

Métodos de minería de datos en
Python

Programación básica en Python

Contenido

1

Listas

2

Funciones

3

Funciones lambda

Listas

Listas

- Son **secuencias ordenadas de información**, a la cual se puede acceder con índices.
- Se representa con barras cuadradas `[]`.

Las listas contienen elementos:

- Usualmente homogéneos (ejemplo: Todos enteros)
- Puede tener elementos combinados (no es común)
- Los elementos de una lista pueden ser reemplazados (mutable).

Listas

1.

```
L = [2, 'a', 4, [1, 2]]  
print(L)
```



```
[2, 'a', 4, [1, 2]]
```

2.

```
len(L)
```



```
4
```

3.

```
L[0]
```



```
2
```

4.

```
L[0] = L[2]+1  
L[0]
```



```
5
```

5.

```
L[3]
```



```
[1, 2]
```

6.

```
L[3][1]
```



```
2
```

7.

```
L[4]
```

```
IndexError                                Traceback (most recent call last)  
<ipython-input-48-1eef6b78def1> in <module>()  
----> 1 L[4]
```



```
IndexError: list index out of range
```

SEARCH STACK OVERFLOW

Listas

Se pueden añadir elementos al final de la lista con **L.append(k)**

```
L = [2, 6, 7]  
L.append(10)  
L
```

```
[2, 6, 7, 10]
```

Qué es el punto?

- Las listas son objetos de Python
- Los objetos tienen datos
- Los objetos además tienen métodos y funciones
- Para acceder a los métodos o funciones de un objeto se usa: **object_name.do_something()**

Listas

Las siguientes funciones permiten **remove** elementos de una lista

➤ **Remove un elemento específico:** *L.remove(element)*

```
L = [2,1,3,6,3,7,0]
```

```
L.remove(2)
```

```
L
```

```
[1, 3, 6, 3, 7, 0]
```

Listas

- Las siguientes funciones permiten **remove** elementos de una lista
 - Indicando un **índice específico**: `del(L[index])`

```
L = [2,1,3,6,3,7,0]  
L.remove(2)  
L
```

```
[1, 3, 6, 3, 7, 0]
```

```
del(L[1])  
L
```

```
[1, 6, 3, 7, 0]
```


Listas

- Las siguientes funciones permiten **remove** elementos de una lista
 - **Remove el último elemento de la lista:** *L.pop()*

```
del(L[1])
```

```
L
```

```
[1, 6, 3, 7, 0]
```

```
L.pop()
```

```
L
```

```
[1, 6, 3, 7]
```

Listas

Convertir palabras a listas de caracteres y volver a unirlos:

➤ Convertir cadena de caracteres a lista: *list(s)*

```
s = "I<3 Python"  
print(list(s))
```

```
['I', '<', '3', ' ', 'P', 'y', 't', 'h', 'o', 'n']
```

Listas

Convertir palabras a listas de caracteres y volver a unirlas:

➤ Partir una lista de caracteres: *s.split()*

```
print(s.split('<'))
```

```
['I', '3 Python']
```

Listas

Convertir palabras a listas de caracteres y volver a unirlos:

➤ Convertir lista de caracteres en cadena de texto: `".join(L)`

```
L = ['a', 'b', 'c']  
print(''.join(L))  
print('_'.join(L))
```

abc

a_b_c

Listas

Realizar operaciones con listas que contengan números

➤ *Ordenar*

```
L = [9,6,0,3]  
print(sorted(L))
```

[0, 3, 6, 9]

```
L = [9,6,0,3]  
L.sort()  
L
```

[0, 3, 6, 9]

Listas

Realizar operaciones con listas que contengan números

➤ *Ordenar inversamente*

```
L = [9,6,0,3]  
L.reverse()  
L
```

```
[3, 0, 6, 9]
```

```
L = [9,6,0,3]  
L.sort()  
L.reverse()  
L
```

```
[9, 6, 3, 0]
```

Listas

Adicionalmente las listas permiten:

➤ *Ser clonadas*

```
cool = ['azul', 'verde', 'gris']
newList = cool[:]
newList.append('negro')
print(newList)
print(cool)
```

```
['azul', 'verde', 'gris', 'negro']
['azul', 'verde', 'gris']
```

➤ *Las listas pueden ser mutadas a través de procesos de iteraciones:*

```
L1 = [1,2,3,4,2,5,2,6]

def remove_dups(lista):
    for e in lista:
        lista.remove(e)
    return(lista)

remove_dups(L1)

[4, 5, 2, 6]
```



Funciones

Cómo programamos?

Hasta ahora:

1. Manejamos diferentes tipos de estructuras y operaciones
2. Sabemos escribir códigos para resolver problemas específicos
3. Cada archivo generado es una pieza del código
4. Cada código es una secuencia de instrucciones

Problemas con éste enfoque:

1. Fácil de usar en problemas de “mundo pequeño”
2. Desordenado para problemas de gran escala
3. Difícil realizar seguimiento a los detalles

¿Cómo se sabe si se suministra la información correcta en la parte correcta del script?

Buenas prácticas

Más código nos es necesariamente lo mejor

Los buenos programadores se miden por:

- La funcionalidad de sus códigos
- La introducción de **funciones**
- La utilización de mecanismos que permitan **descomponer** y **abstraer** la información

Estructuras con Descomposición

En programación el código se divide en **módulos**, los cuales:

- Son **autocontenidos**
- Se usan para **dividir el código**
- Se destinan a ser **reutilizados**
- Mantienen el **código organizado**
- Mantienen el **código coherente**

Una forma de descomponer el código se realiza a través de **funciones**

Generar Abstracción

En programación, piense que cada pieza del código compone un juego de video:

- El usuario final **no puede** ver los detalles de programación
 - El usuario final **no necesita** ver los detalles de programación
- El usuario final **no quiere** ver los detalles de programación

*La abstracción del código se realiza por documentos de especificación de **funciones***

Funciones

- ❑ Una función es un bloque de código organizado y reutilizable que se utiliza para realizar una única acción relacionada.
- ❑ Las funciones se encuentran estáticas hasta que son **llamadas o invocadas** en un programa.
- ❑ Python ofrece muchas funciones integradas y la opción de crear funciones propias: funciones definidas por el usuario.

Funciones

Características:

1. Inicia con una **palabra clave**
2. Tienen un **nombre**
3. Tienen **parámetros** (0 o más)
4. Tienen **comentarios** (opcional pero deseable)
5. Tienen un **cuerpo de código**
6. **Retornan algo**, algún resultado.

Funciones

keyword nombre Parámetros/
Argumentos (Pueden ser obligatorios u opcionales)

```
def par(n):
```

```
## n: Un entero positivo  
## Retorna True si n es par, False en caso contrario
```

Comentarios
al código

```
print("El número es par?: ")  
return n%2 == 0
```

Cuerpo
del código

par(4) → Función
invocada

Funciones: Cuerpo

```
def par(n):
```

```
    ## n: Un entero positivo
```

```
    ## Retorna True si n es par, False en caso contrario
```

```
    print("El número es par?: ")
```

```
    return n%2 == 0
```

keyword

par(4)

Corre algunas instrucciones

Expresión a
evaluar y
retornar

Funciones: Variable

- El **parámetro real** se vincula con el **parámetro formal** cuando se invoca la función
- Un nuevo **entorno** es creado cuando se invoca una nueva función
 - El **alcance** es la asignación de nombres a objetos

```
def f(x):  
    x = x + 1  
    print('in f(x): x =', x)  
    return x
```

➔ **Parámetro formal**

Definición de la función

```
x = 3
```

```
z = f(x)
```

➔ **Parámetro actual**

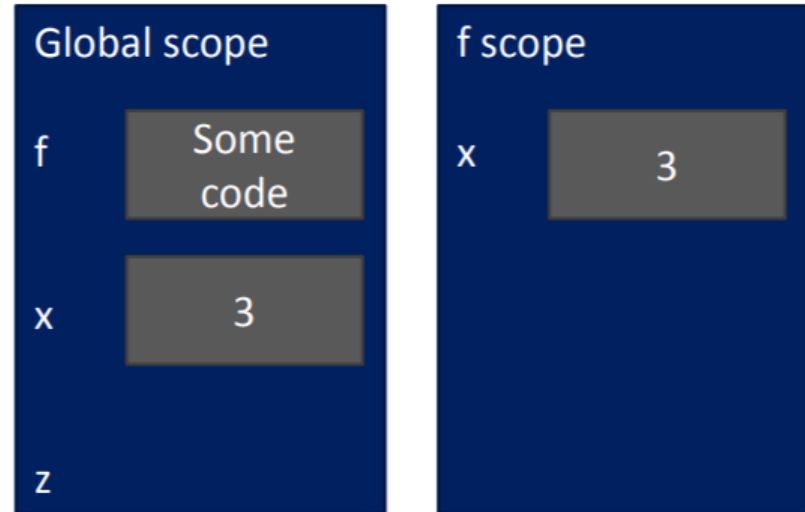
Ejecuta la función:

- Inicializa la variable x
- Invoca la función f(x)
- Asigna el resultado de la función a la variable z

Funciones: Variable

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x
```

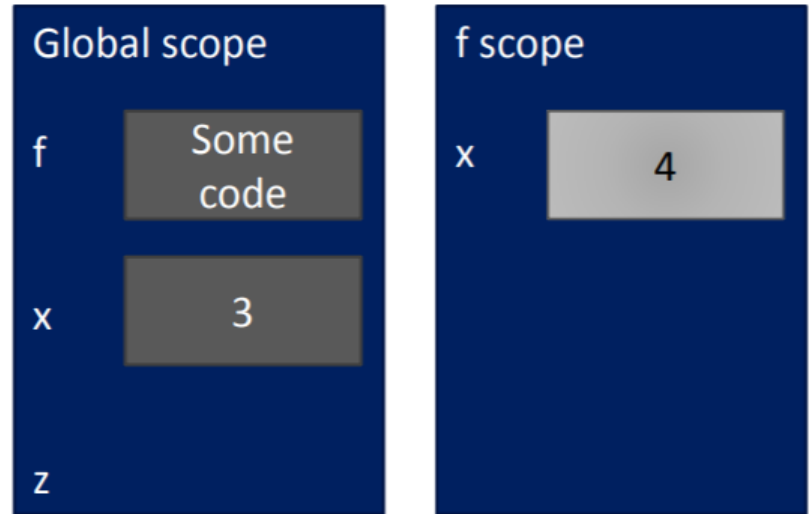
```
x = 3
z = f( x )
```



Funciones: Variable

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x
```

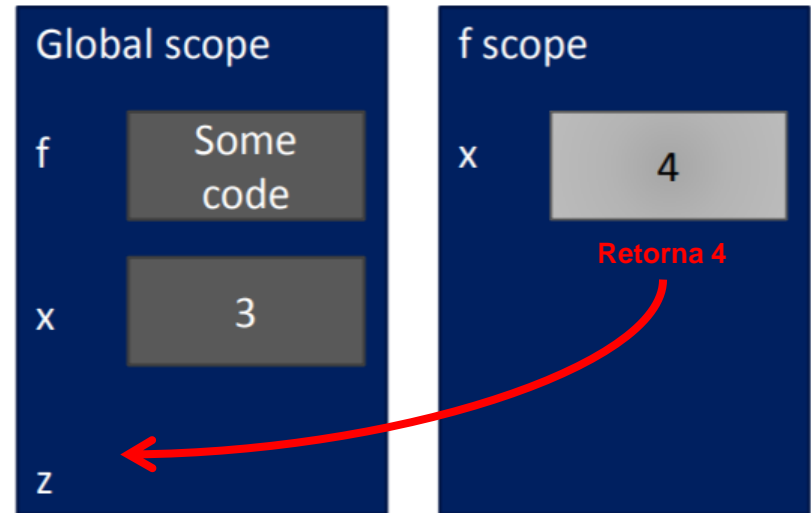
```
x = 3
z = f( x )
```



Funciones: Variable

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x
```

```
x = 3
z = f( x )
```



Funciones: Variable

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x
```

```
x = 3
z = f( x )
```

Global scope	
f	Some code
x	3
z	4

Funciones: Variable

```
def conocer(nombre):  
    """  
    Esta función  
    saluda a la persona  
    que ingrese su nombre  
    como parámetro  
    """  
    print("Hola, " + nombre + ". ¡Buenos días!")
```

```
conocer("Carlos")
```

Hola, Carlos. ¡Buenos días!

Funciones: Variable

```
def absolute_value(num):  
    if num >= 0:  
        return num  
    else:  
        return -num
```

```
print(absolute_value(2))
```

 2

```
print(absolute_value(-4))
```

 4

Funciones como argumentos

```
def conocer(nombre, mensaje = "Buenos días"):
    """
    Esta función
    saluda a la persona
    que ingrese su nombre
    como parámetro

    Si no se provee el mensaje,
    el valor por defecto será "Buenos días"
    """
    print("Hola", nombre + ', ' + mensaje)
```

```
conocer("María")
```

Hola María, Buenos días

```
conocer("Ana María", "¿Cómo estás?")
```

Hola Ana María, ¿Cómo estás?

Funciones como argumentos

```
def func_a():  
    print('Función A')
```

```
def func_b(y):  
    print('Función B')  
    return y
```

```
def func_c(z):  
    print('Función C')  
    return z
```

```
print(func_a())
```

```
print(5 + func_b(2))
```

```
print(func_c(func_b(2)))
```

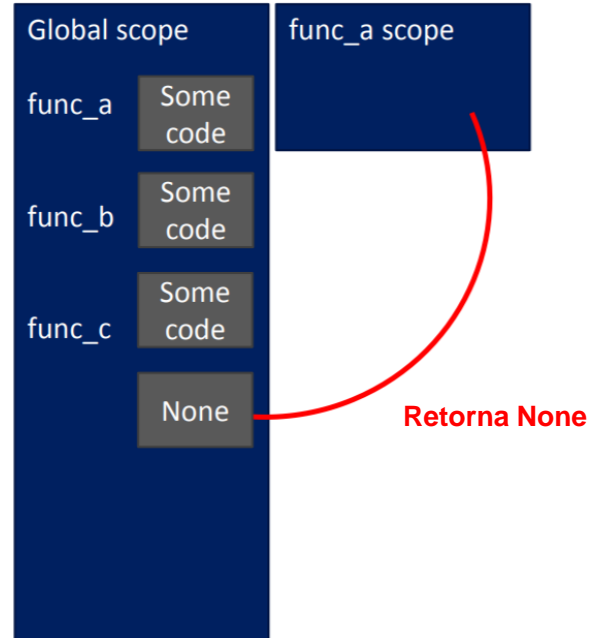
Llama a la función A: No toma argumentos

Llama a la función B: Toma un argumento

Llama a la función C: Toma un argumento, en este caso otra función

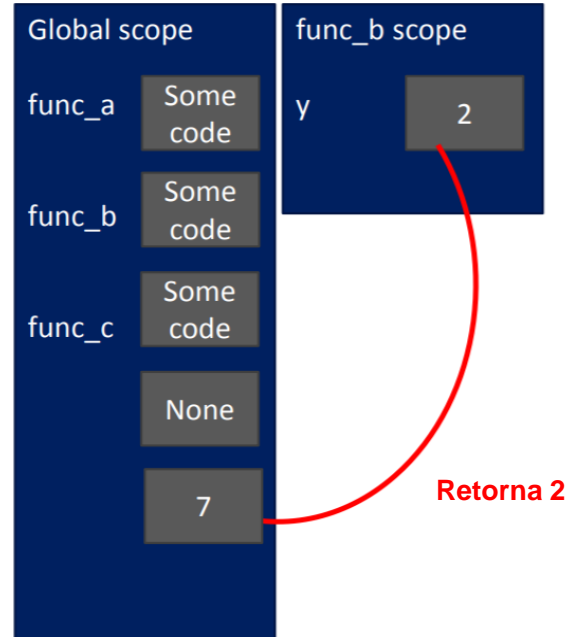
Funciones como argumentos

```
def func_a():  
    print('Función A')  
def func_b(y):  
    print('Función B')  
    return y  
def func_c(z):  
    print('Función C')  
    return z  
print(func_a())  
print(5 + func_b(2))  
print(func_c(func_b(2)))
```



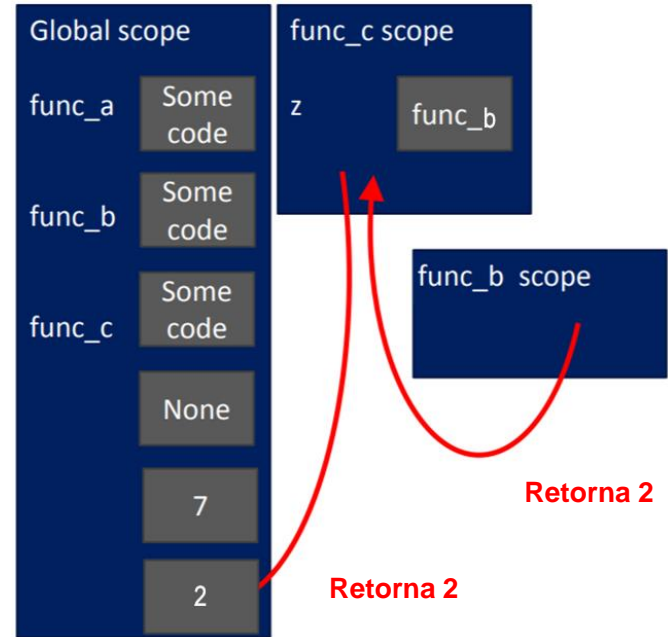
Funciones como argumentos

```
def func_a():  
    print('Función A')  
def func_b(y):  
    print('Función B')  
    return y  
def func_c(z):  
    print('Función C')  
    return z  
print(func_a())  
print(5 + func_b(2))  
print(func_c(func_b(2)))
```



Funciones como argumentos

```
def func_a():  
    print('Función A')  
def func_b(y):  
    print('Función B')  
    return y  
def func_c(z):  
    print('Función C')  
    return z  
print(func_a())  
print(5 + func_b(2))  
print(func_c(func_b(2)))
```



Funciones como elemento

```
def suma(a,b):
```

```
    return(a+b)
```

```
def elevar(c):
```

```
    return(c**2)
```

```
elevar(suma(2,3))
```

25

```
def suma(a,b=0):
```

```
    return(a+b)
```

```
def elevar(c):
```

```
    return(c**2)
```

```
elevar(suma(2))
```

4

```
def suma(a,b=0):
```

```
    return(a+b)
```

```
def elevar(c,d):
```

```
    return(c**d)
```

```
elevar(suma(2),3)
```

8

Expresiones lambda

Expresiones lambda

- Las expresiones lambda se usan idealmente cuando necesitamos hacer algo simple y estamos más interesados en hacer el trabajo rápidamente en lugar de nombrar formalmente la función.
- Las expresiones lambda también se conocen como funciones anónimas.
- Como sabes, una función en Python se define con la palabra reservada *def*. Sin embargo, una función anónima se define con la palabra reservada *lambda*.

Definir una función anónima en Python

La sintaxis para definir una *función lambda* es la siguiente:

```
lambda parámetros: expresión
```

A continuación, te detallo las características principales de una *función anónima*:

- Son funciones que pueden definir cualquier número de parámetros pero una única expresión. Esta expresión es evaluada y devuelta.
- Se pueden usar en cualquier lugar en el que una función sea requerida.
- Estas funciones están restringidas al uso de una sola expresión.
- Se suelen usar en combinación con otras funciones, generalmente como argumentos de otra función.

Ejemplo

```
# Función Lambda para calcular el cuadrado de un número
square = lambda x: x ** 2
print(square(5))

# Funcion tradicional para calcular el cuadrado de un numero
def square1(num):
    return num ** 2
print(square(5))
```

25

25

Ejemplos

```
lambda_func = lambda x: True if x**2 >= 10 else False  
print(lambda_func(3))  
print(lambda_func(4))
```

False

True

Ejemplos

```
my_dict = {"A": 1, "B": 2, "C": 3}  
sorted(mi_dict, key=lambda x: my_dict[x]%3)
```

```
['C', 'A', 'B']
```

```
print(my_dict["A"]%3)  
print(my_dict["B"]%3)  
print(my_dict["C"]%3)
```

```
1  
2  
0
```

¡Gracias!

¿Preguntas?

Python is the
easier language
to learn.
No brackets,
no main.



You get errors
for writing an
extra space



Reto

Crear una función que al insertar una lista de *números* arroje la siguiente salida:

Los números ingresados fueron [1, 2, 3, 4]

La suma de los números es: 10

El valor de la multiplicación de los números es: 24

El valor de los números elevados al cuadrado y sumados es: 331776

Condiciones:

- Tiene que servir con cualquier lista de números de cualquier tamaño la lista
- No puede usar librerías, sólo código escrito a mano (Usar for o while Sí se puede, funciones o funciones lambda)