Python – Funções

Introdução à Programação

SI1

Conteúdo

- Funções
  - Conceitos
  - Objetivos
  - Modularização
  - Exemplos
  - Exercícios

Funções

- O que são?
- Qual a função de uma função?

Imagine o código

```
matriz1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
matriz2 = [[2, 4, 3], [2, 1, 7], [4, 3, 2]]
matriz3 = [[4, 3, 2], [1, 1, 6], [2, 1, 4]]

#Imprime matriz 1
for nlinha in range(3):
    linha = ""
    for ncol in range(3):
        linha = linha + str(matriz1[nlinha][ncol])
    print(linha)

#Imprime matriz 2
for nlinha in range(3):
    linha = ""
    for ncol in range(3):
        linha = linha + str(matriz2[nlinha][ncol])
    print(linha)

#Imprime matriz 2
for nlinha in range(3):
    linha = ""
    for ncol in range(3):
        linha = linha + str(matriz3[nlinha][ncol])
    print(linha)
```

Imagine o código

```
matriz1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
matriz2 = [[2, 4, 3], [2, 1, 7], [4, 3, 2]]
matriz3 = [[4, 3, 2], [1, 1, 6], [2, 1, 4]]

#Imprime matriz 1
for nlinha in range(3):
    linha = ""
    for ncol in range(3):
        linha = linha + str(matriz1[nlinha][ncol])
    print(linha)

#Imprime matriz 2
for nlinha in range(3):
    linha = ""
    for ncol in range(3):
        linha = linha + str(matriz2[nlinha][ncol])
    print(linha)

#Imprime matriz 2
for nlinha in range(3):
    linha = ""
    for ncol in range(3):
        linha = linha + str(matriz3[nlinha][ncol])
    print(linha)
```

Repetição De Código!

Qual a utilidade da ***função***?

- Evita repetição de código
- Deixa o código Menor
- Mais legível
- Mais modularizado

Definindo Funções

Definição da função inicia com "*def*"

```
def funcao_que_faz_algo(arg1, arg2):
    """Texto de documentação"""
    linha1
    linha2
    return alguma_coisa
```

Definindo Funções

Definição da função inicia com “*def*”

Nome da função

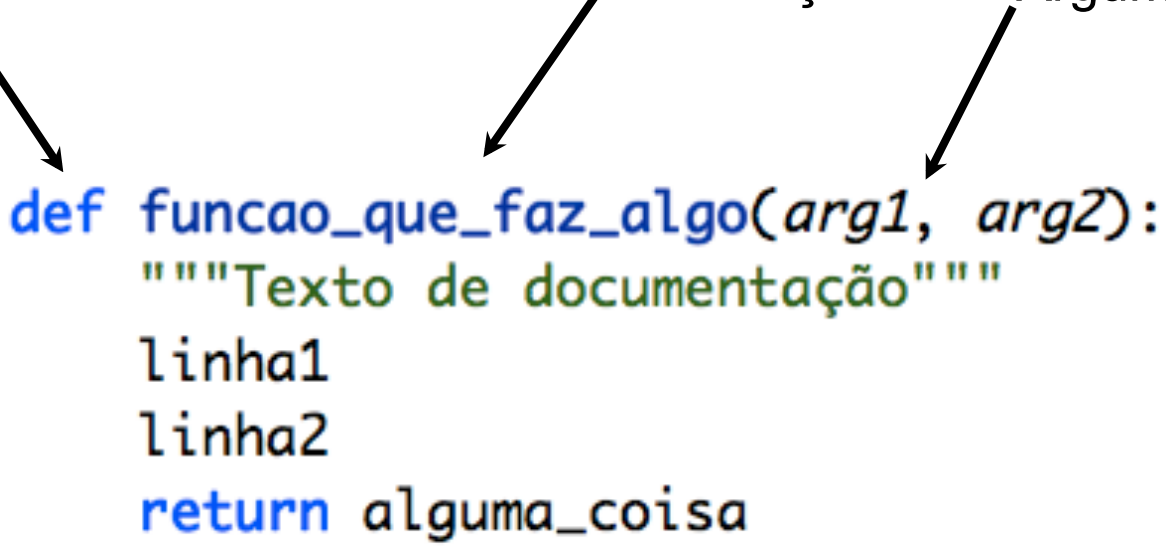
```
def funcao_que_faz_algo(arg1, arg2):
    """Texto de documentação"""
    linha1
    linha2
    return alguma_coisa
```

Definindo Funções

Definição da função inicia com "*def*"

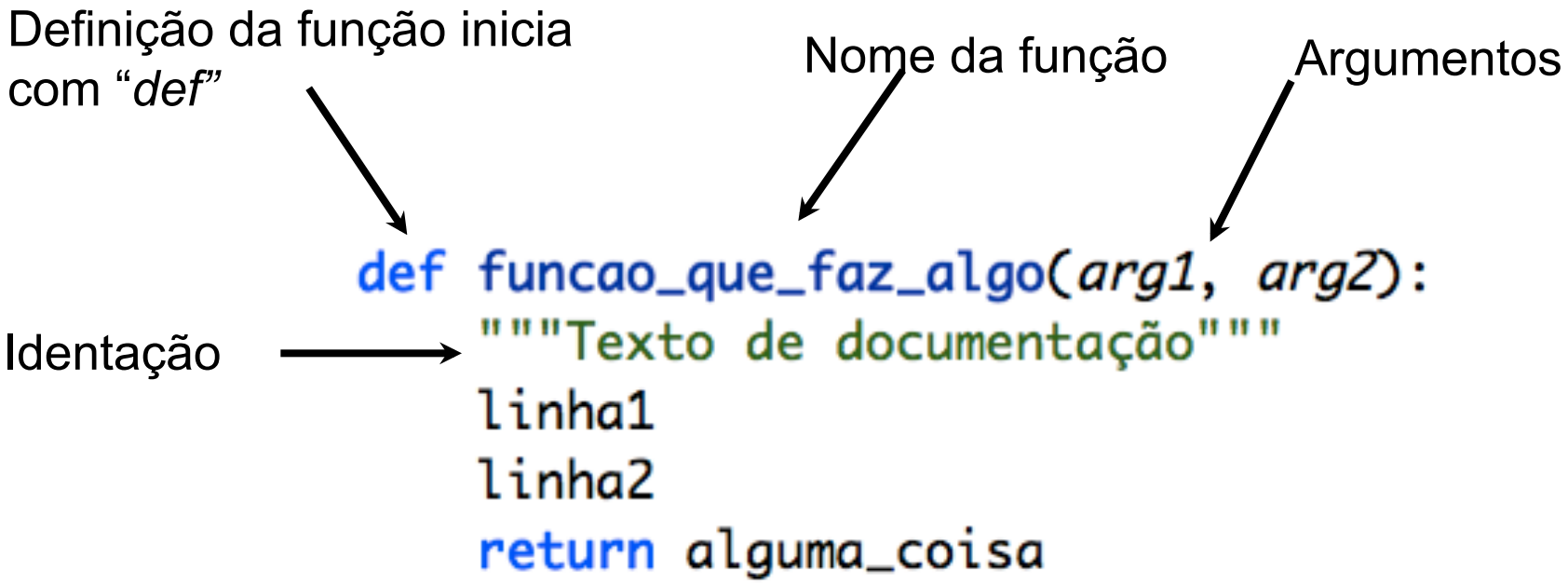
Nome da função

Argumentos

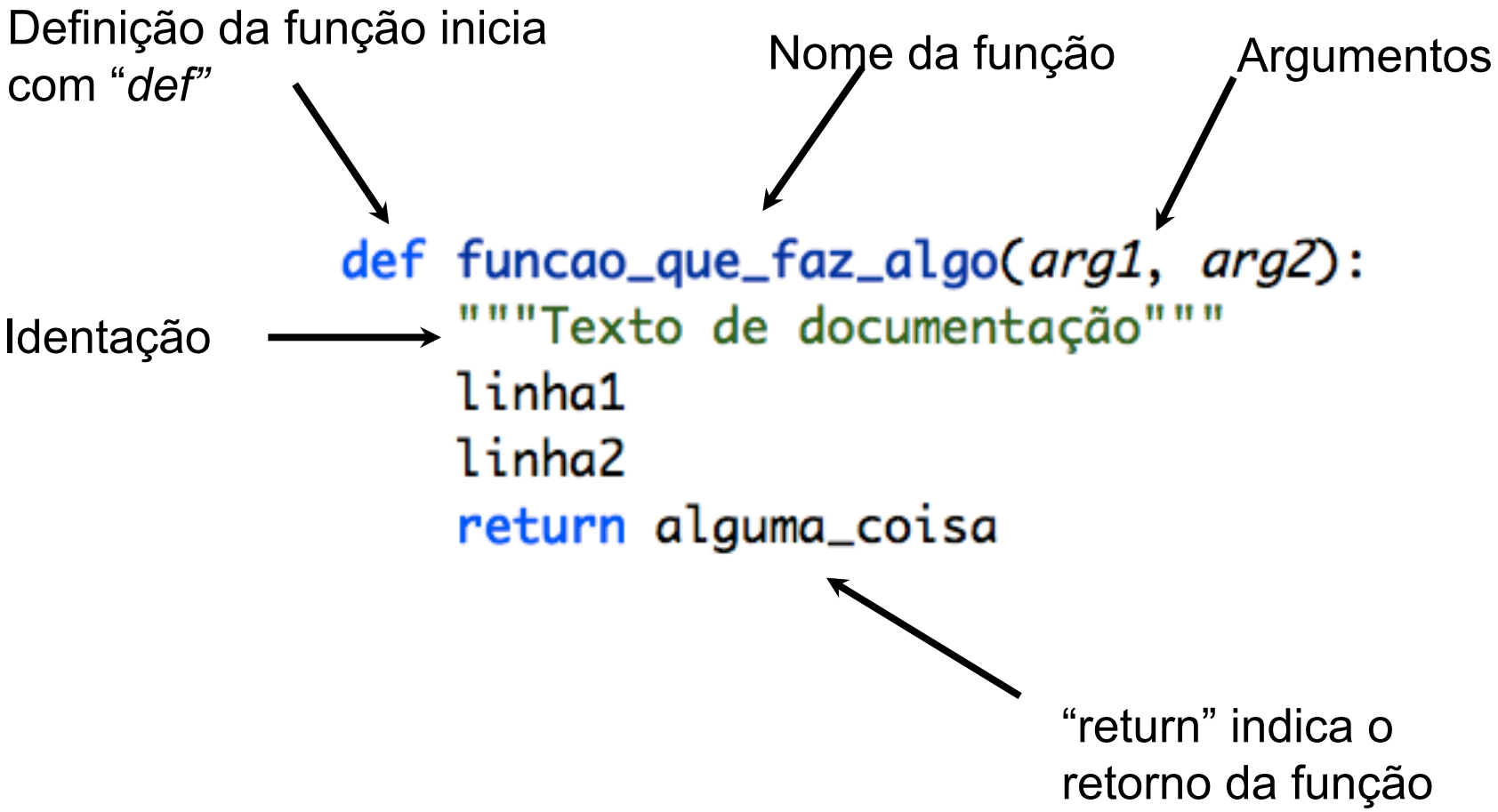


```
def funcao_que_faz_algo(arg1, arg2):
    """Texto de documentação"""
    linha1
    linha2
    return alguma_coisa
```

Definindo Funções



Definindo Funções



Programação Estruturada

- Usa o princípio de "Dividir para Conquistar"
- Programas são divididos em **sub-programas**
  - Cada sub-programa é chamado por meio de um **identificador** e uma lista de **parâmetros** de **entrada**
    - Permite especificar como um problema pode ser resolvido *no geral*
    - O mesmo sub-programa pode ser invocado para resolver diversos problemas de **mesma natureza** mas com valores específicos diferentes

Resultado de Funções

- Uma função tipicamente computa **um** ou **mais valores**
- Para indicar o valor a ser devolvido como o resultado da função, usa-se o comando return, que tem o formato

  return *expressão*

  – onde a *expressão* é opcional *e* designa o valor a ser retornado

Funções

- `return` → termina a função retornando um valor

- O valor default de return → **None**
  - Similar a **NULL**, **void**, ou **nil** em outras linguagens;

- Se a função chegar ao fim sem o uso explícito do `return`, então também será retornado o valor **None**

Valor de retorno

- Toda função em Python retorna algum valor;
- Mesmo funções que não possuem a cláusula **return**.
- Funções que não possuem return retornam **None**.
- **None** é uma constante especial definida na linguagem;
    - Similar a **NULL**, **void**, ou **nil** em outras linguagens;
- É lógicamente equivalente à False;
- O interpretador não imprime **None**.

Exemplo 1

```
#Define a funcao
def somador(valor1, valor2):
    soma = valor1 + valor2
    return soma

#chama a funcao
res = somador(3, 4)
print(res)
```

Exemplo 2

```
#Declaracao da funcao
def imprime_msg(nomePessoa):
    print("O usuario" + nomePessoa + "escreveu uma mensagem")

#Chamando a funcao
nome = input("Digite o seu nome: ")
imprime_msg(nome)
```

Exemplo 3

```
import random

#Declaracao de funcao que cria lista aleatoria
def criaListaAleatoria (tamanho):
    lista = []
    for i in range(tamanho):
        valor = random.randint(1,10)
        lista.append(valor)
    return lista, max(lista), min(lista)

#Chamada funcao
tam = 5
li, maxli, minli = criaListaAleatoria(tam)
```

Exercício

- Fazer uma função que recebe três argumentos, e que retorne o produto desses três argumentos.
- Fazer uma função que receba como parametro um numero inteiro e retorne o fatorial desse numero (não usar recursividade).

Abstração

- Técnica de programação que nos permite pensar num problema em **diversos níveis**
  - Quando pensamos num problema macroscopicamente, não estamos preocupados com detalhes
- ***Dividir para conquistar***:
  - Um problema é dividido em diversos sub-problemas
  - As soluções dos **sub-problemas** são combinadas numa solução do problema maior

Múltiplos argumentos

- Podem receber um número arbitrário de ‘keywords’

```
>>>def n_media(**notas):
...     '''Este método calcula a média das
...     notas passadas como argumento
...     '''
...     return sum(notas.values())/float(len(notas))
>>>n_media(10,6)
8
>>>n_media(10, 6, 4, 8)
7
>>>n_media(10, 6, 4, 8, 3, 9, 6, 2)
6
```

Múltiplos argumentos

- Podem receber inclusive listas e dicionários.

```
>>>def n_media(*notas, **knotas):
...     ret = 0
...     if notas:
...         ret sum(notas)/float(len(notas))
...     elif knotas:
...         ret sum(notas.values())/float(len(notas))
...     return ret
>>>a = [ 10, 6, 4, 8, 3, 9, 6, 2]
>>>n_media(*a)
6
>>>d = [ "nota1":10,"nota2": 6,"nota3": 4,"nota4": 8]
>>>n_media(**a)
7
```

Funções

- Em Python, sub-programas têm o nome de ***funções***
- Formato geral:

```
def nome (arg, arg, ... arg):
        comando
        . . .
        comando
```

- Onde:
  - **nome** é o nome da função
  - **args** são especificações de argumentos da função
    - Uma função pode ter 0, 1 ou mais argumentos
  - **comandos** contêm as instruções a ser executadas quando a função é invocada

Resultado de Funções

- Ao encontrar o comando return, a função termina imediatamente e o controle do programa **volta ao ponto** onde a função foi chamada
- Se uma função chega a seu fim sem nenhum valor de retorno ter sido especificado, o valor de retorno é None

Funções x Procedimentos

- Procedimento

```
>>> def  fib(n):
    #escreve a serie de Fibonacci ate n
    a, b = 0, 1
    print(a)
    while b < n:
        print (b)
        a, b = b, a+b


>>> fib (8)
0
1
1
2
3
5
```

Funções x Procedimentos

- Função

```
>>> def fib2(n):
        # Retorna a lista contendo
        # a serie de Fibonacci ate n
        result = []
        a, b = 0, 1
        result.append(a)
        while b < n:
            result.append(b)
            a, b = b, a+b
        return result

>>> fib2(8)
[0, 1, 1, 2, 3, 5]
```

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 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...
```

Argumentos

```
>>> def mostra_info (obrigatorio, nome = "joao", idade = 20):
    print ("obrigatorio: %s\nnome: %s\nidade: %d" %(obrigatorio, nome, idade))

>>> mostra_info('teste')

obrigatorio: teste
nome: joao
idade: 20
>>> mostra_info('teste',10)

obrigatorio: teste
nome: 10
idade: 20
>>> mostra_info('teste', idade=10)
obrigatorio: teste
nome: joao
idade: 10
```

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

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

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
```

Variáveis Locais e Globais

- Variáveis **definidas em funções** são ***locais***, isto é, só podem ser usadas nas funções em que foram definidas
- Variáveis definidas **fora de funções** são conhecidas como variáveis **globais**
  - Em uma função **pode-se** ler o conteúdo de uma variável global
  - Para **alterar uma variável global**, deve-se declara-la no corpo da função com **global**

Exemplo

```
>>> def f():
        print(a)


>>> a = 1
>>> f()
1
>>> def f():
        a = 5


>>> f()
>>> print a
1
>>> def f():
        global a
        a = 5


>>> f()
>>> print(a)
5
```

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

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)
9
>>>f = lambda a, b: a + b
>>>f(2,7)
9
>>>def incrementador(n):
...     return lambda a: a + n
>>>f = incrementador(10)
>>>f(5)
15
>>>f(20)
30
>>>f = lambda a, b: (a != b and a > b) and a or b
>>>f(10,3)
10
>>>f(3,5)
5
```

Notação lambda

```
>>>def soma(a,b):
    return a + b

>>>f = soma
>>>f(2, 7)
Saida: 9

>>>f = lambda a, b: a + b
>>>f(2,7)
9

>>>f = lambda a: a*a*a
>>>f(3)
8
```

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]
>>>
```

- >>>#e assim por diante
- >>>#map(func,seq1[,seq2,seq3,...])
  #retorna uma lista

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

Gerar uma função que retorna o número de parâmetos passados e uma string com todas os parâmetros concatenados como string.
Ex:
Entrada
funcao(1,3,'flavio,'avaty')
Saída
(4,'13flavioavaty')

Dado um conjunto de palavras [‘fita’, ‘Adenilton’, ‘armario’, ‘gaveta’, ‘Bruna’, ‘adentro’, ‘folga’, ‘impressora’]. Montar um filtro que remova todas as palavras que comecem com ‘A’ ou ‘a’.

Calcule o quadrado dos números pares entre o intervalo de 0 a 10.

EXERCÍCIOS

Exercícios

1. Faça uma função chamada somaImposto. A função possui dois parâmetros :

a) taxaImposto, que é a porcentagem de imposto sobre vendas

b) 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 (Arquivos Separados)

2. Faça uma função que informe a quantidade de dígitos de um determinado número inteiro informado pelo usuário.

3. Fazer uma função que recebe um argumento inteiro. A função retorna o valor de caractere ‘P’, se seu argumento for positivo, e ‘N’, se seu argumento for zero ou negativo.

4. Faça uma função que retorne o reverso de um número inteiro informado. Por exemplo: 127 -> 721.

Exercícios

5. Considere a seguinte fórmula para calcular o mdc (máximo divisor comum) de dois números inteiros positivos:

- mdc(a, b) = b, se b divide a (ou seja, a%b == 0)
- mdc(a, b) = mdc(b, a%b), caso contrário

Escreva uma função em Python que, dados dois números, retorne o máximo divisor comum entre eles. **Usar recursividade**.

6. Criar um programa que leia uma lista de pessoas, obtenha os nomes das pessoas que comecem com ‘a’ e imprima a maior delas (o maior nome).

- Organize as os módulos da seguinte forma.
  - Módulo Principal
  - Módulo só para manipular lista
  - Módulo só para imprimir.
- A execução do algoritmo deverá ser feita através de chamada de funções dos módulos.

Exercícios

7. 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.

Exercícios

8. 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

9. 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