



UNIVERSIDAD
DE MÁLAGA



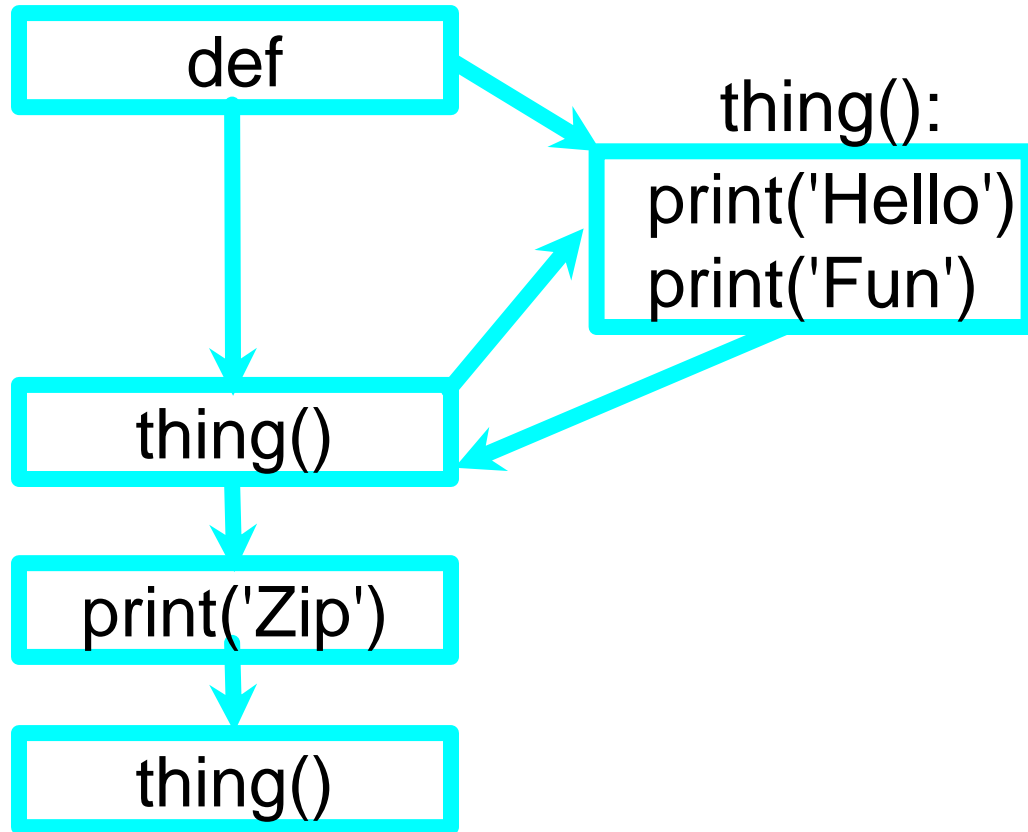
Funciones

Unidad 4

Python para principiantes
dgm@uma.es



Pasos almacenados (y reutilizados)



Programa:

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

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

Salida:

Hello
Fun
Zip
Hello
Fun

Llamaremos "**funciones**" a estas piezas reutilizables

Funciones en Python

Hay dos tipos de funciones en Python.

- **Funciones internas** proporcionadas como parte de Python - `print()`, `input()`, `type()`, `float()`, `int()` ...
- Funciones que **definiremos nosotros mismos** y que podremos usar después

Trataremos a las funciones internas como "nuevas" palabras reservadas
(es decir, evitaremos usarlas como nombres de variables)

Definición de Funciones

- En Python una función es algún código reutilizable que toma **argumento(s)** como entradas, hace algún **procesamiento**, y **retorna** un resultado o resultados
- Las funciones se definen utilizando la palabra reservada **def**
- Podemos llamar/**invocar** a las funciones utilizando el **nombre** de la función, paréntesis y los argumentos en una expresión

Argumento

big = max('Hello world')

Asignación

'w'

Resultado

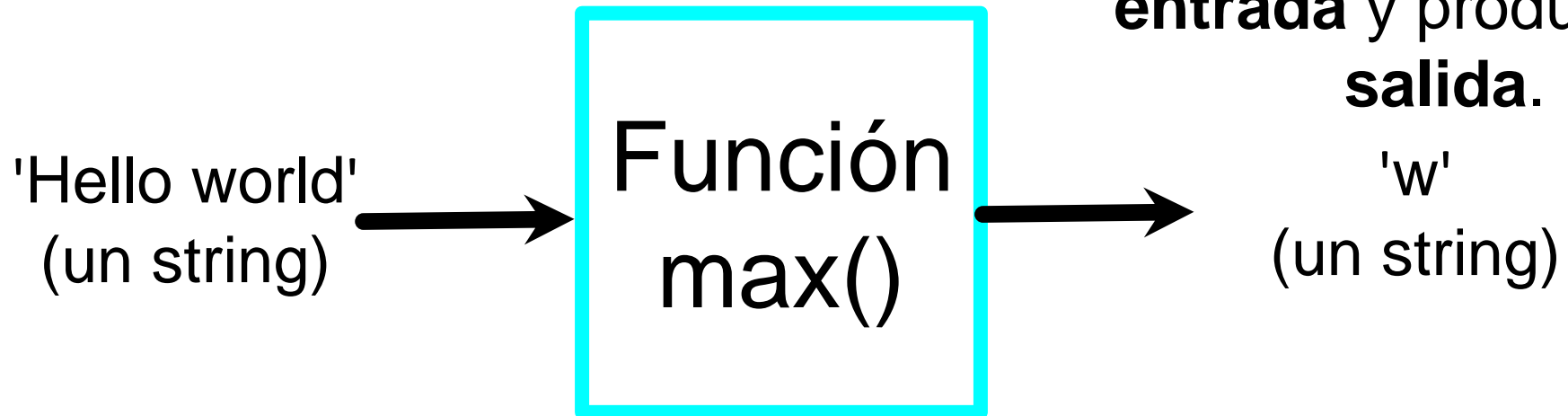
```
>>> big = max('Hello world')
>>> print(big)
w
>>> tiny = min('Hello world')
>>> print(tiny)

>>>
```

Función max

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

Una función es algún código almacenado que podemos usar. Una función toma una **entrada** y produce una **salida**.



Guido escribió este código

Función max

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

Una función es algún código almacenado que podemos usar. Una función toma una **entrada** y produce una **salida**.

'Hello world'
(un string)



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



'w'
(un string)

Guido escribió este código

Conversiones de tipo

- Cuando pones un entero y un flotante en una expresión, el entero es convertido implícitamente a flotante
- Podemos controlar esto con las funciones internas **int()** y **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
>>>
```


Conversiones de cadenas

- También se pueden usar `int()` y `float()` para conversiones entre **strings** y **enteros**
- Obtendremos un error si el string no tiene un **número bien formado**

```
>>> 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()
```

Funciones escritas por nosotros...

Construyendo nuestras propias funciones

- Creamos una nueva función utilizando la palabra clave **def** seguida por el **nombre** de la función y por los **parámetros** (opcionalmente) entre paréntesis
- **Indentamos** el código de la función
- Esto define la función pero **no ejecuta** el cuerpo de la función

```
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

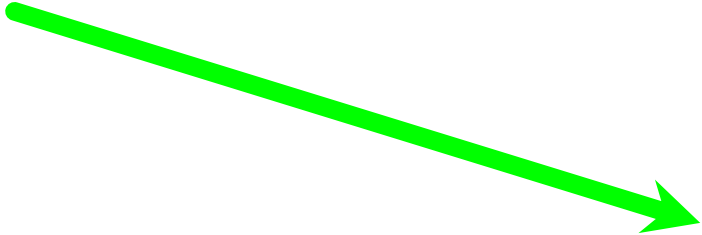
Definiciones y Uso

- Una vez que hemos definido una función, podemos llamarla (o **invocarla**) tantas veces como queramos
- En esto consiste el patrón **almacenar y reutilizar**

```
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

- Un argumento es un **valor** que pasamos a la función como **entrada** cuando llamamos a la función
- Utilizamos argumentos para que la función pueda hacer **diferentes cosas** cuando la llamemos en **diferentes ocasiones**
- Ponemos los argumentos en paréntesis después del **nombre** de la función

big = max('Hello world')



Argumento

Parámetros

- Un parámetro es una **variable** que usamos en la definición de una función. Es un "**manejador**" que permite al código de la función **acceder a los argumentos** utilizados en una invocación en particular de una función.

```
>>> 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

- Frecuentemente una función toma sus argumentos, hace algún cálculo, y **retorna un valor** para ser usado como resultado de la función en la expresión de llamada. La palabra clave **return** se utiliza para esto.

```
def greet():  
    return "Hello"  
  
print(greet(), "Glenn")  
print(greet(), "Sally")
```

```
Hello Glenn  
Hello Sally
```

Valores de Retorno

- Una función “fructífera” (o no vacía) es aquella que produce un resultado (o valor de retorno)
- La sentencia return finaliza la ejecución de la función y "envía de vuelta" el resultado de la función

```
>>> 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, Parámetros y Resultados

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

Argumento → 'Hello world'

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

Parámetro

'w'
↑
Resultado

Múltiples Parámetros / Argumentos

- Podemos definir más de un parámetro en la definición de una función
- Simplemente añadimos más argumentos cuando llamemos a la función
- Los argumentos y parámetros se "**emparejan**" **por el orden** usado y deben coincidir en número

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

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

8

Funciones void (vacías)

- Cuando una función no retorna un valor, se dice que estamos llamando a una función vacía (void)
- Las funciones que retornan valores se denominan "no vacías"
- Esto no quiere decir que hayan "fallado", sino que no retornan nada, simplemente **ejecutan un conjunto de instrucciones**

Función o No función...

- Organiza tu código en "**párrafos**" – captura una **idea** completa y "dale un nombre"
- No te repitas – haz que funcione una vez y entonces **reutiliza**
- Si algo se vuelve demasiado largo o complejo, **divídelo** en trozos lógicos y pon estos trozos en funciones
- Cuando tengas muchas funciones con código que utilizas una y otra vez, haz una **librería** – quizás compartir con otros...

Resumen

- Funciones
- Funciones internas
- Conversiones de tipo (int, float)
- Conversiones de cadenas
- Parámetros
- Argumentos
- Resultados(funciones no vacías)
- Funciones vacías (void)
- ¿Por qué utilizar funciones?

Ejercicio

Reescribe el cálculo del pago por horas de la unidad anterior y escribe una función llamada `calculapago` que toma dos parámetros (horas y precio) y **retorna** el total a pagar.

Introduzca las horas: 45

Introduzca el precio/hora: 10

Total: 475.0

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

Ejercicio

Es decir, debes definir una función con la siguiente cabecera:

```
def calculapago(horas,precio_hora):
```

La función realizará el cálculo del total a pagar y acabará retornando el pago total con **return**

Desde fuera de la función, deberías poder hacer algo así como:

```
print(calculapago(horas,precio_hora))
```

Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance (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.

Continue...

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

... Insert new Contributors and Translators here
Spanish Version: Daniel Garrido (dgm@uma.es)