4	**Clase: Librerías y paquetes**
4.1	Presentación
	- La librería estándar
	- Navegar la documentación del lenguaje
	- Paquetes. Uso de pip
4.2	Práctica
	- Buscar y aplicar funciones integradas de la LE
	- Importar paquetes, descargar paquetes con pip
	- Crear módulos propios

Módulos

En Python, un módulo es un archivo .py que contiene definiciones de funciones, variables, constantes y otras herramientas, que pueden ser llamadas por otro archivo .py

Cuando programamos en Python, nuestro código tiene dos partes: La "parte declarativa" donde definimos funciones y otros elementos, y el "cuerpo principal" (main) donde llamamos funciones y ejecutamos instrucciones.

```
#mi_modulo.py
################################################################################
                                #Parte declarativa#
################################################################################
def fib(n):    # los primeros n numeros de la serie
   a, b = 0, 1
   for numero in range(n):
      print(b, end=' ')
      a, b = b, a+b
   print()


################################################################################
                                #Cuerpo principal (main)#
################################################################################


print("Bienvenido al módulo!")
```

Cualquier archivo .py que escribamos puede funcionar como módulo. Para llamar un módulo desde otro utilizamos la palabra reservada import. Una vez importado el módulo, podemos acceder a sus elementos usando un punto (.) y el nombre de la función/variable que queramos llamar. Por ejemplo:

```
#principal.py
import mi_modulo


mi_modulo.fib(4)
```

Y la salida será:

```
Bienvenido al módulo!
1, 1, 2, 3
```

Tenemos que tener presente que, al importar un módulo, el código que esté en su cuerpo principal -si existe- será ejecutado. Por eso, a menos que sea necesario ejecutar código al momento de importar -por ejemplo, para crear archivos necesarios o inicializar variables- una buena práctica es que nuestros módulos solo contengan definiciones y solamente ejecutemos códigos desde un archivo principal.

Otra consideración es que el import del ejemplo funcionará siempre y cuando *mi_modulo.py* esté en la misma carpeta que el archivo *principal.py*. Este comportamiento se puede cambiar de dos formas: modificando la variable nativa de Python sys.path; o estructurando nuestro código mediante paquetes. Un **paquete** es una colección de módulos que se organizan de una determinada manera dentro de una carpeta (pueden estar todos los módulos dentro de la carpeta principal o dentro de subcarpetas, creando una jerarquía de módulos). Por ejemplo, un paquete puede tener la siguiente estructura:

paquete/

- paquetito1/
  - modulo1.py
  - modulo2.py
- paquetito2/
  - modulo1.py
  - modulo2.py

Para llamar módulos contenidos en una estructura como esta, separamos los nombres de los paquetes con un punto:

```
import paquete.paquetito1.modulo2

paquete.paquetito1.modulo2.funcion1()
```

Como vemos, para referenciar el módulo que acabamos de importar tenemos que usar su nombre completo. Como esto puede volverse tedioso, otra forma de importar módulos dentro de paquetes es:

```
from paquete.paquetito1 import modulo2

modulo2.funcion1()
```

Puede leer más sobre esto en la documentación oficial del lenguaje traducida al español: http://docs.python.org.ar/tutorial/3/modules.html

La Librería Estándar

Una de las grandes ventajas de programar en forma modular -la escritura de programas como un conjunto de funciones independientes que uno puede llamar- es que hace que el código que escribimos sea reutilizable. Con esto en mente, los desarrolladores de Python incluyen junto a cada versión del lenguaje un conjunto de módulos para una gran variedad de tareas para que el programador haga uso de ellos sin necesidad de escribirlos él mismo. Este conjunto es lo que se conoce como la librería estándar.

Por ejemplo, supongamos que queremos simular la tirada de un dado, por lo que vamos a necesitar una función que nos dé un numero al azar entre 1 y 6. Para esto podemos usar el módulo random de la LE. Los paquetes de la librería estándar se encuentran en las carpetas contenidas en la variable sys.path, lo que significa que podemos importarlos desde cualquier programa sin importar en qué carpeta nos encontremos:

```
#dado.py

import random


def dado():

        return random.randint(1,6) #devuelve un numero al azar entre 1 y 6


print(dado())
```

Otros paquetes de la librería estándar incluyen:

- math #funciones matemáticas
- datetime #manejo de fechas y horas
- smtplib #envió y manejo de correo electrónico

y muchos(!!) más. Para consultar la lista completa, la mejor opción es la documentación oficial del lenguaje (https://docs.python.org/3/library/index.html) pero un buen paseo inicial por las funcionalidades de la LE es el tutorial de Python Argentina (http://docs.python.org.ar/tutorial/3/stdlib.html).

Paquetes fuera de la LE

Si estamos buscando una funcionalidad más específica que la que encontramos en la librería estándar, tenemos que descargar los paquetes que necesitemos y colocarlos en un lugar desde el cual podamos llamarlos sin problemas.

Para este fin, existe en internet un repositorio central de todos los paquetes creados por terceros y publicados para el resto de la comunidad conocido como el Python Package Index. Podemos buscar este repositorio en la web y consultar los paquetes por categoría para encontrar el deseado, o podemos usar herramientas que faciliten este proceso tales como **pip**.