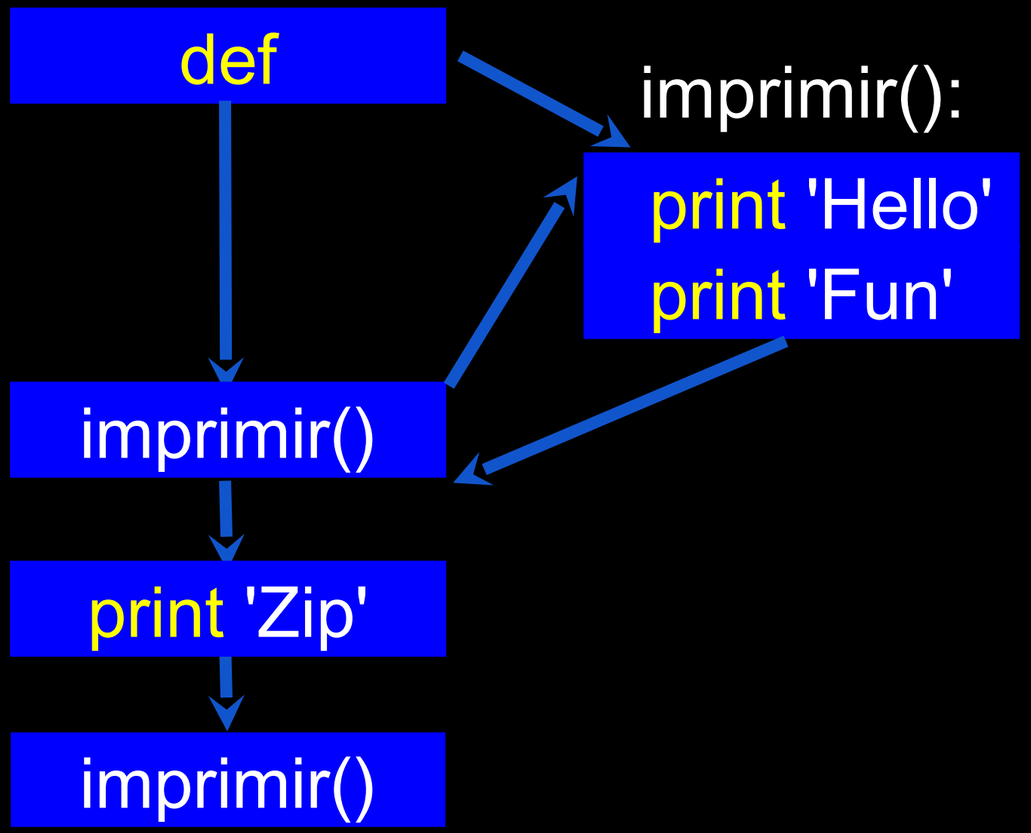
Funções

Prof. Alberto Costa Neto

Programação em Python

Passos armazenados (e reusados)



imprimir():

```
print 'Hello'
print 'Fun'
```

Programa:

```
def imprimir():
    print 'Hello'
    print 'Fun'

imprimir()
print 'Zip'
imprimir()
```

Saída:

```
Hello
Fun
Zip
Hello
Fun
```

Chamamos estes pedaços de código fonte reusáveis de “funções”

Funções em Python

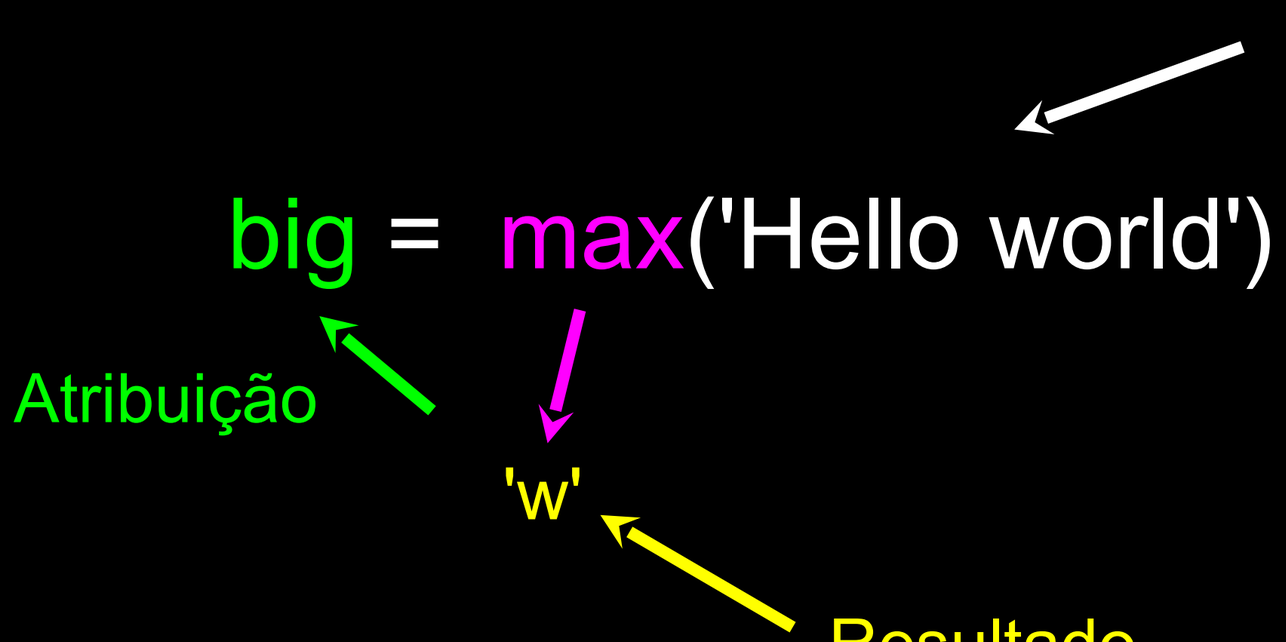
- Há 2 tipos de funções em Python.
  - Funções Built-in que são providas como parte da linguagem Python - raw_input(), type(), float(), int() ...

  > 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 uma função é um código fonte reusável que recebe argumento(s) como entrada, computa algo, e então retorna um resultado ou resultados
- Definimos uma função usando a palavra reservada def
- Chamamos/Invocamos uma função ao usar o nome da função, parênteses, e argumentos em uma expressão



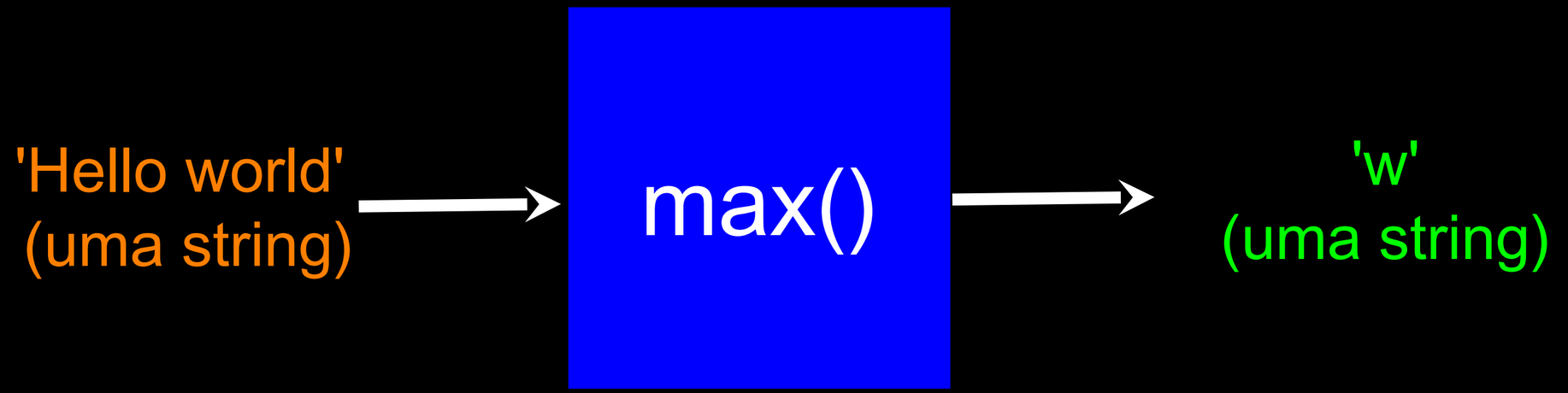

```
>>> big = max('Hello world')
>>> print big
w
>>> tiny = min('Hello world')
>>> print tiny

>>>
```

Função max

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

Uma função é um código fonte armazenado que podemos usar. Uma função recebe uma entrada e produz uma saída.

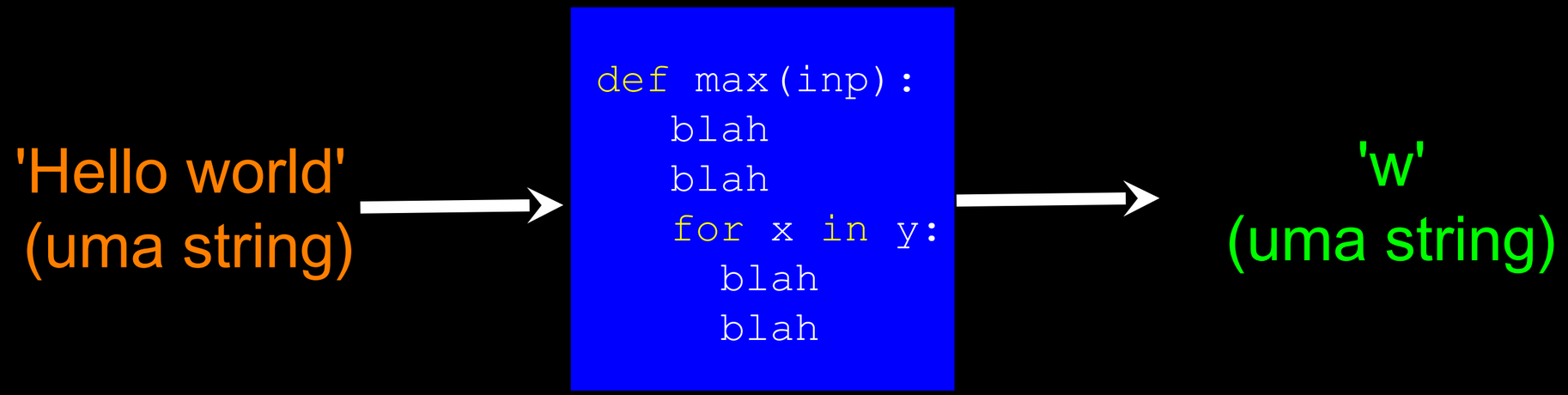


Guido escreveu este código

Função max

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

Uma função é um código fonte armazenado que podemos usar. Uma função recebe uma entrada e produz uma saída.



Guido escreveu este código

Funções de Conversão de Tipos

- Quando você inclui um inteiro e um ponto flutuante em uma expressão, o inteiro é implicitamente convertido para um float
- Você pode controlar isto com as funções built-in int() e float()

```
>>> print float(99) / 100
0.99
>>> i = 42
>>> type(i)
<type 'int'>
>>> f = float(i)
>>> print f
42.0
>>> type(f)
<type 'float'>
>>> print 1 + 2 * float(3) / 4 - 5
-2.5
>>>
```

Conversões de String

- Você também pode usar int() e float() para converter entre strings e inteiros
- Obterá um erro se a string não contiver caracteres numéricos

```
>>> sval = '123'
>>> type(sval)
<type 'str'>
>>> print sval + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str'
and 'int'
>>> ival = int(sval)
>>> type(ival)
<type 'int'>
>>> print ival + 1
124
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
```

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 imprimir_letra():
    print 'Eu sou um lenhador, e estou bem.'
    print 'Eu durmo a noite toda e trabalho o dia todo.'
```

imprimir_letra():

```
print 'Eu sou um lenhador, e estou bem.'
print 'Eu durmo a noite toda e trabalho o dia todo.'
```

```
x = 5
print 'Ola'

def imprimir_letra():
    print 'Eu sou um lenhador, e estou bem.'
    print 'Eu durmo a noite toda e trabalho o dia todo.'

print 'Oi'
x = x + 2
print x
```

```
Ola
Oi
7
```

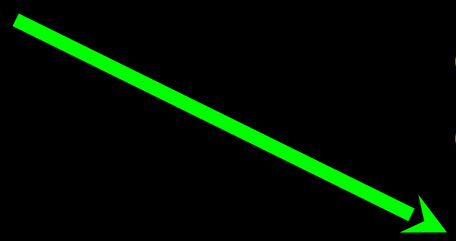
Definições e Usos

- Uma vez que tenhamos definido uma função, podemos chamar (ou invocar) a função quantas vezes quisermos
- Este é o padrão armazenar e reusar

```
x = 5
print 'Ola'

def imprimir_letra():
    print 'Eu sou um lenhador, e estou bem.'
    print 'Eu durmo a noite toda e trabalho o dia todo.'

print 'Oi'
imprimir_letra()
x = x + 2
print x
```



```
Ola
Oi
Eu sou um lenhador, e estou bem.
Eu durmo a noite toda e trabalho o dia todo.
7
```

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')
```

Argumento

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('es')
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"     Hello Glenn
print cumprimentar(), "Sally"     Hello Sally
```

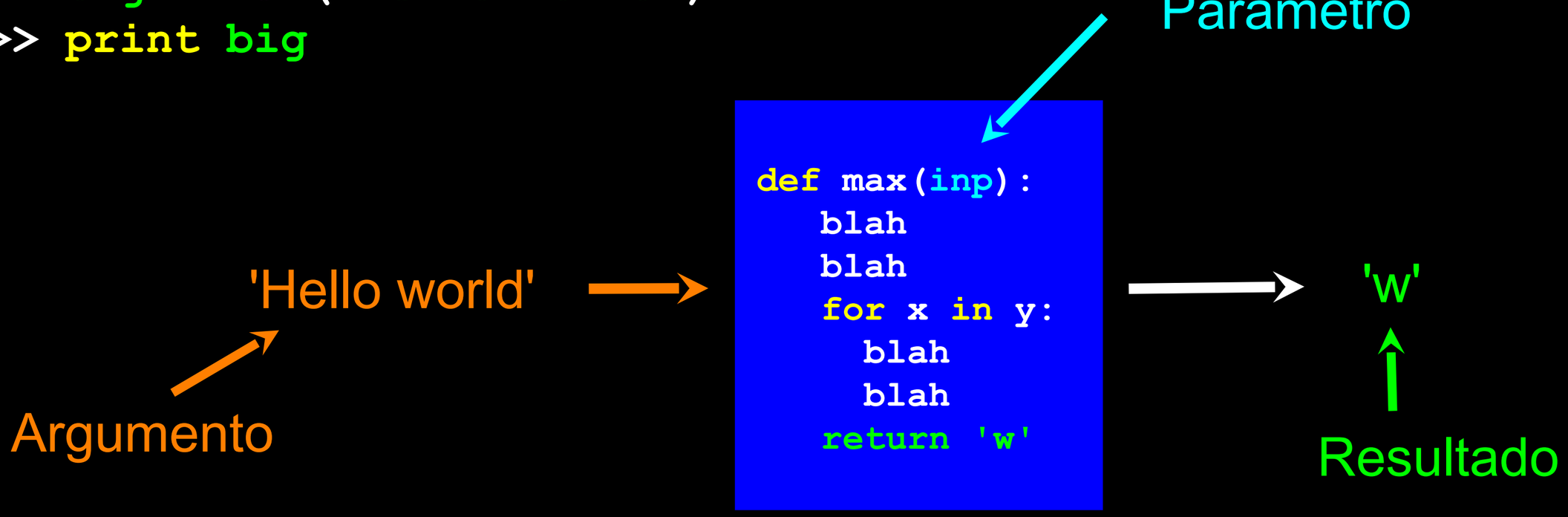
Valor de Retorno

- 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'),'Sally'
Ola Sally
>>> print cumprimentar('fr'),'Michael'
Bonjour Michael
>>>
```

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 somar(a, b):
    somado = a + b
    return somado

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

8

Funções Void

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

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...

Acknowledgements / Contributions
Agradecimentos / Contribuições

...

Initial Development: Charles Severance, University of Michigan School of Information

These slides were translated and adapted by Alberto Costa Neto (albertocn.sytes.net) of the Federal University of Sergipe