



PYTHON

ÍNDICE

- Elementos sintácticos del lenguaje
- Funciones
- Clases y objetos
- Colecciones de objetos
- Elementos de programación funcional
- Gestión de errores
- Operaciones de entrada-salida
- Módulos





Plataforma de desarrollo

¿Que es Python?

- Python es un lenguaje de programación que fue creado a finales de los años 80 por el holandés Guido van Rossum, fan del grupo humorístico Monty Python, de ahí el nombre que le puso al lenguaje de programación.

Características de Python

- Simpleza
- Sintaxis clara
- Propósito general
- Lenguaje interpretado
- Lenguaje de alto nivel
- Lenguaje orientado a objetos
- Open source
- Extensas librerías

A large stack of books of various colors and sizes, primarily in shades of blue, red, and black, is arranged diagonally across the left side of the slide. They appear to be old or well-used, with visible wear and discoloration on the spines.

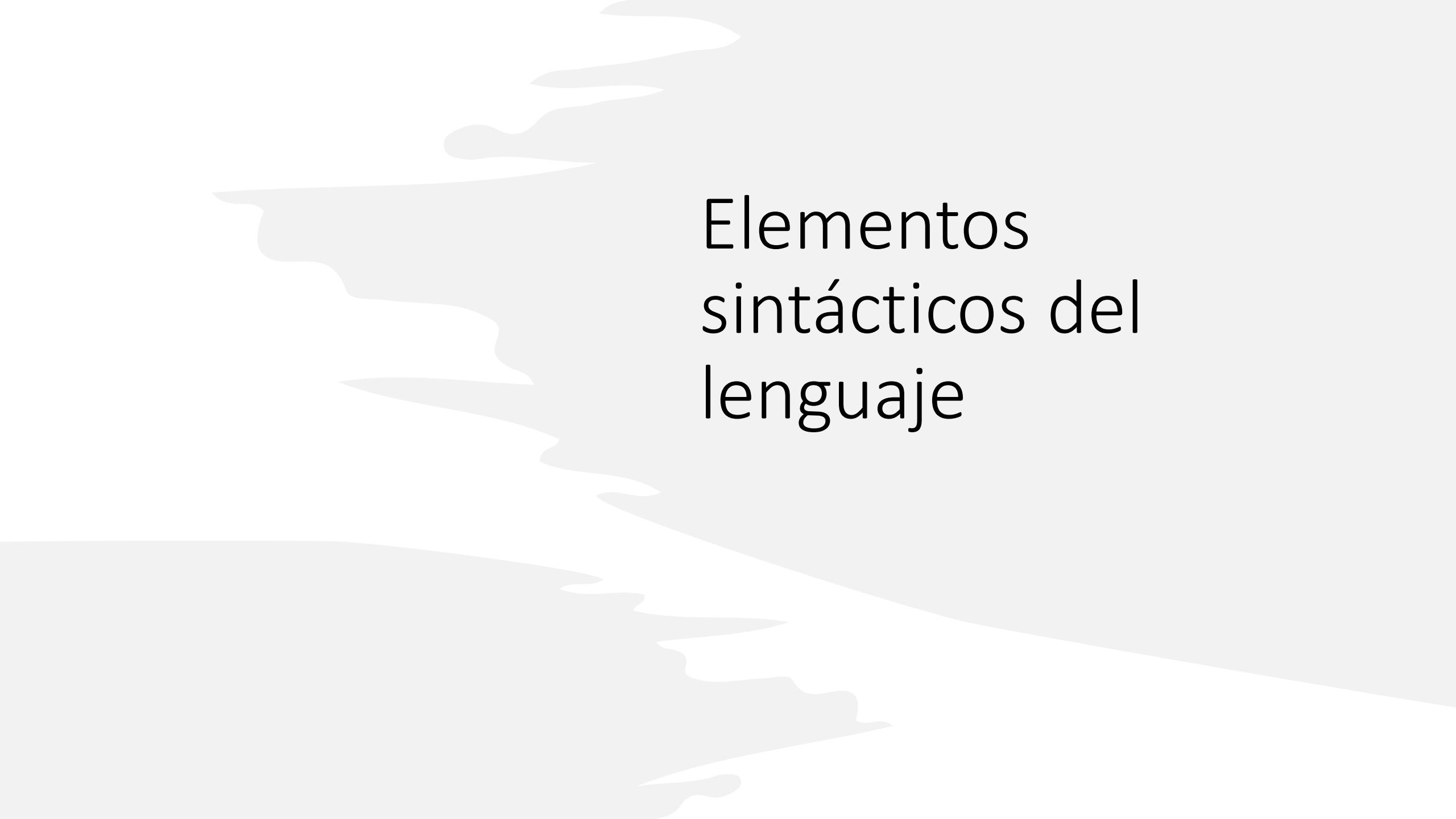
Ventajas

- **Legible:** sintaxis intuitiva y estricta
- **Productivo:** ahorra mucho código
- **Portable:** para todo sistema operativo
- **Recargado:** viene con muchas librerías por defecto.

Principales entornos de desarrollo

- Son muchos los entornos que podemos utilizar:
 - IDLE
 - Visual Studio Code
 - Jupiter
 - Anaconda
 - Spyder
 - PyCharm





Elementos sintácticos del lenguaje

Declaración de variables

- Las variables, a diferencia de los demás lenguajes de programación, no debes definirlas, ni tampoco su tipo de dato, ya que al momento de iterarlas se identificará su tipo.
 - Debe comenzar por letra o _
 - Es case sensitive, diferencia mayúsculas de minúsculas.
 - Para saber el tipo de dato type(variable)

Tipos de datos

Tipo de dato	Descripción
Entero	Número sin decimales, tanto positivo como negativo, incluyendo el 0.
Real	Número con decimales, tanto positivo como negativo, incluyendo el 0.
Complejo	Número con parte imaginaria.
Cadenas de texto	Texto.
Booleanos	Pueden tener dos valores: True o False
Conjuntos	Colección de elementos no ordenados y no repetidos.
Lista	Vector de elementos que pueden ser de diferentes tipos de datos.
Tuplas	Lista inmutable de elementos.
Diccionario	Lista de elementos que contienen claves y valores.

```
sumando1 = int(input("Introduzca el primer sumando: "))  
sumando2 = int(input("Introduzca el segundo sumando: "))  
print("Resultado de la suma: ", sumando1 + sumando2)
```

Conversiones entre tipos de datos

- **Números enteros**
- Toda entrada de usuario se interpreta como una cadena de texto, para poder transformarlo en un numero, se utiliza la instrucción int.

Imprimir variables de cadenas de texto

- El comando **print** lo utilizamos para mostrar información por pantalla. Podemos mostrar uno o varios elementos separados por comas.
- A través de este comando podemos utilizar 2 operadores:
- **sep**; Nos permite establecer una cadena de texto que actuará como separador de los elementos a mostrar.
 - **end**; Será la cadena que se muestre al final.

```
# Comando print  
  
print("Hola")  
print("Juan", "Pedro", "Maria", "Luis")  
  
#Parametro sep  
print("Juan", "Pedro", "Maria", "Luis", sep=' | ')  
  
#Parametro end  
print("Juan", "Pedro", "Maria", "Luis", sep=',', end='.')
```

```
Python 3.7.7 (v3.7.7:d7c567b081  
[Clang 6.0 (clang-600.0.57)] or  
Type "help", "copyright", "crec  
>>>  
= RESTART: /Users/anaisabelvego/  
mplo2_print.py  
Hola  
Juan Pedro Maria Luis  
Juan | Pedro | Maria | Luis  
Juan,Pedro,Maria,Luis.  
>>> |
```

Entradas de texto por teclado

- El comando **input** se utiliza para leer información de entrada, desde el teclado.

```
# Comando input
print('Cual es tu nombre? ')
nombre = input()
print('Hola ', nombre, 'Bienvenido al curso !!')

# Otra forma
nombre = input('Cual es tu nombre? ')
print('Hola ', nombre, 'Bienvenido al curso !!')
```

Operadores aritméticos

Operador	Significado
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo
**	Potencia
//	División entera

Operadores de asignación

Operador	Significado
<code>+=</code>	Suma
<code>-=</code>	Resta
<code>*=</code>	Multiplicación
<code>/=</code>	División
<code>%=</code>	Módulo
<code>**=</code>	Potencia
<code>//=</code>	División entera

Operadores relacionales

Operador	Significado
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual que
!=	Distinto que

Operadores lógicos

Los operadores lógicos que puedes utilizar son los siguientes:

- **and**: operador lógico que realiza la operación lógica ‘Y’ entre dos elementos. El resultado será true si ambos elementos son true, en caso contrario será false.
- **or**: operador lógico que realiza la operación lógica ‘O’ entre dos elementos. El resultado será true si uno de los dos elementos es true, en caso contrario será false.
- **not**: operador lógico que realiza la operación lógica ‘NO’. El resultado será true si el elemento es false, y será false si es true.

Operadores de identidad

- Intentan identificar si los objetos son realmente el mismo objeto o si son diferentes objetos.

```
frutas1 = ["manzana", "pera"]
frutas2 = ["manzana", "pera"]
frutas3 = frutas1
```

```
frutas3 is frutas1
```

True

Operadores de pertenencia

- Los operadores de pertenencia permiten verificar si un valor está dentro de una lista de valores o si un objeto está dentro de una lista de objetos.

```
frutas1 = ["manzana", "pera", "naranja"]  
frutas2 = "pera"
```

```
frutas2 in frutas1
```

```
True
```

```
# not in
```

```
frutas2 not in frutas1
```

```
False
```

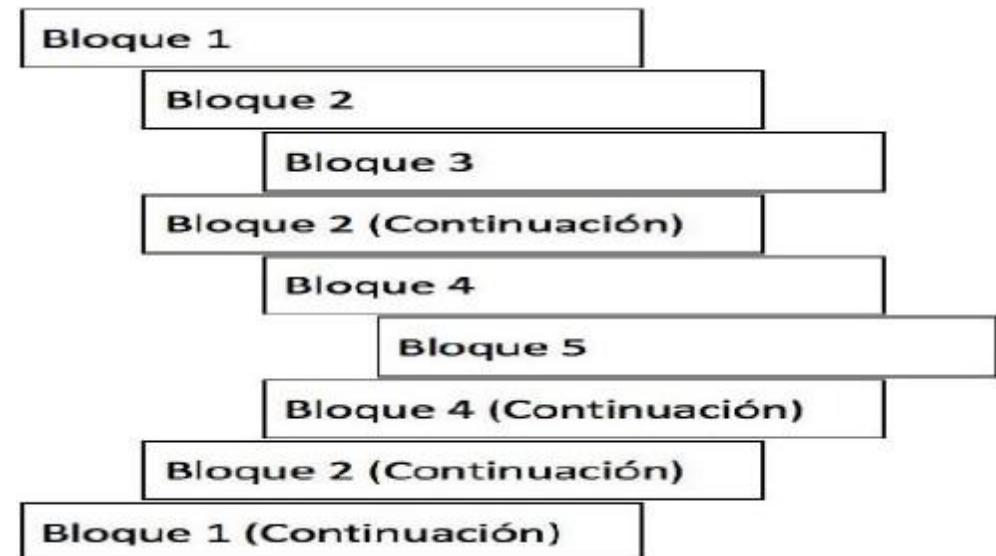
```
frutas3 = "melocoton"  
frutas3 not in frutas1
```

```
True
```

```
|
```

Bloques e indentación

- Un bloque es un grupo de sentencias de código fuente que contiene una o más sentencias. Los bloques están delimitados por su inicio y su fin, y la forma de delimitarlos es específica de cada lenguaje de programación.
- **Indentación** significa mover un bloque de texto hacia la derecha insertando espacios o tabuladores, para así separarlo del margen izquierdo y distinguirlo más fácilmente dentro del texto.



A black dumbbell lies diagonally across the frame, resting on a light-colored, textured surface that appears to be concrete or stone. The dumbbell has a black handle and two black weight plates. The background is plain white.

EJERCICIO 1

Condicionales y bucles

Condicional IF

```
if numero1>numero2 :  
    BloqueInstrucciones1  
elif numero1==numero2 :  
    BloqueInstrucciones2  
else :  
    BloqueInstrucciones3
```

Match

```
switch( variable ){
    case valor1: accion1;
    case valor2: accion2;
    case valor3: accion3;
    ...
    case valorN: accionN;

    default: accionD;
}
```

Bucle FOR

- En Python, los bucles for se ejecutan sobre elementos iterables, como pueden ser listas, tuplas, cadenas de texto o diccionarios.

```
lista = [1,2,3,4,5,6,7,8,9]  
for item in lista:  
    print(item, end=" ")
```

*for Variable in ColeccionIterable:
 BloqueInstrucciones*

```
i = 0  
while i<10:  
    print(i,end=" ")  
    i = i + 1
```

Bucle WHILE

*while Condición:
 BloqueInstrucciones*

EJERCICIO 2

Funciones

Funciones

```
def Saludar():
    print("¡Hola Time of Software!")
Saludar()
```

*def NombreFuncion (parámetros):
 BloqueInstrucciones
 return ValorRetorno*

Retornar mas de un elemento

- Se pueden devolver más de un elemento con return

```
def SumarRestar(param1, param2):  
    return param1 + param2, param1 - param2  
  
numero1 = int(input("Introduce el primer numero: "))  
numero2 = int(input("Introduce el segundo numero: "))  
resultadosuma, resultadoresta = SumarRestar(numero1,numero2)  
print("El resultado de la suma es: ", resultadosuma)  
print("El resultado de la resta es: ", resultadoresta)
```

Funciones con numero variable de argumentos

- En estas funciones los parámetros se reciben como una lista, por lo que tendrás que iterarla para poder procesarlos todos.

```
def Sumar(*valores):  
    resultado = 0  
    for item in valores:  
        resultado = resultado + item  
    return resultado
```

```
resultado = Sumar(23,56,3,89,78,455)  
print("El resultado de la suma es: ", resultado)
```

A close-up, low-angle photograph of a barbell and weights. The barbell is made of dark metal and has two weight plates attached. The top plate is engraved with '15 KG'. The background is a light-colored, textured surface.

EJERCICIO 3

Colecciones de objetos

Listas

```
lista = ["ordenador","teclado","raton"]
print(len(lista))
print(lista[0])
print(lista[1])
print(lista[2])
```

- Una lista es un conjunto ordenado de elementos que puede contener datos de cualquier tipo. Es el tipo de colección más flexible de todos.
- Las listas pueden contener elementos del mismo tipo o elementos de diferentes tipos.
- En Python las listas se delimitan con corchetes “[]”, con los elementos separados por comas.
- La propiedad len devuelve la longitud, es decir, el numero de elementos que contiene la lista.
- El primer elemento de una lista es el elemento 0, no el 1.

Tuplas

```
tupla = ("ordenador","teclado","raton")
print(tupla)
print(len(tupla))
print(tupla[0])
print(tupla[1])
print(tupla[2])
```

- Las tuplas son un conjunto de elementos ordenados e inmutables.
- Las tuplas en Python o tuples son muy similares a las listas, pero con dos diferencias.
- Son inmutables, lo que significa que no pueden ser modificadas una vez declaradas, y en vez de inicializarse con corchetes se hace con (). Dependiendo de lo que queramos hacer, las tuplas pueden ser más rápidas.

Conjuntos

- Los set en Python son un tipo de dato que permite almacenar varios elementos y acceder a ellos de una forma muy similar a las listas pero con ciertas diferencias:
- Los elementos de un set son únicos, lo que significa que no puede haber elementos duplicados.
- Los set son desordenados, lo que significa que no mantienen el orden de cuando son declarados.

```
# Conjuntos

conjunto_colores = {"rojo", "verde", "azul"}

conjunto_colores
{'azul', 'rojo', 'verde'}

for color in conjunto_colores:
    print(color)

azul
rojo
verde

conjunto_colores[0]

-----
TypeError                                         Traceback (most recent call last)
<ipython-input-12-6ccc7b158046> in <module>
      1 conjunto_colores[0]

TypeError: 'set' object does not support indexing
```

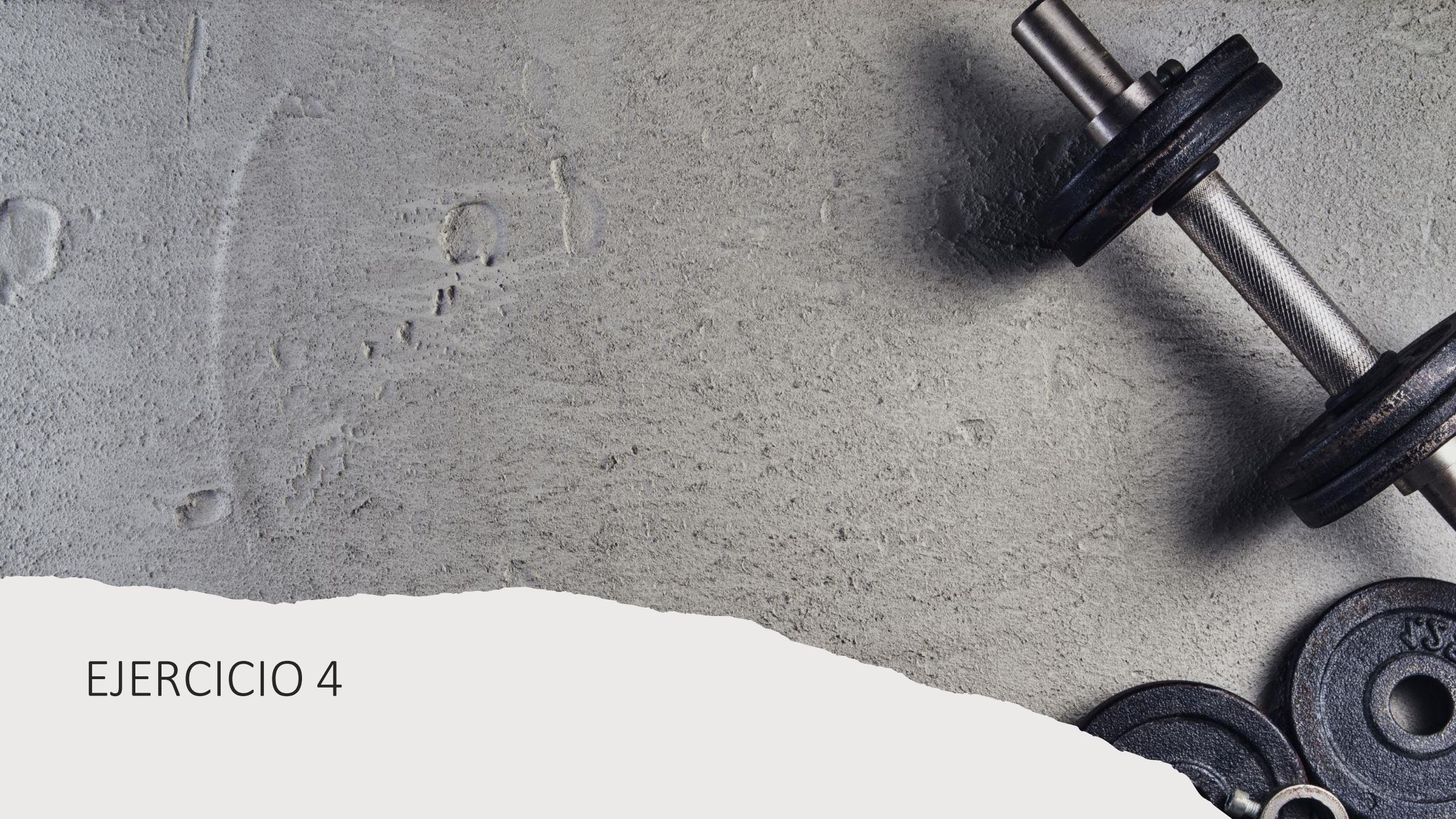
```
con|
```

Diccionarios

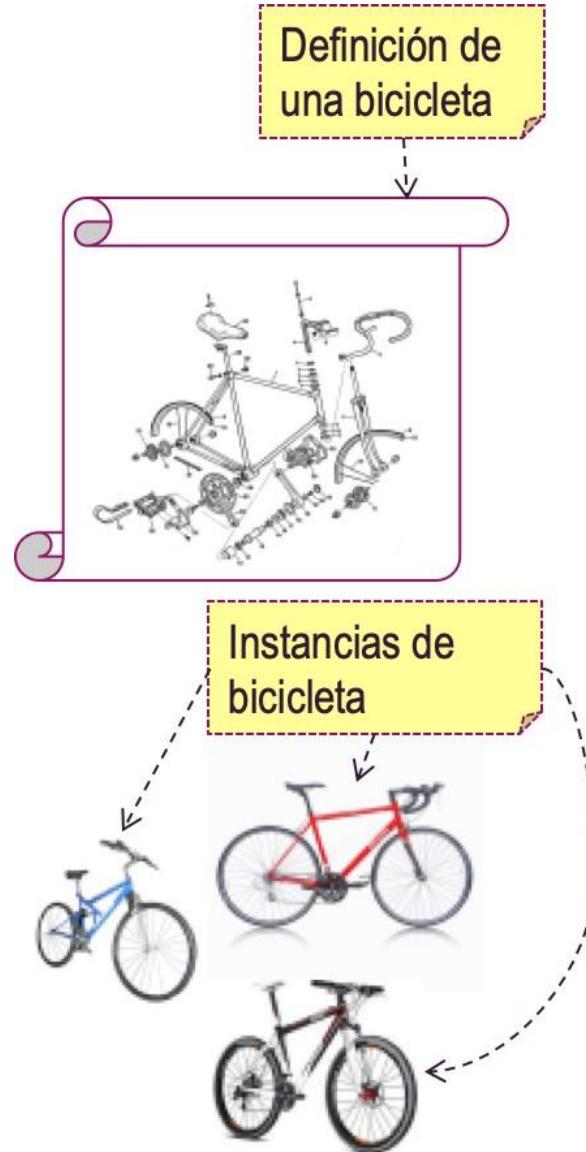
- Los diccionarios son colecciones de elementos compuestos por una clave y un valor asociado. Las claves en los diccionarios no pueden repetirse.
- En Python los diccionarios se delimitan por corchetes “{ }”, con los elementos separados por comas y la clave separada del valor mediante dos puntos.

```
mesestraducidos = {"Enero" : "January",
                    "Febrero" : "February",
                    "Marzo" : "March",
                    "Abril" : "April",
                    "Mayo" : "May",
                    "Junio" : "June",
                    "Julio" : "July",
                    "Agosto" : "August",
                    "Septiembre" : "September",
                    "Octubre" : "October",
                    "Noviembre" : "November",
                    "Diciembre" : "December"}
print(mesestraducidos["Noviembre"])
print(mesestraducidos["Mayo"])
```

EJERCICIO 4



Clases y objetos



Conceptos y miembros

- **Clase:** Una clase es la definición de las características concretas de una determinada tipología de objetos. Es decir, cuáles son los atributos y métodos de los que van a disponer todos los objetos de ese tipo. Equivale a un tipo de dato.
- **Objeto:** Un objeto es una agregado de atributos (información) y métodos (comportamiento), creado según la definición de una clase. A un objeto concreto que pertenece a una clase se le suele llamar instancia.

```
class Punto:  
    def __init__(self,x,y):  
        self.X = x  
        self.Y = y  
    def MostrarPunto(self):  
        print("El punto es (",self.X,",",self.Y,")")  
  
p1 = Punto(4,6)  
p1.MostrarPunto()
```

Clases

- Clases es uno de los conceptos con más definiciones en la programación, pero en resumen sólo son la representación de un objeto.
- Para definir la clase usas class y el nombre. En caso de tener parámetros los pones entre paréntesis.
- Para crear un constructor haces una función dentro de la clase con el nombre init y de parámetros self (significa su clase misma),

Composición



- Consiste en dividir el software en diferentes componentes, resolver cada uno por separado y por último unirlos en una solución única.
- Ante un problema complejo es más fácil resolverlo cuando se divide en piezas manejables.

Composición

- Composicion: Clases como atributos de otras clases

```
class Punto:  
    def __init__(self, x, y):  
        self.X = x  
        self.Y = y  
    def MostrarPunto(self):  
        print("El punto es (" ,self.X, "," ,self.Y, ")")
```

```
class Triangulo:  
    def __init__(self, v1,v2,v3):  
        self.V1 = v1  
        self.V2 = v2  
        self.V3 = v3  
    def MostrarVertices(self):  
        self.V1.MostrarPunto()  
        self.V2.MostrarPunto()  
        self.V3.MostrarPunto()
```

```
v1 = Punto(3,4)  
v2 = Punto(6,8)  
v3 = Punto(9,2)  
triangulo = Triangulo(v1,v2,v3)  
triangulo.MostrarVertices()
```

Encapsulación

- La definición de atributos privados se realiza incluyendo los caracteres _ entre la palabra “self.” y el nombre del atributo.

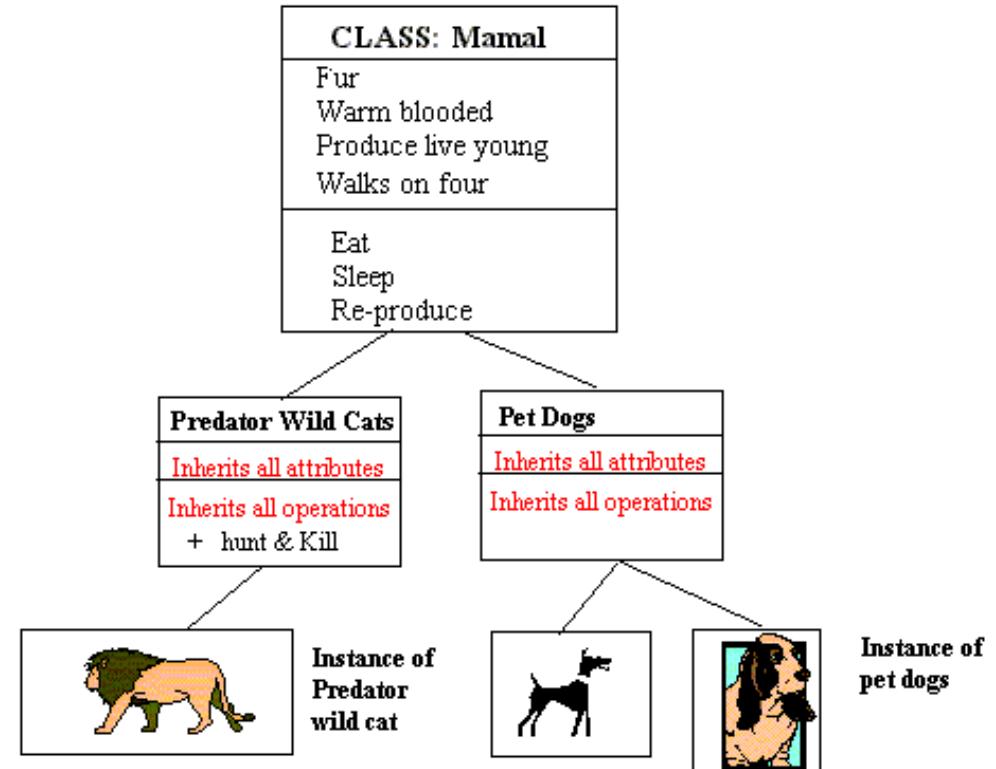
```
class PuntoPublico:  
    def __init__(self, x, y):  
        self.X = x  
        self.Y = y
```

```
class PuntoPrivado:  
    def __init__(self, x, y):  
        self.__X = x  
        self.__Y = y  
    def GetX(self):  
        return self.__X  
    def GetY(self):  
        return self.__Y  
    def SetX(self, x):  
        self.__X = x  
    def SetY(self, y):  
        self.__Y = y
```

```
publico = PuntoPublico(4,6)  
privado = PuntoPrivado(7,3)  
print("Valores punto publico:", publico.X, ", ", publico.Y)  
print("Valores punto privado:", privado.GetX(), ", ", privado.GetY())  
publico.X = 2  
privado.SetX(9)  
print("Valores punto publico:", publico.X, ", ", publico.Y)  
print("Valores punto privado:", privado.GetX(), ", ", privado.GetY())
```

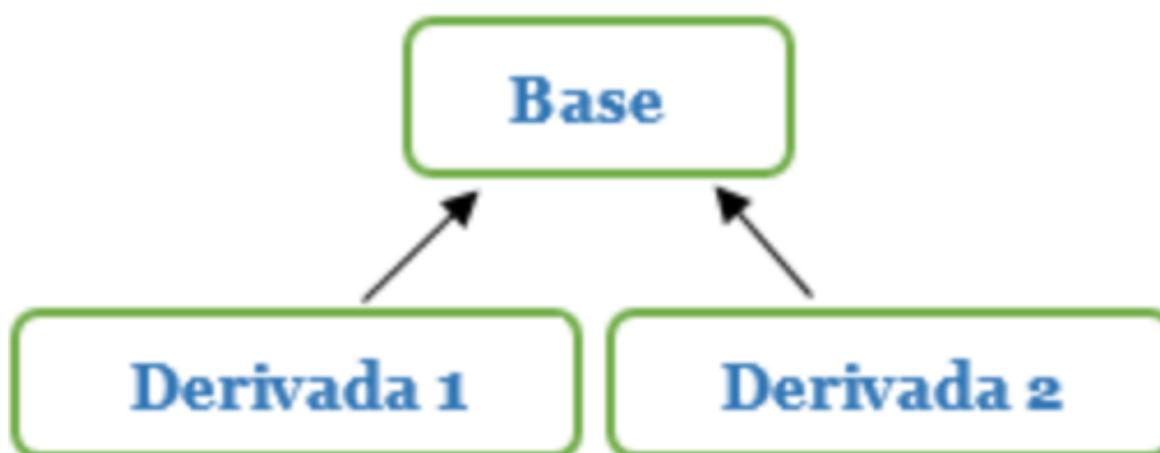
Herencia

- Las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación.
 - Los objetos heredan las propiedades, relaciones y los comportamientos de todas las clases a las que pertenecen.
 - La herencia organiza y facilita el polimorfismo y el encapsulamiento, permitiendo compartir y extender las propiedades y el comportamiento sin tener que volver a implementarlo.

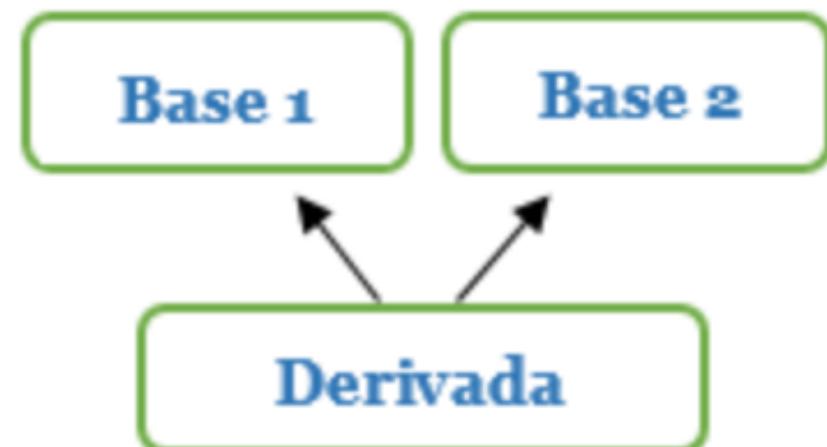


Herencia múltiple

Herencia simple



Herencia múltiple



Herencia múltiple

- En Python si que se permite la herencia múltiple, clase que hereda de varias clases.

```
class Negocio(Hotel, Restaurante):  
    def __init__(self):  
        self.__Nombre = ""  
        self.__Direccion = ""  
        self.__Telefono = 0  
    def SetNombre(self, nombre):  
        self.__Nombre = nombre  
    def SetDireccion(self, direccion):  
        self.__Direccion = direccion  
    def SetTelefono(self, telefono):  
        self.__Telefono = telefono  
    def MostrarNegocio(self):  
        print("#####")  
        print("Negocio:")  
        print("\tNombre:", self.__Nombre)  
        print("\tDireccion:", self.__Direccion)  
        print("\tTelefono:", self.__Telefono)  
        self.MostrarHotel()  
        self.MostrarRestaurante()  
        print("#####")
```



EJERCICIO 5

Elementos de programación funcional

Funciones anidadas

```
def SumarRestar(param1, param2):
    return Sumar(param1,param2), Restar(param1,param2)

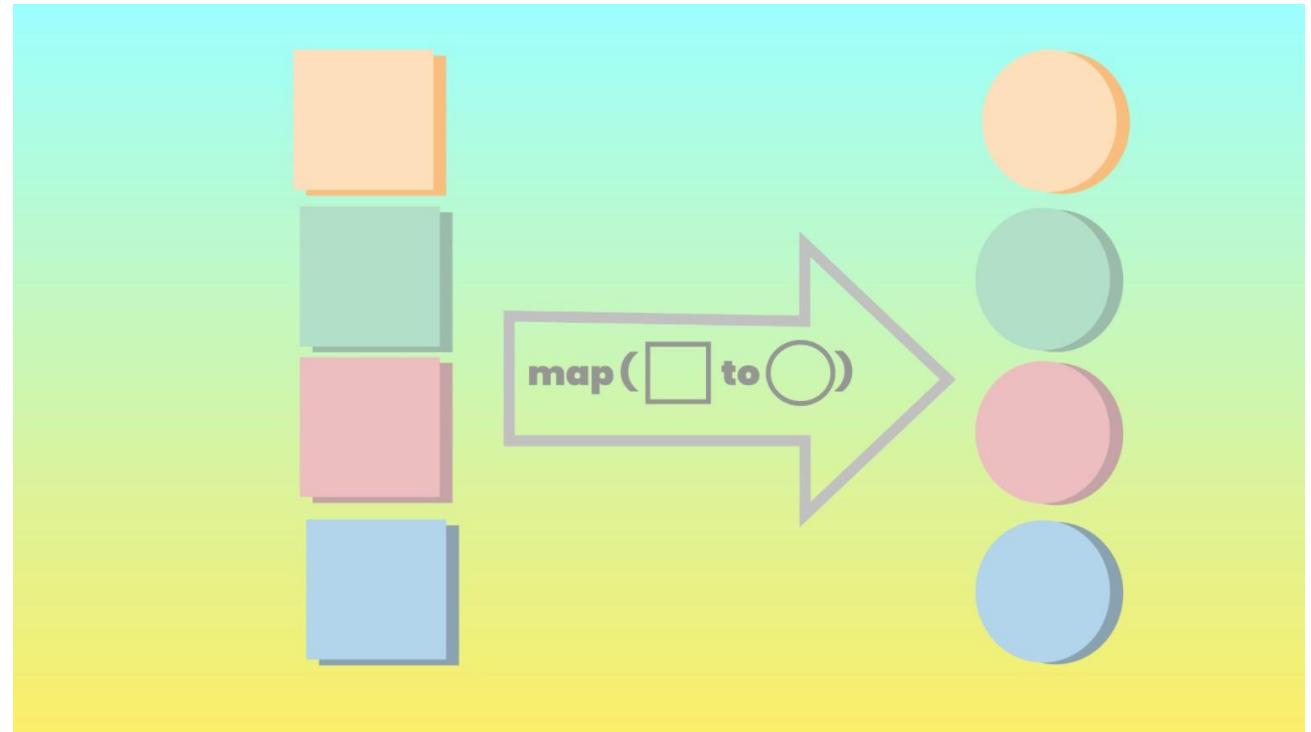
def Sumar(sumando1, sumando2):
    return sumando1 + sumando2

def Restar(minuendo, sustraendo):
    return minuendo - sustraendo

numero1 = int(input("Introduce el primer numero: "))
numero2 = int(input("Introduce el segundo numero: "))
resultadosuma, resultadoresta = SumarRestar(numero1,numero2)
print("El resultado de la suma es: ", resultadosuma)
print("El resultado de la resta es: ", resultadoresta)
```

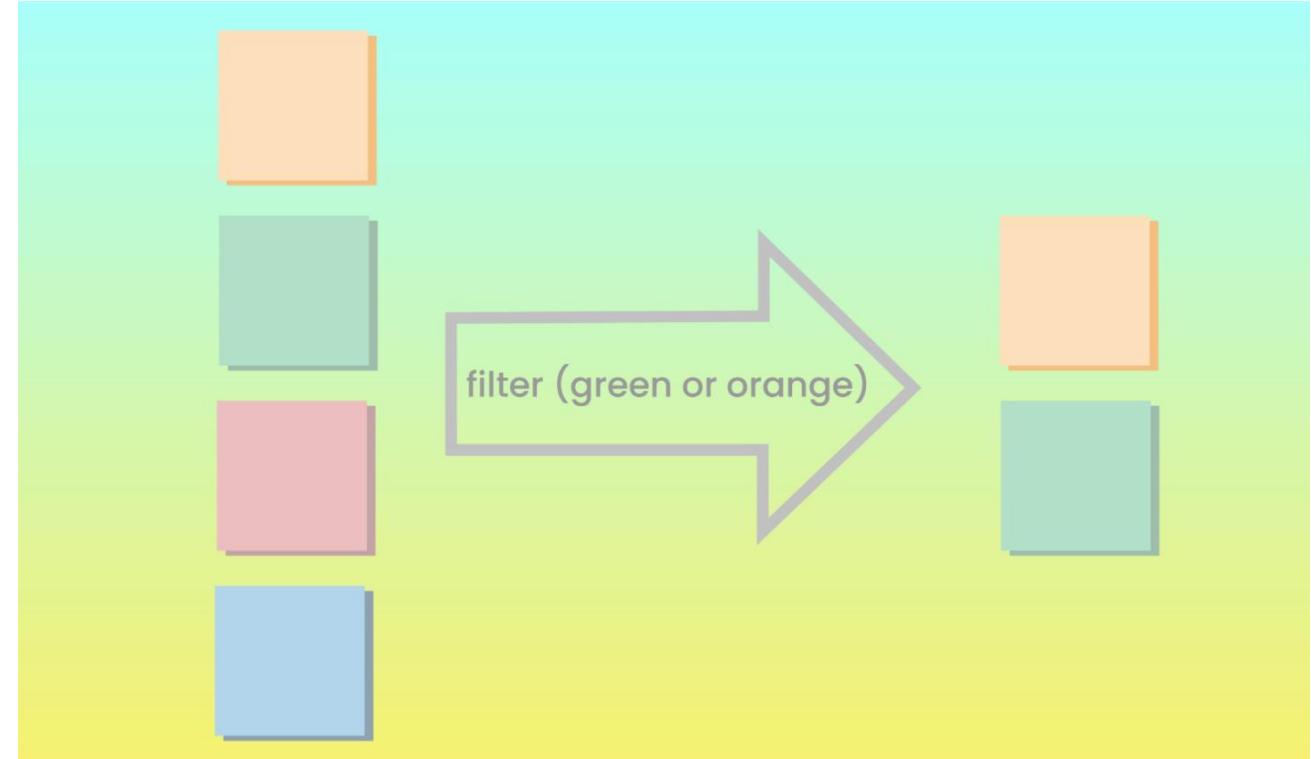
Función map

- La función map nos permite aplicar una función sobre cada uno de los elementos de un colección (Listas, tuplas, etc...).

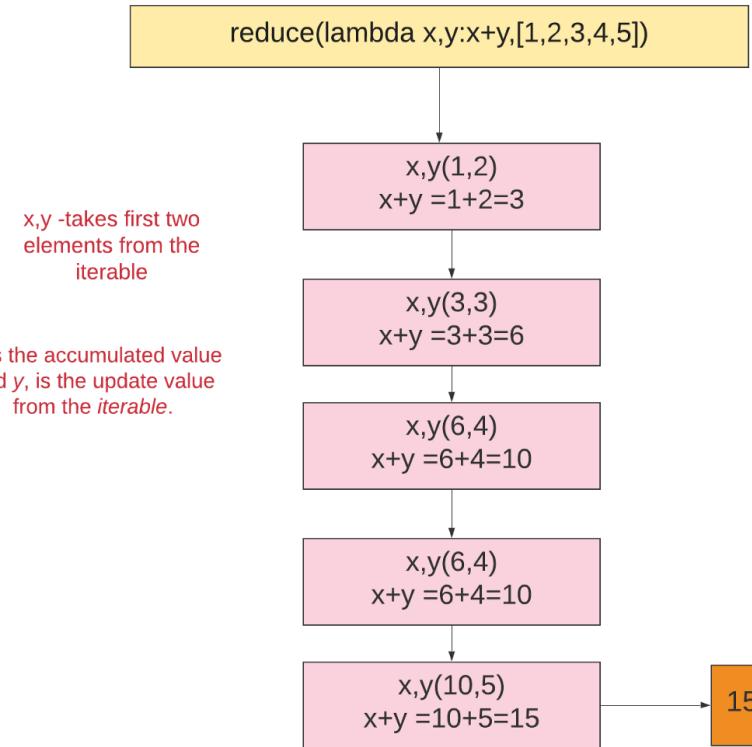


Función filter

- La función filter, es quizás, una de las funciones más utilizadas al momento de trabajar con colecciones. Cómo su nombre lo indica, esta función nos permite realizar un filtro sobre los elementos de la colección.



Función reduce

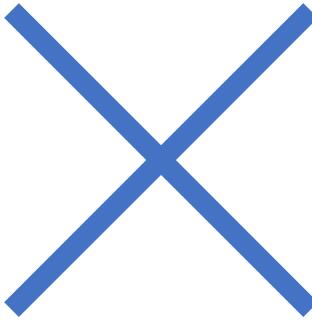


lambda argumentos : cuerpo de la función

Funciones lambda

- Una función lambda es una función anónima, una función que no posee un nombre. En Python la estructura de una función lambda es la siguiente.

EJERCICIO 6



Gestión de errores

Tipos de excepciones

En Python existen diferentes tipos de excepciones que pueden controlarse, todas ellas derivan de una serie de excepciones base.

- **Exception:** tipo de excepción más genérica, de ella derivan todas las excepciones existentes en Python.
- **ArithmetError:** tipo de excepción genérica para errores aritméticos.
- **BufferError:** tipo de excepción genérica para errores relacionados con buffers.
- **LookupError:** tipo de excepción genérica para errores relacionados con acceso a datos de colecciones.

```
try:  
    print(3/0)  
except:  
    print("ERROR: Division por cero")
```

try-except

try-except-finally

```
print("¡Iniciando programa!")
try:
    print(3/0)
except:
    print("ERROR: Division erronea")
finally:
    print("¡Programa acabado!")
```

```
print("¡Iniciando programa!")
try:
    print(3/1)
except:
    print("ERROR: Division erronea")
else:
    print("¡No se han producido errores!")
finally:
    print("¡Programa acabado!")
```

try-except-else-finally

- Consiste en añadir un bloque de instrucciones else que se ejecuta cuando no se lanzan excepciones y no se desea incluir ese bloque de código dentro del bloque final.

Captura varias excepciones

```
print("¡Iniciando programa!")
try:
    print(3/0)
except ZeroDivisionError:
    print("ERROR: Division por cero")
except:
    print("ERROR: General")
else:
    print("¡No se han producido errores!")
finally:
    print("¡Programa acabado!")
```

A close-up, low-angle shot of a row of black dumbbells in a gym. The dumbbells are arranged in a curve, with the handles pointing towards the bottom left. The background is dark and out of focus.

EJERCICIO 7



BBDD



CREATE



READ



UPDATE



DELETE

C R U D

Cheatography

SQLite Cheat Sheet

by fettobse (fettobse) via cheatography.com/27664/cs/8070/

Basics

```
sqlite      Will create a new  
filename.db   database
```

Create Tables

```
CREATE TABLE name (  
    colname type constraints,  
    colname type constraints  
)
```

This will create a new table. For types and constraints look right.

Insert Data

```
INSERT INTO tablename (column2,  
                      column4)  
VALUES  
(data1, data2),  
(data3, data4);
```

You don't have to fill every column of the table. It is often useful to leave out columns with primary keys or default value insertion.

Update Tables

```
UPDATE tablename  
SET column5 = value5  
WHERE expression
```

This will update the given column with the given value, where the expression evaluates true (usually specify primary key).

Alter Tables

The ALTER TABLE function is limited in SQLite.

You can either rename the table with:

```
ALTER TABLE tablename RENAME TO  
newtablename
```

Or add a new column with:

```
ALTER TABLE tablename ADD COLUMN  
columndefintion
```

It is not possible to change column definitions of existing tables.

Drop Table

```
DROP TABLE tablename
```

Select from Table

```
SELECT table1.column1,  
       table2.column2,  
       FROM table1, table2  
       WHERE expression  
       ORDER BY table1.column1 order
```

The ORDER BY is optional, possible ways to order are ASC (ascending) and DESC (descending).

Data Types

NULL null value

INTEGER Integer value

REAL 8-byte floating point number

TEXT Text with UTF encoding

BLOB Blob-Data (stored like input)

BOOLEAN Will be stored as Integer (0 for false, 1 for true)

Constraints

PRIMARY KEY Sets column as primary key.
The constraints NOT NULL and UNIQUE will be set automatically.

NOT NULL Prevents the column from empty values (null).

UNIQUE Each value in the column has to be unique.

INTEGER PRIMARY KEYS will automatically become the ROWID and therefore will be automatically incremented. This column can be accessed as ROWID, OID, _ROWID_ or the column name.

Obtener el cursor:

```
cursor = conexion.cursor()
```

Crear la tabla:

```
sql = "CREATE TABLE ALUMNOS (nombre TEXT, edad INTEGER, nota REAL)"  
cursor.execute(sql)
```

Efectuar el commit:

```
conexion.commit()
```

Crear tabla

Insertar un registro:

```
cursor.execute("INSERT INTO ALUMNOS VALUES ('Juan', 26, 7.5)")
```

Insertar varios registros desde una lista:

```
lista_alumnos = [('Pedro', 22, 4.5), ('Maria', 25, 6.8), ('Jorge', 24, 9.2)]
sql = "INSERT INTO ALUMNOS VALUES (?, ?, ?)"
cursor.executemany(sql, lista_alumnos)
```

Efectuar el commit:

```
conexion.commit()
```

Insertar datos

```
cursor.execute("SELECT * FROM ALUMNOS")
```

```
cursor.execute("SELECT * FROM ALUMNOS where nota < 5")
```

```
cursor.execute("SELECT * FROM ALUMNOS where edad >= 23 ORDER BY nombre DESC")
```

Consultas

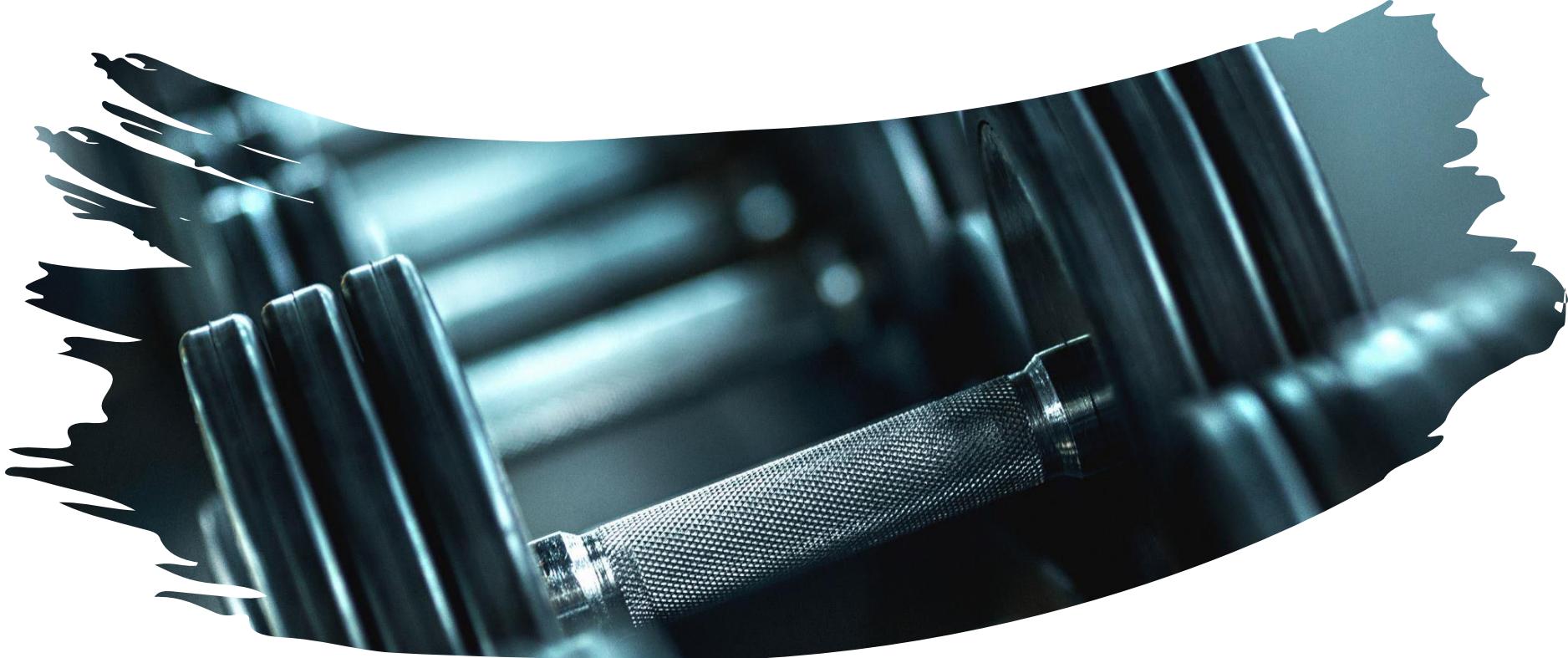
 Modificar:

```
cursor.execute("UPDATE ALUMNOS SET nota = 5 where nombre = 'Pedro' ")
```

 Eliminar:

```
cursor.execute("DELETE FROM ALUMNOS where nombre = 'Juan' ")
```

Modificar y eliminar datos



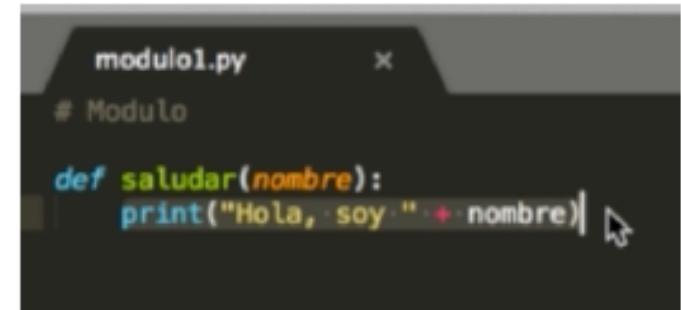
EJERCICIO 9

Módulos

Modulos en Python

- Un módulo es un fichero que contiene un conjunto de funciones que puedes incluir y usar en tu aplicación.

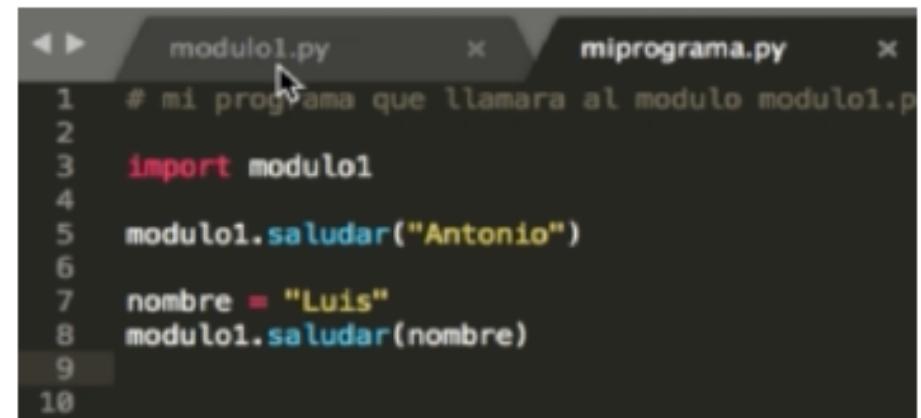
Declaración del módulo



```
modulo1.py
# Modulo

def saludar(nombre):
    print("Hola, soy " + nombre)
```

Uso del módulo

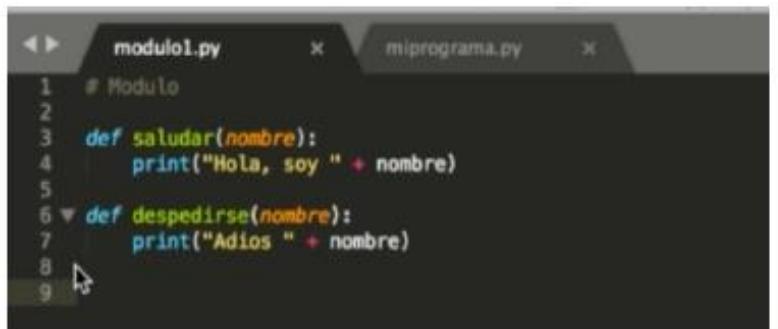


```
miprograma.py
# mi programa que llamara al modulo modulo1.py

1 # mi programa que llamara al modulo modulo1.py
2
3 import modulo1
4
5 modulo1.saludar("Antonio")
6
7 nombre = "Luis"
8 modulo1.saludar(nombre)
9
10
```

Módulos

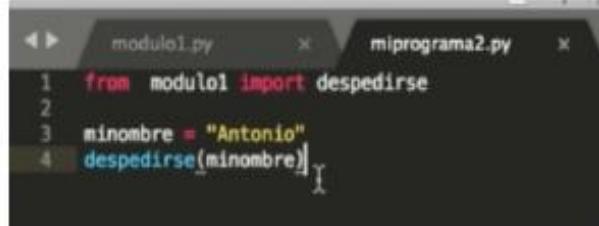
Declaración de otra función



```
modulo1.py
1 # Modulo
2
3 def saludar(nombre):
4     print("Hola, soy " + nombre)
5
6 def despedirse(nombre):
7     print("Adios " + nombre)
8
9

miprogramma.py
```

Uso del módulo



```
modulo1.py
1 from modulo1 import despedirse
2
3 minombre = "Antonio"
4 despedirse(minombre)

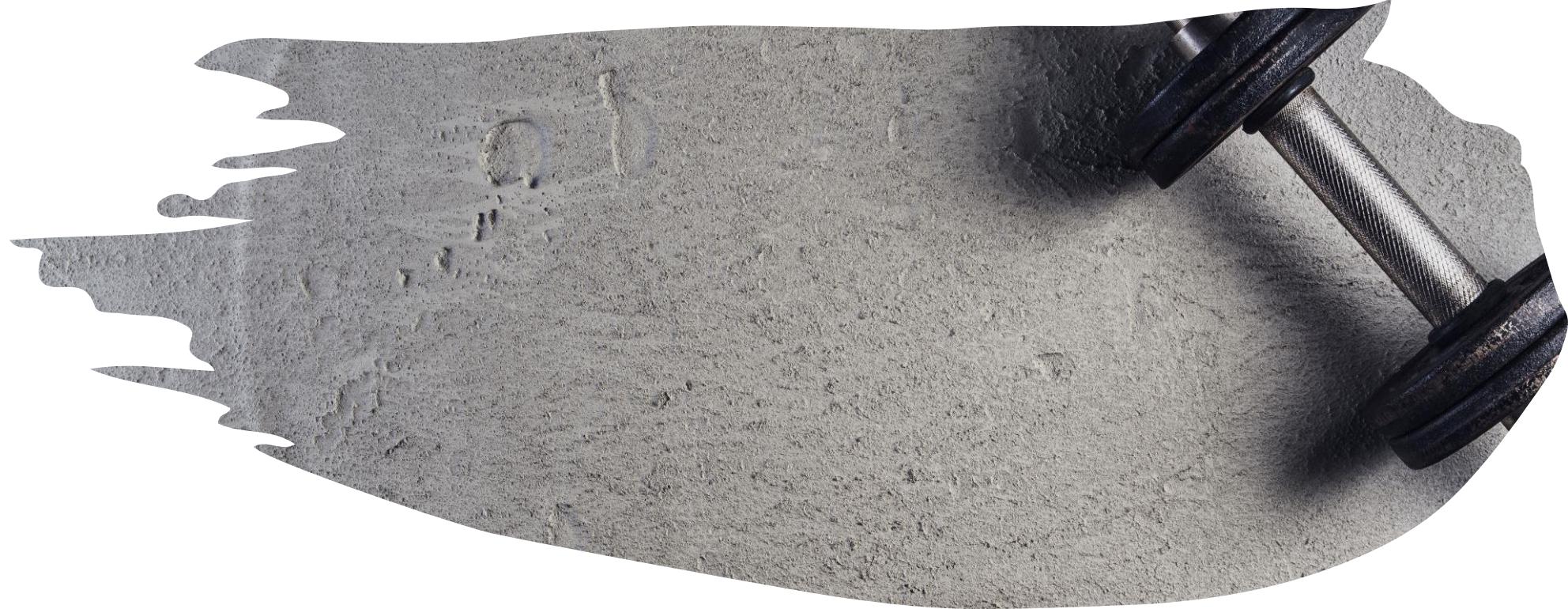
miprogramma2.py
```

Con alias

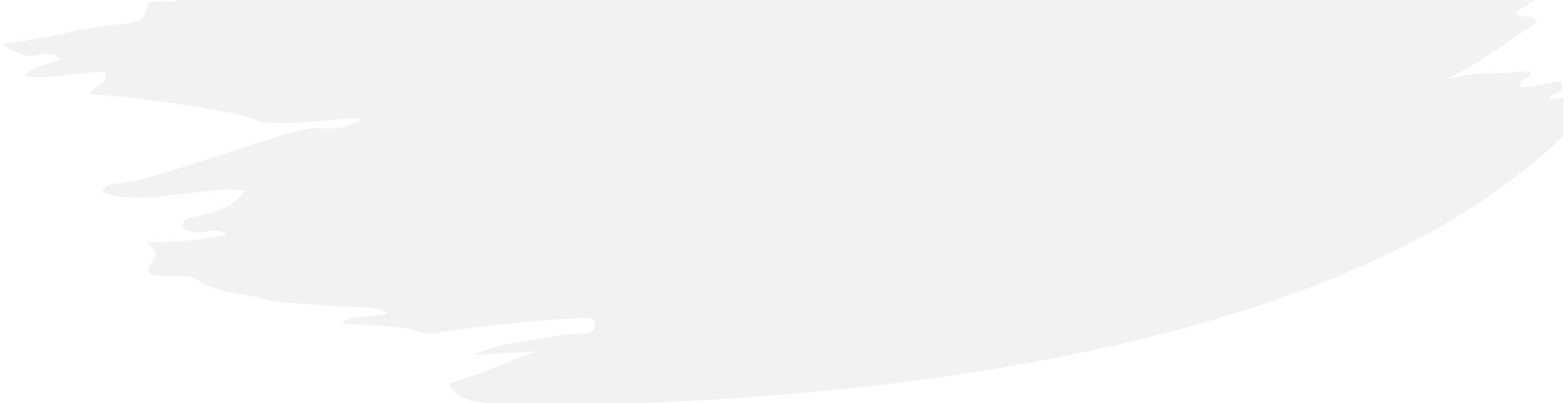


```
modulo1.py
1 from modulo1 import despedirse as adios
2
3 minombre = "Antonio"
4
5 adios(minombre)

miprogramma2.py
```



EJERCICIO 10



Pandas

Pandas

- Esta librería es fundamental en el tratamiento de datos.
- Nos permite filtrar datos, manipularlos, transformarlos, etc.
- Podemos leer y escribir diferentes tipos de ficheros como csv y excel. Tambien en BBDD.
- Veremos dos de las tres estructuras que tiene Pandas:
 - Series
 - Data Frames

Series

Características:

- Son tipo vectores o arrays de una dimension
- Todos los datos tienen que ser del mismo tipo
- Contiene indices, es decir, un elemento de la serie puede asociarse con ese indice. Esto es muy util a la hora de acceder a los datos

Acceder a los elementos de una serie:



Por posición:

```
#Ejemplo 1: quiero los dos primeros elementos  
print('Ejemplo 1: \n ', serieDesdeDict[:2], '\n')
```

```
#Ejemplo 2: quiero los tres ultimos elementos  
print('Ejemplo 2: \n ', serieDesdeDict[-1])
```



Por indice

```
#Ejemplo 3: quiero el elemento cuyo indice es el 0  
print('Ejemplo 3: \n ', serieDesdeDict[0], '\n')
```

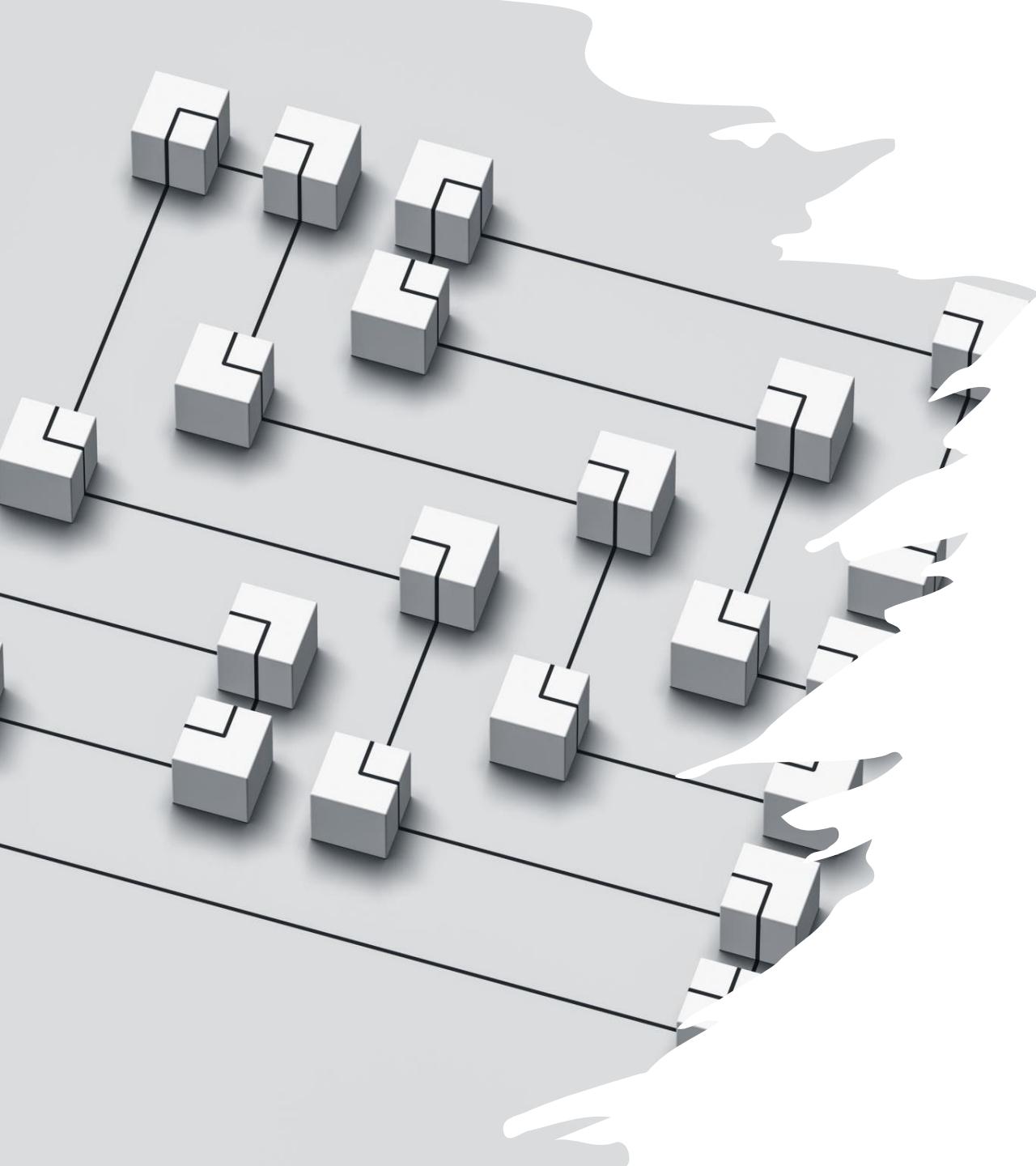
```
#Ejemplo 4: quiero los elementos cuyos indices son el 1 y el 4  
print('Ejemplo 4: \n ', serieDesdeDict[[1, 4]])
```

```
serieFiltro = serieDesdeDict.apply(lambda x: x in ['Pedro', 'Juan'])
print(serieFiltro)
```

Apply()

Método apply junto con funciones lambdas:

- La función apply() aplica una función a cada elemento a lo largo de la serie.
- La sintaxis es la siguiente:
- La ventaja de apply es que nos evitamos usar los bucles, que realmente son más lentos, sobre todo con datos muy grandes.



Data Frames

Características:

- Son tablas, es decir, son de 2 dimensiones.
- Cada columna es una serie.
- Contiene dos índices, uno columnar y otro por fila como las series. Y estos deben ser únicos.

Data frame a partir de una lista:

```
dfDesdeLista = pd.DataFrame(data)
```

A partir de una lista de lista:

```
data2 = [['Juan', 4.5], ['Pedro', 8.9], ['Estefania', 1.4], ['Ana', 5.6], ['Esteban', 7.8]]
columnasNombres = ["Nombre", "Notas"]
filas = list(range(5))

dfDesdeListDeLista = pd.DataFrame(data2, columns=columnasNombres, index=filas)
```

A partir de un diccionario:

```
dictNombreNotas1 = { "Id": [1,2,3,4,5] ,
                     "Nombre" : ['Juan', 'Pedro', 'Estefania', 'Ana', 'Esteban'] ,
                     "Apellido":['Garcia', 'Sanchez', 'Lopez', 'Garcia', 'Gonzalez' ],
                     "Notas": [4.5, 8.9, 1.4, 5.6, 7.8]}

dfDesdeDict1 = pd.DataFrame(dictNombreNotas1)
```

A partir de la lectura de un csv:

```
dfDatosPersonales = pd.read_csv('C:/DatosPersonales.csv',sep=";")
```

Atributos de DataFrame

Atributos de los data frames:

- info: nos devuelve toda la información referente al data frame
- shape: nos devuelve el número de filas y columnas del data frame
- size: devuelve el número de elementos en el data frame
- columns: nos da una lista con el nombre de las columnas
- index: nos devuelve una lista con el nombre de los índices fila
- head(n): nos da los n primeros elementos del data frame

Acceder a los datos

Acceder a los datos:

- Por posicion se usa "iloc"
- Por nombre del indice fila es traves de "loc"
- Por nombre de columna es df[nombre de la columna] o la lista con los nombres de las columnas si queremos varias a la vez

Insertar nuevos datos

- insert: nos permite agregar una nueva columna, siendo el dato que se inserta una lista.
 - Este método tiene tres parametros:
 - loc:le decimos donde agregar la columna
 - column: nombre de la nueva columna
 - value: la lista de datos a insertar

```
resultado = ['Desaprobado', 'Aprobado', 'Desaprobado', 'Desaprobado', 'Aprobado']
dfDesdeDict1.insert(loc = 3, column = 'Resultado', value = resultado)
print(dfDesdeDict1)
```

JOIN

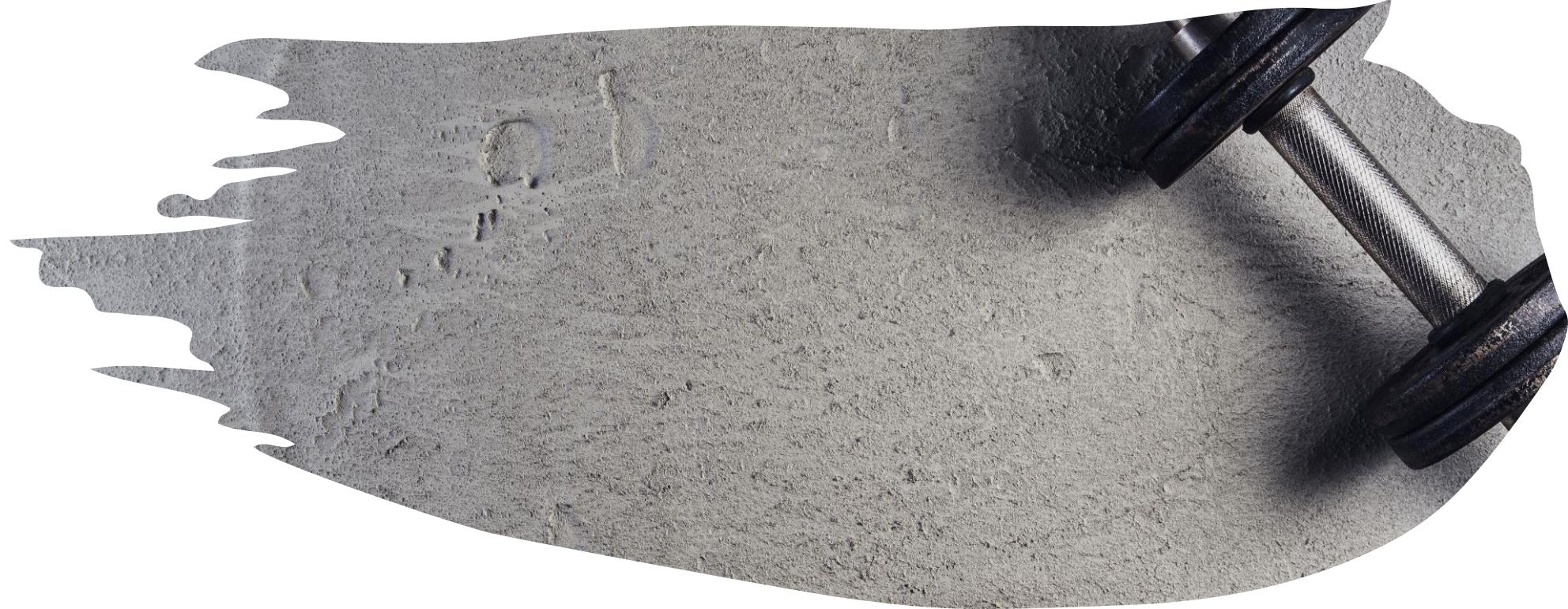
merge: Para poder unir dos data frames a través de una o varias columnas se utiliza el método merge. Existen cuatro tipos de uniones:

- inner: une solo lo que tienen en común ambos data frames
- left: unimos el df1 con el df2 sin perder información del df1
- right: unimos el df1 con el df2 sin perder información del df2
- outer: unimos ambos data frames sin perder nada de información

```
dfTotal = dfDatosPersonales.merge(dfDesdeDict3, how = 'inner', on = ['Id'])
```

```
dfTotal[(dfTotal['Resultado'] == 'Desaprobado') & (dfTotal['Apellido'] == 'Garcia')]  
    .sort_values('Notas', ascending=False)
```

Filtrar valores



EJERCICIO 11

Introducción a la biblioteca estándar del lenguaje

Librerías más utilizadas

- **numpy**; calculo matemático, vectores, matrices, números aleatorios, ...etc
- **pandas**; tratamiento de datos
- **matplotlib**; para dibujar graficas
- **sklearn**; regresiones, arboles decisión, regularización LASSO, Ridge, clasificacion
- **graphviz**; para visualizar arboles