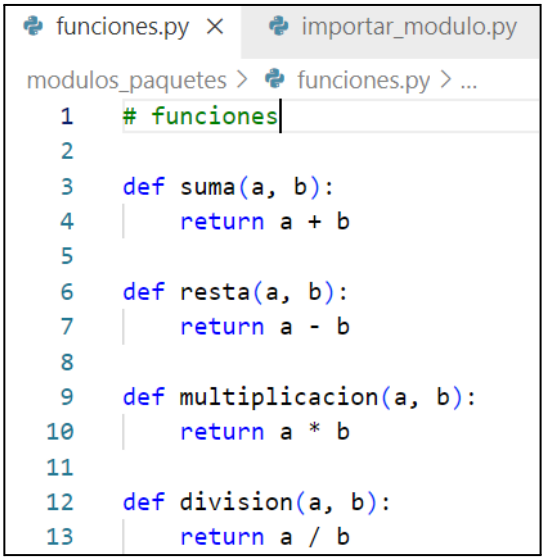
1. Módulos

- Un módulo en Python es un archivo con extensión ***.py*** que contiene código Python en forma de funciones, clases, variables, etc. y que puede ser utilizado (importado) en otros módulos.
- El uso de módulos facilita la organización de los proyectos y además permite la reutilización de código.
- Existen módulos estándar y módulos creados por el programador.
- Para utilizar dentro de un módulo, código desarrollado en otro, es necesario **importar el módulo que contiene el código.**

1.1. Formas de importar un módulo.

Veamos mediante ejemplos las distintas posibilidades que existen para importar módulos.
Partimos de un módulo llamado ***funciones.py***, que como se observa en la imagen de la derecha contiene la definición de varias funciones.



- **Opción 1. import nombre_modulo.** Para usar las funciones y variables definidas en este módulo debemos utilizar la forma: **nombre_modulo.nombre_función.**

```
1  import funciones
2
3  print(funciones.suma(5, 3))
4  print(funciones.resta(5, 3))
5  print(funciones.multiplicacion(5, 3))
```

- **Opción 2. import nombre_modulo as alias.** En este caso utilizamos un alias para el módulo, acortando su nombre. Si se importan varios módulos con el mismo alias, prevalece el último.

```
1   import funciones as f
2
3   print(f.suma(5, 3))
4   print(f.resta(5, 3))
5   print(f.multiplicacion(5, 3))
```

- **Opción 3. from módulo import función/funciones/*.** Importar una función de un módulo, varias separadas por comas, o bien usar asterisco "*" para importarlas todas. Debe tenerse en cuenta que si un módulo contiene gran cantidad de funciones y se importan todas, incluso las que no se van a usar, estaremos consumiendo memoria de forma innecesaria.

```
1   from funciones import suma, resta
2
3   print(suma(5, 3))
4   print(resta(5, 3))
```

1.2. Ejecutar un módulo como un script.

- Un módulo puede ser ejecutado como un script o como punto de entrada de un programa cuando se pasa directamente como parámetro al intérprete de Python. En este caso el módulo debe llevar **al final del mismo** la sentencia **if __name__=='__main__':**, seguida de alguna llamada a una función, pass, etc. De esta forma podemos probar las funciones del módulo desde el mismo módulo.

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

if __name__ == '__main__':
    print(suma(2, 3))
```

2. Paquetes.

- **Un paquete es un directorio que contiene otros paquetes y/o módulos. El código de estos paquetes suele estar relacionado entre sí.**
- La distribución del código de una aplicación en paquetes mejora la organización y reutilización.
- Para que un directorio sea considerado un paquete debe tener un archivo vacío con el nombre ***__init__.py.***

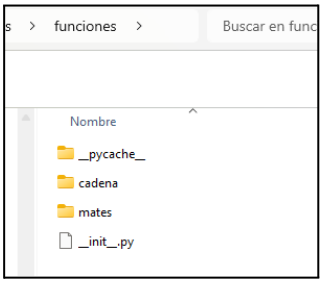
- Los paquetes pueden contener subpaquetes, o paquetes dentro de paquetes. Esto permite dar un paso más en la organización de archivos del proyecto. Los subpaquetes también deben contener el archivo ***__init__.py.***
- Para importar un módulo contenido en un paquete debe usarse la forma:
    - from paquete import modulo
    - from paquete.modulo import funciones/*
    - from paquete.subpaquete.modulo import funciones/*

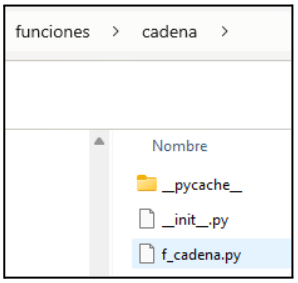
3. Paquetes redistribuibles.

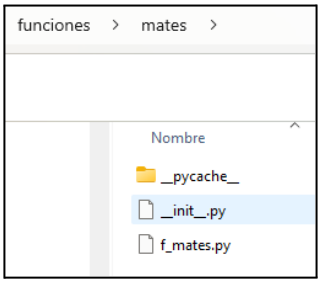
- **Todo lo visto hasta ahora relacionado con módulos y paquetes generaría una excepción (ModuleNotFoundError) si los directorios o archivos que contienen las funciones/clases no están en la misma carpeta que el archivo desde el que se importan**.
- Para solucionar este problema y no tener que copiar/pegar los directorios con módulos y paquetes en cada proyecto, pueden crearse **paquetes redistribuibles.**
- Su función es la de reutilizar el código contenido dentro de los módulos en diferentes aplicaciones.
- Para ello se debe transformar el código ya desarrollado en un paquete redistribuible que deberá ser instalado en el directorio virtual de la aplicación donde va a ser usado, mediante “pip” o cualquier otro gestor de paquetes.

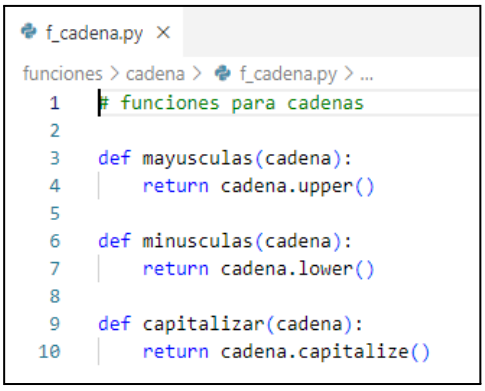
3.1. Pasos para crear un paquete redistribuible. Ejemplo.

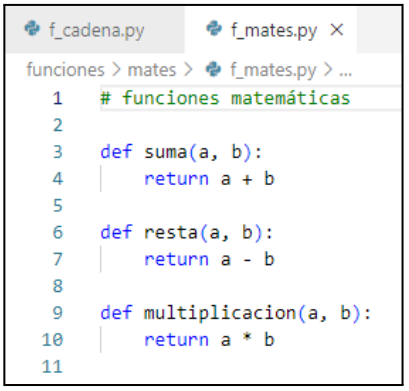
- Crear la estructura de paquetes/módulos/código.



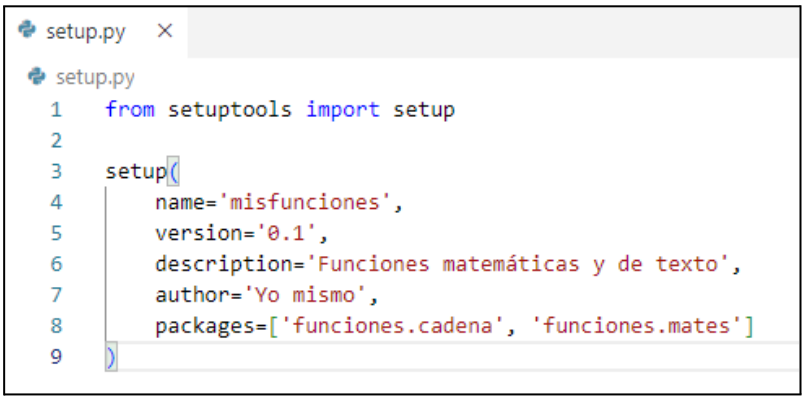








- Crear el archivo ***setup.py*** en la misma carpeta que contiene la estructura de paquetes mencionada. Este archivo contiene la descripción del redistribuible que se creará. Estudia la imagen siguiente.



  - Línea 1. Importar la función "setup". Si el IDE nos informa de que no la encuentra deberá instalarse en el directorio virtual de trabajo el paquete "setuptools" mediante ***pip install setuptools***.
  - Línea 3 y siguientes. Descripción del paquete a crear mediante la llamada a la función "setup". El parámetro más importante es "packages" donde indicamos los paquetes que contendrá nuestro redistribuible.
- **Desde un terminal,** crear el redistribuible mediante ***python setup.py sdist***. Esta orden debe generar una carpeta llamada "dist" que contendrá el archivo ***misfunciones-0.1.tar.gz***. El parámetro "name" que se utilizó en la llamada a la función "setup" contiene el nombre del redistribuible.
- *Instalar en el directorio virtual de trabajo este paquete redistribuible, mediante* ***pip install misfunciones-0.1.tar.gz.***
- Importar las funciones en nuestro archivos Python:
  - from funciones.mates.f_mates import *
  - from funciones.cadena.f_cadena import *
  - …
- El archivo "XXX.tar.gz" podrá ser compartido como se estime, e instalado en otros equipos y/o directorios virtuales.

3.2. Variable sys.path.

Otra forma de importar módulos que están en ubicaciones distintas a la del proyecto que se está creando, es la modificación del valor de la variable ***sys.path***. Esta variable guarda las rutas donde Python buscará los módulos, por tanto si añadimos nuevas rutas a esta variable, podrán usarse los módulos contenidos en ellas. Estudia el ejemplo contenido en la imagen.

```
1 import sys
2 sys.path.append('C:/Users/jara2/Documents/python_apps/Curso2425/POO/Unidad_01/nueva_ruta')
3 from masfunciones import *
4
5 print(alreves('Hola mundo'))
```

4. Espacios de nombre.

- Es una colección aislada de nombres que hacen referencia a objetos como variables, funciones o clases.
- Podemos verlo como un diccionario en el que las claves son los nombres de los objetos y los valores, los objetos propiamente dichos.
- Existen los siguientes espacios de nombres:
    - **Built-in namespace** (espacio de nombres incorporado). Contiene los nombres de todos los objetos incorporados por Python. Aquí están los nombres print, super, True, int…
    - **Global namespace** (espacio de nombres global). Contiene los nombres definidos a nivel de programa principal o de módulo.
    - **Local namespace** (espacio de nombres local). Contiene los nombres definidos dentro de una función o clase.
    - **Enclosing namespace** (espacio de nombres encerrado). Es un espacio de nombres local encerrado en otro espacio local. Se da cuando la definición de una función está incluida dentro de otra.
- **Ámbito de una variable (Scope).**
    - El ámbito de una variable viene determinado por el lugar donde ha sido definida.
    - **Cuando dentro de una función se hace referencia a un nombre, este se busca en primer lugar en el espacio de nombres local, luego en el global y por último en el incorporado.**
    - Se pueden definir variables globales dentro de una función mediante **“global variable = valor”**.

5. Enlaces de interés.

- https://j2logo.com/python/tutorial/espacios-de-nombres-modulos-y-paquetes/.
- https://ellibrodepython.com/modulos-python.
- https://realpython.com/python-namespaces-scope/.