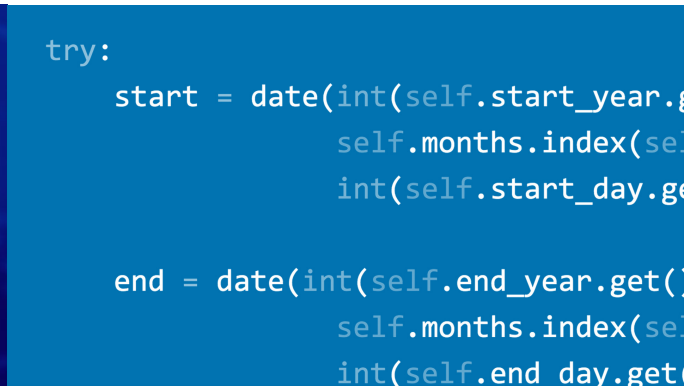
Introducción a Python

funciones

Miguel Carrasco
miguel.carrasco@uai.cl

Temario

- Estructuras de control
- Listas
- Funciones integradas
- Funciones
  - Definición
  - Paso de parámetros
  - Scope de variables
  - Ejemplos y ejercicios
  - Manejo de excepciones



Relevancia

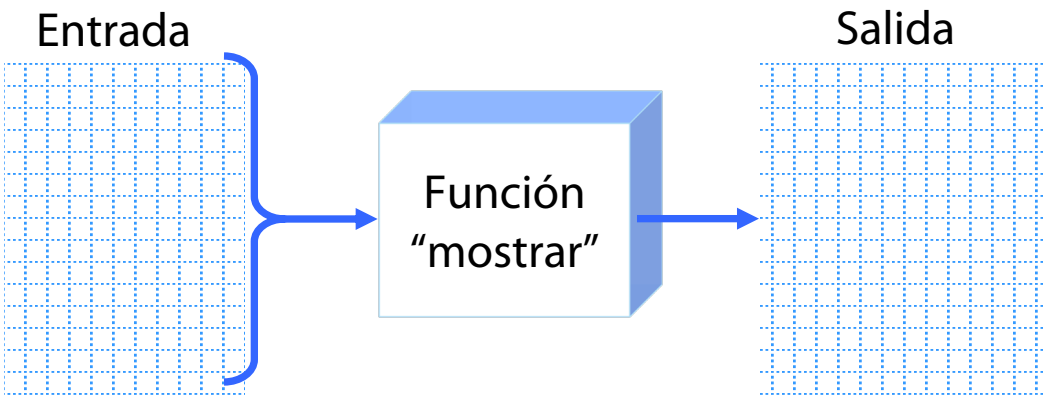
Las funciones nos permiten:

- Ordenar código
- Escribir código de manera rápida
- Compartir código con otros
- Reusar código en distintas partes de uno o más programas
- Entre otras cosas

Definición

Las funciones son pequeños partes de código que pueden ser reutilizados nuevamente.

- Son independientes de nuestro programa.
- Es un subprograma que realiza una tarea específica.
- Puede recibir cero o más valores del programa que los llama y devolver 0 o 1 valor a dicho programa.



Definición

Las funciones se definen antes del código principal.

La función más sencilla no recibe ni retorna ningún valor.

Función

```
def nombreFuncion():
        <instrucciones>
```

```
def holaMundo():
        print("Nuestra primera funcion")
```

Función definición

Las funciones se definen antes del código principal.

La función más sencilla no recibe ni retorna ningún valor.

Una función se puede invocar desde cualquier parte, simplemente escribiendo su nombre como una instrucción

Función

```
def nombreFuncion():
        <instrucciones>
```

invocación o llamado

```
def holaMundo():
        print("Nuestra primera funcion")

holaMundo()
```

Función definición

Una función puede recibir uno o más parametros (valores). Estos valores son copiados dentro de la función y pueden ser empleados dentro de ella. Sólo la función los puede emplear.

Función

```
def nombreFuncion(<var1>,…, <varn>):
        <instrucciones>
```

```
def mostrarPantalla(entrada):
      print (entrada)


mostrarPantalla("holaMundo")
```

invocación o llamado

Función definición

La función puede finalizar en cualquier parte de su código al alcanzar la instrucción return.

Función

```
def nombreFuncion(<var1>,…, <varn>):
        <instrucciones>
        return
```

```
def sumaNumeros(num1, num2):
    resultado = num1 + num2
    print(resultado)
    return


sumaNumeros(21, 13)
```

invocación o llamado

Función definición

La función puede finalizar en cualquier parte de su código al alcanzar la instrucción return.

Función

```
def nombreFuncion(<var1>,…, <varn>):
        <instrucciones>
        return
```

```
def numEsPar(num):
    if (num%2==0):
        print ("SI es par")
        return
    print ("NO es par")
    return

numEsPar(21)
```

invocación o llamado

Función definición

La función puede devolver un valor y retornarlo. El valor devuelto debe estar como argumento de return()

Función

```
def nombre_funcion(parámetro_1, parámetro_2, …):
        <instrucciones>
        return <valor>
```

invocación o llamado

```
def holaMundo(nombre):
    return(f'Hola {nombre} desde la función')

nombre_usuario = input('Ingresa tu nombre')
saludo = holaMundo(nombre_usuario)
print(saludo)
```

Función definición

La función puede devolver un valor y retornarlo. El valor devuelto debe estar como argumento de return()

Función

```
def nombreFuncion(<var1>,…, <varn>):
        <instrucciones>
        return(<resultado>)
```

```
def numEsPar(num):
    if (num%2==0):
        return(1)
    return(0)


if (numEsPar(21)==1):
    print("ES par")
else:
    print("NO es par")
```

invocación o llamado

Return permite que se retorne el resultado a quien invoca la función

Función definición

La función puede devolver un valor y retornarlo. El valor devuelto debe estar como argumento de return()

Función

```
def nombreFuncion(<var1>,…, <varn>):
        <instrucciones>
        return(<resultado>)
```

```
def numPrimo(num):
    for i in range(2,num,1):
        if (num%i==0):
            return(0)
    return(1)

if (numPrimo(21)==0):
    print ("NO primo")
else:
    print ("SI primo")
```

invocación o llamado

Return permite que se retorne el resultado a quien invoca la función

Resumen

No retorna ningún valor

```
def holaMundo():
    print('Primera funcion')
```

llamado

```
holaMundo()
```

Recibe un parámetro de entrada

```
def mostrarPantalla(entrada):
      print (entrada)
      return
```

llamado

```
mostrarPantalla("holaMundo")
```

Recibe un parámetro de entrada

```
def numPrimo(num):
    for i in range(2,num,1):
        if (num%i==0):
            print ("NO primo")
            return
    print ("SI primo")
    return
```

llamado

```
numPrimo(21)
```

Envía y retorna valores

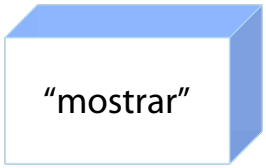
```
def holaMundo(nombre):
      return f'Hola {nombre}'
```

llamado

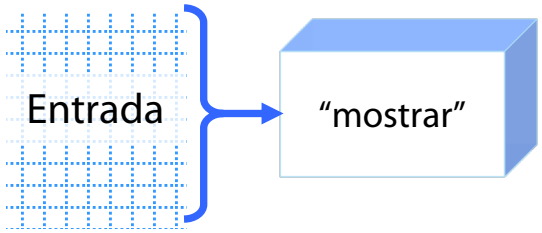
```
usuario = input('Ingresa nombre:')
saludo = holaMundo(usuario)
print(saludo)
```

Resumen

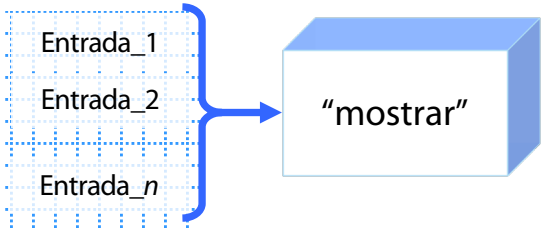
No retorna ningún valor



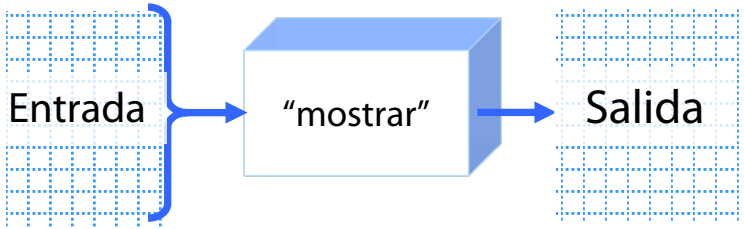
Recibe un parámetro de entrada

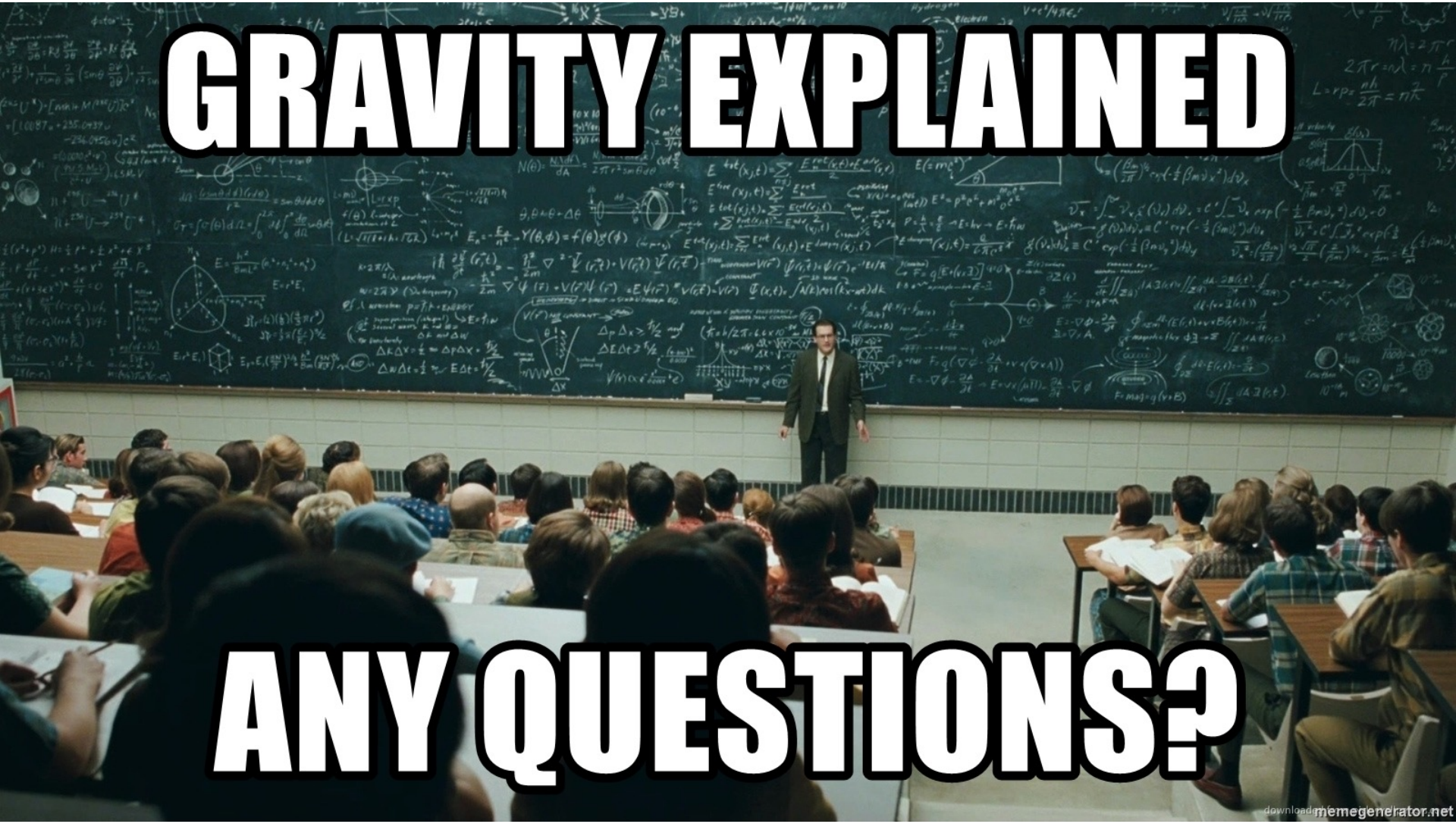


Recibe varios parámetros de entrada



Envía y retorna valores




GRAVITY EXPLAINED
ANY QUESTIONS?
memegenerator.net

Temario

- Estructuras de control
- Listas
- Funciones integradas
- Funciones
  - Definición
  - Scope de variables
  - Paso de parámetros
  - Ejemplos y ejercicios
  - Manejo de excepciones


```
try:
    start = date(int(self.start_year.
                     self.months.index(se
                     int(self.start_day.ge

    end = date(int(self.end_year.get(
                     self.months.index(se
                     int(self.end_day.get
```

Scope

Las variables tienen un determinado alcance según donde éstas sean creadas y empleadas.

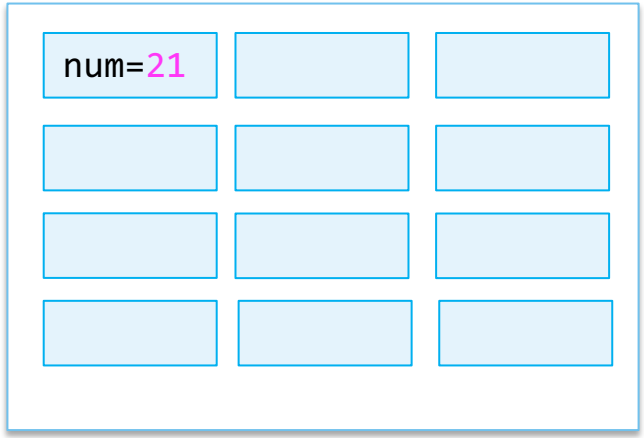
Las variables que son creadas dentro de una función, solo “existen”, mientras el programa se encuentre dentro de ésta.

```
num = 21

def mifuncion(num):
    num = num+1
    print('dentro de la función', num)
    return

mifuncion(num)
print('fuera de la funcion', num)
```


A veces me acuerdo que lo q pasa en las vegas se queda en las vegas
Luego me acuerdo q no voy a ir y se me pasa
Imagen creada en GeneradorMemes.com

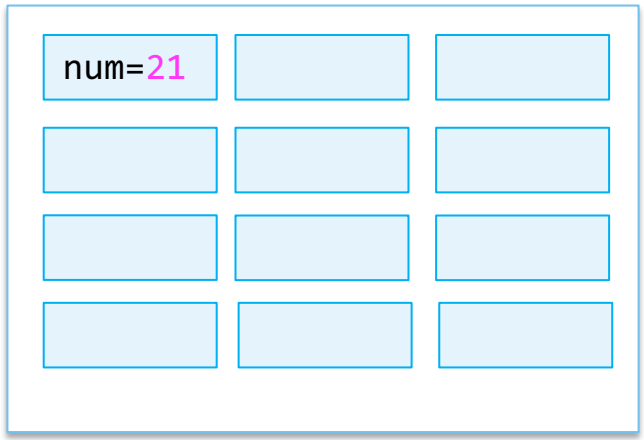
Scope

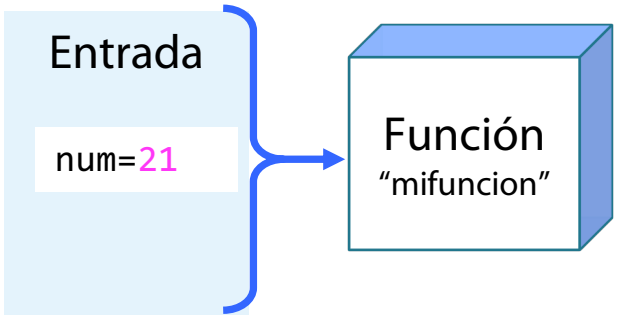
```
num = 21

def mifuncion(num):
    num = num+1
    print('dentro de la función', num)
    return

mifuncion(num)
print('fuera de la funcion', num)
```



Scope

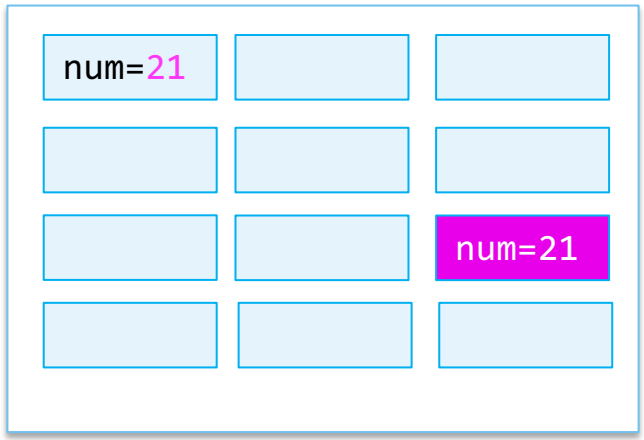
```
num = 21

def mifuncion(num):
    num = num+1
    print('dentro de la función', num)
    return

mifuncion(num)
print('fuera de la funcion', num)
```



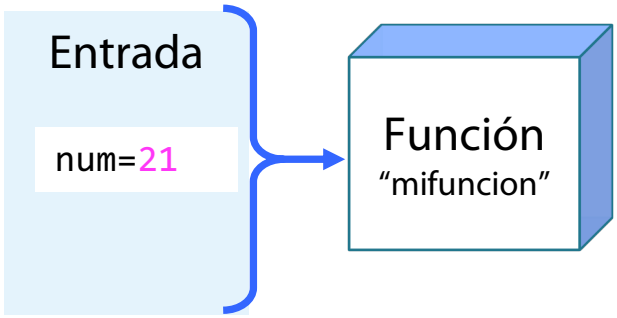


Scope

```
num = 21

def mifuncion(num):
    num = num+1
    print('dentro de la función', num)
    return

mifuncion(num)
print('fuera de la funcion', num)
```

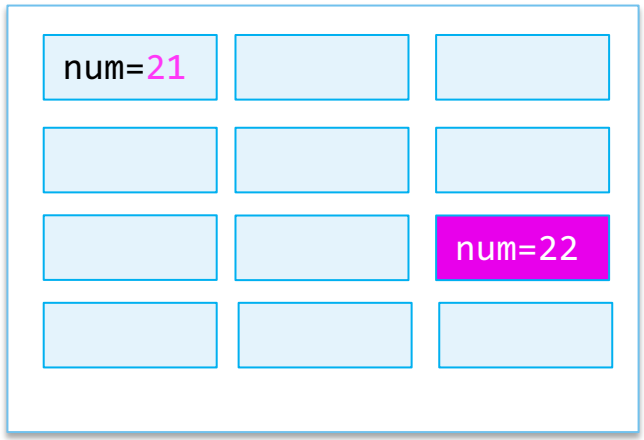


esta es otra variable num en la memoria

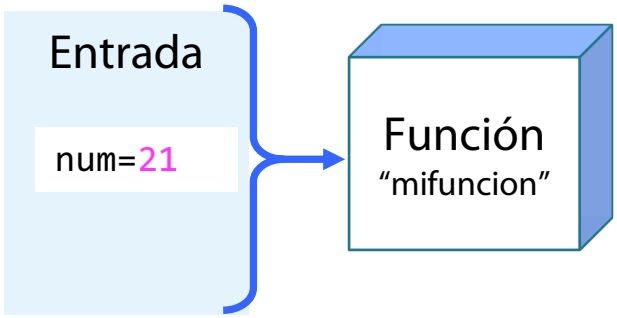


Scope

```
num = 21

def mifuncion(num):
    num = num+1
    print('dentro de la función', num)
    return

mifuncion(num)
print('fuera de la funcion', num)
```

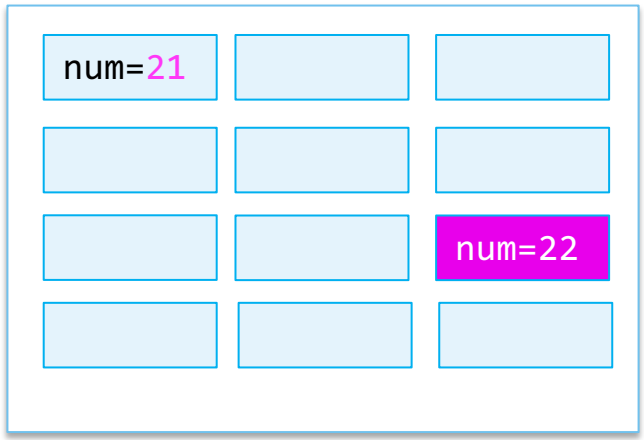


esta es otra variable num en la memoria

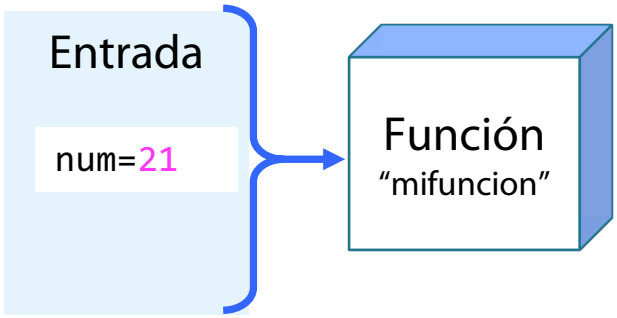


Scope

```
num = 21

def mifuncion(num):
    num = num+1
    print('dentro de la función', num)
    return

mifuncion(num)
print('fuera de la funcion', num)
```

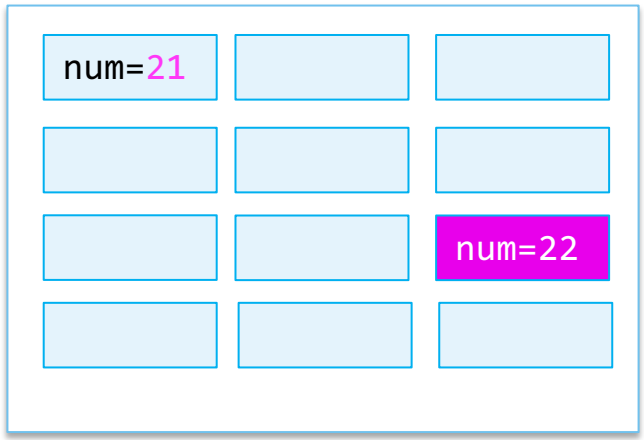


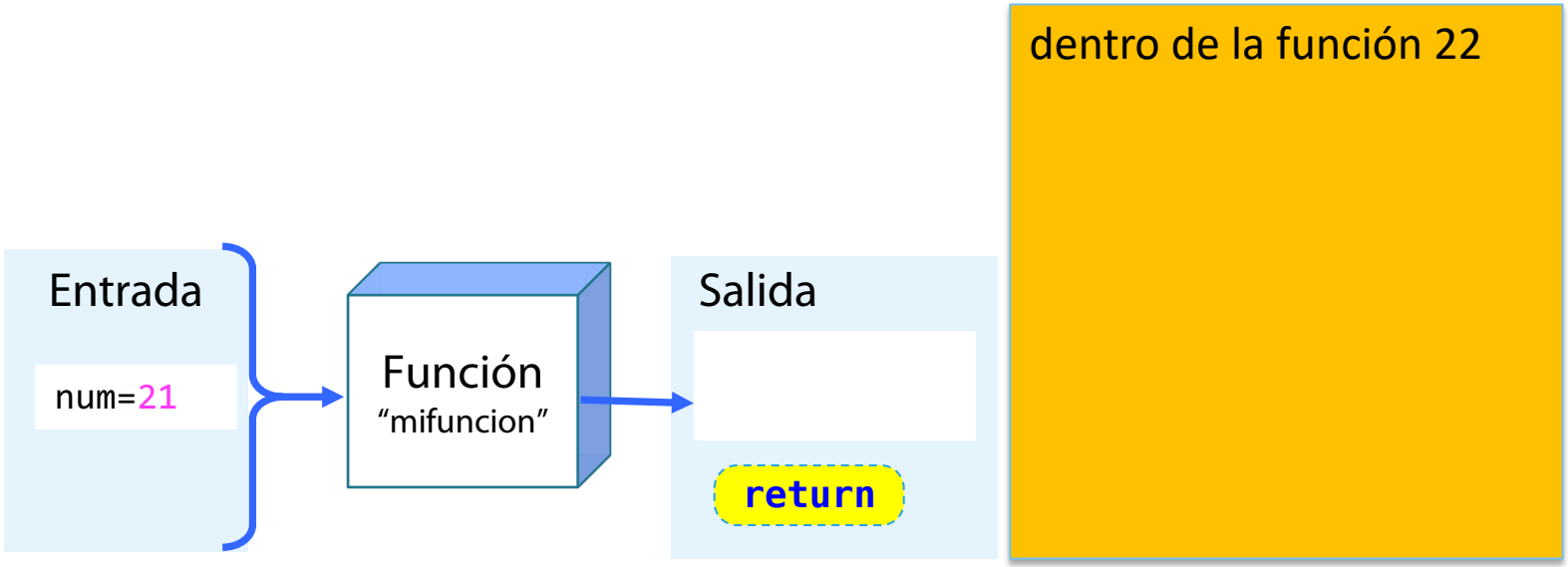
dentro de la función 22



Scope

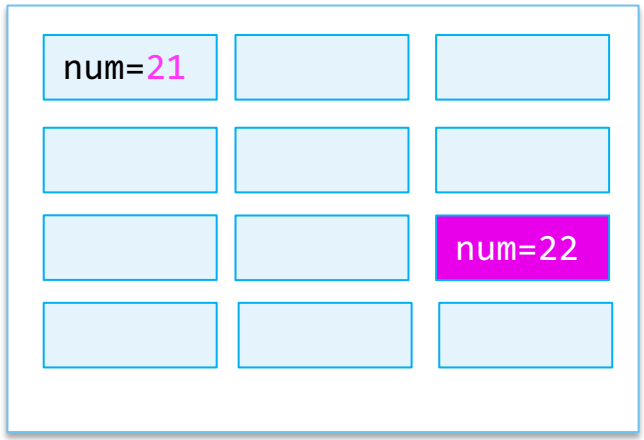
```
num = 21

def mifuncion(num):
    num = num+1
    print('dentro de la función', num)
    return

mifuncion(num)
print('fuera de la funcion', num)
```

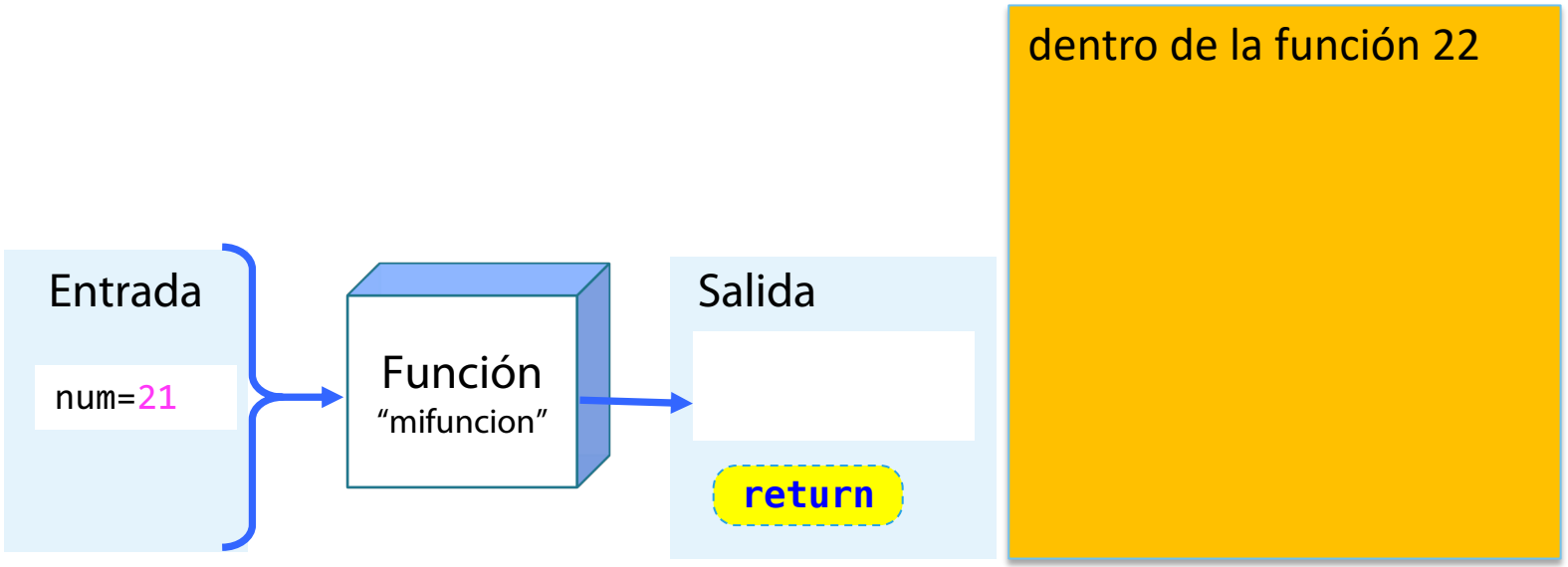




Scope

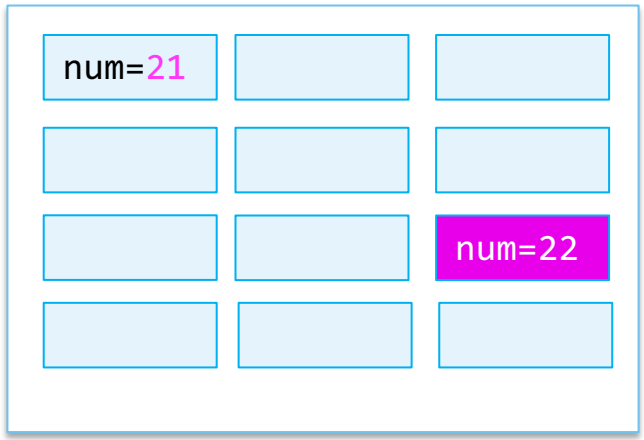
```
num = 21

def mifuncion(num):
    num = num+1
    print('dentro de la función', num)
    return

mifuncion(num)
print('fuera de la funcion', num)
```

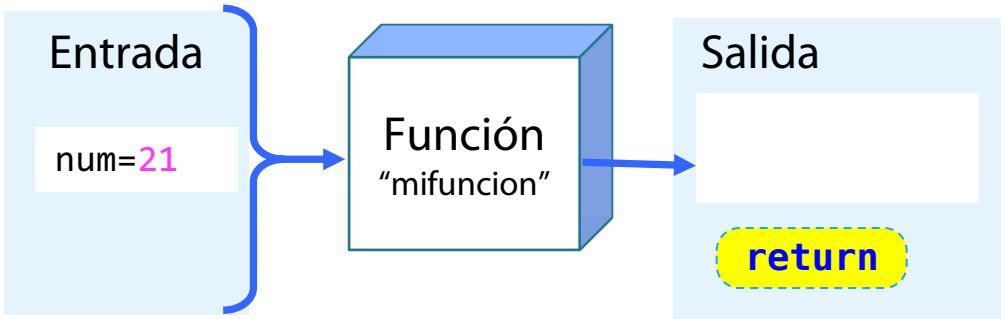




Scope

```
num = 21

def mifuncion(num):
    num = num+1
    print('dentro de la función', num)
    return

mifuncion(num)
print('fuera de la funcion', num)
```





dentro de la función 22
fuera de la función 21

Temario

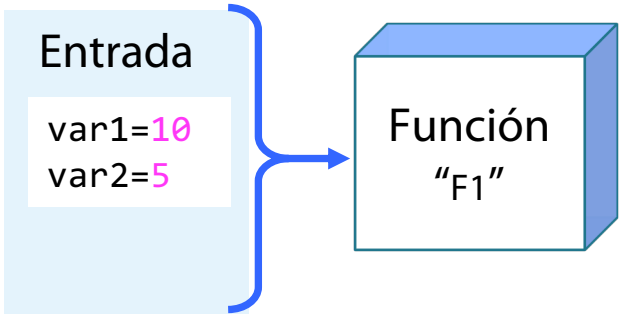
- Estructuras de control
- Listas
- Funciones integradas
- Funciones
  - Definición
  - Scope de variables
  - Paso de parámetros
  - Ejemplos y ejercicios
  - Manejo de excepciones


```
try:
    start = date(int(self.start_year.
                     self.months.index(se
                     int(self.start_day.ge

    end = date(int(self.end_year.get(
                     self.months.index(se
                     int(self.end_day.get
```

Paso de valores

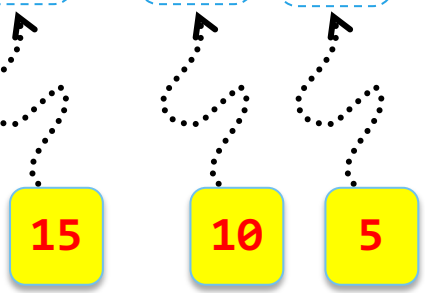
```
var1 = 10
var2 = 5
var2 = F1(var1, var2)

print("var1 es ", var1)
print("var2 es ", var2)
```

Suponga que tenemos la función **F1** a la cual enviamos una copia del valor 10 y 5.



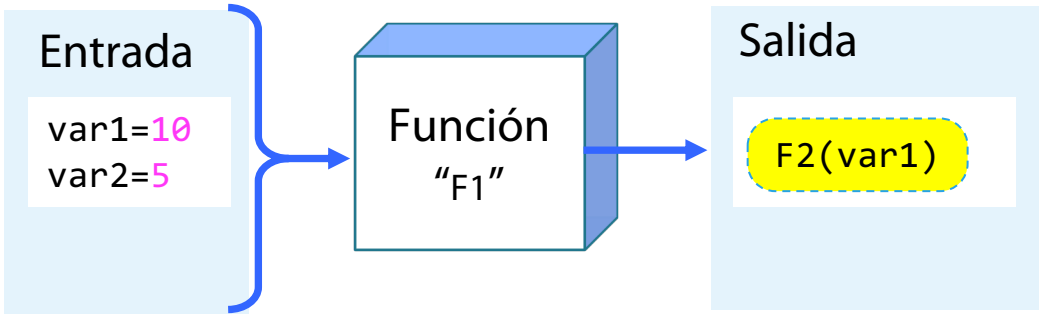
Paso de valores

```
def F1(var1, var2):
    var1 = var1+var2
```




```
var1 = 10
var2 = 5
var2 = F1(var1, var2)

print("var1 es ", var1)
print("var2 es ", var2)
```

Suponga que tenemos la función **F1** a la cual enviamos una copia del valor 10 y 5.



Paso de valores

```
def F1(var1, var2):
        var1 = var1+var2
        return (F2(var1))
```

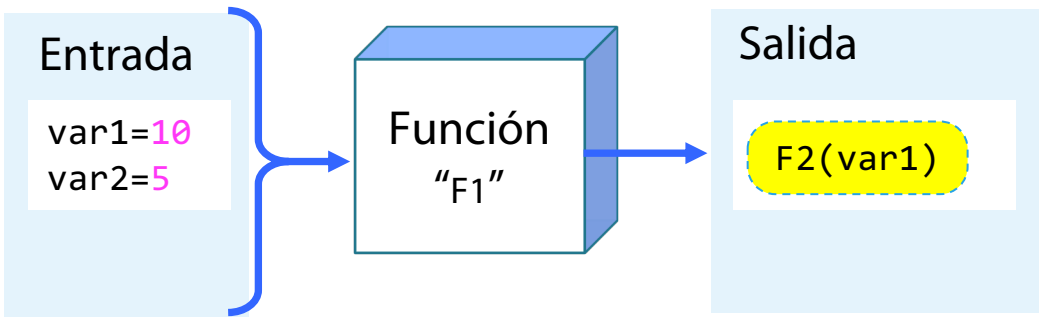
`return` permite que se retorne el resultado a quien invoca la función

15

```
var1 = 10
var2 = 5
var2 = F1(var1, var2)

print("var1 es ", var1)
print("var2 es ", var2)
```

Suponga que tenemos la función **F1** a la cual enviamos una copia del valor 10 y 5

Una vez la función alcanza la instrucción **return**, se prepara para retornar el resultado, hasta obtener el valor de todas las variables.

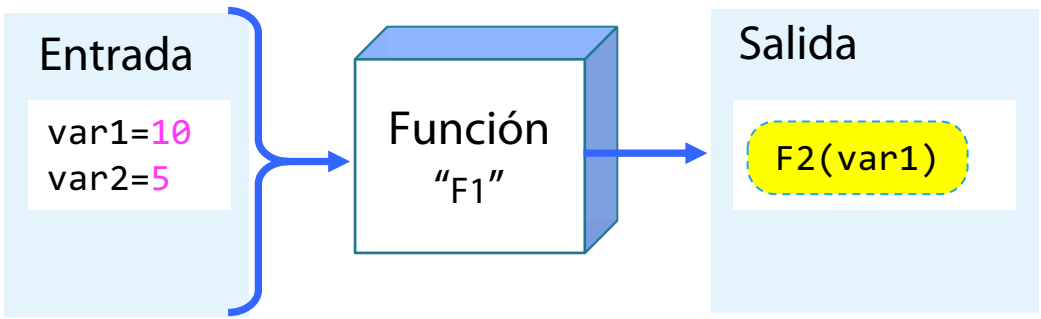


Paso de valores

```
def F1(var1, var2):
        var1 = var1+var2
        return (F2(var1))


def F2(aux):
        aux = aux*2
        return (aux)


var1 = 10
var2 = 5
var2 = F1(var1, var2)

print("var1 es ", var1)
print("var2 es ", var2)
```

Una vez la función alcanza la instrucción return, se prepara para retornar el resultado, hasta obtener el valor de todas las variables.

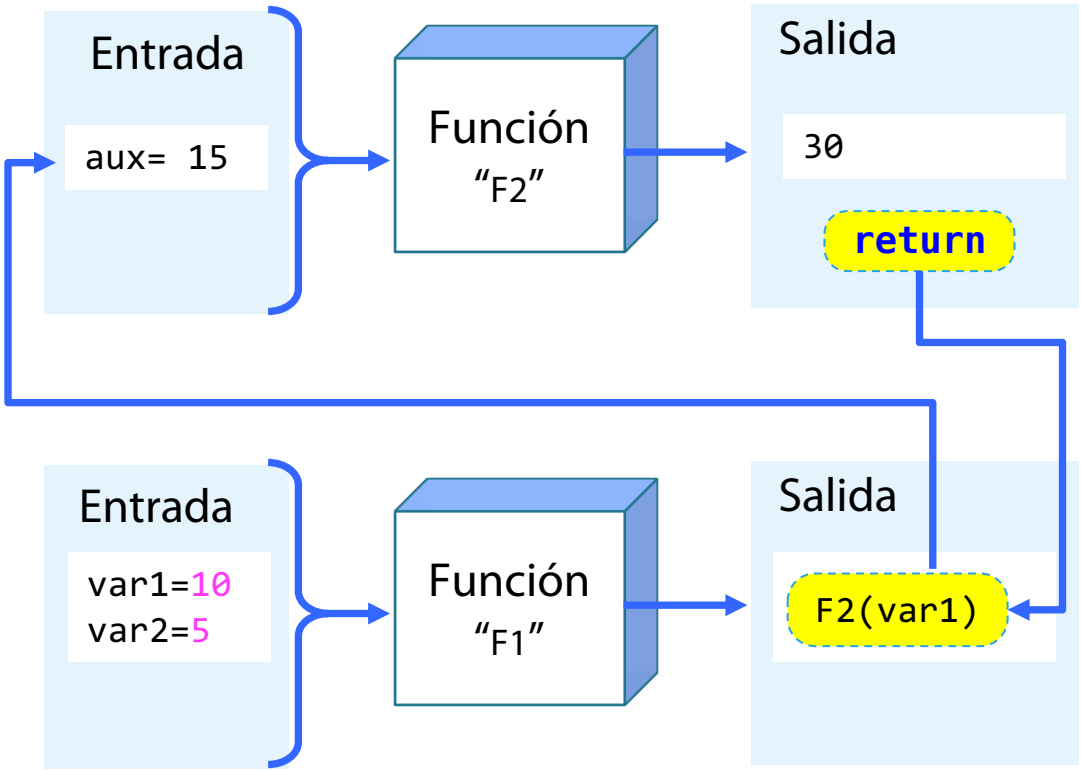
Como observamos en el programa, la función debe retornar otra función. Esto significa que debemos **evaluar** esa función y calcular su resultado



Paso de valores

```
def F1(var1, var2):
        var1 = var1+var2
        return (F2(var1))


def F2(aux):
        aux = aux*2
        return (aux)


var1 = 10
var2 = 5
var2 = F1(var1, var2)

print("var1 es ", var1)
print("var2 es ", var2)
```

El programa ingresa a otra función y devuelve el resultado a quien lo invocó.

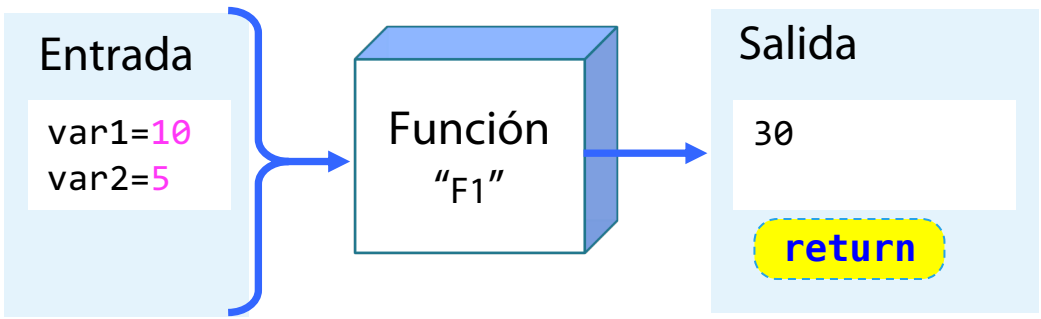


Paso de valores

```
def F1(var1, var2):
        var1 = var1+var2
        return (F2(var1))


def F2(aux):
        aux = aux*2
        return (aux)


var1 = 10
var2 = 5
var2 = F1(var1, var2)

print("var1 es ", var1)
print("var2 es ", var2)
```

Observe como la función F1 ahora posee un valor. Este valor será retornado a quien llamó a la función.

30



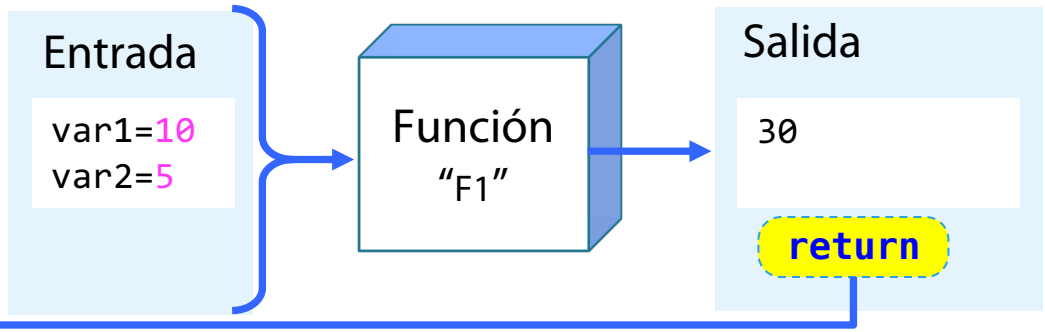
Paso de valores

```
def F1(var1, var2):
        var1 = var1+var2
        return (F2(var1))
```

¿Qué podemos hacer con este valor? Resp. Lo podemos asignar a otra variable o bien mostrar por pantalla

La función invocada se transforma en un valor

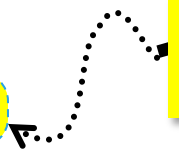
```
var1 = 10
var2 = 5
var2 = F1(var1, var2)

print("var1 es ", var1)
print("var2 es ", var2)
```



Paso de valores

```
def F1(var1, var2):
        var1 = var1+var2
        return (F2(var1))
```

```
var1 = 10
var2 = 5
var2 = 30

print("var1 es ", var1)
print("var2 es ", var2)
```



La función invocada se transforma en un valor

Para retornar el resultado, empleamos la instrucción return.

¿Qué podemos hacer con este valor?

Resp. Lo podemos asignar a otra variable o bien mostrar por pantalla

Traspaso por valor

Se denomina a la forma por defecto en que un valor es traspasado del código principal o función a otra función.

```
def F1(var1, var2):
        var1 = var2/var1
        return (var1)

var1 = 10
var2 = 5
var2 = F1(var1, var2)
print("var1 es ", var1, " y var2 es ", var2 )
```

A la variable var1 de F1 se le asigna el valor 5, mientras que a la variable var2 de F1 se le asigna el valor 10.

El valor de var1 (10) y var2 (5) se envían a la función F1, no la variable

Como se puede observar, se traspasan los valores de las variables **NO las variables en sí.**

Temario

- Estructuras de control
- Listas
- Funciones integradas
- Funciones
  - Definición
  - Paso de parámetros
  - Scope de variables
  - Ejemplos y ejercicios
  - Manejo de excepciones

```
try:
    start = date(int(self.start_year.
                     self.months.index(se
                     int(self.start_day.ge

    end = date(int(self.end_year.get(
                     self.months.index(se
                     int(self.end_day.get
```

Ejercicio

Tiempo : 5 minutos

Cree una función que reciba un texto (varTexto) y un número (varNum) como parámetros y muestre por pantalla el texto un número varNum de veces.

```
def repetirTexto(varTexto, varNum):
     for i in range(varNum)
        print (varTexto)
```

Ejercicio

Tiempo : 10 minutos ¿Qué realizan las siguientes funciones?

```
def F1(var1, var2 , var3):
    if (var1<var2):
         var1=var2
    if (var1<var3):
         var1=var3
    print (var1)
    return
```

Muestra el mayor de los 3 números

```
def F2(var1, var2 , var3):
    flag = 1
    if (var1>var2):
         flag = 0
    else:
         if (var2>var3):
              flag = 0
    return flag
```

retorna 1 si los números ingresados están ordenados de menor a mayor, 0 en caso contrario.

Variables locales

Cada función tiene sus propias variables (no las comparte con ninguna otra función).

Los parámetros también son variables propias de la función.

Por ejemplo, las 3 variables de la función F1, son distintas a las 3 variables existentes en algoritmo.

invocación o llamado

```
def F1(var1, var2 , var3):
        print( "en la funcion", var1+var2 +var3 )

F1(4,5,6)
print( "en algoritmo", var1+var2 +var3)
```

Estas variables no han sido definidas en el código principal, por lo cual se produce un error

Ejercicio

Tiempo : 5 minutos ¿Qué valores se muestran por pantalla?

```
def suma(varA, varB):
        varA = varA+varB
        return (varA)

var1 = 10
var2 = 5
var2 = suma(var1, var2)
print("var1 es ", var1, " y var2 es ", var2 )
```

El programa mostrará que var1 es 10 y var2 es 15.

Ejercicio

Tiempo : 10 minutos

```
def cambios(x, y)
        3->
        x = x+1
        4->
        y = 20
        return y

a = 10
1->
b = 5
2->
b= cambios(b, a)
5->
```

Las flechas numeradas identifican instantes de tiempo de ejecución del siguiente código. Por ejemplo: 1-> señaliza el instante inmediatamente después de ejecutar la instrucción a=10. Completa la tabla con los valores de las variables en los instantes de tiempo señalados. Ponga XX en los bloques en los que la variable está fuera de su ámbito y -- si el valor de la variable no ha sido inicializado.

	a	b	x	y
1->	10	--	XX	XX
2->				
3->				
4->				
5->				

Ejercicio

Tiempo : 10 minutos

Escriba una función en Python que reciba dos arreglos, y en caso que ambos arreglos tengan la misma cantidad de dimensiones calcule la distancia entre los puntos. Caso contrario devuelva -1.
Hint: la biblioteca numpy tiene las funciones square, sum y sqrt

Ejemplo:
Para el código
P1 = numpy.array([0, 0])
P2 = numpy.array([1, 1])
dist(P1,P2)
Debe mostrar
La distancia es 1.4142

Ejercicio

Tiempo : 10 minutos

Escriba una función en Python que reciba dos arreglos, y en caso que ambos arreglos tengan la misma cantidad de dimensiones calcule la distancia entre los puntos. Caso contrario devuelva -1.
Hint: la biblioteca numpy tiene las funciones square, sum y sqrt

```
import numpy
import math

def dist(p1,p2):
  if (len(p1)!=len(p2)):
    return(-1)
  temp=p1-p2
  temp=pow(temp,2)
  temp=numpy.sum(temp)
  return(math.sqrt(temp))

P1 = numpy.array([0, 0])
P2 = numpy.array([1, 1])
print(dist(P1,P2))
```

Ejercicio

Tiempo : 10 minutos

Escriba una función en Python que reciba dos arreglos, y en caso que tengan el mismo número de elementos, se compare elemento por elemento.

La función deberá retornar un arreglo de 3 valores, en el primer indice se indicará el número de elementos del arreglo A que son menores que el arreglo B. El segundo índice, el número de elementos donde ambos arreglos son iguales, y el tercer índice, el número de elementos del arreglo A que son mayores que el arreglo B.

Ejemplo:
Para el código

```
P1 = numpy.array([0, 0, 2, 4, 1, 2])
P2 = numpy.array([1, 1, 3, 4, 0, 2])
comp(P1,P2)
```

Deberá retornar el arreglo [3, 2, 1]

Ejercicio

Tiempo : 10 minutos

Escriba una función en Python que reciba dos arreglos, y en caso que tengan el mismo número de elementos, se compare elemento por elemento.

La función deberá retornar un arreglo de 3 valores, en el primer indice se indicará el número de elementos del arreglo A que son menores que el arreglo B. El segundo índice, el número de elementos donde ambos arreglos son iguales, y el tercer índice, el número de elementos del arreglo A que son mayores que el arreglo B.

```
import numpy
def comp(Array1,Array2):
  res = numpy.array([-1,-1,-1])
  if (len(Array1)!=len(Array2)):
    return(res)

  res[0]=numpy.sum(Array1<Array2)
  res[1]=numpy.sum(Array1==Array2)
  res[2]=numpy.sum(Array1>Array2)
  return(res)

P1 = numpy.array([0, 0, 2, 4, 1, 2])
P2 = numpy.array([1, 1, 3, 4, 0, 2])
print(comp(P1,P2))
```

Temario

- Estructuras de control
- Listas
- Funciones integradas
- Funciones
  - Definición
  - Paso de parámetros
  - Scope de variables
  - Ejemplos y ejercicios
  - Manejo de excepciones

```
try:
    start = date(int(self.start_year.
                     self.months.index(se
                     int(self.start_day.ge

    end = date(int(self.end_year.get(
                     self.months.index(se
                     int(self.end_day.get
```

Exception

Suponga que su programa requiere emplear datos ingresados por teclado, donde inevitablemente puedan haber errores de parte del usuario. ¿Cómo podemos evitar que el programa se detenga?

```
def foo():
    res= 1/0
    return(res)

foo()
```

Exception

error generado por Python

```
Traceback (most recent call last):

  File "<ipython-input-100-7b7d59c006ee>", line 1, in <module>

runfile('/Users/mlacarrasco/Dropbox/_BACKUP/Asignaturas/2019/sem_01/MagisterDataScience/clases/modulo_6_bibliotecas/exeption_test.py', wdir='/Users/mlacarrasco/Dropbox/_BACKUP/Asignaturas/2019/sem_01/MagisterDataScience/clases/modulo_6_bibliotecas')

  File "/opt/anaconda3/lib/python3.7/site-packages/spyder_kernels/customize/spydercustomize.py", line 827, in runfile
    execfile(filename, namespace)

  File "/opt/anaconda3/lib/python3.7/site-packages/spyder_kernels/customize/spydercustomize.py", line 110, in execfile
    exec(compile(f.read(), filename, 'exec'), namespace)

  File "/Users/mlacarrasco/Dropbox/_BACKUP/Asignaturas/2019/sem_01/MagisterDataScience/clases/modulo_6_bibliotecas/exeption_test.py", line 16, in <module>
    foo()

  File "/Users/mlacarrasco/Dropbox/_BACKUP/Asignaturas/2019/sem_01/MagisterDataScience/clases/modulo_6_bibliotecas/exeption_test.py", line 12, in foo
    res= 1/0

ZeroDivisionError: division by zero
```

```python
def foo():
    res= 1/0
    return(res)

foo()
```

Exception

Afortunadamente tenemos una herramienta muy útil para evitar que nuestro programa falle en forma precipitada. Esta herramienta se conoce como try / except

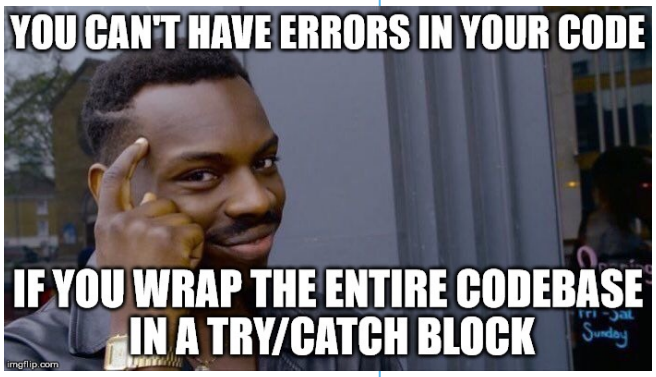
```
import sys

def foo(bar=None):
    res= 1/0
    return(res)

try:
    foo()
except:
    exc_type, exc_value, tb = sys.exc_info()
    print(exc_type)
```

El programa evalúa la función, y en caso que ésta tenga algún problema, enviará a una excepción

Con esto podemos saber cuál es el error generado por el sistema

Afortunadamente tenemos una herramienta muy útil para evitar que nuestro programa falle en forma precipitada. Esta herramienta se conoce como try / except

```
import sys

def foo(bar=None):
    res= 1/0
    return(res)

try:
    foo()
except:
    exc_type, exc_value, tb = sys.exc_info()
    print(exc_type)
```



error generado por Python

```
<class 'ZeroDivisionError'>
```

Exception

Afortunadamente tenemos una herramienta muy útil para evitar que nuestro programa falle en forma precipitada. Esta herramienta se conoce como try / except

```
import sys

def foo(bar=None):
    res= 1/0
    return(res)

try:
    foo()
except ZeroDivisionError:
    print("Error de division por cero")
else:
    print(”Función ejecutada sin errores")
```

Podemos capturar un error específico y definir un error para cada caso

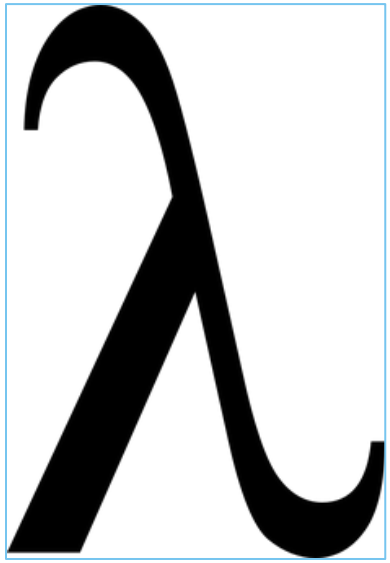

Listen kid...
I don't have much time...
The only way to write bug free code is...

Temario

- Estructuras de control
- Listas
- Funciones integradas
- Funciones
- Funciones avanzadas
  - lambda , map, filter, reduce

```
try:
    start = date(int(self.start_year.
                     self.months.index(se
                     int(self.start_day.ge

    end = date(int(self.end_year.get(
                     self.months.index(se
                     int(self.end_day.get
```

Como hemos indicado en los puntos anteriores, toda función en Python posee un nombre. Sin embargo, existe una función especial que no posee nombre, y se conoce como función lambda.



Está pensada para realizar tareas simples y sencillas (y reducir las líneas de código).

Ahora bien, pueden ser algo complejas en un inicio, pero son muy empleadas y útiles en diferentes escenarios.

After 5 days of Python:

After 10 years of Python:

Comencemos por una función ya conocida y simplifiquemosla

```
def mifuncion(num):
    num = num+1
    print('dentro de la función', num)
    return(num)

num =  mifuncion(22)
print(num)
```

```
def mifuncion(num):
    num = num+1
    return(num)

num =  mifuncion(22)
print(num)
```

```
def mifuncion(num):
    return(num+1)

num =  mifuncion(22)
print(num)
```

Comencemos por una función ya conocida y simplifiquemosla

```
def mifuncion(num):
    return(num+1)

num =  mifuncion(22)
print(num)
```

```
def mifuncion(num): return(num+1)

num =  mifuncion(22)
print(num)
```

```
lambda num: num+1
```

función lambda: no tiene nombre por ello se conoce como función anónima. Tiene el mismo funcionamiento que una del tipo procedural.

Ya temenos un nuevo tipo de función, que puede ser asignada a una variable. ¿Cómo? Sí..a una variable del tipo función.

```
lambda num: num+1
```

En este punto ejecutamos nuestra función anónima

```
incremento = lambda num: num+1

print(incremento(2))
```

```
3
```

Ejemplos

Veamos dos ejemplos que operaciones que ya hemos realizado anteriormente

Esta función anónima retorna la lista en sentido inverso

```
lista = [1,2,3,4]
ret = lambda lst: lst[::-1]

print(ret(lista))
```

con dos parámetros

```
mayor = lambda x,y: max(x,y)

print(mayor(5,9))
```

Map

Otra función que normalemente se utiliza en combinación con lambda es la función map. La función map permite ejecutar o llamar a otra función y en cada llamado recorrer el valor de una una lista, arreglo, o diccionario.

```
def mifuncion(num):
    return(num+1)

lista =[2,5,1,4]

print(map(mifuncion, lista))
```

```
<map object at 0x7f18480e3460>
```

Retorna un objeto de tipo map (que en este caso es una lista) (aka iterator)

```
def mifuncion(num):
    return(num+1)

lista =[2,5,1,4]

print(list(map(mifuncion, lista)))
```

```
[3,6,2,5]
```

Con la función list podemos ver el contenido de la lista

Map+ lambda

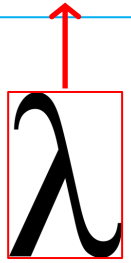
Otra función que normalemente se utiliza en combinación con lambda es la función map. La función map permite ejecutar o llamar a otra función y en cada llamado recorrer el valor de una una lista, arreglo, o diccionario.

```
def mifuncion(num):
    return(num+1)

lista =[2,5,1,4]

print(list(map(mifuncion, lista)))
```

versión sin lambda

```
lista =[2,5,1,4]
print(list(map(lambda i:i+1, lista)))
```

versión con lambda



Filter

La función filter nos permite filtrar una lista empleando el llamado a una función. Solo cuando el valor sea verdadero, éste será almacenado en una lista. Comparemos las dos formas posibles (con y sin lambda)

```
def mifuncion(num):
    return(num%2)

lista =[2,5,1,4]

print(list(filter(mifuncion, lista)))
```

versión sin lambda

```
lista =[2,5,1,4]

print(list(filter(lambda i:i%2, lista)))
```

versión con lambda

Filter

La función filter nos permite filtrar una lista empleando el llamado a una función. Solo cuando el valor sea verdadero, éste será almacenado en una lista. Comparemos las dos formas posible (con y sin lambda)

versión sin lambda

```
def mifuncion(num):
    if (num>3):
          return 1
    else
          return 0

lista = [2,5,1,4]

print(list(filter(mifuncion, lista)))
```

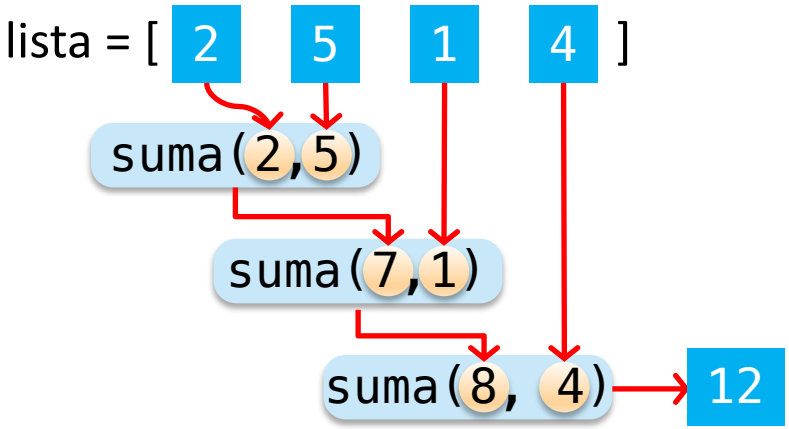
versión con lambda

```
lista =[2,5,1,4]

print(list(filter(lambda i:1 if i>3 else 0, lista)))
```

La función reduce nos permite aplicar el resultado parcial de una función una y otra vez empleando los resultados parciales anteriores.
Esta función recibe dos argumentos

```
from functools import reduce

def suma(a,b):
    return(a+b)

lista =[2,5,1,4]

print(reduce(suma, lista))
```

versión sin lambda

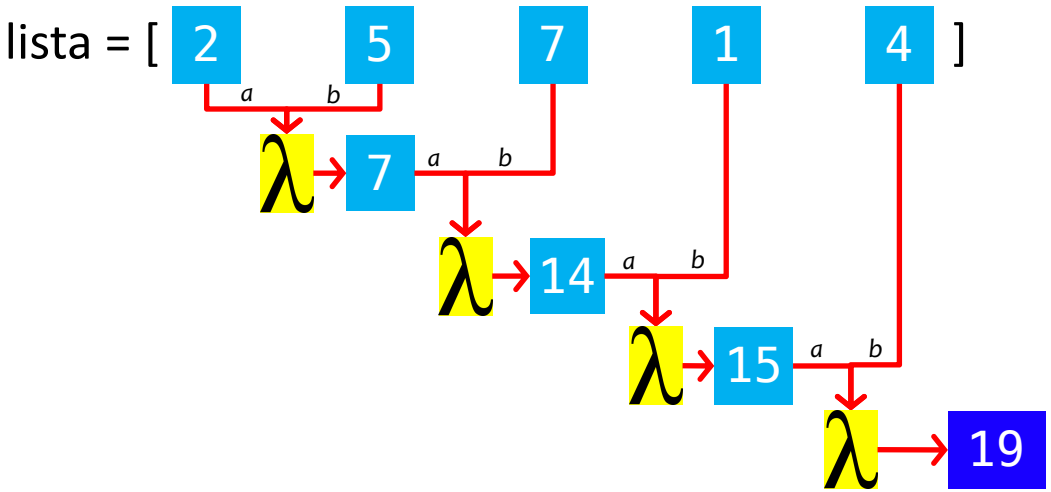


Reduce

La función reduce nos permite aplicar el resultado parcial de una función una y otra vez empleando los resultados parciales anteriores. Esta función recibe dos argumentos.

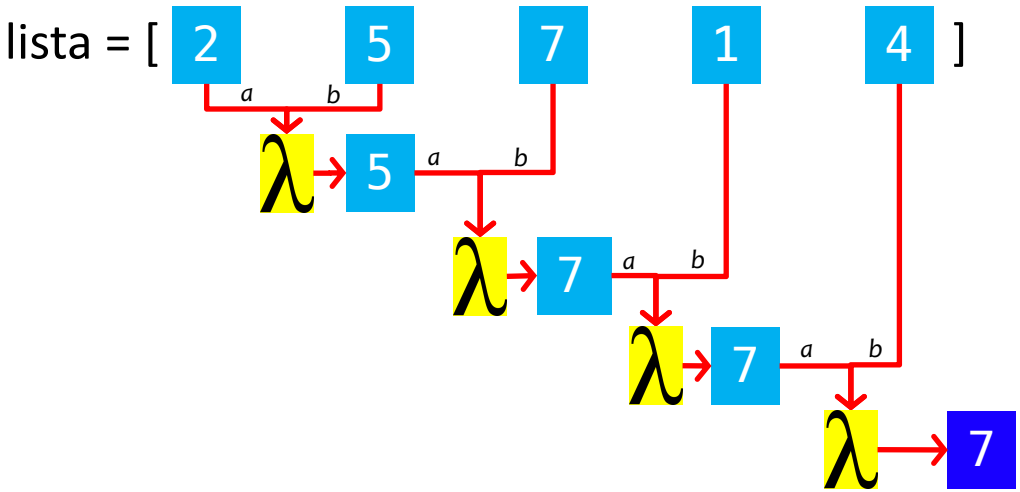
```
from functools import reduce

lista =[2, 5, 7, 1, 4]

print(reduce(lambda a,b:a+b, lista))
```

versión con lambda



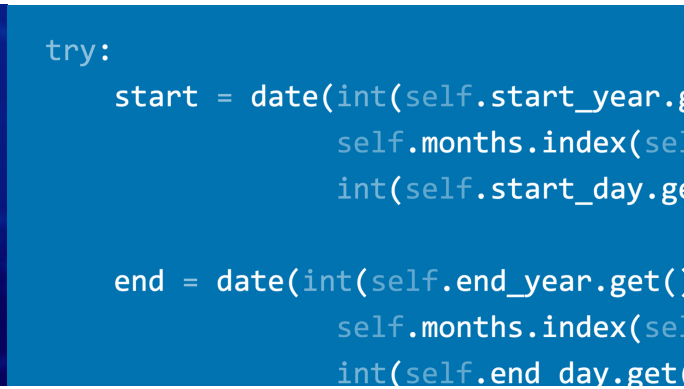
La función reduce nos permite aplicar el resultado parcial de una función una y otra vez empleando los resultados parciales anteriores. Esta función recibe dos argumentos.

```
from functools import reduce

lista =[2, 5, 7, 1, 4]

print(reduce(lambda a,b: a if a>b else b, lista))
```



Temario

- Estructuras de control
- Listas
- Funciones integradas
- Funciones
- Funciones avanzadas
- Decorators (muy avanzado 😎)



Decorators

Los decorators corresponde a una función que utiliza otra función y extiende su comportamiento agregando una nueva funcionalidad sin modificar su estructura.

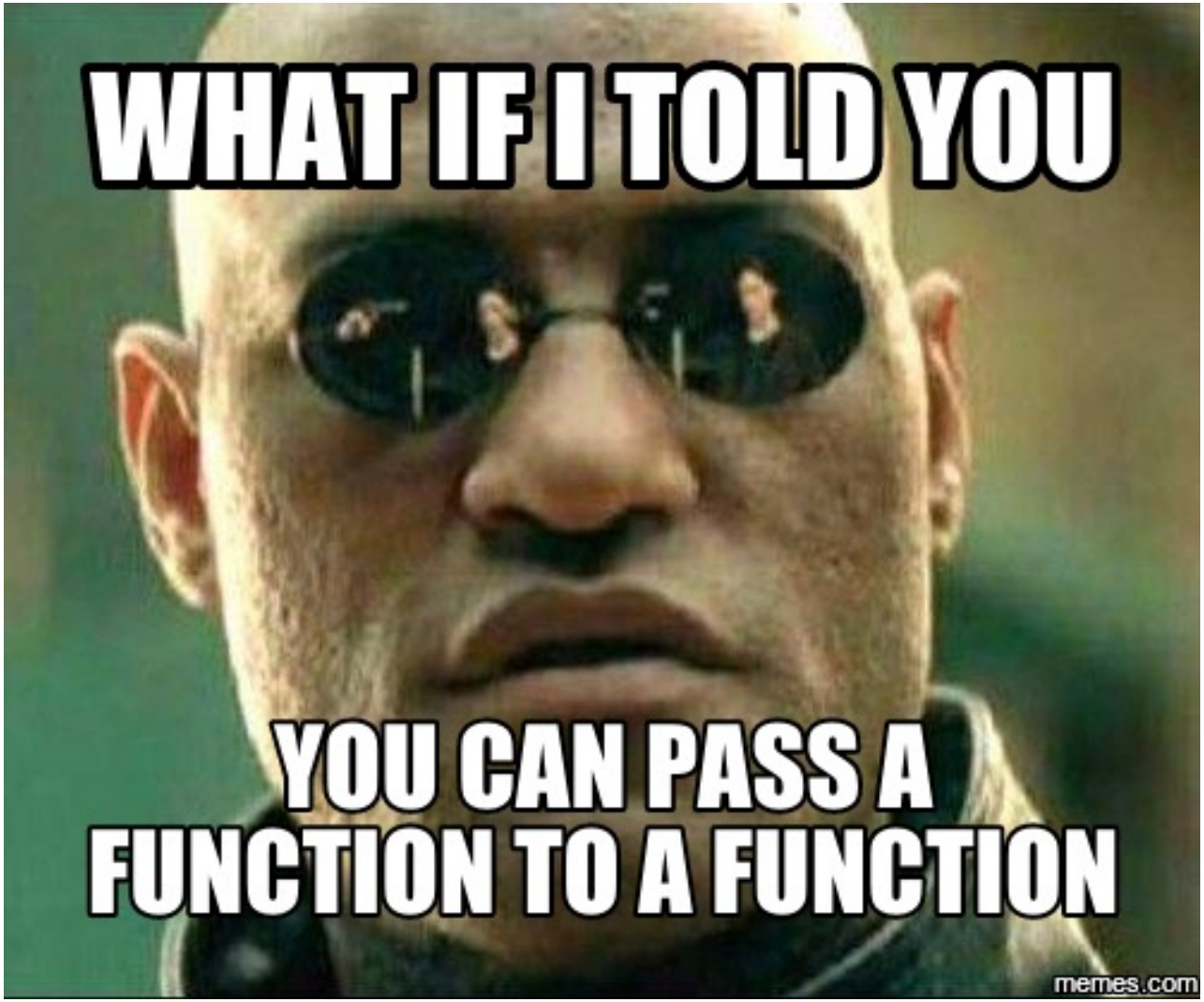
Comencemos por una función ya conocida y analicemos su uso ...

```
def mifuncion(num):
    return(num*num)

num =  mifuncion(4)
```

Toda función retorna un valor según sus argumentos

llamado a la función

argumento o parámetro


WHAT IF I TOLD YOU
YOU CAN PASS A FUNCTION TO A FUNCTION
memes.com

Los decorators corresponde a una función que utiliza otra función y extiende su comportamiento agregando una nueva funcionalidad sin modificar su estructura.

Comencemos por una función ya conocida y analicemos su uso ... y extendamos su funcionalidad

```
def funcion_generica(funcion):

    return(funcion(4))



def mifuncion(num):
    return(num*num)

num = funcion_generica(mifuncion)
```

Esta función recibe como argumento otra función. Esto se conoce como *First-Class Object*

llamado a la función

argumento es una función

Comencemos por una función ya conocida y analicemos su uso ... y extendamos su funcionalidad

```
def funcion_generica(funcion):

    return(funcion(4))


def mifuncion_A(num):
    return(num*num)

def mifuncion_B(num):
    return(num/2)


val_A = funcion_generica(mifuncion_A)

val_B = funcion_generica(mifuncion_B)
```

llamado a la función. Puede ser cualquiera pero debe recibir un sólo parámetro

16

2

Ahora agrupemos ambas funciones en una sola (conocido como *inner function*)

```
def funcion_generica():

    def mifuncion_A(num):
        return(num*num)

    def mifuncion_B(num):
        return(num/2)

    print(mifuncion_A(4))
    print(mifuncion_B(4))


funcion_generica()
```

Dentro de la función puedo definir funciones

llamado a la función

Agregemos un argumento a nuestra función genérica para que podamos interactuar con nuestras funciones internas

```
def funcion_generica(args):

    def mifuncion_A(num):
        return(num*num)

    def mifuncion_B(num):
        return(num/2)

    if args > 4:
        return(mifuncion_A(args))
    else:
        return(mifuncion_B(args))

funcion_generica(6)
```

Puedo cambiar el curso de la función según el argumento indicado

llamado a la función

Agregemos un argumento a nuestra función genérica para que podamos interactuar con nuestras funciones internas

```
def funcion_generica(args):

    def mifuncion_A(num):
        return(num*num)

    def mifuncion_B(num):
        return(num/2)

    if args > 4:
        return(mifuncion_A)
    else:
        return(mifuncion_B)

f1 = funcion_generica(6)
f2 = funcion_generica(3)
```

Observe que no tiene argumentos

Es posible asignar a una variable una función

Como podemos ver, tenemos acceso a las funciones internas de nuestra función a través de variables

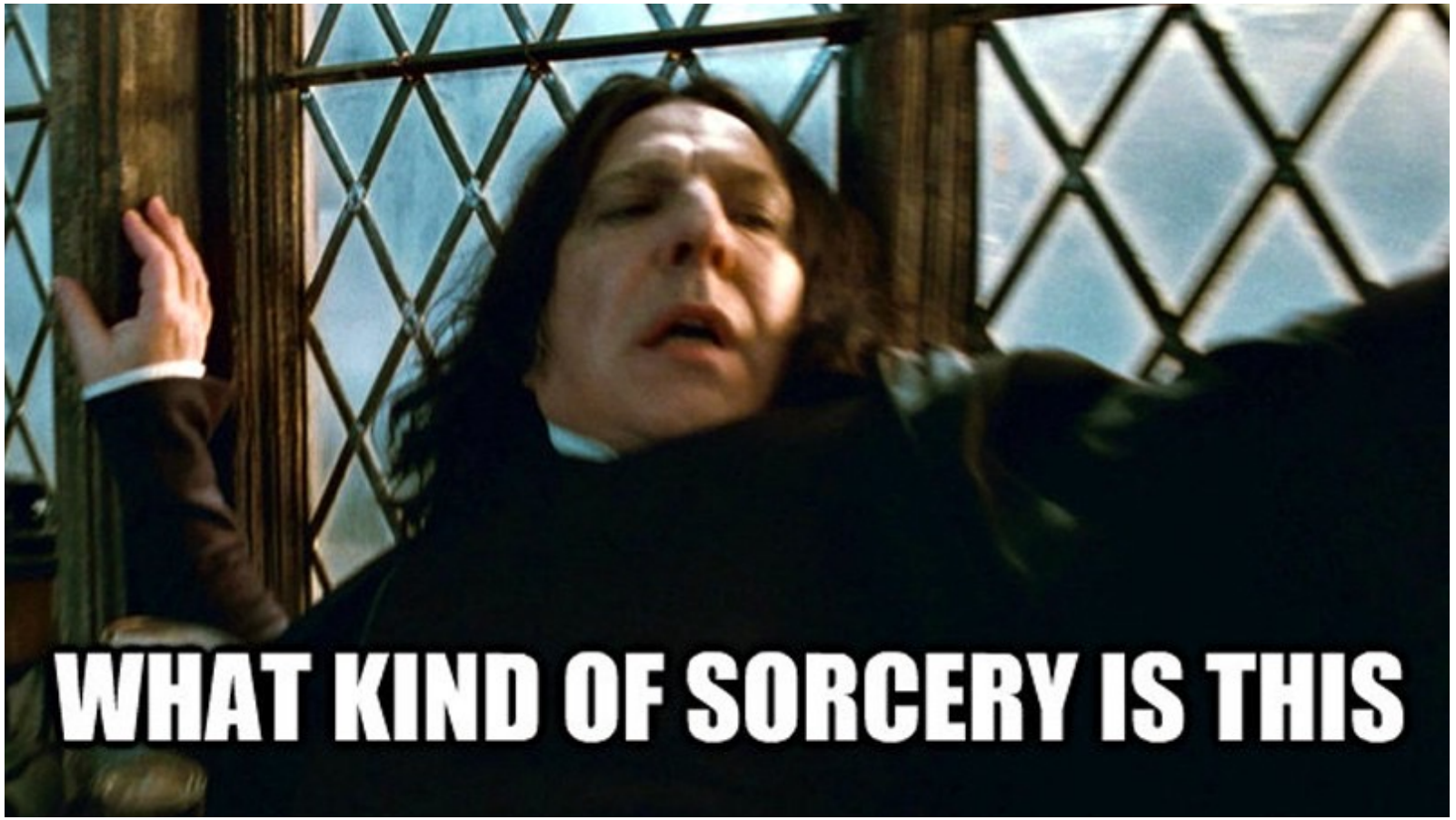
```
def funcion_generica(args):

    def mifuncion_A(num):
        return(num*num)

    def mifuncion_B(num):
        return(num/2)

    if args > 4:
        return(mifuncion_A)
    else:
        return(mifuncion_B)

f1 = funcion_generica(6)
f2 = funcion_generica(3)

print(f1) → <function funcion_generica.<locals>.mifuncion_A at 0x7f6341059d40>
print(f2) → <function funcion_generica.<locals>.mifuncion_B at 0x7f6341059e60>
```

estas dos variables son funciones

Como podemos ver, tenemos acceso a las funciones internas de nuestra función a través de variables

```
def funcion_generica(args):

    def mifuncion_A(num):
        return(num*num)

    def mifuncion_B(num):
        return(num/2)

    if args > 4:
        return(mifuncion_A)
    else:
        return(mifuncion_B)

f1 = funcion_generica(6)
f2 = funcion_generica(3)

print(f1(10))    100
print(f2(3))     1.5
```

estas dos variables son funciones


WHAT KIND OF SORCERY IS THIS

Ahora que hemos visto las operaciones básicas entre funciones y funciones con funciones, analicemos los *decorators*

```
def mi_decorador(funcion):

    def wrapper():
            print('El area es:')
            return(funcion(5))

    return(wrapper)


def mi_area(num):
    return(num*num)

area =  mi_decorador(mi_area)

print(area())
```

```
El area es:
25
```

El decorator toma la función y le agrega una funcionalidad

Función que vamos aplicar un decorador

Recuerde que retornamos una función

Ahora que hemos visto las operaciones básicas entre funciones y funciones con funciones, analicemos los *decorators*

```
def mi_decorador(funcion):

    def wrapper():
            print('El area es:')
            return(funcion(5))

    return(wrapper)


@mi_decorador
def mi_area(num):
    return(num*num)


print(mi_area())
```

Observe que se retorna una función

es lo mismo que decir:
`area = mi_decorador(mi_area)`

Función que vamos aplicar un decorador

```
El area es:
25
```

Ahora que hemos visto las operaciones básicas entre funciones y funciones con funciones, analicemos los *decorators*

```
def mi_decorador(funcion):

    def wrapper():
          print('El area es:')
          return(funcion(5))

    return(wrapper)


@mi_decorador
def mi_area(num):
     return(num*num)


print(mi_area(7))
```

Ojo que esta función no recibe argumentos. Y qué pasaría si damos uno

```
TypeError: wrapper() takes 0 positional arguments but 1 was given
```

Ahora que hemos visto las operaciones básicas entre funciones y funciones con funciones, analicemos los *decorators*

```
def mi_decorador(funcion):

    def wrapper(*args, **kwargs):
          print('El area es:')
          return(funcion(*args, **kwargs))

    return(wrapper)


@mi_decorador
def mi_area(num):
     return(num*num)


print(mi_area(7))
```

```
El area es:
49
```

args: son los argumentos de la función. No tiene límite de argumentos. El símbolo * realiza la operación *unpacking* (desde Python 3.7)

kwargs: son los nombres de las variables que ingresan a la función. No tiene límite de argumentos. El símbolo ** realiza la operación *unpacking dos veces*


SO MUCH.
FUN.
memegenerator.net

Si no sabemos qué hay dentro del decorador podemos emplear algunos atributos

```
def mi_decorador(funcion):

    def wrapper(*args, **kwargs):
          print('El area es:')
          return(funcion(*args, **kwargs))

    return(wrapper)


@mi_decorador
def mi_area(num):
     return(num*num)


print(mi_area.__name__)
```

```
wrapper
```

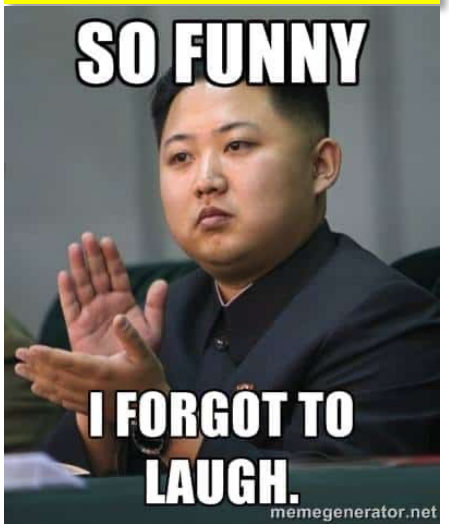
con el atributo .__name__ podemos conocer las funciones definidas dentro del decorator

Si no sabemos qué hay dentro del decorador podemos emplear algunos atributos

```
import functools

def mi_decorador(funcion):
   @functools.wraps(funcion)
    def wrapper(*args, **kwargs):
          print('El area es:')
          return(funcion(*args, **kwargs))

    return(wrapper)


@mi_decorador
def mi_area(num):
     return(num*num)


print(mi_area.__name__)
```

```
mi_area
```

Este decorador encapsula todo lo que está dentro del decorador



Ahora tú...

Tiempo : 10 minutos

Implemente un decorador que calule el tiempo que toma realizar la ejecución de la función "loop_variable". (Hint. utilice `time.perf_counter()`)

```
import time
def contador_tiempo(func):









@contador_tiempo
def loop_variable(num):
    cont = 0
    for i in range(num):
        cont = i+cont

loop_variable(100)
```

es lo mismo que decir:
`contador_tiempo(loop_variable)`

Ahora tú...

Tiempo : 10 minutos

Implemente un decorador que calule el tiempo que toma realizar la ejecución de la función "loop_variable". (Hint. utilice `time.perf_counter()`)

```
import time
def contador_tiempo(func):

     def wrapper_contador_tiempo(*args, **kwargs):
         ini = time.perf_counter()
         func(*args, **kwargs)
         delta = time.perf_counter()-ini
         return(f'Terminado en {delta} segundos')

     return(wrapper_contador_tiempo)

@contador_tiempo
def loop_variable(num):
    cont = 0
    for i in range(num):
       cont = i+cont

loop_variable(100)
```