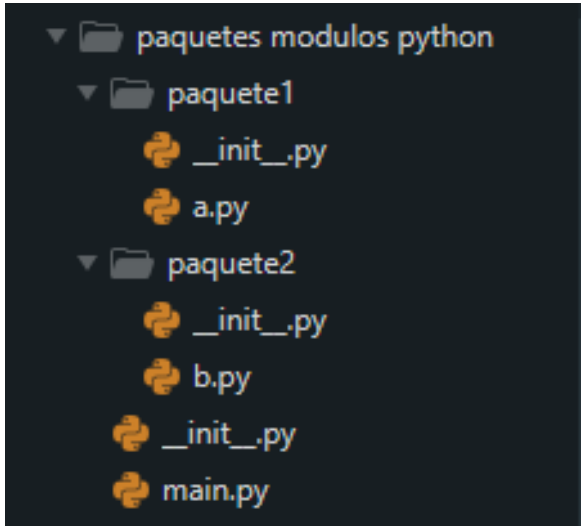
# Módulos y paquetes

Los módulos y paquetes son archivos de código fuente que contiene funciones, clases o variables preprogramadas. Los módulos y paquetes facilita la sistematización y la reutilización de código, permitiendo la creación de programas más grandes sin tener que escribir el código desde cero.




Ing. Antonio Torres Sarmiento

# Módulos

- Poner definiciones en un archivo y usarlos en un script o en una instancia del intérprete.
- Usar una función útil que se ha escrito en distintos programas sin copiar su definición en cada programa.
- Las definiciones de un módulo pueden ser importadas a otros módulos o al módulo principal.
- El nombre de archivo es el nombre del módulo con el sufijo **.py** agregado.
- Puede contener tanto declaraciones ejecutables como definiciones de funciones.
- Estás declaraciones se ejecutan únicamente la primera vez que el módulo se encuentra en una declaración **import**.

# Módulos

- La declaración **from** es una variante para importar los nombres de un módulo directamente al espacio de nombres del módulo que hace la importación.
- Con **from** no se introduce en el espacio de nombres local el nombre del módulo desde el cual se está importando.
- Los módulos pueden importar otros módulos.
- Es costumbre pero no obligatorio ubicar todas las declaraciones **import** al principio del módulo.
- Los nombres de los módulos importados, si se colocan en el nivel superior de un módulo (fuera de cualquier función o clase), se añaden al espacio de nombres global del módulo.
- Cada módulo tiene su propio espacio de nombres privado, que es utilizado como espacio de nombres global por todas las funciones definidas en el módulo.

# Módulos

- Se pueden utilizar las variables globales de un módulo con la misma notación que se utiliza para referirse a sus funciones.
- Con from nombre_modulo import * se importan todos los nombres que un módulo define (exceptuando a las definiciones privadas).
- Es costumbre pero no obligatorio ubicar todas las declaraciones **import** al principio del módulo.
- Los nombres de los módulos importados, si se colocan en el nivel superior de un módulo (fuera de cualquier función o clase), se añaden al espacio de nombres global del módulo.
- Si el nombre del módulo es seguido por **as**, el nombre siguiendo **as** queda ligado directamente al módulo importado. También se puede utilizar cuando se utiliza **from** con efectos similares.

# Ejecutar módulos como scripts

- Al ejecutar **python nombre_modulo.py <arguments>** el código en el módulo será ejecutado, tal como si se hubiese importado.

- **__name__** tendrá el valor de ”__main__”. (__name__ es una variable especial de Python que contiene el nombre del módulo Python como String. Es útil para determinar cómo se ha ejecutado un módulo. Si el módulo ha sido ejecutado directamente, __name__ se establecerá en “__main__”. Si el módulo ha sido importado por otro módulo, __name__ se establecerá en el nombre de archivo del módulo sin la extensión .py).

- Para utilizar el archivo como **script**, se debe agregar se siguiente código al final del módulo:

```
If __name__ == “__main__”
        import sys
        [bloque de código]
```

Solo se ejecutará si el módulo es ejecutado como archivo principal. Si el módulo se importa, este código no se ejecutará.

# Algunos módulos estándar

- **os**: provee funciones para interactuar con el sistema operativo.
- **math**: proporciona acceso a las funciones matemáticas definidas en el estándar de C.
- **random**: provee herramientas para realizar selecciones al azar.
- **statistics**: calcula propiedades de estadística básica (la media, mediana, varianza, etc) de datos numéricos.
- **datetime**: ofrece clases para gestionar fechas y tiempos.
- **etc**.

# Paquetes

- Es una forma de estructurar el espacio de módulos en Python.
- Colección de módulos.
- Permite un sistema jerárquico de archivos.
- Los archivos **__init__.py** son necesarios para que Python trate los directorios que contienen el archivo como paquetes. Puede ser simplemente un archivo vacío, pero también puede ejecutar el código de inicialización del paquete.
- Al usar **from package import item**, el ítem puede ser tanto un submódulo (o subpaquete) del paquete, o algún otro nombre definido en el paquete, como una función, clase, o variable.
- En la sintaxis como **import item.subitem.subsubitem**, cada ítem excepto el último debe ser un paquete; el mismo puede ser un módulo o un paquete pero no puede ser una clase, función o variable definida en el ítem previo.

# Importar * desde un paquete

- Importa todos los submódulos que están presente en el paquete.

- Si el código del **__init__.py** de un paquete define una lista llamada **__all__**, se toma como la lista de los nombres de módulos que deberían ser importados cuando se hace **from package import *.**

- Si no se define **__all__**, no se importan todos los submódulos del paquete al espacio de nombres actual; sólo se asegura que se haya importado el paquete (posiblemente ejecutando algún código de inicialización que haya en __init__.py) y luego importa aquellos nombres que estén definidos en el paquete. Esto incluye cualquier nombre definido (y submódulos explícitamente cargados) por **__init__.py**.

Nota:
1) Usar **import *** puede tardar mucho y el importar sub-módulos puede tener efectos secundarios no deseados que sólo deberían ocurrir cuando se importe explícitamente el sub-módulo.
2) Se considera una mala práctica en código de producción.

Ing. Antonio Torres Sarmiento

# Referencias internas en paquetes

- Se puede escribir import relativos con la forma **from module import name**.
- Los imports usan puntos adelante para indicar los paquetes actuales o paquetes padres involucrados en el import relativo.
- Los imports relativos se basan en el nombre del módulo actual.

Nota: Los módulos pensados para usarse como módulo principal de una aplicación Python siempre deberían usar **import** absolutos.