

# INTRODUCCIÓN A LA PROGRAMACIÓN

**Python - Módulos y Paquetes.**



# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

La modularidad es la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan **independiente** como sea posible de la aplicación en sí y de las restantes partes.

Estos módulos son **archivos** que alojan **código independiente con una determinada funcionalidad**, y cuando hablamos de independencia nos referimos a que pueden ser **reutilizados por otras aplicaciones** y no dependen directamente de la aplicación original en la cual lo utilizamos.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Cualquier programa Python no trivial se organizaría en varios archivos, conectados entre sí mediante importaciones. Python, como la mayoría de los otros lenguajes de programación, utiliza esta estructura de programa modular, donde las funcionalidades se agrupan en unidades reutilizables. En general, podemos distinguir tres tipos de archivos en una aplicación Python de múltiples archivos:

- ✓ Archivo de nivel superior.
- ✓ Módulos de biblioteca estándar.
- ✓ Módulos definidos por el usuario.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

**Archivo de nivel superior:** un archivo de Python, o secuencia de comandos, que es el punto de entrada principal del programa. Este archivo se ejecuta para iniciar su aplicación.

```
import myModulo
Import random
def main():
    print("Hello World!")
X = random.random()

if __name__ == "__main__":
    main()
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

**Módulos de biblioteca estándar:** módulos precodificados que están integrados en el paquete de instalación de Python, como herramientas independientes de la plataforma para interfaces del sistema, secuencias de comandos de Internet, construcción de GUI y otros.

Un ejemplo claro es el módulo “**math**” por ejemplo que en su código incluye varias operaciones matemáticas que podemos llamar como una función luego de importar el módulo.

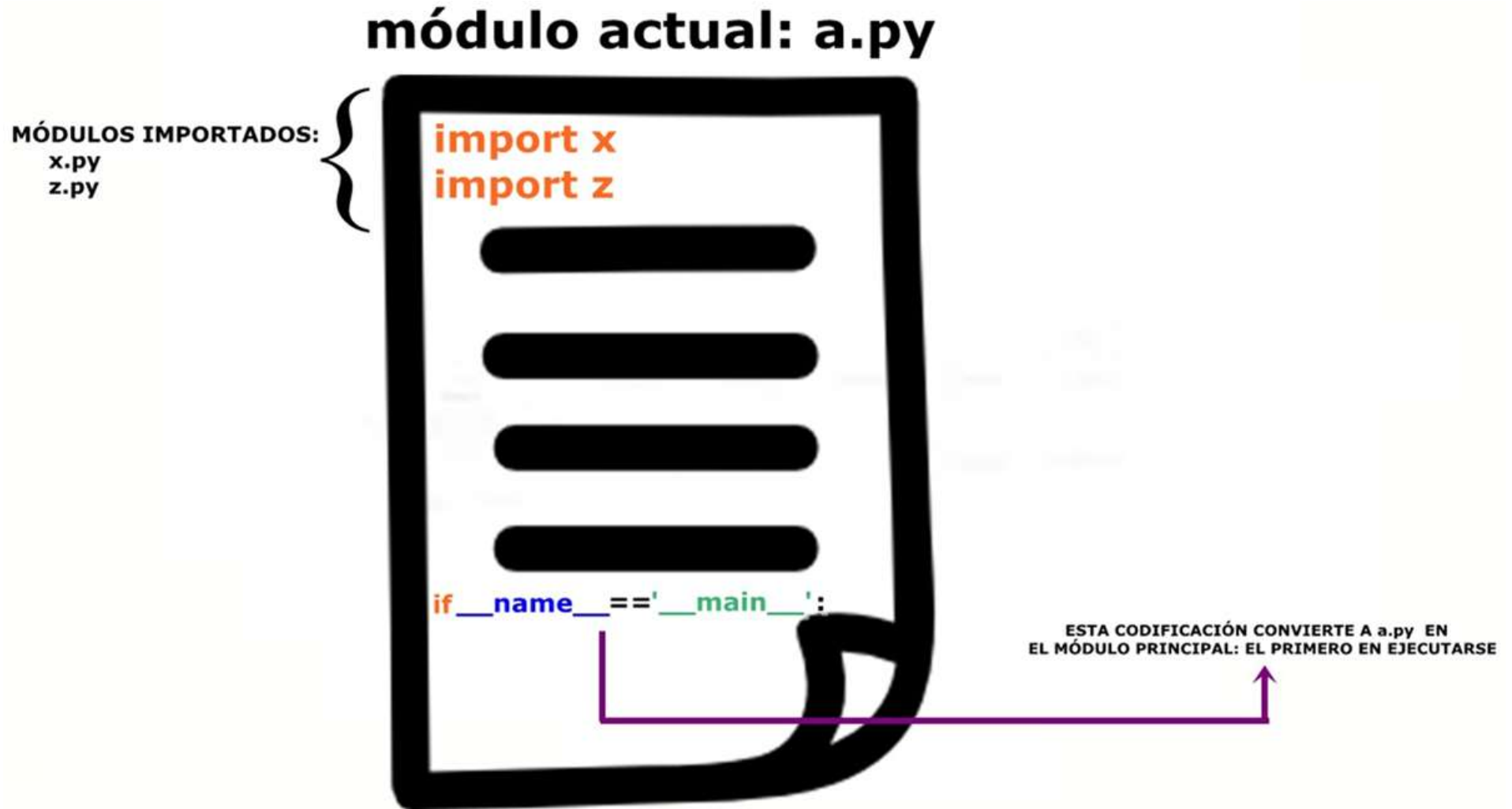
# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

**Módulos definidos por el usuario:** archivos de Python que se importan al archivo de nivel superior, o entre sí, y proporcionan funcionalidades independientes. Por lo general, estos archivos no se inician directamente desde el símbolo del sistema y están hechos a medida para el propósito del proyecto.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.



# INTRODUCCIÓN A LA PROGRAMACIÓN

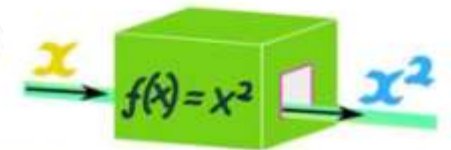
## Programación modular.

- Un módulo es un contenedor lleno de **funciones**, se pueden empaquetar tantas funciones como se deseen en un módulo y distribuirlo por todo el mundo.
- Se deber **agrupar las funciones por afinidad** y asignar un nombre claro e intuitivo al módulo que las contiene.
- Los módulos se pueden **agrupar en paquetes**.

## Función

### Definición

- Subalgoritmo, parte de un algoritmo principal que **permite resolver una tarea específica**
- Puede ser invocada desde otras partes del algoritmo cuantas veces se necesite
- Puede recibir valores (parámetros o argumentos)
- Puede devolver valores





# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Los **módulos** son entidades que permiten una **organización y división lógica** de nuestro código. Los **ficheros** son su contrapartida física. Cada archivo Python **.py** almacenado en disco equivale a un módulo.

Un **paquete** es un **conjunto de módulos** que tienen una **finalidad específica** dentro del programa que los utiliza, almacenados dentro de una misma carpeta.

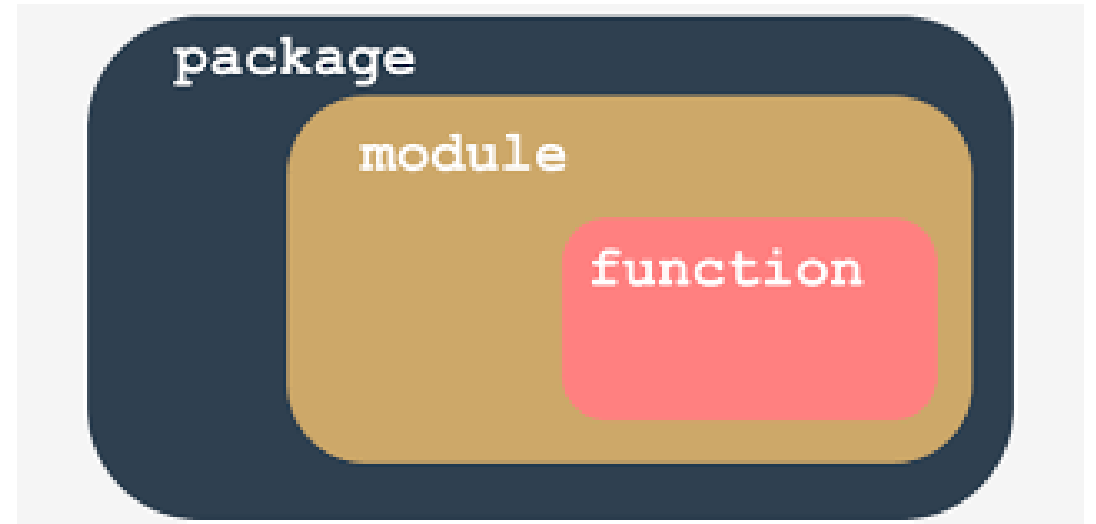
Crear un paquete no solo es ubicar los módulos dentro de una carpeta, **estos módulos deben tener una relación particular**, de lo contrario no sería necesario almacenarlos juntos en la misma ubicación.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Si los módulos sirven para organizar el código, los paquetes sirven para organizar los módulos. Los paquetes permiten agrupar módulos relacionados.

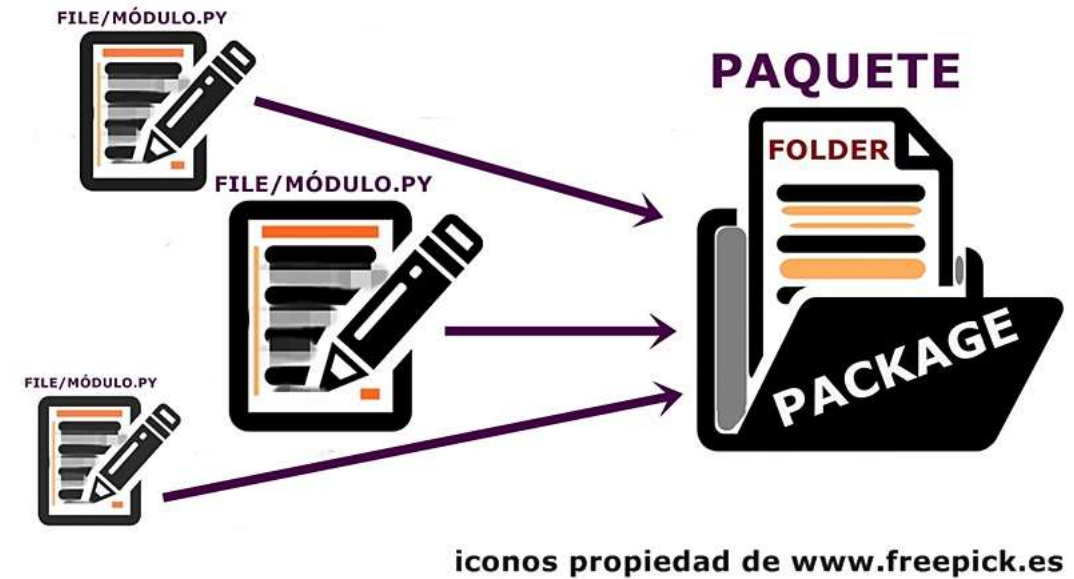
Mientras los módulos se corresponden a nivel físico con los archivos, los paquetes se representan mediante directorios.



# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

En Python, cada uno de nuestros archivos .py se denominan módulos. Estos módulos, a la vez, pueden formar parte de paquetes. Un paquete, es una carpeta que contiene archivos .py. Pero, para que una carpeta pueda ser considerada un paquete, debe contener un archivo de inicio llamado **`__init__.py`**. Este archivo, no necesita contener ninguna instrucción. De hecho, puede estar completamente vacío.

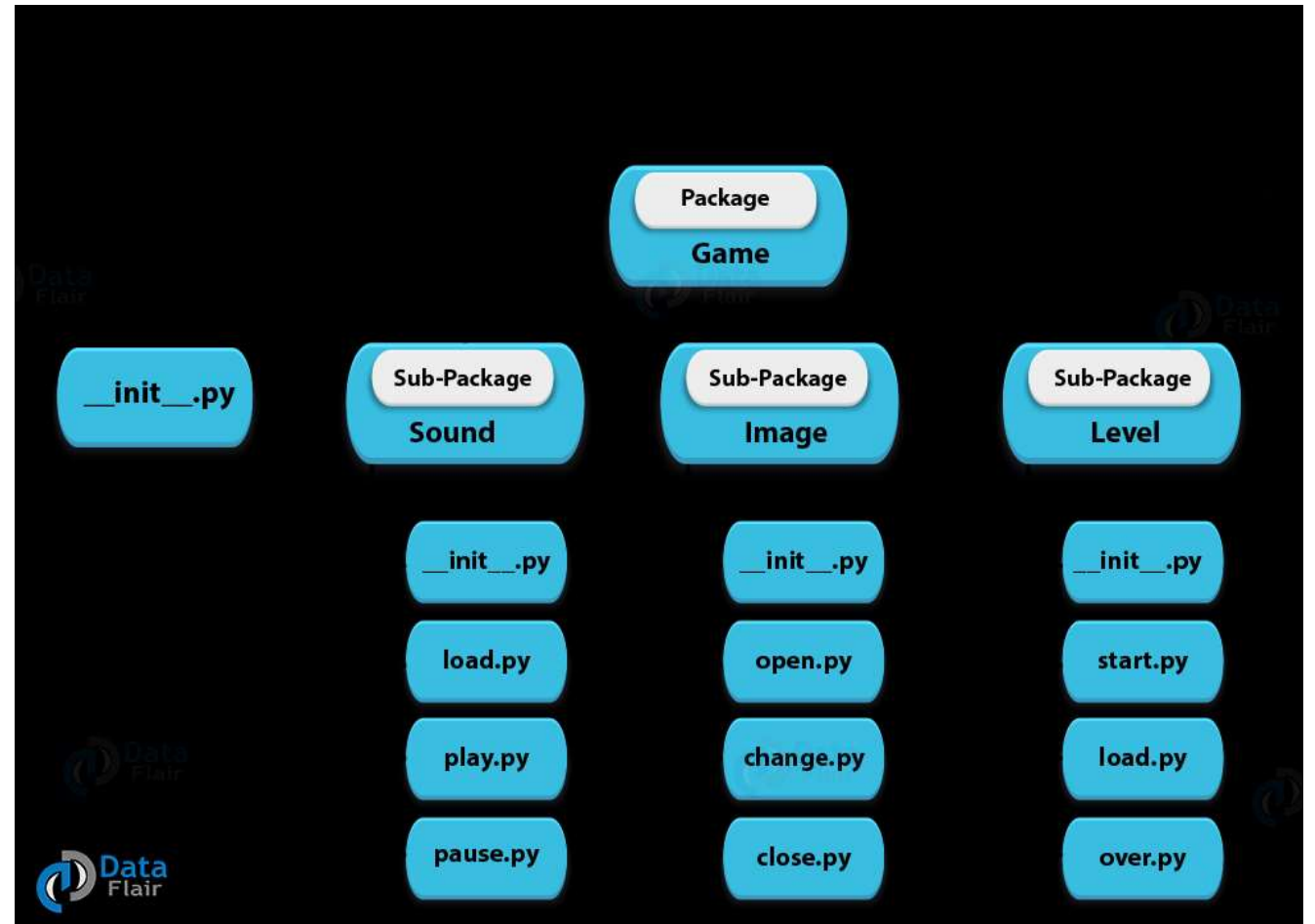


En Python3.x, estos archivos son opcionales y puede usarlos si es necesario.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Los paquetes, a la vez, también pueden contener otros sub-paquetes.



Y los módulos, no necesariamente, deben pertenecer a un paquete.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

De lo dicho hasta ahora se deducen varias conclusiones importantes:

- Un módulo es una parte de un programa.
- Los módulos nos permiten descomponer el programa en partes más o menos independientes y manejables por separado.
- Los módulos, en general, agrupan colecciones de funciones interrelacionadas.
- Los módulos, además de funciones, pueden contener datos en forma de constantes y variables manipulables desde el interior del módulo así como desde el exterior del mismo usando las funciones que forman el módulo.
- A nivel práctico, los módulos se programan físicamente en archivos separados del resto del programa.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Python incluye una gran colección de módulos. Hay más de 200 módulos para tareas de programación comunes.

Estos módulos son llamados **biblioteca estándar**, puedes encontrarlos en la mayoría de las instalaciones en todas las plataformas.

### ¿Qué es?

- La librería estándar contiene varios tipos de componentes.
  - Tipos de datos que pueden ser considerados como parte del núcleo de Python (números y listas).
  - Funciones internas y excepciones.
  - La mayor parte de la librería está constituida por un amplio conjunto de módulos que permiten expandir la funcionalidad de Python.
- Se distribuye junto con el intérprete.
- Con cada nueva versión de Python, se mejora y amplía la funcionalidad.

<http://docs.python.org/library/index.html>

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Algunos de los módulos de la librería estándar son los siguientes:

- ❑ **random**: módulo que permite realizar elecciones al azar.
- ❑ **math**: módulo que añade funciones matemáticas.
- ❑ **statistics**: módulo que permite utilizar funciones estadísticas básicas.
- ❑ **datetime**: módulo que permite manejar fechas y tiempos de forma sencilla.
- ❑ **os**: módulo para interactuar con el sistema operativo.
- ❑ **shutil**: módulo para administrar ficheros y grupos de ficheros.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Para incluir módulos en tus programas tienes que realizarlo con la sentencia **import** de la siguiente forma al principio de tus programas:

**import** *NombreLibrería*

Una vez importada la librería podrás utilizar en tus programas todas las clases, constantes y funciones que incluyan.

Los módulos pueden importar otros módulos. Es costumbre pero no obligatorio ubicar todas las declaraciones import al principio del módulo.



# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulo random.

El módulo **random** va a permitirte hacer elecciones de forma aleatoria, tanto elegir al azar un valor de una lista como generar números aleatorios u obtener un número al azar dentro de un rango.

Algunas de las funciones que veremos son las siguientes:

**randrange** Función que devuelve un número aleatorio en un rango especificado por parámetro.

```
print("Número aleatorio(1-100): ", random.randrange(100))
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulo random.

**sample** Función que devuelve una lista de números aleatorios dentro de un rango establecido por parámetro. La función tiene dos parámetros, el primero es el rango sobre el que se seleccionarán los números aleatorios y el segundo es el número de elementos que tendrá la lista que se devuelve.

```
print("Lista aleatoria de seis números(1-50): “  
      ,random.sample(range(50), 6))
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

Python - Módulo random.

**random** Función que devuelve un número en coma flotante.

```
print("Número aleatorio en coma flotante: ",random.random())
```

**choice** Función que selecciona un elemento de la lista que recibe por parámetro al azar.

```
print("Elección aleatoria [Rojo,Verde,Amarillo,Azul,Rosa]: ",  
      random.choice(['Rojo', 'Verde', 'Amarillo', 'Azul', 'Rosa']))
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulo `math`.

El módulo **`math`** te permitirá incluir en tus programas funciones y constantes matemáticas para facilitarte aquellos desarrollos que realices que requieran operaciones matemáticas complejas.

Veremos aquellas funciones y constantes más importantes y relevantes, para ver todas las funciones y constantes incluidas te recomendamos que las consultes en la documentación oficial incluida en la página web de Python.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulo math.

Algunas de las funciones que veremos son las siguientes:

**fabs** Función que retorna el valor absoluto del número especificado como parámetro.

```
print("El valor absoluto de -7 es: ", math.fabs(-7))
```

**factorial** Función que calcula la factorial del número especificado como parámetro.

```
print("El factorial de 9 es: ", math.factorial(9))
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulo math.

**gcd** Función que calcula el máximo común divisor de dos números especificados como parámetros.

```
print("El máximo común divisor de 39 y 26 es: ", math.gcd(39,26))
```

**isnan** Función que comprueba si el parámetro no es un número. La función retornará False en caso de que sea un número y True en caso contrario.

```
print("isnan(8): ",math.isnan(8))
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

Python - Módulo math.

**log** Función que calcula el logaritmo en base X del número especificado como parámetro. La función tiene dos parámetros, el primero es el número del que se quiere calcular el logaritmo y el segundo la base del logaritmo.

```
print("El logaritmo en base 10 de 540 es: ", math.log(540,10))
```

**pow** Función que calcula la potencia del número especificado como parámetro. La función tiene dos parámetros, el primero de ellos es la base y el segundo el exponente.

```
print("El valor de 2 elevado a 10 es: ", math.pow(2,10))
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

Python - Módulo `math`.

**sqrt** Función que calcula la raíz cuadrada del número especificado como parámetro.

```
print("La raíz cuadrada de 144 es: ", math.sqrt(144))
```

**degrees** Función que retorna el valor en **grados** del ángulo en radianes recibido como parámetro.

```
print("El ángulo 1.57 en grados es: ", math.degrees(1.57))
```



# INTRODUCCIÓN A LA PROGRAMACIÓN

Python - Módulo `math`.

**radians** Función que retorna el valor en **radianes** del ángulo en grados recibido como parámetro.

```
print("El ángulo 90 en radianes es: ", math.radians(90))
```

**sin** Función que calcula el seno en radianes del ángulo especificado como parámetro.

```
print("El seno de un ángulo de 180 es: ", math.sin(math.radians(180)))
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

Python - Módulo math.

**cos** Función que calcula el coseno en radianes del ángulo especificado como parámetro.

```
print("El coseno de un ángulo de 180 es: ",  
      math.cos(math.radians(180)))
```

**tan** Función que calcula la tangente en radianes del ángulo especificado como parámetro.

```
print("La tangente de un ángulo de 180 es: ",  
      math.tan(math.radians(180)))
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

Python - Módulo `math`.

**asin** Función que calcula el arcoseno en radianes del ángulo especificado como parámetro.

```
print("El arcoseno de 1 es: ", math.degrees(math.asin(1)))
```

**acos** Función que calcula el arcocoseno en radianes del ángulo especificado como parámetro.

```
print("El arcocoseno de 1 es: ", math.degrees(math.acos(1)))
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

Python - Módulo math.

**atan** Función que calcula la arcotangente en radianes del ángulo especificado como parámetro.

```
print("La arcotangente de 1 es: ", math.degrees(math.atan(1)))
```

**pi**: tiene el valor de la constante  $\pi$ .

```
print("El valor de pi es: ", math.pi)
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

Python - Módulo math.

**e:** tiene el valor de la constante  $e$ .

```
print("El valor de e es: ", math.e)
```

**tau:** tiene el valor de la constante  $\tau$ .

```
print("El valor de tau es: ", math.tau)
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulo `statistics`.

El módulo **`statistics`** te va a permitir realizar cálculos estadísticos sobre conjuntos de datos.

Las funciones más importantes del módulo son las siguientes:

**`mean`** Función que calcula la media de un listado de números pasados como parámetro.

**`median`** Función que calcula la mediana de un listado de números pasados como parámetro.

# INTRODUCCIÓN A LA PROGRAMACIÓN

Python - Módulo statistics.

**median\_low** Función que calcula la mediana inferior de un listado de números pasados como parámetro.

**median\_high** Función que calcula la mediana superior de un listado de números pasados como parámetro.

**mode** Función que calcula la moda de un listado de números pasados como parámetro.

**variance** Función que calcula la varianza de un listado de números pasados como parámetro.

# INTRODUCCIÓN A LA PROGRAMACIÓN

Python - Módulo statistics.

```
import statistics
import random
valores = random.sample(range(10), 8)
print("Número aleatorios generados: ", valores)
print("Media: ", statistics.mean(valores))
print("Mediana: ", statistics.median(valores))
print("Mediana inferior: ", statistics.median_low(valores))
print("Mediana superior: ", statistics.median_high(valores))
print("Moda: ", statistics.mode(valores))
print("Varianza: ", statistics.variance(valores))
```



# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulo datetime.

El módulo **datetime** va a permitirte manipular fechas de forma sencilla.

Las funciones que vamos a ver son las siguientes:

**datetime.now** Función que retorna la fecha y la hora de ahora mismo.

```
print("Ahora mismo es: ", datetime.datetime.now())
```

**date.today** Función que retorna el día de hoy.

```
print("Hoy es: ", datetime.date.today())
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulo datetime.

**date** Función que permite crear un objeto de tipo fecha pasándole como parámetros el año, el mes y el día.

```
fecha = datetime.date(2017,11,29)  
print(fecha)
```

**datetime** Función que permite crear un objeto de tipo Datetime pasándole como parámetros el año, el mes, el día, la hora, los minutos, los segundos y los microsegundos.

```
fechahora = datetime.datetime(2016,2,21,14,00,00,000)  
print(fechahora)
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulo datetime.

```
import datetime
fecha = datetime.datetime(2016,2,21,14,00,00,000)
print(fecha)
print("Año: ",fecha.year)
print("Mes: ",fecha.month)
print("Día: ",fecha.day)
print("Hora: ",fecha.hour)
print("Minutos: ",fecha.minute)
print("Segundos: ",fecha.second)
print("Microsegundos: ",fecha.microsecond)
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulo datetime.

# Calcula la diferencia entre dos datetime.

```
import datetime
```

```
fin = datetime.datetime.now()
```

```
inicio = datetime.datetime(2016,2,21,14,00,00,000)
```

```
print("Resta de fechas:")
```

```
print("1.- ", fin)
```

```
print("2.- ", inicio)
```

```
resultado = fin - inicio
```

```
print("Resultado: ",resultado)
```

# La forma de acceso a los elementos y el cálculo de la diferencia entre dos objetos se realizan de la misma forma con objetos de tipo date, pero teniendo en cuenta que únicamente están compuestos por año, mes y día.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulo os.

El módulo **os** nos va a permitir interactuar con el sistema operativo y manejar todas aquellas funcionalidades que ofrece.

Vamos a aprender a utilizar las siguientes funciones del módulo os:

**getcwd** Función que devuelve el directorio actual de trabajo de la aplicación.

```
print("Directorio de trabajo actual: ",os.getcwd())
```

**chdir** Función que cambia el directorio de trabajo de la aplicación al pasado por parámetro.

```
os.chdir("/Users/alfre/Downloads/")  
print("Nuevo directorio de trabajo: ",os.getcwd())
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

Python - Módulo os.

**getpid** Función que devuelve el identificador del proceso del aplicativo.

```
print("ID proceso: ", os.getpid())
```

**getuid** Función que devuelve el identificador del usuario del proceso del aplicativo.

```
print("ID usuario: ", os.getuid())
```

**listdir** Función que lista el contenido del directorio de trabajo actual.

```
print("El contenido del directorio actual es: ", os.listdir("./"))
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

Python - Módulo os.

**mkdir** Función que crea un nuevo directorio dentro del directorio de trabajo actual.

```
os.mkdir("NuevoDirectorio")
```

**rename** Función que renombra un fichero.

```
os.rename("ejemplo.txt","nuevonombre.txt")
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulo shutil.

El módulo **shutil** nos va a permitir administrar ficheros de forma sencilla.

**copy** Función que realizar la copia del fichero parametrizado en uno nuevo con el nombre especificado por parámetro.

```
shutil.copy("ejemplo.txt","nuevoejemplo.txt")
```

**move** Función que mueve el fichero especificado por parámetro a la ruta especificada por parámetro.

```
shutil.move("nuevonombre.txt","NuevoDirectorio")
```



# INTRODUCCIÓN A LA PROGRAMACIÓN

Python - Módulo shutil.

**rmtree** Función que elimina el directorio especificado como parámetro y todo su contenido.

```
shutil.rmtree("NuevoDirectorio")
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Podemos recurrir a innumerables paquetes y módulos tanto de la librería estándar de python como de paquetes desarrollados por terceros miembros de diferentes comunidades orientadas a la programación. Así utilizar módulos de terceros nos evita el tener que “reprogramar lo ya programado”.

Para importar un módulo o paquete de terceros primero debemos ubicarlo, conocer qué es lo que hace y en que nos va a servir. Donde habitualmente puedes encontrar muchos paquetes y módulos es en **PyPi.org** además de su documentación donde podrás comprender mejor la implementación del mismo.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Luego de encontrar el módulo o paquete que necesitas para instalarlo debes usar “pip”.

**pip install** *nombre\_del\_modulo/paquete*

lo que procederá a instalar el modulo o paquete y luego bastará con importarlo.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - PIP Installs Packages.

PIP es el acrónimo de «**PIP Installs Packages**». Es una herramienta de línea de comandos que te permitirá instalar, reinstalar y desinstalar módulos o paquetes de Python, además de resolver automáticamente las dependencias de cada uno de estos módulos o paquetes. El comando que se utiliza para ello es `pip`. Se instalará con Python por defecto si utilizas la versión Python 3.4 o superior

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - PIP Installs Packages.

Para instalar un módulo/paquete de Python, utiliza el siguiente comando:

```
pip install nombre-del-paquete
```

Para instalar una versión específica de un módulo/paquete de Python con PIP, agrega el siguiente modificador con la versión del paquete:

```
pip install nombre-de-paquete==1.0.0
```

Para ver los detalles de cualquier módulo/paquete que tengas instalado:

```
pip show nombre-de-paquete
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - PIP Installs Packages.

Para buscar un módulo/paquete determinado en el repositorio:

```
pip search "consulta"
```

Para ver una lista con todos los módulos/paquetes instalados:

```
pip list
```

Para ver una lista de aquellos módulos/paquetes que no están actualizados a su última versión, agrega el modificador `--outdated`:

```
pip list --outdated
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - PIP Installs Packages.

Para actualizar un módulo/paquete y eliminar las versiones anteriores:

```
pip install nombre-de-paquete --upgrade
```

Para reinstalar completamente un módulo/paquete:

```
pip install nombre-de-paquete --upgrade --force-reinstall
```

Para eliminar completamente un módulo/paquete, usa este comando:

```
pip uninstall nombre-paquete
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Para importar un módulo, se utiliza la instrucción **import**, seguida del **nombre del paquete** más el **nombre del módulo** (sin el .py) que se desee importar.

**import** modulo      # importar un módulo que no pertenece a un paquete

**import** paquete.modulo1 # importar un módulo que está dentro de un paquete

**import** paquete.subpaquete.modulo1

La instrucción **import** seguida de **nombre\_del\_paquete.nombre\_del\_modulo**, nos permitirá hacer uso de todo el código que dicho módulo contenga.



# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Para **acceder** (desde el módulo donde se realizó la importación), a **cualquier componente del módulo importado**, se realiza mediante el **namespace**, seguido de un **punto** (.) y el nombre del **elemento** que se desee obtener. En Python, un **namespace**, es el nombre que se ha indicado luego de la palabra import, es decir **la ruta del módulo**:

modulo.elemento

paquete.modulo1.elemento

paquete.subpaquete.modulo1.elemento

```
import math
```

```
X = math.pi
```

```
print("El valor de  $\pi$  es" , X)
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Es posible también, abreviar los **namespaces** mediante un **alias**. Para ello, durante la importación, se asigna la palabra clave **as** seguida del alias con el cuál nos referiremos en el futuro a ese namespace importado:

```
import modulo as m
```

```
import paquete.modulo1 as pm
```

```
import paquete.subpaquete.modulo1 as psm
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Luego, para **acceder** a cualquier elemento de los módulos importados, el namespace utilizado será el **alias** indicado durante la importación:

m. elemento

pm. elemento

psm. elemento

```
import math as mates  
X = mates.pi  
print("El valor de  $\pi$  es" , X)
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

En Python, es posible también, importar de un módulo **solo los elementos que se desee utilizar**. Para ello se utiliza la instrucción **from** seguida del namespace, más la instrucción **import** seguida del elemento que se desee importar:

**from** paquete.modulo1 **import** elemento

En este caso, se accederá directamente al elemento, sin recurrir a su namespace:

```
from math import pi
X = pi
print("El valor de  $\pi$  es" , X)
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Es posible también, importar **más de un elemento** en la misma instrucción. Para ello, cada elemento irá separado por una **coma (,)** y un **espacio en blanco**:

```
from paquete.modulo1 import elemento _1, elemento _2
```

```
from math import pi, tau
X = pi
Y = tau
print("El valor de  $\pi$  es" , X)
Print("el valor de la constante  $\tau$  es", Y)
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Pero ¿qué sucede si los elementos importados desde módulos diferentes tienen los mismos nombres? En estos casos, habrá que prevenir fallos, utilizando alias para los elementos:

```
from paquete.modulo1 import elemento_1 as C1
```

```
from paquete.subpaquete.modulo1 import elemento_1 as CS1
```

```
from math import pi as constante  
X = constante  
print("El valor de  $\pi$  es" , X)
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

De forma alternativa, también es posible importar **todos** los elementos de un módulo, sin utilizar su **namespace** pero tampoco alias:

```
from paquete.modulo1 import *
```

Es decir, que todos los elementos importados se **accederá** usando su nombre sin anteponer nada:

```
from math import *  
X = pi  
print("El valor de  $\pi$  es" , X)
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Con esta sintaxis importaremos todas las definiciones del módulo excepto aquellas cuyo nombre comience por un guión bajo (\_).

Las definiciones con nombres que comienzan por \_ son consideradas privadas o internas al módulo, lo que significa que no están concebidas para ser usadas por los usuarios del módulo y que, por tanto, no forman parte de su interfaz (no deben usarse fuera del módulo).

En general, los programadores no suelen usar esta funcionalidad ya que puede introducir todo un conjunto de definiciones desconocidas dentro del módulo importador, lo que incluso puede provocar que se sobrescriban definiciones ya existentes.



# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Existen dos funciones bastante útiles para explorar la información de un módulo, las funciones **dir()** y **help()**.

La función **dir()** permite ver cuáles son los componentes de un módulo.

La función **help()** permite leer información más detallada del módulo o de alguno de sus componentes.

**dir(módulo)**

**help(módulo.elemento)**

```
import math  
dir(math)  
help(math)
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

**Crear** un módulo en Python es muy simple. Un módulo no es más que un fichero que contiene instrucciones y definiciones (variables, funciones, clases, ...). El fichero debe tener extensión **.py** y su **nombre** se convierte en el **nombre del módulo**.

Dado que los nombres de archivo de los módulos se convierten en nombres de variables en Python cuando se importan, no se permite nombrar módulos con palabras reservadas de Python.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

En el siguiente ejemplo se crea el módulo **moduloseries.py** y un programa **programa.py** lo importa y hace uso de la función `sumadesde1aN` (que para un número  $n$ , suma todos los números que van desde 1 a  $n$ ).

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Vamos a crear el módulo **moduloseries.py**:

**moduloseries.py:**

**def sumadesde1aN(pnumero):**

''' Suma todos los números consecutivos desde 1 hasta el número expresado. Si el valor es menor que 1 devuelve 0 '''

ntotal= 0

if pnumero >0:

    for nnum in range(1,pnumero+1):

        ntotal = ntotal + nnum

return ntotal

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Vamos a crear un programa **programa.py** que importa el modulo **moduloseries.py**:

**programa.py:**

```
import moduloseries
```

```
valor = 10
```

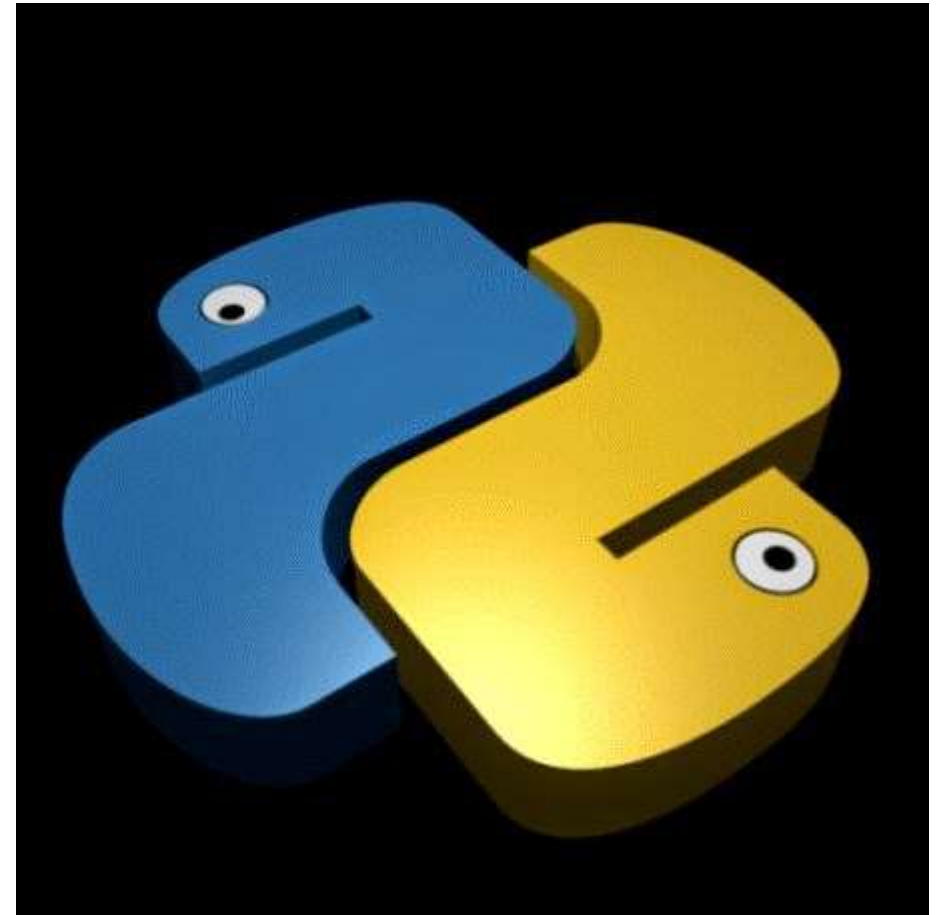
```
suma = moduloseries.sumadesde1aN(valor)
```

```
print('Suma desde 1 a '+' + str(valor) + ':' + str(suma))
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Un archivo de texto básico que contiene código Python que está destinado a ser ejecutado directamente por el cliente generalmente se denomina script.



# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

La forma correcta de crear scripts en Python es:

- Definir una función main que será ejecutada por el script.
- Añadir todas las funciones necesarias que serán llamadas por main.
- Al final del módulo se añade una llamada a main con los parámetros necesarios y se realiza la ejecución.

Para una correcta ejecución se suele emplear:

```
if __name__ == '__main__':
```

y añadir la lógica del script o la llamada a main justo después.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

**multiplier.py:**

```
def multiply(a, b):  
    return float(a) * float(b)
```

**def main():**

```
    a = input('Inserte un número: ')  
    b = input('Inserte otro número: ')  
    res = multiply(a, b)  
    print(f'La multiplicación es {res}')
```

```
if __name__ == '__main__':  
    main()
```



# INTRODUCCIÓN A LA PROGRAMACIÓN


## Python - Módulos y Paquetes.

Para poder lanzar scripts en Python se suele hacer desde una consola de comandos siguiendo los siguientes pasos:

- Ubicar la consola en el mismo directorio que el módulo a ejecutar.  
Para usuarios de MacOS X, Linux o Windows se suele usar `cd` .
- Lanzar el intérprete de Python usando como primer parámetro el nombre del fichero.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

 Símbolo del sistema

```
C:\>python elpythonista/python_scripts/multiplier.py
Inserte un número: 5.89
Inserte otro número: 2.34
La multiplicación es 13.782599999999999

C:\>cd elpythonista/python_scripts/

C:\elpythonista\python_scripts>python multiplier.py
Inserte un número: 4.5
Inserte otro número: 8
La multiplicación es 36.0

C:\elpythonista\python_scripts>
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

Python - Módulos y Paquetes.

```
tuxskar@o-mbp: ~/PycharmProjects/elpythonista/python_scripts 1
→ ~ cd PycharmProjects/elpythonista/python_scripts
→ python_scripts ls
multiplier.py
→ python_scripts python multiplier.py
Inserte un número: 5
Inserte otro número: 8.9
La multiplicación es 44.5
→ python_scripts █
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Utilizando la librería psutil se puede conocer el % de uso de cada CPU del sistema, la memoria usada y la memoria disponible en el mismo.

### **cpu\_mem\_stats.py:**

```
import psutil
```

```
if __name__ == '__main__':
```

```
    perc_cpu = psutil.cpu_percent(interval=1, percpu=True)
```

```
    mem_virt = int(psutil.virtual_memory().used / (1024 ** 2))
```

```
    avail_mem = int(psutil.virtual_memory().available * 100 /
```

```
psutil.virtual_memory().total)
```

```
    print(f"Estado actual del PC: La CPU está al {perc_cpu}%
```

```
        Usando {mem_virt} Mb de memoria
```

```
        Quedando {avail_mem}% memoria libre")
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Cuando creamos nuestros propios módulos podemos importarlos para utilizar sus componentes. También podemos usarlos como **scripts**.

Un script es un archivo con una secuencia de instrucciones que se ejecutan cada vez que el script se llama.

Para esto, es necesario haber incluido **al final del módulo** la sentencia:

```
if __name__ == '__main__':
```

y el **código** que **ejecutar después de la misma**.

Para ejecutar nuestro modulo como script podemos hacerlo desde la terminal de Python.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Podemos usar la variable `__name__` para crear comportamiento diferente en nuestros programas a partir de la forma en que se está ejecutando.

Una de las razones para hacerlo es que, a veces, se escribe un módulo (un archivo `.py`) que **se puede ejecutar directamente**, pero que alternativamente, también **se puede importar** y reutilizar sus funciones, clases, métodos, etc en otro módulo.

Con esto conseguimos que la **ejecución** sea **diferente** al **ejecutar** el módulo directamente que al **importarlo** desde otro programa.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Vamos a crear el módulo **demo** contenido en el archivo **demo.py**:

En este archivo, definiremos dos funciones `cuadrado` y `cubo`. Tenemos dos opciones: ejecutar ese módulo como un script o importar ese módulo desde otro, para utilizar sus funciones.

### **demo.py:**

```
def cuadrado(n):
```

```
    return(n**2)
```

```
def cubo(n):
```

```
    return(n**3)
```

```
if __name__ == "__main__":
```

```
    a = input('Inserte un número: ')
```

```
    b = input('Inserte otro número: ')
```

```
    print(cuadrado(a))
```

```
    print(cubo(b))
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Cuando incluimos la declaración **import** para la carga del módulo correspondiente no se le dice a Python donde buscar.

Por defecto Python buscará primero en el directorio donde tengamos el programa principal ó donde resida el intérprete Python. Si no lo encuentra ahí, seguidamente buscará en la variable de entorno PYTHONPATH. Si tampoco tiene éxito, buscará por las bibliotecas estándar que vienen en la distribución base de Python.



# INTRODUCCIÓN A LA PROGRAMACIÓN

## Python - Módulos y Paquetes.

Una manera de ver las rutas por las cuales Python intenta cargar los módulos mediante **import** es cargando el módulo de sistema **sys** (en el intérprete), y a continuación ejecutar **sys.path**. Esto nos devolverá una lista con todas las rutas a las que Python accede para buscar los módulos requeridos.

# INTRODUCCIÓN A LA PROGRAMACIÓN

Python - Módulos y Paquetes.

*“Fin del tema”*

