Definição de Funções

Prof. Alberto Costa Neto

Programação em Python

Funções em Python

- Há 2 tipos de funções em Python.
  - Funções Built-in que são providas como parte da linguagem Python - input(), type(), float(), int() ...

  > Funções que nós definimos e então as utilizamos

- Tratamos os nomes das funções built-in como “novas” palavras reservadas (ou seja, evitamos usá-los como nomes de variáveis)

Definição de Funções

- Em Python, assim como em outras linguagens de programação, uma função é um código fonte reusável que recebe argumento(s) como entrada, computa algo, e então retorna um resultado ou resultados

Construindo nossas próprias Funções

- Nós criamos uma nova função usando a palavra chave def seguida por parâmetros (opcionalmente) dentro de parênteses
- O corpo da função deve ser indentado
- Isto define a função, mas *não* executa o corpo da função

```
def aviso():
    print('Sistema indisponivel')
    print('Contacte o administrador:')
    print('adm@empresa.com.br')
```

Chamando/Invocando Funções

- Chamamos/Invocamos uma função ao usar o nome da função, parênteses e argumentos em uma expressão ou comando
- Por exemplo, para chamar a função aviso, utilizamos:

```
aviso()
```

Definições e Usos

- Uma vez que tenhamos definido uma função, podemos chamar (ou invocar) a função quantas vezes quisermos

- Reuso gera lucro em vários ramos da indústria!

Passos armazenados (reusados)

Programa:

```
def aviso():
    print('Sistema indisponivel')
    print('Contacte o administrador')
    print('adm@empresa.com.br')

print('Primeiro erro')
aviso()
print('Erro novamente')
aviso()
```

Saída:

```
Primeiro erro
Sistema indisponivel
Contacte o administrador
adm@empresa.com.br
Erro novamente
Sistema indisponivel
Contacte o administrador
adm@empresa.com.br
```

Chamamos estes pedaços de código fonte reusáveis de “funções”

Argumentos

- Um argumento é um valor que passamos para dentro de uma função (como sua entrada) quando chamamos a função
- Usamos argumentos para fazer com que a função execute tipos diferentes de trabalho quando a chamamos em situações diferentes
- Os argumentos são colocados, entre parênteses e separados por vírgula, depois do nome da função

```
big = max('Hello world')
```

Parâmetros

- Um parâmetro é uma variável que usamos dentro da definição da função
- É o mecanismo que permite acessar os argumentos de uma invocação específica de uma função.

```
>>> def cumprimentar(ling):
...     if ling == 'br':
...         print('Ola')
...     elif ling == 'fr':
...         print('Bonjour')
...     else:
...         print('Hello')
...
>>> cumprimentar('en')
Hello
>>> cumprimentar('br')
Ola
>>> cumprimentar('fr')
Bonjour
>>>
```

Valor de Retorno

Geralmente uma função recebe argumentos, computa algo, e retorna um valor a ser usado como o valor da função na expressão que a chamou. A palavra chave return é usada para isso.

```
def cumprimentar():
    return "Hello"

print(cumprimentar(), "Glenn")
print(cumprimentar(), "Sally")
```

```
Hello Glenn
Hello Sally
```

Valor de Retorno

- Uma função útil é uma que produz um resultado (ou valor de retorno)
- O comando return encerra a execução da função e “devolve” o resultado da função

```
>>> def cumprimentar(ling):
...     if ling == 'br':
...         return 'Ola'
...     elif ling == 'fr':
...         return 'Bonjour'
...     else:
...         return 'Hello'
...
>>> print(cumprimentar('en'),'Glenn')
Hello Glenn
>>> print(cumprimentar('br'),'Maria')
Ola Maria
>>> print(cumprimentar('fr'),'Michael')
Bonjour Michael
>>>
```

Funções Void

- Quando uma função não retorna um valor, a chamamos de função “void”

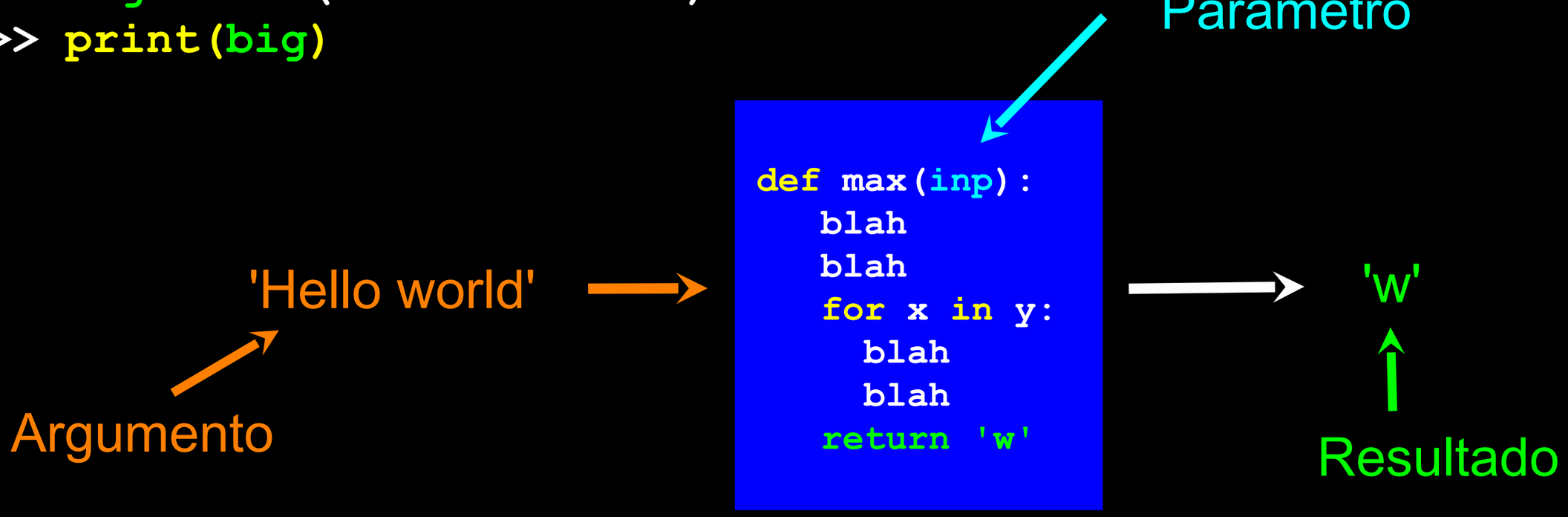
```
def cumprimentar(ling):
    if ling == 'br':
        print('Ola')
    elif ling == 'fr':
        print('Bonjour')
    else:
        print('Hello')
```

Melhor assim!

```
def cumprimentar(ling):
    if ling == 'br':
        return 'Ola'
    elif ling == 'fr':
        return 'Bonjour'
    else:
        return'Hello'
```

Argumentos, Parâmetros e Resultado

```
>>> big = max('Hello world')
>>> print(big)
w
```



Múltiplos Parâmetros / Argumentos

- Podemos definir mais de um parâmetro na definição da função
- Ao chamarmos a função, simplesmente passamos mais argumentos
- O número e a ordem dos argumentos deve casar com os parâmetros

```
def maior2(a, b):
    if a > b:
        return a
    else:
        return b

x = maior2(3, 5)
print(x)
```

Múltiplos Parâmetros / Argumentos

```
def maior2(a, b):
    if a > b:
        return a
    else:
        return b

def maior3(p1, p2, p3):
    m12 = maior2(p1,p2)
    m123 = maior2(p3,m12)
    return m123

print( maior3(3,7,5) )
```

Usar funções é muito bom

- Organiza o código fonte em “parágrafos” - capture um raciocínio completo e escolha um bom nome para a função
- DRY - *Don’t repeat yourself* – Faça apenas uma vez e reuse
- Se algo ficou muito grande e complexo, quebre em pedaços lógicos e coloque estes pedaços em funções
- Crie uma biblioteca (*library*) de coisas comuns que você faz repetidamente – talvez compartilhar com seus amigos...