Python – Funções

Introdução à Programação

SI2

Exercício

- Fazer uma função que receba como parametro um numero inteiro e retorne o fatorial desse numero (não usar recursividade).
- Defina uma função que pesquisa um valor em uma lista. Se o valor for encontrado, devemos retornar a posição deste na lista. Caso não seja encontrado, devemos retornar None
- Calcule a média de valores em uma lista

Média

```
def media(L):
  total = 0
  for e in L:
    total+=e
  return total/len(L)
```

Média

```
def soma(L):
  total = 0
  for e in L:
    total+=e
  return total

def media(L):
  return(soma(L)/len(L))
```

Pesquisa

```
def pesquise(L,valor):
  for x,e in enumerate(L):
    if e == valor:
      return x
  return None

L = [10,20,25,30]
print(pesquise(L,25))
print(pesquise(L,27))
```

Fatorial

```
def fatorial(n):
  fat = 1
  while n>1:
    fat*=n
    n-=1
  return fat
```

Fatorial

```
def fatorial(n):
  fat = 1
  x=1
  while x<=n:
    fat*=x
    x+=1
  return fat
```

Escopo

- O escopo de uma variável é o alcance que ela tem, de onde ela pode ser acessada.
- Variáveis Globais
- Variáveis Locais

Escopo

```
#Escopo Global
x = 99  #x e func : global
def func(y):
    #Escopo local
    z = x + y      #x é uma variável global
    return z

func(1)      #resultado = 100
```

Escopo

```
X = 'Spam'
def func():
    X = 'NI'
    print(X)

func()
print(X)
```

Escopo

```
X = 'Spam'
def func():
    X = 'NI'
    print(X)

func()
print(X)
```

Saídas:

NI
Spam

Escopo

- Palavras Reservadas
  - **global –** permite que a variável local assim definida altere o conteúdo da variável global.
  - **nonlocal –** permite que a variável local tenha escopo um nível acima.

Escopo

```
X = 'Spam'
def func():
    global X
    X = 'NI'
    print(X)

func()
print(X)
```

Escopo

```
X = 'Spam'
def func():
    global X
    X = 'NI'
    print(X)

func()
print(X)
```

Saídas:

NI
NI

Escopo

```
X = 'Spam'
def func():
    X = 'NI'
    def nested():
        print(X)
    nested()

func()
print(X)
```

Escopo

```
X = 'Spam'
def func():
    X = 'NI'
    def nested():
        print(X)
    nested()

func()
print(X)
```

Saídas:

NI
Spam

Escopo

```
X="teste"
def func():
    X = 'NI'
    def nested():
        X = 'Spam'
        print(X)
    nested()
    print(X)

func()
print (X)
```

Escopo

```
X="teste"
def func():
    X = 'NI'
    def nested():
        X = 'Spam'
        print(X)
    nested()
    print(X)

func()
print (X)
```

Saídas:

Spam
NI
teste

Escopo

```
X="teste"
def func():
    X = 'NI'
    def nested():
        nonlocal X
        X = 'Spam'
        print(X)
    nested()
    print(X)

func()
print (X)
```

Escopo

```
X="teste"
def func():
    X = 'NI'
    def nested():
        nonlocal X
        X = 'Spam'
        print(X)
    nested()
    print(X)

func()
print (X)
```

Saídas:

Spam
Spam
teste

Escopo

```
X="teste"
def func():
    X = 'NI'
    def nested():
        global X
        X = 'Spam'
        print(X)
    nested()
    print(X)

func()
print (X)
```

Escopo

```
X="teste"
def func():
    X = 'NI'
    def nested():
        global X
        X = 'Spam'
        print(X)
    nested()
    print(X)

func()
print (X)
```

Saídas:

Spam
NI
Spam

Recursividade

- É um princípio muito **poderoso** para construção de algoritmos
- A solução de um problema é **dividido** em
  - Casos simples:
    - São aqueles que podem ser resolvidos **trivialmente**
  - Casos gerais:
    - São aqueles que podem ser resolvidos **compondo soluções** de casos mais simples

Funções Recursivas

- Algoritmos recursivo onde a solução dos casos genéricos requerem **chamadas à própria função**
- Exemplo: Sequência de Fibonacci
  - O **primeiro** e o **segundo** termo são **0** e **1**, respectivamente
  - O ***i*-ésimo** termo é a soma do **(*i*-1)-ésimo** e o **(*i*-2)-ésimo** termo

Fatorial

- Fatorial(1) = 1
- Fatorial(i) = i * Fatorial(i – 1)

```
def fatorial(n):
  if n==0 or n==1:
    return 1
  else:
    return n*fatorial(n-1)
```

Fibonacci

- Fibonacci(0) = 0
- Fibonacci(1) = 1
- Fibonacci(i) = Fibonacci(n-1) + Fibonacci(n-2)

```
def fibonacci(n):
   if n<=1:
      return n
   else:
      return fibonacci(n-1)+fibonacci(n-2)
```

Recursividade

Exemplo

```
def fibRec(n):
    if n == 1: return 0
    elif n == 2: return 1
    else: return fibRec(n-1) + fibRec(n-2)

for i in range (1, 8):
    print fibRec(1)
```

```
0
1
1
2
3
5
8
```

Funções Recursivas

- Exemplo: Fatorial
  - **Fatorial(1) = 1**
  - **Fatorial(i) = i * Fatorial(i – 1)**

```
def fatRecursivo(num):
    if (num == 1):
        return 1
    else:
        return num * fatRecursivo(num - 1)


>>> fatRecursivo(6)
720
```

MDC

- mdc(a,b) = a (b=0)
- mdc(a,b) = mdc(b, a%b)

```
def mdc(a,b):
    if b == 0:
        return a
    return mdc(b, a % b)

print("MDC 10 e 5 --> %d" % mdc(10,5))
print("MDC 32 e 24 --> %d" % mdc(32,24))
print("MDC 5 e 3 --> %d" % mdc(5,3))
```

MMC

- mmc(a,b) = |a x b| / mdc(a, b)

```
def mmc(a,b):
    return abs(a*b) / mdc(a,b)

print("MMC 10 e 5 --> %d" % mmc(10,5))
print("MMC 32 e 24 --> %d" % mmc(32,24))
print("MMC 5 e 3 --> %d" % mmc(5,3))
```

Validação

- Funções são úteis para validar entrada de dados

```
def faixa_int(txt,min,max):
   while True:
      v = int(input(txt))
      if v<min or v>max:
         print("Erro! %d < v < %d" % (min, max))
      else:
         return v
```

Exercício

- Fazer uma função que valida uma variável string. A função deve receber como parâmetros uma string, o número mínimo e máximo de caracteres. Retorne verdadeiro se o tamanho da string estiver entre os valores de mínimo e máximo, e falso em caso contrário.

Validação

```
def valida_string(s,min,max):
   tam = len(s)
   return (min <= tam) and (tam <= max)
```

```
def valida_string(s,min,max):
   tam = len(s)
   return min <= tam <= max
```

Argumentos de funções

- **Argumentos (ou parâmetros)** são variáveis que recebem valores iniciais na **chamada** da função
- Essas variáveis são ***locais***
- Se uma função define ***n*** argumentos, a sua chamada **deve incluir valores** para todos eles
  - **Exceção**: argumentos com valores **default**

Exemplo

```
>>> def f(x):
        return x*x

>>> print f(10)
100
>>> print x
....
NameError: name 'x' is not defined
>>> print f()
....
TypeError: f() takes exactly 1 argument (0 given)
```

Argumentos default

- É possível dar valores ***default*** a argumentos
  - Se o chamador não especificar valores para esses argumentos, os **defaults** são usados
- Formato:

```
def nomeFuncao (arg1=default1, ..., argN=defaultN)
```

- Se apenas alguns argumentos têm default, esses devem ser os ***últimos***

Exemplo

```
def barra():
   print(“*” * 40)

def barra(n=40,caractere=“*”):
   print(caractere * n)

>>> barra(10)
**********
>>> barra(10, “-”)
----------
```

Exemplo

```
>>> def f(nome,saudacao="Oi",pontuacao="!!"):
      return saudacao+","+ nome + pontuacao

>>> print f("Joao")
Oi,Joao!!
>>> print f("Joao","Parabens")
Parabens,Joao!!
>>> print f("Joao","Ah","...")
Ah,Joao...
```

Exemplo

```
def soma(a,b,imprime=False):
   s = a+b
   if imprime:
      print(s)
   return s

#ERRO
def soma(imprime=True,a,b):
   s = a+b
   if imprime:
      print(s)
   return s
```

Nomeando Parâmetros

- Até agora vimos parâmetros sendo passados na mesma ordem em que foram definidos.
- Quando especificamos os nomes dos parâmetros, podemos passá-los em qualquer ordem

```
def retangulo(largura,altura,caractere="*"):
    linha = caractere*largura
    for i in range(altura):
        print(linha)
```

Exemplo

```
def retangulo(largura,altura,caractere="*"):
   linha = caractere*largura
   for i in range(altura):
      print(linha)

>>> retangulo(3,2)
***
***
>>> retangulo(altura=3, largura = 2, caractere="-")
--
--
--
```

Observações

- Funções podem ser utilizadas da mesma maneira que outro tipo de dado em Python
- Elas podem ser:
  - Argumentos para outras funções;
  - Valores de retorno de outras funções;
  - Atribuídas para outras variáveis;
  - Partes de tuplas, listas, etc

Exemplo

```
def soma(a,b):
   return a+b
def subtracao(a,b):
   return a-b

def imprime(a,b,foper):
   print(foper(a,b))

imprime(5,4,soma)
imprime(10,4,subtracao)
```

Módulos

- Módulos são arquivos em python (.py)
- Usados para separar o código dependendo de sua funcionalidade
- Facilita organização e **reuso**.

Import

- 1) import modulo
  - Ex:

    import random

    random.randint(1,10)
- 2) from modulo import funcao
  - Ex:
    - from random import randint
    - randint(1,10)
- 3) from modulo import *
  - Ex:
    - from random import *
    - randint(1, 10)
    - randfloat(1,10)

Exemplo

arq1.py

```
def func(x, y):
    if x > y:
        res = x-y
    else:
        res = y-x
    return res
```

arq2.py

```
from arq1 import *
x = input("Digite um valor para x: ")
y = input("Digite um valor para y: ")
func(x,y)
```

arq3.py

```
import arq1
x = input("Digite um valor para x: ")
y = input("Digite um valor para y: ")
arq1.func(x,y)
```

Algumas Funções Especiais!

Notação lambda

- Funções podem ser definidas sem precisar de rótulos!
- Isto é muito útil quando você quer passar uma pequena função como argumento para outra função
- Apenas funções simples (única expressão) podem ser definidas nessa notação.

- Notação lambda tem um histórico rico em linguagens de programação desde I.A. passando por LISP, haskell...

```
lambda <parametros>: <codigo com retorno>
```

Notação lambda

```
def soma(a,b):
  return a+b
f = soma
f(2,7)

f = lambda a,b: a+b
f(2,7)

f = lambda a: a*a*a
f(3)
```

Funções map, reduce e filter

- Função map(func, seq)
- Função interna que aplica uma função func a cada item de um objeto sequência (seq) e retorna uma lista com os resultados da chamada da função.

```
>>> pow2 = lambda a: a**2
>>> pow2(5)
25
>>> map(pow2, [1, 2, 3, 4, 5])
[1, 4, 9, 16, 25]
>>> map(lambda x, y: x + y, [1, 2, 3, 4, 5], [6, 7, 8, 9, 10])
[7, 9, 11, 13, 15]
>>>
```

Funções map, reduce e filter

- Função reduce(func, seq)

- Função interna que aplica a função sobre o valor corrente retornado pela função (func) junto com o próximo item da lista.

```
>>>soma = a,b: a+b
>>>reduce(soma,[1,2,3,4,5,6,7,8,9])
    >>>#este comando executa:
    (((((((1+2)+3)+4)+5)+6)+7)+8)+9)
```

```
>>>fat = lambda x: reduce(lambda a,b:a*b,
xrange(1,x+1))
```

Funções map, reduce e filter

- Função filter(func, seq)
- Função interna que aplica uma função filtro func a cada item de um objeto sequência (seq) e retorna uma lista com os resultados que satisfazem os critérios da função de teste func.

```
>>> filter(lambda a: a>10, [2, 3, 4, 5,77, 49, 38, 2, 485])
[77, 49, 38, 485]
>>> filter(lambda a: a%2, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
[1, 3, 5, 7, 9]
```

Exercícios

Dado um conjunto de palavras [‘fita’, ‘Adenilton’, ‘armario’, ‘gaveta’, ‘Bruna’, ‘adentro’, ‘folga’, ‘impressora’]. Monte uma função usando filter, que remova todas as palavras que comecem com uma determinada letra passada como argumento, independente de ser maiúscula ou minúscula.

EXERCÍCIOS

Exercícios

1. Faça um programa que converta da notação de 24 horas para a notação de 12 horas. Por exemplo, o programa deve converter 14:25 em 2:25 P.M; 6:44 em 6:44 A.M. A entrada é dada em dois inteiros. O programa deve ler várias entradas e chamar uma função para convertê-las e em seguida imprimir a saída.

2. Faça uma função chamada *somaImposto*. A função possui dois parâmetros :

- *taxaImposto*, que é a porcentagem de imposto sobre vendas
- custo, que é o custo de um item antes do imposto.

A função retorna o valor de custo alterado para incluir o imposto sobre vendas.

Exercícios

3. Faça um programa que permita ao usuário digitar o seu nome e em seguida o programa chama uma função que retorna o nome do usuário de trás para frente utilizando somente letras maiúsculas. Dica: lembre−se que ao informar o nome, o usuário pode digitar letras maiúsculas ou minúsculas.

Exercícios

4. Faça um programa que solicite a data de nascimento (dd/mm/aaaa) do usuário e imprima a data com o nome do mês por extenso. O programa deve chamar uma função que retorna o mês convertido. Exemplo:

- Entrada - Data de Nascimento: 29/10/1973
- Saída - Você nasceu em 29 de Outubro de 1973.

Bibliografia

- Livro “Como pensar como um Cientista de Computação usando Python” – Capítulos 3 e 13
  - http://pensarpython.incubadora.fapesp.br/portal
- Python Tutorial
  - http://www.python.org/doc/current/tut/tut.html
- Dive into Python
  - http://www.diveintopython.org/
- Python Brasil
  - http://www.pythonbrasil.com.br/moin.cgi/DocumentacaoPython#head5a7ba2746c5191e7703830e02d0f5328346bcaac