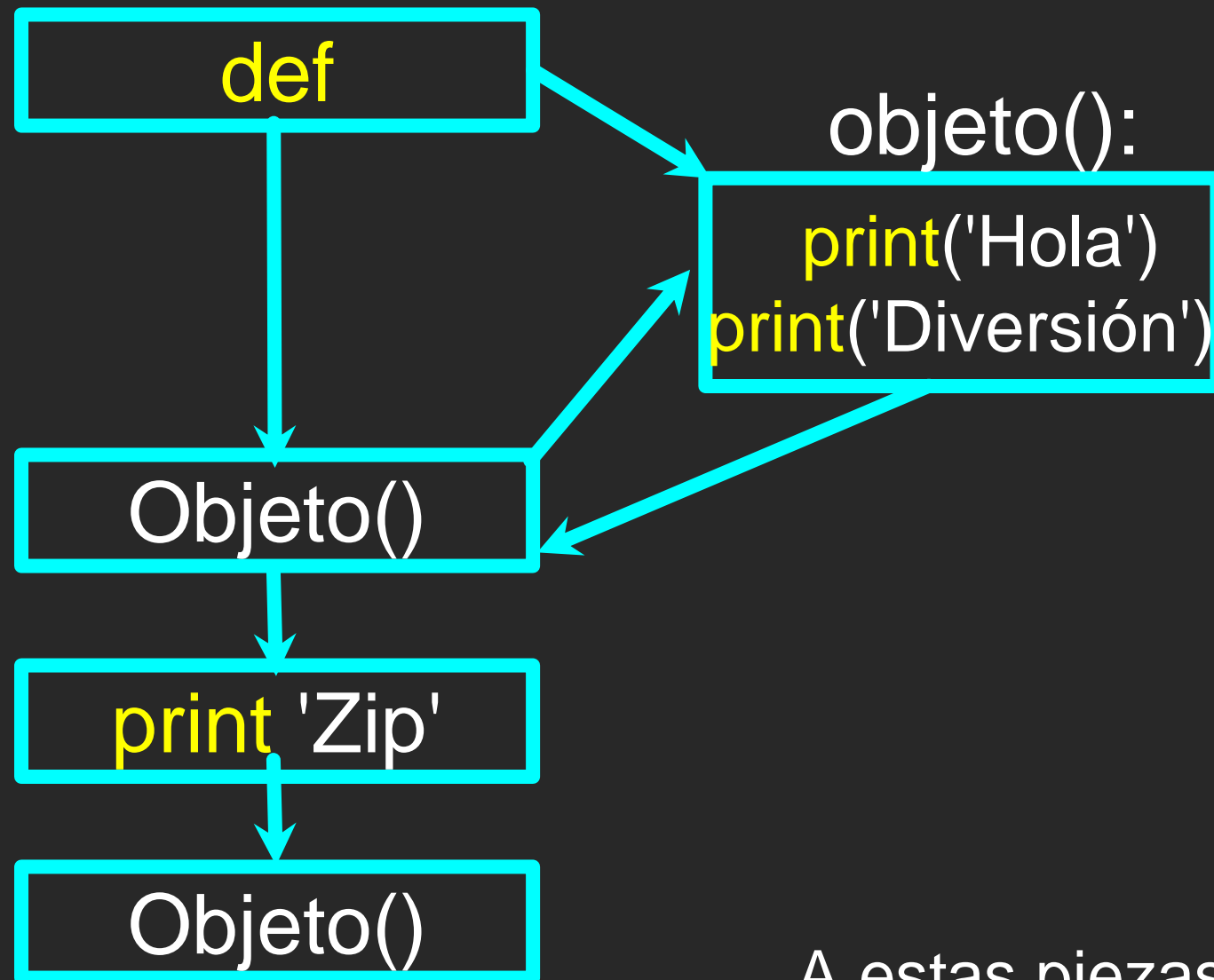


Funciones

Pasos Almacenados (y reutilizados)



Programa:

```
def objeto():  
    print('Hola')
```

```
print('Diversión')
```

```
objeto()  
print('Zip')  
objeto()
```

Resultado:

Hola
Diversión
Zip
Hola
Diversión

A estas piezas de códigos reutilizables las denominamos “funciones”

Funciones de Python

- Existen dos tipos de funciones en Python.
 - **Funciones incorporadas** que se presentan como parte de Python - `print()`, `input()`, `type()`, `float()`, `int()` ...
 - **Funciones que nosotros definimos** y luego utilizamos
- Tratamos a los nombres de las funciones incorporadas como “nuevas” **palabras reservadas** (es decir, las evitamos como nombres de variables)

Definición de la Función

- En Python una **función** es un código reutilizable que toma **argumentos**(s) como input, realiza algunos cálculos y luego devuelve uno o más resultado(s)
- Para definir una **función** utilizamos la palabra reservada **def**
- Llamamos/Invocamos a la **función** utilizando una expresión que contenga el nombre de la función, paréntesis y **argumentos**

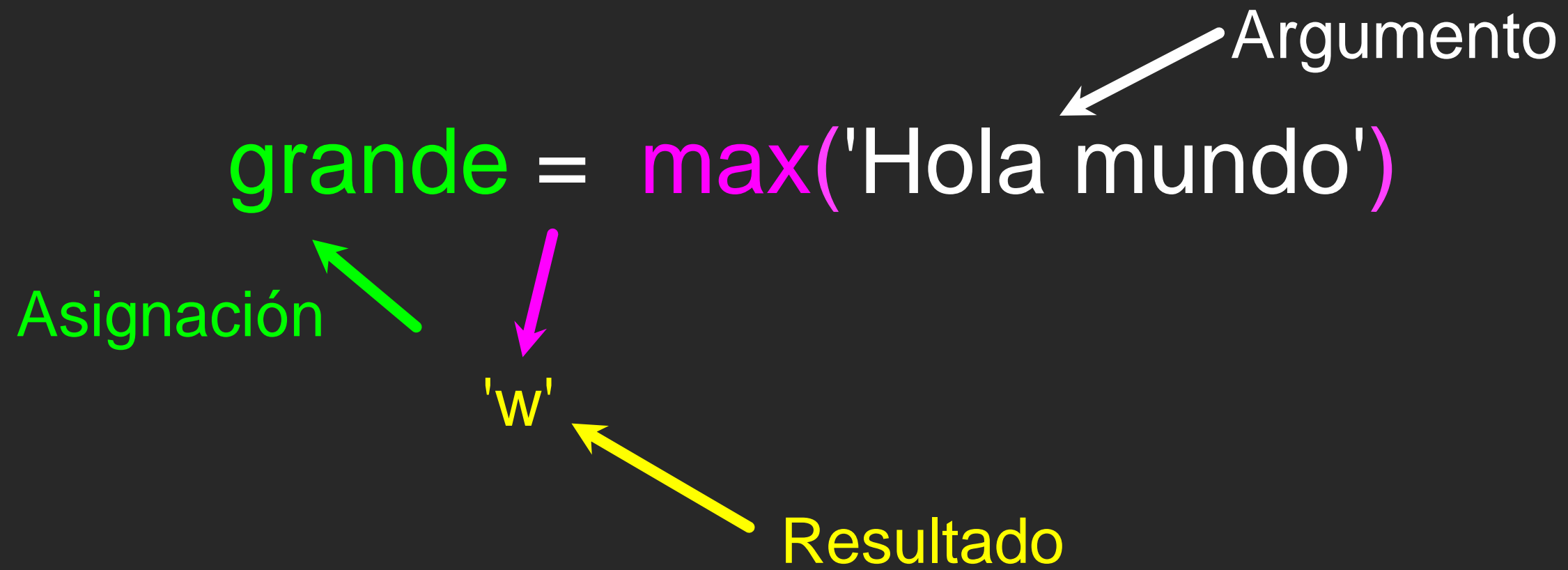
Argumento

Asignación

'w'

Resultado

`grande = max('Hola mundo')`



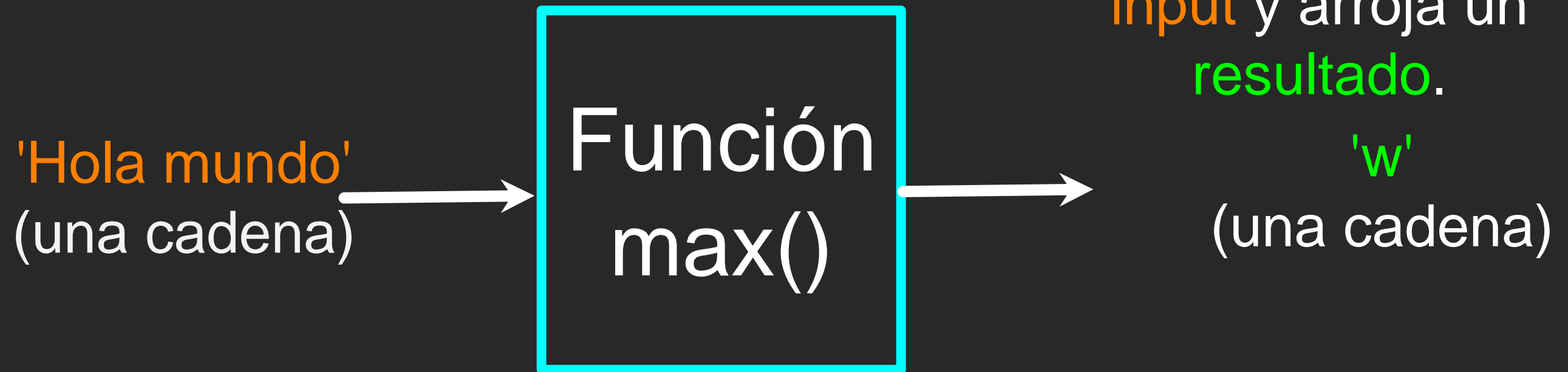
```
>>> grande = max('Hola mundo')
>>> print(grande)
w
>>> pequeño = min('Hola mundo')
>>> print(pequeño)

>>>
```

Función Max

Una función es un código almacenado que nosotros utilizamos. Una función toma un **input** y arroja un **resultado**.

```
>>> grande = max('Hola mundo')  
>>> print(grande)  
w
```



Guido escribió este código

Función Max

Una función es un código almacenado que nosotros utilizamos. Una función toma un **input** y arroja un **resultado**.

```
>>> grande = max('Hola mundo')
>>> print(grande)
w
```

'Hola mundo'
(una cadena)



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



'w'
(una cadena)

Guido escribió este código

Conversiones de Type (Tipo)

- Cuando coloca un número entero y un punto flotante en una expresión, el número entero **implícitamente** se convierte en decimal
- Puede controlar esto con las funciones incorporadas `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 puede usar `int()` y `float()` para convertir entre cadenas y valores enteros
- Se mostrará **error** si la cadena no contiene caracteres numéricos

```
>>> sval = '123'
>>> type(sval)
<class 'str'>
>>> print(sval + 1)
Traza de rastreo (llamada más reciente
a lo último):
  Archivo "<stdin>", línea 1, in
<module>
TypeError: cannot concatenate 'str'
and 'int'
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print(ival + 1)
124
>>> nsval = 'hola bob'
>>> niv = int(nsval)
Traza de rastreo (llamada más reciente
a lo último):
  Archivo "<stdin>", línea 1, in
<module>
ValueError: invalid literal for int()
```

Una Función Propia

Construyendo Nuestras Propias Funciones

- Creamos una nueva función usando la palabra clave **def** seguida de parámetros opcionales entre paréntesis
- Indentamos el cuerpo de la función
- Esto **define** la función pero **no** ejecuta el cuerpo de la función

```
def print_lyrics():  
    print("Soy un leñador, y estoy bien.")  
    print('Duermo toda la noche y trabajo todo el  
día.')
```

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

```
def print_lyrics():  
    print("Soy un leñador, y estoy bien.")  
    print('Duermo toda la noche y trabajo todo el  
día.')
```

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

`print_lyrics():`

```
print "Soy un leñador, y estoy bien."  
print 'Duermo toda la noche y trabajo  
todo el día.'
```

Hola
Yo
7

Definiciones y Usos

- Una vez que hemos **definido** una función, podemos **llamarla** (o **invocarla**) todas las veces que queramos
- Este es el patrón **almacenar** y **reutilizar**

```
x = 5
```

```
print('Hola')
```

```
def print_lyrics():
```

```
    print("Soy un leñador, y estoy bien.")
```

```
    print('Duermo toda la noche y trabajo todo el  
día.')
```

```
print('Yo')
```

```
print_lyrics()
```

```
x = x + 2
```

```
print(x)
```

Hola

Yo

Soy un leñador, y estoy bien.

Duermo toda la noche y trabajo
todo el día.

7

Argumentos

- Un **argumento** es un valor que informamos a la **función** como su **entrada (input)** cuando llamamos a la función
- Utilizamos **argumentos** para poder instruir a la **función** que realice diferentes tareas cuando la llamamos en **diferentes** oportunidades
- Colocamos los **argumentos** entre paréntesis luego del **nombre** de la función

grande = **max**('Hola mundo')

Argumento



Parámetros

Un **parámetro** es una variable que utilizamos **en** la función **definition** (**definición**). Es una “handle” (palanca) que permite al código de la función acceder a los **argumentos** para invocar una función en particular.

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


Valores de Retorno

A menudo, una función tomará sus argumentos, hará algunos cálculos, y **retornará** un valor que se usará como el valor de la llamada de la función en la **expresión de llamada**. La palabra clave **return (retorno)** se utiliza para esto.

```
def saludo ():  
    return "Hola"
```

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

```
Hola Glenn  
Hola Sally
```

Valor de Retorno

- Una **función** “fructífera” es la que arroja un **resultado** (o **valor de retorno**)
- El enunciado **return** termina la ejecución de la **función** y “devuelve” el **resultado** de la **función**

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

Argumentos, Parámetros, y Resultados

```
>>> grande = max('Hola mundo')  
>>> print(grande)  
w
```

Parámetro

Argumento
→
'Hola mundo'

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

→
'w'
↑
Resultado

Múltiples Parámetros / Argumentos

- Podemos definir más de un **parámetro** en la **definición** de la **función**
- Simplemente agregamos más **argumentos** cuando llamamos a la **función**
- Hacemos coincidir el número y orden de los argumentos y parámetros

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

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

8

Funciones Nulas (no fructíferas)

- Cuando una función no retorna un valor, la denominamos una función “**void**” (**nula**)
- Las funciones que retornan valores son las funciones “fructíferas”
- Las funciones **Void (Nulas)** son “no fructíferas”

Funcionar o no funcionar...

- Organice su código en “párrafos”; capture una idea completa y “póngale un nombre”
- No se repita, hágalo funcionar una vez y luego reutilícelo
- Si algo se vuelve demasiado largo o complejo, desglose en bloques lógicos y coloque esos bloques en funciones
- Haga una biblioteca de objetos comunes que usted repite todo el tiempo, tal vez deba compartirlo con sus amigos...

Síntesis

- Argumentos
- Resultados (funciones fructíferas)
- Funciones Void (nulas, no fructíferas)
- ¿Por qué usar funciones?
- Funciones
- Funciones incorporadas
 - Conversiones de Type (tipo) (int, float)
 - Conversiones de cadenas
- Parámetros