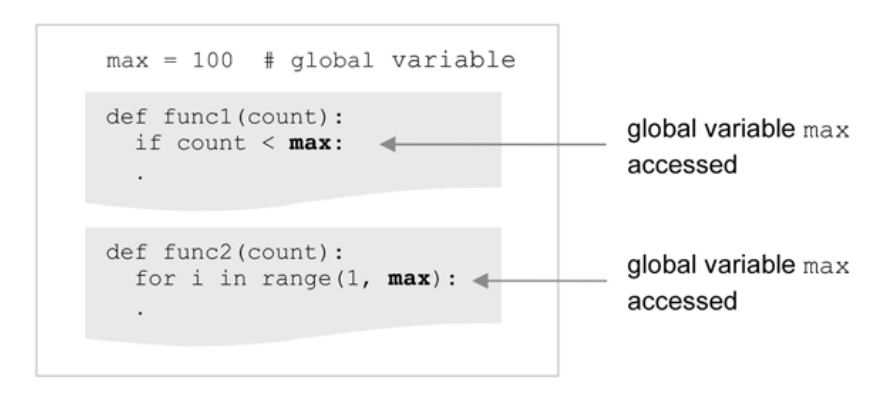
**EXERCÍCIO TUTORIADO 3**

O que será exibido ao digitarmos as seguintes linhas de comando?

|  |  |
| --- | --- |
| >>> def addup(first, last, incr=1):  if first > last:  sum = -1  else:  sum=0  for i in range(first,last+1, incr):  sum=sum+i  return sum | (a)  >>> print(addup(1,10))  (b)  >>> print(addup(1,10,2))  (c)  >>> print(addup(first=1,last=10))  (d)  >>> print(addup(incr=2, last=10, first=1)) |

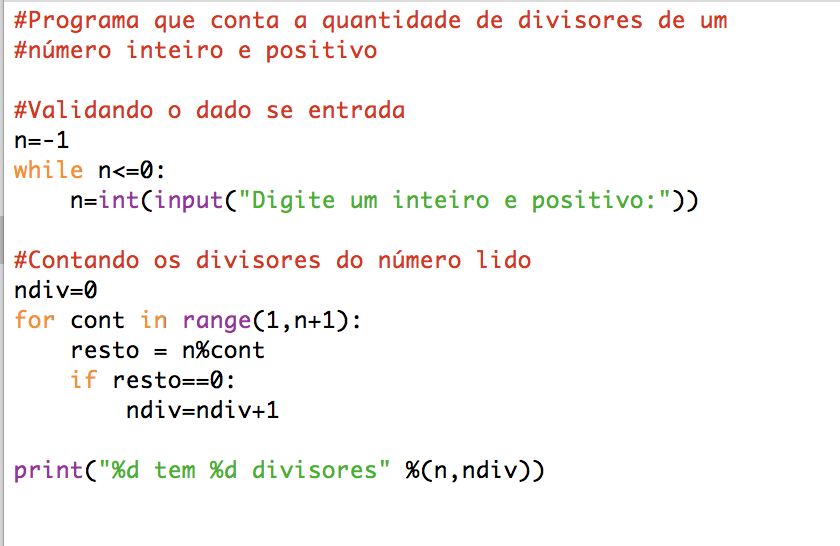
**Escopo de Variáveis**

* Uma variável é chamada de **variável local** se somente é acessível a partir de dentro de uma dada função. O seu ciclo de vida, ou seja, o período de tempo que a variável existirá, é igual a 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 disponível 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.

****

**EXERCÍCIO TUTORIADO 4**

Considere o programa que conta a quantidade de divisores de um número inteiro positivo. Reescreva o programa modularizado. Crie uma função para a entrada de dados e outra função para a contagem dos divisores de um número dado. Lembre-se de escrever a parte principal que faz a chamada das funções e exibe o resultado.



**EXERCÍCIOS DE LABORATÓRIO**

**1.** Escreva um programa que verifica se um número é primo, usando a lógica de contar divisores do exercício da aula, usando a estrutura de controle de fluxo mais adequada e as funções já criadas. A função **ePrimo** recebe como entrada um número inteiro e positivo n e retorna um booleano True ou False.

**2.** Vamos ampliar o problema de cálculo de números primos. Dados n números inteiros positivos, escreva um programa para calcular a **soma dos que são primos**. O seu programa deve usar necessariamente todas as funções criadas anteriormente. A função **somaPrimos** recebe como entrada a quantidade de números qtd e retorna um inteiro representando a soma dos primos.

**3.** Considere o problema de **conversão de temperatura**. Escreva um programa modularizado que permite ao usuário converter uma faixa de temperatura de Fahrenheit para Celsius (O usuário deve digitar F) e de Celsius para Fahrenheit (O usuário deve digitar C). Para a construção do programa você deve escrever as seguintes funções:

(a). exibeMsg() - apenas exibe uma msg para o usuário dizendo o que o programa faz e informando como deve ser a entrada de dados. Não tem parâmetro de entrada e não retorno.

(b). getConvertTo() - a função não tem parâmetro de entrada e retorna “F” ou “C”

(c). exibeFahrenheitTOCelsius(start, end) – essa função recebe como entrada o intervalo de temperatura a ser exibido, faz a conversão de temperatura e mostra a temperatura convertida para Celsius

(d). exibeCelsiusTOFahrenheit(start, end) – essa função recebe como entrada o intervalo de temperatura a ser exibido, faz a conversão de temperatura e mostra a temperatura convertida para Fahrenheit

**EXERCÍCIOS EXTRAS**

1. Uma chamada de função pode ser feita em qualquer lugar dentro de um programa em que o tipo do retorno seja apropriado. (TRUE ou FALSE)
2. Uma expressão pode conter mais do que uma chamada de função. (TRUE ou FALSE)
3. Chamadas de funções podem conter argumentos que sejam chamadas de função. (TRUE ou FALSE)
4. Toda função que retorna um valor deve conter pelo menos um parâmetro. (TRUE ou FALSE)
5. Uma variável global em Python é uma variável que é definida fora de qualquer definição de função. (TRUE ou FALSE)
6. Para a definição de função seguinte e uma chamada de função associada, liste todos os parâmetros formais e todos os argumentos atuais.

**def somefunction (n1, n2):**

**.**

**.**

**#main**

**num1=10**

**somefunction(num1, 15)**

1. Para a definição de função seguinte, indique se cada chamada de função é apropriada ou não. Se inapropriada, explique porque.

**#Função que calcula o maior divisor comum de n1 e n2, com o requisito que n1 <= n2, e n1 e n2 #inteiros**

**def gdc (n1, n2):**

**.**

**.**

|  |  |
| --- | --- |
| (a)  a=10  b=20  result=gcd(a,b) | (d)  a=10  b=20  c=30  result=gcd(gcd(a,b),c) |
| (b)  a=10.0  b=20  result=gcd(a,b) | (e)  a=10  b=20  c=30  print(gcd(a, gcd(c,b))) |
| (c)  a=10  b=20  result=gcd(b,a) |  |

1. Faça um programa modularizado que calcule e apresente o **fatorial** de um número inteiro e natural fornecido pelo usuário.

Exemplo: 5! = 5 x 4 x 3 x 2 x 1=120.

Por definição 0! = 1.

O seu programa deve contar uma função que lê um número inteiro e positivo, bem como uma função, chamada fatorial, que recebe um inteiro e positivo n e retorna um inteiro e positivo, representando a soma dos dígitos do número dado.

1. Dado um número inteiro positivo, escreva um programa modularizado para calcular a **soma de seus dígitos**. O seu programa deve contar uma função que lê um número inteiro e positivo, bem como uma função, chamada somaDigitos, que recebe um inteiro e positivo n e retorna um inteiro e positivo, representando a soma dos dígitos do número dado.
2. A **série de Fibonacci** é formada pela sequência 1, 1, 2, 3, 5, 8, 13, 21, 34, ... Escreva um programa modularizado que apresente a série de Fibonacci até o *n-ésimo* termo (n > 0). O seu programa deve contar uma função que lê um número inteiro e positivo, bem como uma função, chamada Fibonacci, que recebe um inteiro e positivo n e retorna um inteiro e positivo, representando *n-ésimo* termo.
3. Escreva um programa modularizado que calcule o **MMC (mínimo múltiplo comum)** entre dois números naturais. O seu programa deve contar uma função que lê um número inteiro e positivo, bem como uma função, chamada MMC, para calcular o MMC entre dois números inteiros e positivos e retornar o MMC entre eles.
4. Escreva um algoritmo que calcule o **MDC (máximo divisor comum)** entre dois números naturais. O seu programa deve contar uma função que lê um número inteiro e positivo, bem como uma função, chamada MDC, para calcular o MDC entre dois números inteiros e positivos e retornar o MDC entre eles.