

Funções



UNINASSAU

Prof^ª. Esp. Sônia Gomes de Oliveira

PAULISTA - 2024

O que são Funções em Python?

Funções nada mais é do que blocos de códigos que realiza uma tarefa específica. Geralmente são utilizadas para evitar que determinada parte do código seja repetida várias vezes.

Por exemplo:

Professor 1: Realizou a soma das notas dos alunos e calculou a média total, de 20 alunos. Ele utilizou uma função para fazer isso.

Professor 2: Tinha 30 alunos e pegou a função do **professor 1** para realizar o processo com os alunos dele também.



Função - Como Declarar

A função sempre se inicia com a primeira letra minúscula.

Para definir uma função deve utilizar a sintaxe: **def**

Função

Nome da Função

Parâmetros

```
def mostrarPrint ():  
    print('O print está dentro de uma função')
```

```
def mostrarPrint ():  
    return 'Olá'
```

Como chamar ?

Se dentro da função estiver com o comando de saída **print**, para chamar a função, declara só o nome dela.

```
mostrarPrint()
```

Como chamar ?

Caso tenha o comando **return** deve chamar a função com o print.

```
print(mostrarPrint())
```

Função - Como Declarar

Vamos criar uma função que imprima a soma de dois números.

```
def soma_numeros():  
    num1 = float(input("Primeiro numero: "))  
    num2= float(input("Segundo numero: "))  
    soma = num1+ num2  
    print('Soma:', soma)  
  
soma_numeros()
```

Parâmetros nas Funções

As funções podem receber e retornar dados. Podemos enviar dados para uma função através de seus parâmetros.

Como definir uma função com parâmetros

```
def recebe_nome(nome):  
    print(nome)  
  
nome = input('Digite seu nome: ')  
recebe_nome(nome)
```

Tenho que especificar o nome, porque minha função pede um parâmetro de nome.

Parâmetros nas Funções

Como tratar se caso não digitar um parâmetro em uma função que pede parâmetro?

Iremos definir um parâmetro padrão

```
def recebe_nome(nome='oi'):  
    print(nome)
```

```
recebe_nome()
```

Esqueceu o
parâmetro

console

oi

Funções com Retorno de Dados

As funções também podem retornar valores através da palavra reservada **return**.

Como definir uma função com retorno

```
def recebe_num(numero1, numero2):  
    soma = numero1 + numero2  
    return soma
```

```
num1 = int(input('Digite n1: '))  
num2 = int(input('Digite n2: '))  
print(recebe_num(num1,num2))
```


Palavra reservada Pass

Caso você deseje definir uma função sem dados nenhum, ou seja, sem código, saiba que isso irá disparar um erro de indentação, porque as funções não podem estar vazias.

Para resolver isso, basta utilizar a palavra reservada **pass** ou **três pontos**.

```
def sem_nada():  
    pass
```

```
def preencho_depois():  
    ...
```


Função com apenas uma linha

A linguagem Python permite a criação de funções com apenas uma linha de código.

```
def soma(n1,n2): return n1+n2
def subtracao(n1,n2): return n1-n2

valor1 = int(input('Numero 1: '))
valor2 = int(input('Numero 2: '))
print(soma(valor1,valor2))
print(subtracao(valor1,valor2))
```



Variáveis Locais e Globais

O que são variáveis Locais??

São variáveis que só são visíveis dentro da função.

Quem está fora da função não sabe quais variáveis existe dentro dela.

```
def funcao():  
    variavel_local = 'Olá'  
    print(variavel_local)  
  
funcao()
```

O que são variáveis Globais?

São variáveis universais que pode ser usada em todas as funções, ela deve ser definida **fora** das funções.

```
variavel_Global= ''

def funcao1():
    variavel_Global = 'oi'
    print(variavel_Global)

def funcao2():
    variavel_Global = 'Outro dado'
    print(variavel_Global)

funcao1() #chamando a função
funcao2()
```

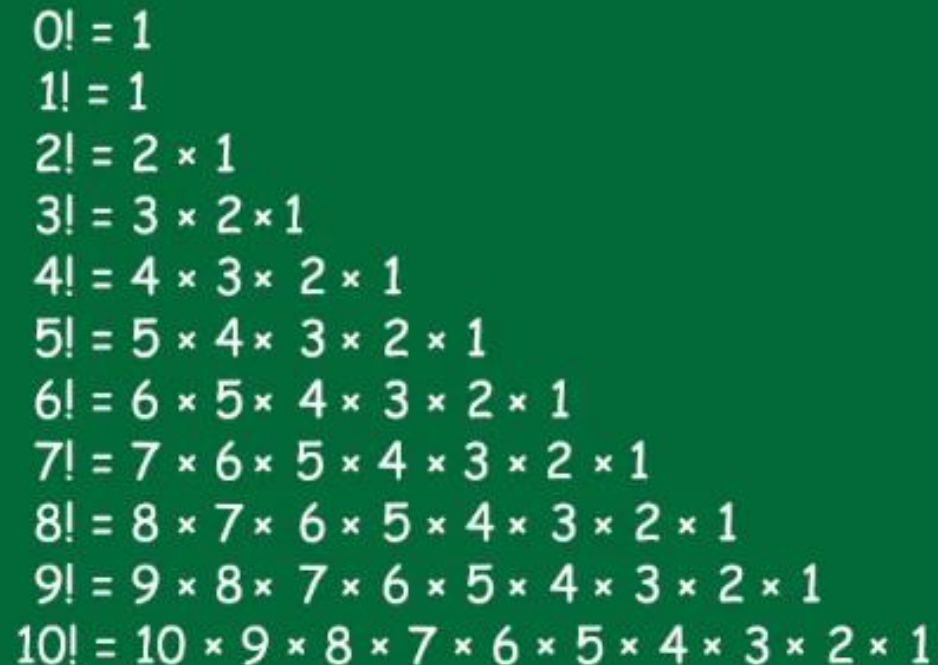


Funções Recursivas

Funções Recursivas

Funções recursivas, são funções que precisam chamar elas mesmas dentro de um algoritmo para resolver um determinado problema.

Exemplo: Calcular um fatorial de um número



0! = 1
1! = 1
2! = 2 × 1
3! = 3 × 2 × 1
4! = 4 × 3 × 2 × 1
5! = 5 × 4 × 3 × 2 × 1
6! = 6 × 5 × 4 × 3 × 2 × 1
7! = 7 × 6 × 5 × 4 × 3 × 2 × 1
8! = 8 × 7 × 6 × 5 × 4 × 3 × 2 × 1
9! = 9 × 8 × 7 × 6 × 5 × 4 × 3 × 2 × 1
10! = 10 × 9 × 8 × 7 × 6 × 5 × 4 × 3 × 2 × 1

O **fatorial (!)** de um número **n**, representado por **n!**, é a **multiplicação de n por seus antecessores maiores ou iguais a 1**. Essa operação é muito comum em análise combinatória.

Funções Recursivas

Exemplo: Calcular um fatorial de um número

```
✓ def calcular_fatorial(valor):  
    if valor == 0:  
        return 1  
    else:  
        return valor * calcular_fatorial(valor-1)  
  
✓ while True:  
    valor = int(input('Informe o fatorial: '))  
    if valor < 0:  
        print('Não é aceito valores negativos')  
    else:  
        print(calcular_fatorial(valor))
```

Console

```
Informe o fatorial: -1  
Não é aceito valores negativos  
Informe o fatorial: 0  
1  
Informe o fatorial: 3  
6  
Informe o fatorial: 4  
24
```

O Fatorial não aceita valores negativos, dar erro de recursão, por isso devemos fazer um tratamento de erro.

Função Recursiva Mais detalhada

```
def recursao(valor):  
    while valor == 0:  
        return 1  
    print(f'{valor}!', end=' ')  
    if valor > 1:  
        print('x', end=' ')  
    else:  
        print('==>', end=' ')  
    return valor * recursao(valor-1)
```

```
while True:  
    valor=int(input('Informe o valor: '))  
    while valor < 0:  
        print('Não permite números negativos')  
        valor = int(input('Informe o valor: '))  
    print(recursao(valor))
```

Console

```
Informe o valor: -1  
Não permite números negativos  
Informe o valor: 1  
1! ==> 1  
Informe o valor: 3  
3! x 2! x 1! ==> 6  
Informe o valor: 4  
4! x 3! x 2! x 1! ==> 24
```

Outra forma de resolução sem o uso de funções

```
fatorial= int(input('Insira um valor positivo: '))
valor = fatorial
recursao=1
print('calculando {}!'.format(valor), end='')
while valor <0:
    print('Não será aceito numeros negativos')
    fatorial = int(input('Insira um valor positivo: '))
    valor = fatorial
else:
    while valor > 0:
        print(f'{valor}', end='')
        if valor > 1:
            print('x', end='')
        else:
            print('=', end='')
        recursao = recursao * valor
        valor = valor - 1
    print(recursao)
```

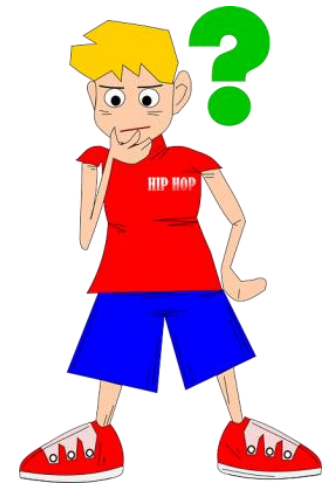


Números Aleatórios

Por que aprender sobre números aleatórios?

A vantagem dos números aleatórios são enormes, principalmente em jogos... Por exemplo, retornar um personagem aleatório ou um inimigo, girar uma roleta de sorteio e etc.

Já passou alguma vez de esquecer a senha e ter que fazer uma recuperação, através de um código numérico ?



Àquele código são números aleatórios 😊

Como gerar um número inteiro aleatório?

Temos primeiro que importar uma biblioteca chamada de `random`. Em seguida utilizaremos a função `randint`.

A função `randint` tem dois parâmetros (início, fim)

```
import random  
  
print(random.randint(1,10))
```

Retorna os números inteiros entre 1 a 10

E números flutuantes é possível retornar?

Sim... Através da função `uniform()`.

```
import random  
  
print(random.uniform(2.1, 10.1))
```

Como funciona o randrange()?

Parecido com o range(), a diferença é que o range retorna todos os valores entre os intervalos, enquanto o randrange retorna apenas 1.

```
import random  
  
print(random.randrange(1,10))  
print(random.randrange(10))
```

Retorna os números entre 1 até 9

Retorna os números entre 0 até 9



Como sortear nomes aleatórios?

Deve importar a biblioteca **random** e utilizar o comando **choice**.

```
import random
a1 = str(input('Primeiro aluno: '))
a2 = str(input('Segundo aluno: '))
a3 = str(input('Terceiro aluno: '))
a4 = str(input('Quarto aluno: '))

print(f'Sorteio: {random.choice([a1,a2,a3,a4])}')
```

Como o choice só aceita um elemento por vez, utiliza-se uma lista para sortear todos os nomes.

Como embaralhar nomes?

Deve importar a biblioteca **random** e utilizar o comando **sample(palavra, numero_caracteres)**

```
from random import sample

1 usage
def embaralhar(nome):
    frase = sample(nome, len(nome))
    for item in frase:
        print(item, end='')

nome = input('Digite algo: ')
embaralhar(nome)
```

Se não utilizar o laço de repetição, irá ser mostrado em uma lista

Digite algo: Ana Paula

Pn aaluAa

Embaralhando vários nomes e mostrando na tela

```
lista = []  
1 usage  
def guardar_dados():  
    for i in range(2):  
        nome = input('Digite algo: ')  
        frase = sample(nome, len(nome))  
        #print(f'Frase: {frase}')        juncao = ''.join(frase)  
        #print(f'Juncao: {juncao}')        lista.append(juncao)  
    print(f'Adicionado na lista: {lista}')    for item in lista:  
        print(f'Nomes embaralhados: {item}')
```

```
guardar_dados()
```

Desordenando os caracteres
e indexando em uma lista

Juntando os caracteres da lista
(onde tem espaçamentos, retirar)

Adicionando os caracteres dentro
de outra lista

Mostrando os itens separados da
lista

Resultado

```
lista = []
1 usage
def guardar_dados():
    for i in range(2):
        nome = input('Digite algo: ')
        frase = sample(nome, len(nome))
        #print(f'Frase: {frase}')
        juncao = ''.join(frase)
        #print(f'Juncao: {juncao}')
        lista.append(juncao)
    print(f'Adicionado na lista: {lista}')
    for item in lista:
        print(f'Nomes embaralhados: {item}')

guardar_dados()
```

```
Digite algo: maria
Frase: ['a', 'a', 'r', 'i', 'm']
Juncao: aarim
Digite algo: joao
Frase: ['j', 'a', 'o', 'o']
Juncao: jaoo
Adicionado na lista: ['aarim', 'jaoo']
Nomes embaralhados: aarim
Nomes embaralhados: jaoo
```

Mostrando o resultado em outra função

```
from random import sample
```

```
lista = []
```

```
1 usage
```

```
def mostrar_dados():  
    print(lista)
```

```
1 usage
```

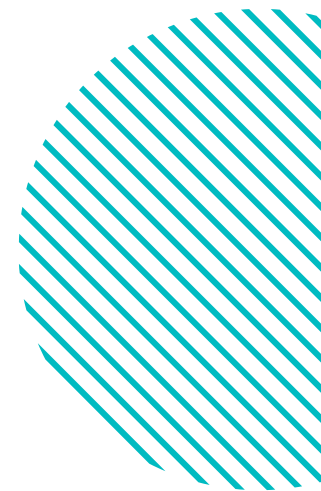

```
def guardar_dados():  
    for i in range(2):  
        nome = input('Digite algo: ')  
        frase = sample(nome, len(nome))  
        juncao = ''.join(frase)  
        lista.append(juncao)
```

```
guardar_dados()
```

```
mostrar_dados()
```

```
Digite algo: paulo  
Digite algo: carlos  
['oluap', 'ocalrs']
```

Lembrando que só é possível imprimir a lista em outra função se ela for uma variável global.

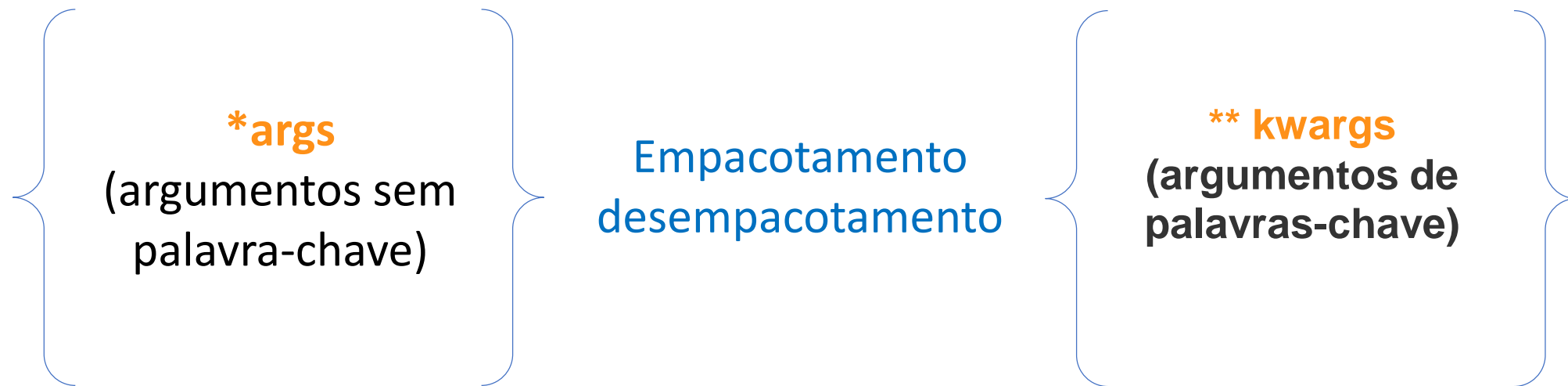


Empacotamento e desempacotamento

Conhecendo os tipos de argumentos

No Python podemos passar inúmeros argumentos para uma função, usando símbolos especiais. Existe dois tipos:

Símbolos especiais usados para passar argumentos



Args - Argumentos não nomeados

O args tem como característica receber valores indefinidos de parâmetros. Sua nomenclatura é composta por: ***args**

```
def soma(*args):  
    print(args)  
    soma = 0  
    for numero in args:  
        soma = soma + numero  
    print(soma)  
  
soma(10,10,10,10,10)
```

(10, 10, 10, 10, 10)
50

Retorna em
formato de
Tupla

```
def somar(*args):  
    print(args)  
    soma = 0  
    for i in range(len(args)):  
        soma = soma + args[i]  
    print(soma)  
  
somar(10,10,2)
```

Desempacotamento de listas

Desempacotamento é o nome dado ao comando que desmembra valores de lista para repassar para os parâmetros da função. Exemplo:

Dado uma função que recebe 4 argumentos:

```
def items(w,x,y,z):  
    print(w,x,y,z)
```

Função com parâmetros

```
lista=[2,3,4,5]  
items(lista)
```

Lista sendo chamada pela função items

Só conseguiu pegar o primeiro elemento. Para resolver isso vamos desempacotar a lista

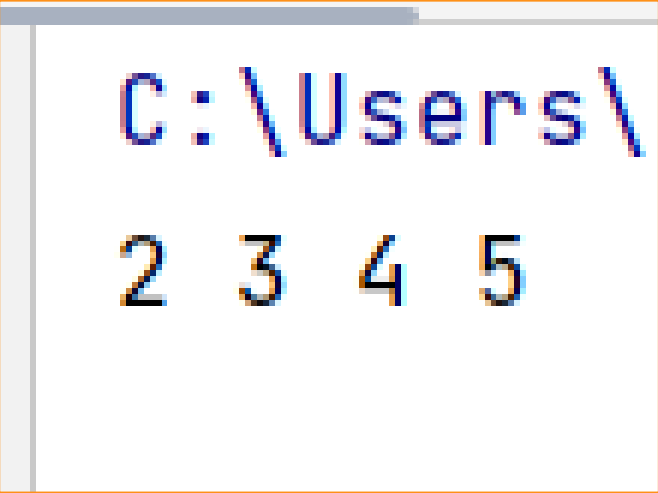
```
TypeError: items() missing 3 required positional arguments: 'x', 'y', and 'z'
```

Desempacotamento de listas

Para desempacotar uma lista utilizamos o **operador *** e o nome do elemento que irá ser desempacotado.

```
def items(w,x,y,z):  
    print(w,x,y,z)
```

```
lista=[2,3,4,5]  
items(*lista)
```



```
C:\Users\  
2 3 4 5
```



Desempacotamento de dicionários

Desempacotamento de dicionários

Para desempacotar um dicionário utilizamos o **operador **** e o nome do elemento que irá ser desempacotado.

```
pessoa = {  
    'nome': 'ana',  
    'sobrenome': 'maria'  
}  
  
continuacao = {  
    'idade': 23,  
    'altura': 1.56  
}  
  
print(pessoa, continuacao)  
dados_completos = {**pessoa, **continuacao}  
print(dados_completos)
```

```
{'nome': 'ana', 'sobrenome': 'maria'} {'idade': 23, 'altura': 1.56}
```

```
{'nome': 'ana', 'sobrenome': 'maria', 'idade': 23, 'altura': 1.56}
```

Kwargs – argumentos nomeados

```
def formulario(**args):  
    print(args)  
  
nome = input('Nome: ')  
idade = int(input('Idade: '))  
formulario(nome_usuario = nome, idade_usuario = idade, peso_usuario = 54.90)
```

chave

valor

Nome: *Julia*

Idade: *45*

`{'nome_usuario': 'Julia', 'idade_usuario': 45, 'peso_usuario': 54.9}`



Módulos em Python

Módulos em Python

Módulos nada mais é do que arquivos que contêm códigos prontos para o uso. Exemplo de Módulos:

Python Module Index

_ a b c d e f g h i j k l m n o p q r s t u v w x z	
time	<i>Time access and conversions.</i>
timeit	<i>Measure the execution time of small code snippets.</i>
tkinter	<i>Interface to Tcl/Tk for graphical user interfaces</i>
token	<i>Constants representing terminal nodes of the parse tree.</i>
tokenize	<i>Lexical scanner for Python source code.</i>
tomllib	<i>Parse TOML files.</i>
trace	<i>Trace or track Python statement execution.</i>
traceback	<i>Print or retrieve a stack traceback.</i>
tracemalloc	<i>Trace memory allocations.</i>
tty (Unix)	<i>Utility functions that perform common terminal control operations.</i>
turtle	<i>An educational framework for simple graphics applications</i>
turtledemo	<i>A viewer for example turtle scripts</i>
types	<i>Names for built-in types.</i>
typing	<i>Support for type hints (see :pep:`484`).</i>

Módulos – Maneiras de importação

Você pode importar os módulos de várias maneiras:

- **Em partes** (só algumas funções específicas): **from** nome_do_módulo(arquivo) **import** objeto1, objeto2.

```
from random import randrange  
print(randrange(1,10)) #retorna números aleatórios
```

- **Por renomeação de arquivo:** **import** nome_do_modulo **as** apelido

```
import random as r  
print(r.randrange(5,10))
```

Módulos – Maneiras de importação

Você pode importar os módulos de várias maneiras:

- **Por nomeação de objeto:** `from` nome_do_modulo `import` objeto `as` apelido

```
from random import randrange as aleatorio  
print(aleatorio(1,10))
```

- **Importação completa:** `from` nome_do_modulo `import *`

```
from random import *  
  
print(randint(1,10))#aqui retorna de 1 a 10  
print(randrange(1,10))#aqui retorna de 1 a 9
```

Criando um módulo

Vamos criar um módulo (arquivo) chamado de calculadora.py

calculadora.py

```
def soma(n1, n2):  
    return n1 + n2  
  
def subtracao(n1, n2):  
    return n1 - n2  
  
def multiplicacao(n1, n2):  
    return n1 * n2  
  
def divisao(n1, n2):  
    return n1 / n2
```

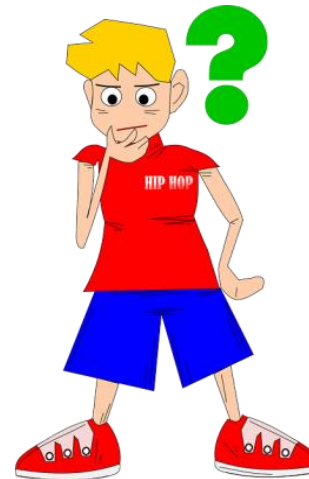
Agora crie um novo **arquivo.py** em branco com o nome da sua preferência.

Criando um módulo

Para você ter acesso ao módulo anterior (calculadora) deverá utilizar o comando **import** logo no início do seu arquivo

import calculadora #prontinho o arquivo já foi importado para seu novo arquivo

Mas, e agora como usar?



Vamos criar um código para utilizar as funções do módulo anterior

```
import calculadora

print("1. Soma\n2. Subtração\n3. Multiplicação\n4. Divisão")
opcao=int(input("Que operação deseja realizar: "))

n1=float(input("Primeiro numero: "))
n2=float(input("Segundo numero: "))

if opcao==1:
    print("Soma:", calculadora.soma(n1,n2))
elif opcao==2:
    print("Subtração:", calculadora.subtracao(n1,n2))
elif opcao==3:
    print("Multiplicação:", calculadora.multiplicacao(n1,n2))
elif opcao==4:
    print("Divisão:", calculadora.divisao(n1,n2))
else:
    print('Opção inválida,tente novamente')
```

calculadora.py

```
def soma(n1, n2):
    return n1 + n2

def subtracao(n1, n2):
    return n1 - n2

def multiplicacao(n1, n2):
    return n1 * n2

def divisao(n1, n2):
    return n1 / n2
```

Vamos criar um código para utilizar as funções do módulo anterior

```
from calculadora import soma

print(soma(1,2))
print(subtracao(1,2))
print(multiplicacao(1,2))
print(divisao(1,2))
```

Neste caso só reconhecerá a função soma, porque só ela foi importada.

calculadora.py

```
def soma(n1, n2):
    return n1 + n2

def subtracao(n1, n2):
    return n1 - n2

def multiplicacao(n1, n2):
    return n1 * n2

def divisao(n1, n2):
    return n1 / n2
```

REFERÊNCIAS

LOPES.E. **Funções em python**. Disponível em: < <https://pythonacademy.com.br/blog/funcoes-em-python>>.

PYTHON PROGRESSIVO. **Funções**. Disponível em:
<<https://www.pythonprogressivo.net/2018/06/Funcao-O-que-e-Para-que-serve-Onde-sao-usadas-funcoes-em-Python.html>>.