

# Funções

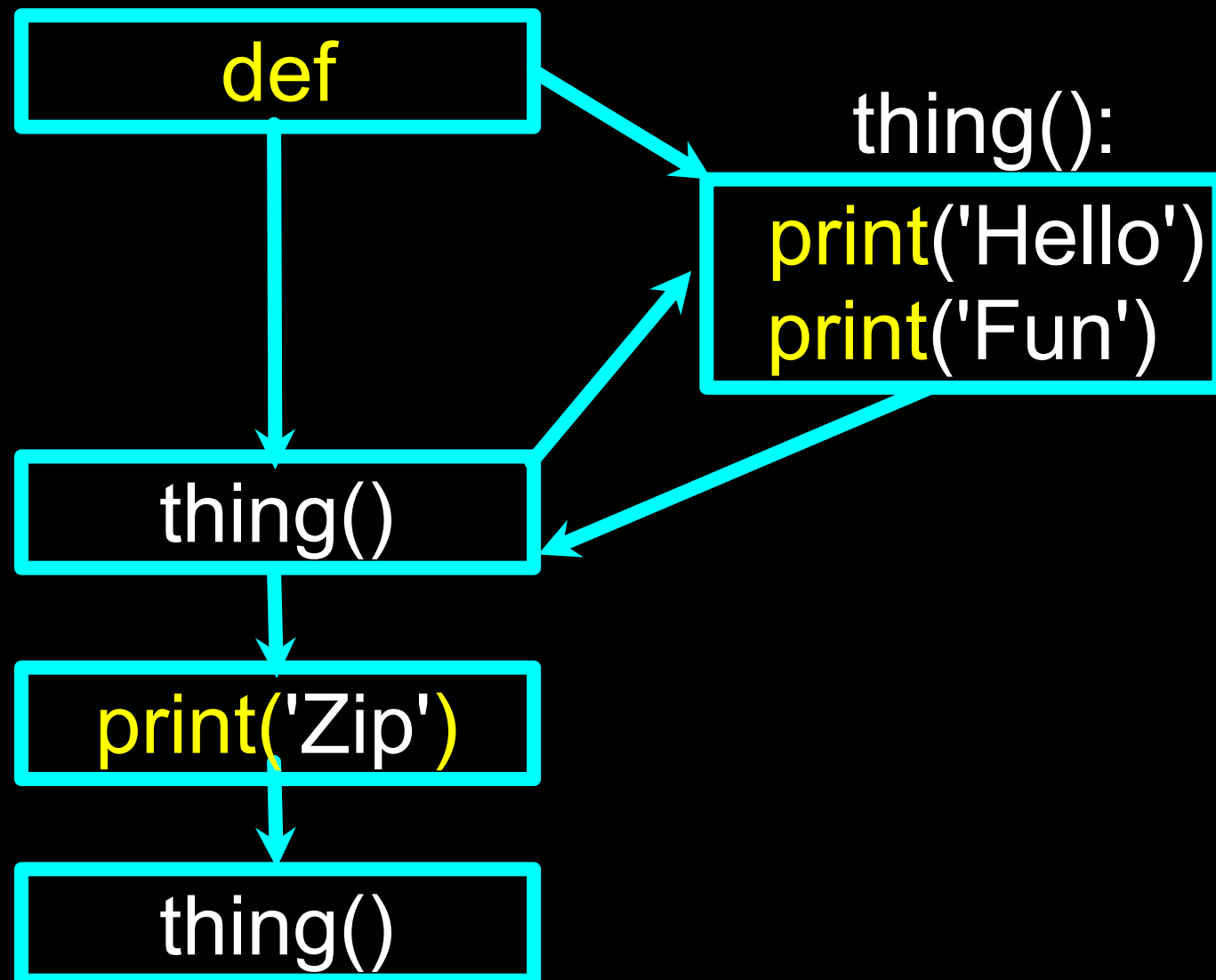
## Capítulo 4



Python for Everybody  
[www.py4e.com](http://www.py4e.com)



# Passos Armazenados (e Reusados)



Programa:

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

```
thing()  
print('Zip')  
thing()
```

Saída:

```
Hello  
Fun  
Zip  
Hello  
Fun
```

Chamamos essas partes reutilizáveis de código de "funções"

# Funções de Python

- Existem dois tipos de funções em Python.
  - **Funções Internas (*built-in functions*)** que são fornecidos como parte de Python - `print()`, `input()`, `type()`, `float()`, `int()` ...
  - **Funções que nos definimos** e depois usamos
- Tratamos os nomes das **funções internas** como “novas” palavras reservadas (ou seja, evitamos que sejam nomes de variáveis)

# Definição de Função

- Em Python, uma **função** é um código reutilizável que recebe **argumentos** como entrada, faz alguma computação e retorna um resultado ou resultados
- Definimos uma **função** usando a palavra reservada **def**
- Chamamos/invocamos a **função** usando o nome da função, parênteses e **argumentos** em uma expressão

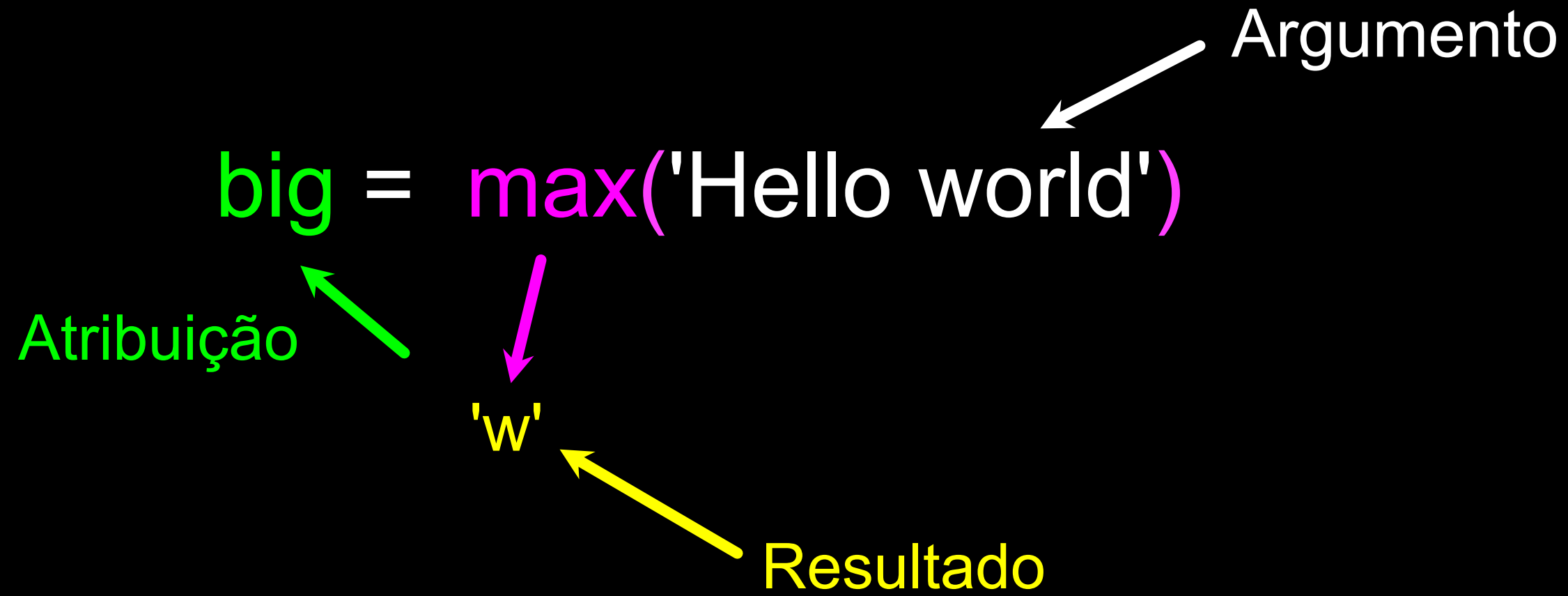
Argumento

`big = max('Hello world')`

Atribuição

`'w'`

Resultado



```
>>> 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 algum código armazenado que usamos. Uma função recebe alguma entrada e produz uma saída.



Guido foi quem escreveu esse código

# Função Max

Uma função é algum código armazenado que usamos. Uma função recebe alguma entrada e produz uma saída.

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

'Hello world'  
(uma string)

```
def max(inp):  
    blah  
    blah  
    for x in inp:  
        blah  
        blah
```

'w'  
(uma string)

Guido foi quem escreveu esse código

# Conversão de Tipos

- Quando você coloca um número inteiro e um ponto flutuante em uma expressão, o número inteiro é **implicitamente** convertido em um número flutuante
- Você pode controlar isso com as funções internas `int()` e `float()`

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



# Conversão de Strings

- Você também pode usar `int()` e `float()` para converter entre seqüências de caracteres e números inteiros
- Você receberá um **erro** se a seqüência não contiver caracteres numéricos

```
>>> sval = '123'
>>> type(sval)
<class '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)
<class '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()
```

Nossas próprias funções...

# Definindo nossas próprias funções

- Criamos uma nova função usando a palavra-chave **def** seguida por parâmetros opcionais entre parênteses
- Recuamos o corpo da função
- Isso **define** a função, mas não executa o corpo da função

```
def print_lyrics():  
    print("I'm a lumberjack, and I'm okay.")  
    print('I sleep all night and I work all day.')
```

`print_lyrics():`

```
print("I'm a lumberjack, and I'm okay.")  
print('I sleep all night and I work all day.')
```

```
x = 5  
print('Hello')
```

```
def print_lyrics():  
    print("I'm a lumberjack, and I'm okay.")  
    print('I sleep all night and I work all day.')
```

```
print('Yo')  
x = x + 2  
print(x)
```

Hello  
Yo  
7

# Definições e Usos

- Depois de **definir** uma função, podemos **chamá-la** (ou **invocá-la**) quantas vezes quisermos
- Esse é o padrão de **armazenamento** e **reutilização**

```
x = 5
print('Hello')

def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')

print('Yo')
print_lyrics()
x = x + 2
print(x)
```

Hello

Yo

I'm a lumberjack, and I'm okay.  
I sleep all night and I work all day.

7

# Argumentos

- Um **argumento** é um valor que passamos para a **função** como sua **entrada** quando chamamos a função
- Usamos **argumentos** para que possamos direcionar a **função** para realizar diferentes tipos de trabalho quando a chamamos em momentos **diferentes**
- Colocamos os **argumentos** entre parênteses após o **nome** da função

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



Argumento

# Parâmetros

Um **parâmetro** é uma variável que usamos **na definição** da função. É um "identificador" que permite que o código na função acesse os **argumentos** para uma chamada de função específica.

```
>>> def greet(lang):  
...     if lang == 'es':  
...         print('Hola')  
...     elif lang == 'fr':  
...         print('Bonjour')  
...     else:  
...         print('Hello')  
...  
>>> greet('en')  
Hello  
>>> greet('es')  
Hola  
>>> greet('fr')  
Bonjour  
>>>
```



# Valores de Retorno

Frequentemente, uma função aceita seus argumentos, calcula um pouco e **retorna** um valor para ser usado como o valor da chamada de função na **expressão de chamada**. A palavra-chave **return** é usada para isso.

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

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

```
Hello Glenn  
Hello Sally
```

# Valores de Retorno

- Uma **função** "proveitosa" é aquela que produz um **resultado** (ou **valor de retorno**)
- A instrução de **retorno** finaliza a execução da **função** e "envia de volta" o **resultado** da **função**

```
>>> def greet(lang):  
...     if lang == 'es':  
...         return 'Hola'  
...     elif lang == 'fr':  
...         return 'Bonjour'  
...     else:  
...         return 'Hello'  
...  
>>> print(greet('en'), 'Glenn')  
Hello Glenn  
>>> print(greet('es'), 'Sally')  
Hola Sally  
>>> print(greet('fr'), 'Michael')  
Bonjour Michael  
>>>
```

# Argumentos, Parâmetros e Resultados

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

Argumento → 'Hello world'

```
def max(inp):  
    blah  
    blah  
    for x in inp:  
        blah  
        blah  
    return 'w'
```

Parâmetros →

→ 'w'  
Resultado

# Mútiplos Parâmetros / Argumentos

- Podemos definir mais de um **parâmetro** na **definição** de **função**
- Simplesmente adicionamos mais **argumentos** quando chamamos a **função**
- Combinamos o número e a ordem dos argumentos e parâmetros

```
def addtwo(a, b):  
    added = a + b  
    return added
```

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

8

# Funções com retorno vazio

- Quando uma função não retorna um valor, chamamos de função “**vazia**” (em inglês “*void*”)
- Funções que retornam valores são funções “dão frutos”
- As funções **vazias** não “dão frutos”

# *“To function or not to function...”*

- Organize seu código em "parágrafos" - capture um pensamento completo e "dê um nome"
- Não se repita - faça funcionar uma vez e depois reutilize
- Se algo ficar muito longo ou complexo, divida-o em partes lógicas e coloque-as em funções
- Faça uma biblioteca de coisas comuns que você faz repetidas vezes - talvez compartilhe isso com seus amigos ...

# Resumo

- Funções
- Funções internas
- Conversão de tipo (int, float)
- Conversões de string
- Parâmetros
- Argumentos
- Resultados
- Funções sem resultado
- Por que usar funções?

## Exercício

Reescreva seu cálculo de pagamento com 1,5x para horas extras e crie uma função chamada `computepay`, que usa dois parâmetros (horas e taxa).

Digite as horas: 45

Digite a taxa: 10

Pay: 475.0

$$475 = 40 * 10 + 5 * 15$$





# Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance ([www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

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

Traduzido para o Português Brasileiro por Filipe Calegario

...