VOLTAR

Funções personalizadas

Criar e desenvolver funções ou métodos personalizados em Python.

NESTE TÓPICO

Marcar tópico

Olá alunos,

Como já vimos, a segunda geração de linguagens de programação: Pascal, C, por exemplo, surgem com um recurso principal, que não existiam nas linguagens anteriores, a possibilidade de criar funções, que em C, são conhecidas como functions e em Pascal, conhecidas como procedures.

Podíamos, então, encapsular instruções em funções, particionando o desenvolvimento do código. Estes recursos acabaram sendo um dos principais pilares das próximas linguagens de programação que surgiram como: C++, Java, C#., as linguagens orientadas a objetos. Tudo na orientação a objetos é encapsulado, blocado.

Vimos que os comandos de decisão: if, else, os comandos de repetição: for, while, são blocos também.

Então, em uma visão, do micro para o macro, podemos dizer que: um bloco if ou for ou while estão em um outro bloco que é uma função ou método e este método está num bloco maior conhecido como classe, estas classes podem estar relacionadas e este conjunto de classes relacionados formam um sistema, ou seja, um sistema de informação de uma empresa pode ser composto por um conjunto de classes relacionados, cada classe, que é um bloco, pode ser composto de uma ou mais funções, cada função, que é um bloco, pode ter vários comandos de decisão ou de repetição, que também são blocos.

Portanto, funções ou métodos servem para realizar funcionalidades, ações do sistema. Em Python podemos ter funções globais, locais ou lambda.

Python fornece uma vasta biblioteca de funções nativas, já implantadas, daí teremos que utilizar os nomes que a linguagem já criou, como por exemplo: print( ), input( ), pow( ), search( ), etc. mas, podemos **personalizar** as **funções**, criar a nossa própria função, "batizando-a" com um nome.

Como toda função em matemática é acompanhada por um par de parêntesis, por exemplo: **f(x)**, onde posso passar um parâmetro, as funções em linguagens de programação também são acompanhadas de parêntesis, onde podemos ou não (é opcional) passar parâmetros ou argumentos.

CRIANDO FUNÇÕES EM PYTHON

Para criarmos as funções em Python, é necessário utilizar a cláusula **def** e a sintaxe básica é:

def nome_funcao (parâmetros):

.....instruções

Em Python, toda função retorna um valor, por isso podemos utilizar a cláusula **return**, apesar de ser opcional a utilização de um valor para **return**, se não passarmos nenhum valor, o interpretador retornará **none.**

EXEMPLO 1:

```
1.  >>> import math
2.  >>> def raizQuadrada():
3.          return math.sqrt(81)
4.
5.  >>> raizQuadrada()
6.  9.0
```

Neste exemplo, utilizamos o interpretador interativo para testar, criamos a função **def** denominando com o nome **raizQuadrada**, note que para criarmos nomes para funções, não utilize acentuação gráfica e nem "ç". Não passamos nenhum parâmetro, por isso o parêntesis após o nome da função está vazio e não esqueça de colocar os dois pontos **":".**

Agora, para utilizarmos a função, basta chama-la pelo nome acompanhado pelo parêntesis (mesmo que vazio), como fizemos na linha **5**.

EXEMPLO 2:

```
1.  >>> def raizQuadrada(numero):
2.          return math.sqrt(numero)
3.
4.  >>> raizQuadrada(81)
5.  9.0
```

No exemplo 2, criamos a função, com o parâmetro **numero**, que neste caso, comporta-se como uma variável e na hora de executarmos a função na linha **4**, além do nome da função, passamos o valor **81** para o parâmetro entre

parêntesis, este valor é atribuído como **argumento** de entrada do parâmetro **numero** e é conhecido como passagem por valor.

EXEMPLOS 3 e 4:

```
1.  >>> def raizQuadrada(num1, num2):
2.          num1 = math.sqrt(num1)
3.          num2 = math.sqrt(num2)
4.          return num1,num2
5.
6.  >>> raizQuadrada(81,9)
7.  (9.0, 3.0)
```

```
1.  >>> def raizQuadrada(num1, num2):
2.          num1 = math.sqrt(num1)
3.          num2 = math.sqrt(num2)
4.          print (num1," , ",num2)
5.
6.
7.  >>> raizQuadrada(81,9)
8.  9.0  ,  3.0
```

Nos exemplos acima, utilizamos dois parâmetros de entrada e na hora de chamarmos a função, passamos dois valores.

No exemplo **4**, não utilizamos o comando **return**, trocamos na linha **4,** pelo comando **print.**

EXEMPLO 5:

```
1.  def mensagem():
2.          print("A raiz quadrada é: ")
3.
4.  def raizQuadrada(num):
5.          num = math.sqrt(num)
6.          mensagem()
7.          print(num)
8.
9.  >>> raizQuadrada (81)
10. A raiz quadrada é:
11. 9.0
```

No EXEMPLO **5**, criamos a função **mensagem** e a função **raizQuadrada**. Dentro da função **raizQuadrada** chamamos a função **mensagem** (linha **6**) e ao executarmos a função **raizQuadrada**, a função **mensagem( )** também será executada, mostrando a saída nas linhas **10** e **11**.

A FUNÇÃO LAMBDA

A função **lambda** na realidade não é uma função e sim uma expressão ou também conhecida como função anônima (sem nome). Dai, já percebemos a diferença para uma função que é denominada, nós não podemos chama-la pelo nome (já que não tem!).

A sintaxe é:

lambda parâmetros: expressão

Lembrando que os parâmetros são opcionais e não se esqueça dos dois pontos **":"**.

EXEMPLO 6:

```
1.  >>> num = lambda: math.sqrt(81)
2.  >>> num()
3.  9.0
```

No exemplo **6**, criamos a variável **num** e passamos a função lambda com a expressão matemática da raiz quadrada de 81. A variável **num** (linha **2**), neste caso, é acompanhada de parêntesis para executar a função **lambda**.

SAIBA MAIS...

Dê uma olhada nos links abaixo para saber mais sobre a linguagem Python:

https://www.python.org/doc/ (https://www.python.org/doc/)

https://wiki.python.org/moin/PythonBooks (https://wiki.python.org/moin/PythonBooks)

Neste tópico vimos como criar funções denominadas, um recurso muito utilizado nas linguagens procedurais e orientadas a objetos. Vimos também como criar a função anônima **lambda**.

Quiz

Exercício Final

Funções personalizadas

INICIAR ❯

Referências

SUMMERFIELD, M. *Programação em Python 3*: Uma introdução completa à linguagem Python. Rio de Janeiro Alta Books, 2012. 495 p.

MENEZES, N. N. C. *Introdução à programação com Python:* algoritmos e lógica de programação para iniciantes. 2. ed. São Paulo: Novatec, 2014. 328 p.

SWEIGART, AL. *Automatize tarefas maçantes com Python:* programação prática para verdadeiros iniciantes. São Paulo: Novatec, 2015. 568 p.

PYTHON, doc. Disponível em: <https://www.python.org/doc/>. Acesso em: Junho/2018.

PYTHON, books. Disponível em: <https://wiki.python.org/moin/PythonBooks>. Acesso em: Junho/2018.

Avalie este tópico

ANTERIOR
Expressões regulares em Python

Índice

PRÓXIMO
Funções de preenchimento

Biblioteca (https://www.uninove.br/conheca-a-uninove/biblioteca/sobre-a-biblioteca/apresentacao/)
Portal Uninove (http://www.uninove.br)
Mapa do Site

Ajuda? (https://ava.uninove.br/seu/AVA/topico/topico.php?idCurso=)