Para el desarrollo de esta actividad se decidió implementar una aplicación interactiva a través de la consola. Donde, el usuario, deberá seguir las instrucciones que aparecen en pantalla para introducir las listas que se han de evaluar y así poder obtener los resultados esperados.

Adicionalmente, para el desarrollo de la Actividad #2, se hará uso de la librería `pickprimes` implementada durante la asignatura `Fundamentos de Python` del presente Máster y que se encuentra disponible en `PyPI` a tarvés del siguiente link: pickprimes

> De igual forma el módulo `pickprimes.py` será incluido en la entrega de la actividad.

```
[(base) marci@Marcials-MacBook-Pro:~/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4$ ]
python Asignacion.py
Bienvenido a la Asignación #4 de BPP

================================================================================
Actividad #1: encontrar el máximo elemento de cada lista dentro de una lista

Por favor indicar cuantas listas se encuentran dentro de la lista: 3

----------------------------------------------------------------
Nota: se debe separar los elelmentos de cada lista con espacio
----------------------------------------------------------------

Ingrese los elementos de la lista #1: 2 4 1
Ingrese los elementos de la lista #2: 1 2 3 4 5 6 7 8
Ingrese los elementos de la lista #3: 100 250 43

================================================================================
Actividad #2: encontrar los números primos en la lista suministrada

----------------------------------------------------------------
Nota: se debe separar los elementos de cada lista con espacio
----------------------------------------------------------------

ingrese los elementos de la lista: 3 5 5 13

================================================================================
RESUMEN:

Resultado de la Actividad #1:
El mayor elemento de cada sub-lista es:
[4, 8, 250]

Resultado de la Actividad #2:
Los números primos encontrados son:
[3, 5, 5, 13]

Gracias por usar la Aplicación!!!!!!
```

Figure 1: Ejemplo de la ejecución del programa

> **Nota:** Queda de parte del usuario/profesor si desea realizar la instalación del paquete o ejecutar el módulo:
>
> 1. En caso de proceder con la instalación del paquete, ejecutar el siguiente comando en la terminal:

```
pip install pickprimes
```

> 2. En caso de decidir ejecutar el módulo, recordar que se debe encontar en el mismo directorio que el script principal: `Asignacion.py`

Estructura del script

El script `Asignacion.py` posee 3 funciones de soporte y un cuerpo principal (`Main`) necesario para la ejecución del mismo.

Se consideró crear un módulo adicional sólo con las funciones de soporte, sin embargo, se decició incluirlas en el script principal para facilitar la tarea de revisión del código por parte del docente y así evitar la entrega de distintos archivos (`Asignacion.py`, `funciones.py`, `pickprimes.py`)

Funciones de soporte:

- `actividad_1()`: contiene las instrucciones necesarias para obtener el máximo elemento de cada sub-lista
  - Se hace uso de la comprensión de listas como fue requerido
- `actividad_2()`: contiene las instrucciones necesarias para obtener los números primos dentro de la lista
  - Se hace uso de la función `filter()` como fue requerido
- `ask()`: Contiene las instrucciones para solicitar al usuario si desea repetir alguna o ambas asignaciones en caso de error

> **Importante** Cabe destacar que se decidió implementar bloques **`try`** / `except` para validar los datos ingresados por el usuario y poder asegurar la estabilidad del mismo durante su ejecución.

Actividad #1

A continuación se muestra el código implementado para la función `actividad_1()`

```
1  def actividad_1():
2      print('=================================================')
3      print('Actividad #1: encontrar el máximo elemento de cada lista
           dentro de una lista\n')
4
5      try:
6          n = int(input('Por favor indicar cuantas listas se encuentran
               dentro de la lista: '))
7
8      except ValueError as e:
9          print('\n-----------------------------------------')
10         print('ERROR:')
11         msg = 'La cantidad de listas debe ser un número entero\n'
12         print(msg,e)
13         print('-----------------------------------------')
14         print('\nPasaremos a la siguiente actividad')
15         return False, msg
16
17     else:
18         print('\n-----------------------------------------')
19         print('Nota: se debe separar los elelmentos de cada lista con
               espacio')
20         print('-----------------------------------------\n')
21
22         try:
23             test_case = [list(map(int,input(f'Ingrese los elementos de
                   la lista #{i}: ').split())) for i in range(1,n+1)]
24
25         except ValueError as e:
26             print('\n-------------------------------------')
27             print('ERROR:')
28             msg = 'Los valores indicados de cada lista deben ser
                   enteros\n'
29             print(msg,e)
30             print('-------------------------------------')
31             print('\nPasaremos a la siguiente actividad')
32             return False, msg
33
34         else:
35             # Creación del resultado de la Actividad #1
36             pdb.set_trace()
37             return [max(lista) for lista in test_case], None
```

Es posible observar que en la línea 36 se ha incluído el comando `pdb.set_trace()` indicando que en dicha línea se detendrá nuestro programa para comenzar con las labores de depuración de errores.

> **Importante** Se debe comentar la línea 36 para poder disfrutar de la mejor experiencia al ejecutar el programa sin el depurador pdb

Observaciones durante la depuración del código

- Tras ejecutar el programa debemos suministrar la información solicitada para construir la lista de listas que vamos a estudiar. De esta forma accederemos a la línea 36 donde se ejecutará el debugger y aparecerá en consola las siglas (pdb)

```
(base) marci@Marcials-MacBook-Pro:~/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4$ python Asignacion_2.py
Bienvenido a la Asignación #4 de BPP

================================================================================
Actividad #1: encontrar el máximo elemento de cada lista dentro de una lista

Por favor indicar cuantas listas se encuentran dentro de la lista: 3

----------------------------------------------------------------
Nota: se debe separar los elelmentos de cada lista con espacio
----------------------------------------------------------------

Ingrese los elementos de la lista #1: 2 4 1
Ingrese los elementos de la lista #2: 1 2 3 4 5 6 7 8
Ingrese los elementos de la lista #3: 100 250 43
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)actividad_1()
-> return [max(lista) for lista in test_case], None
(Pdb)
```

Figure 2: Depuración paso 1

- En la imagen previa, podemos destacar que pdb nos proporciona información valiosa en pantalla
  - La línea que comienza con > indica en que parte del código nos encontramos: se observa el path del fichero, luego del pathse indica la línea del código donde pdb ha detenido la ejecución (aún no se ha ejecutado dicha línea) y luego del paréntesis se muestra el entorno donde nos encontramos
  - La siguiente líena comienza con una flecha (–>) y muestra la instrucción a ejecutar
- Realizamos la defenición del punto de parada #1 a través del comando **break** / b seguido de la línea de código donde se desea definir

```
(base) marci@Marcials-MacBook-Pro:~/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4$ python Asignacion_2.py
Bienvenido a la Asignación #4 de BPP

================================================================================
Actividad #1: encontrar el máximo elemento de cada lista dentro de una lista

Por favor indicar cuantas listas se encuentran dentro de la lista: 3

----------------------------------------------------------------
Nota: se debe separar los elelmentos de cada lista con espacio
----------------------------------------------------------------

Ingrese los elementos de la lista #1: 2 4 1
Ingrese los elementos de la lista #2: 1 2 3 4 5 6 7 8
Ingrese los elementos de la lista #3: 100 250 43
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)actividad_1()
-> return [max(lista) for lista in test_case], None
(Pdb) b 42
Breakpoint 1 at /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py:42
(Pdb)
```

Figure 3: Depuración paso 2

- Una vez definido el punto de parada hacemos uso del comando `step` / `s` para avanzar hasta el punto de parada. Cabe destacar que podríamos hacer uso del comando `next` / `n` de igual forma, la diferencia recae en que `next` continuará a la siguiente línea pero se mantendrá en el entorno actual y, en caso de encontrarse otra función, no pasará al entorno de dicha función. `step`, por el contrario, accedera a la siguiente línea dentro del nuevo entorno haciendo y mostrará en pantalla el mensaje `--call--`

```
(base) marci@Marcials-MacBook-Pro:~/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4$ python Asignacion_2.py
Bienvenido a la Asignación #4 de BPP

================================================================================
Actividad #1: encontrar el máximo elemento de cada lista dentro de una lista

Por favor indicar cuantas listas se encuentran dentro de la lista: 3

----------------------------------------------------------------
Nota: se debe separar los elelmentos de cada lista con espacio
----------------------------------------------------------------

Ingrese los elementos de la lista #1: 2 4 1
Ingrese los elementos de la lista #2: 1 2 3 4 5 6 7 8
Ingrese los elementos de la lista #3: 100 250 43
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)actividad_1()
-> return [max(lista) for lista in test_case], None
(Pdb) b 42
Breakpoint 1 at /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py:42
(Pdb) step
--Call--
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)<listcomp>()
-> return [max(lista) for lista in test_case], None
(Pdb)
```

Figure 4: Depuración paso 3

  - Importante destacar como `pdb` nos indica que nos encontramos en el mismo script (ver `path`) pero en un entorno diferente: `<listcomp>`
  - Este nuevo entorno hace referencia al construcctor de la comprensión de listas
  - Un punto importante es que el constructor no crea una variable local para el resultado de la comprensión o, al menos, no se logró identificarlo

- `pdb`ha ingresado al constructor pero no ha comenzado su ejecución; podemos validarlo al llamar a la variable `lista` y vemos como aún no ha sido definida

```
(base) marci@Marcials-MacBook-Pro:~/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4$ python Asignacion_2.py
Bienvenido a la Asignación #4 de BPP

================================================================================
Actividad #1: encontrar el máximo elemento de cada lista dentro de una lista

Por favor indicar cuantas listas se encuentran dentro de la lista: 3

----------------------------------------------------------------
Nota: se debe separar los elelmentos de cada lista con espacio
----------------------------------------------------------------

Ingrese los elementos de la lista #1: 2 4 1
Ingrese los elementos de la lista #2: 1 2 3 4 5 6 7 8
Ingrese los elementos de la lista #3: 100 250 43
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)actividad_1()
-> return [max(lista) for lista in test_case], None
(Pdb) b 42
Breakpoint 1 at /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py:42
(Pdb) step
--Call--
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)<listcomp>()
-> return [max(lista) for lista in test_case], None
(Pdb) n
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)<listcomp>()
-> return [max(lista) for lista in test_case], None
(Pdb) p lista
*** NameError: name 'lista' is not defined
(Pdb)
```

Figure 5: Depuración paso 4

- Un comando importante valioso es `where` / `w` ya que nos muestra las distintas llamadas de la misma variable en distintos entornos.

```
(base) marci@Marcials-MacBook-Pro:~/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4$ python Asignacion_2.py
Bienvenido a la Asignación #4 de BPP

================================================================================
Actividad #1: encontrar el máximo elemento de cada lista dentro de una lista

Por favor indicar cuantas listas se encuentran dentro de la lista: 3

----------------------------------------------------------------
Nota: se debe separar los elelmentos de cada lista con espacio
----------------------------------------------------------------

Ingrese los elementos de la lista #1: 2 4 1
Ingrese los elementos de la lista #2: 1 2 3 4 5 6 7 8
Ingrese los elementos de la lista #3: 100 250 43
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)actividad_1()
-> return [max(lista) for lista in test_case], None
(Pdb) b 42
Breakpoint 1 at /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py:42
(Pdb) step
--Call--
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)<listcomp>()
-> return [max(lista) for lista in test_case], None
(Pdb) n
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)<listcomp>()
-> return [max(lista) for lista in test_case], None
(Pdb) p lista
*** NameError: name 'lista' is not defined
(Pdb) w
  /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(89)<module>()
-> result, msg1 = actividad_1()
  /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)actividad_1()
-> return [max(lista) for lista in test_case], None
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)<listcomp>()
-> return [max(lista) for lista in test_case], None
(Pdb)
```

Figure 6: Depuración paso 5

  - Los entornos se encuentrar organizados de forma descendente, donde la llamada más reciente se encuentrá en la parte inferior
  - Podemos observar por el indicador > que nos encontramos en el entorno `<listcomp>`

  - Sin embargo, se encuentran activos otros dos entornos donde se han producido las llamadas a la función analizada
    * `asignacion_1` es la función donde se ha definido la comprensión de listas, lo que indica que <`listcomp`> es un entorno independiendte de la función son sus propias variables
    * <`module`> es el entorno principal/global del script

- Podemos pasar y probar el estado de variables en los otros entornos a través de los comandos `up`/ `u` y `down` / `d`
  - Por ejemplo validemos la variable `test_case`
  - Importante: siempre debemos estar atentos al cursor > que nos indica el entorno activo de `pdb`

```
(Pdb) w
  /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(89)<module>()
-> result, msg1 = actividad_1()
  /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)actividad_1()
-> return [max(lista) for lista in test_case], None
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)<listcomp>()
-> return [max(lista) for lista in test_case], None
(Pdb) u
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)actividad_1()
-> return [max(lista) for lista in test_case], None
(Pdb) p test_case
[[2, 4, 1], [1, 2, 3, 4, 5, 6, 7, 8], [100, 250, 43]]
(Pdb) d
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)<listcomp>()
-> return [max(lista) for lista in test_case], None
(Pdb) p test_case
*** NameError: name 'test_case' is not defined
(Pdb)
```

Figure 7: Depuración paso 6

- Procedemos a supervisar la construcción de la lista, para ello hacemos uso del comando `next` / `n`
- Una vez, comienza la construcción de la lista podemos observar la variable local `lista` y podemos calcular su máximo con el método `max()`
- En lugar de hacer uso del comando print (`p`) de `pdb`, se usó el comando `display`

```
(Pdb) n
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)<listcomp>()
-> return [max(lista) for lista in test_case], None
(Pdb) display lista
display lista: [2, 4, 1]
(Pdb) display max(lista)
display max(lista): 4
(Pdb)
```

Figure 8: Depuración paso 7

- La ventaja de `display` es que nos permite llevar una trazabilidad del cambio de las variables con cada ejecución

```
(Pdb) n
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)<listcomp>()
-> return [max(lista) for lista in test_case], None
display lista: [1, 2, 3, 4, 5, 6, 7, 8]  [old: [2, 4, 1]]
display max(lista): 8  [old: 4]
(Pdb)
```

Figure 9: Depuración paso 8

```
(Pdb) n
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)<listcomp>()
-> return [max(lista) for lista in test_case], None
display lista: [100, 250, 43]  [old: [1, 2, 3, 4, 5, 6, 7, 8]]
display max(lista): 250  [old: 8]
(Pdb)
```

Figure 10: Depuración paso 9

- Una vez se ha validado todas las listas de `test_case` podemos observar como `pdb` muestra el mensaje `--Return--` indicando que ha finalizado la función/método y nos muestra su resultado al lado del nombre del entorno `<listcomp>`

```
(Pdb) n
--Return--
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)<listcomp>()->[4, 8, 250]
-> return [max(lista) for lista in test_case], None
(Pdb)
```

Figure 11: Depuración paso 10

- La siguiente línea del depurador nos muestra el `--Return--` de la función `asignacion_1` el cual ha sido definido como un `tuple` con segundo elemento `None`

```
(Pdb) n
--Return--
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)actividad_1()->([4, 8, 250], None)
-> return [max(lista) for lista in test_case], None
(Pdb)
```

Figure 12: Depuración paso 11

- Una vez de vuelta al entorno principal/global `<module>` y luego de haber ejecutado la función `asignacion_1`, podemos imprimir el valor de la variable `result` que debe ser la lista con los elementos máximos de cada lista de `test_case`

```
(Pdb) n
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(90)<module>()
-> primes, msg2 = actividad_2()
(Pdb) p result
[4, 8, 250]
(Pdb)
```

Figure 13: Depuración paso 12

- Para finalizar la depuración del código y continuar con el programa hacemos uso del comando **`continue`** / `c`

```
(Pdb) n
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)<listcomp>()
-> return [max(lista) for lista in test_case], None
(Pdb) display lista
display lista: [2, 4, 1]
(Pdb) display max(lista)
display max(lista): 4
(Pdb) n
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)<listcomp>()
-> return [max(lista) for lista in test_case], None
display lista: [1, 2, 3, 4, 5, 6, 7, 8]  [old: [2, 4, 1]]
display max(lista): 8  [old: 4]
(Pdb) n
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)<listcomp>()
-> return [max(lista) for lista in test_case], None
display lista: [100, 250, 43]  [old: [1, 2, 3, 4, 5, 6, 7, 8]]
display max(lista): 250  [old: 8]
(Pdb) n
--Return--
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)<listcomp>()->[4, 8, 250]
-> return [max(lista) for lista in test_case], None
(Pdb) n
--Return--
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(42)actividad_1()->([4, 8, 250], None)
-> return [max(lista) for lista in test_case], None
(Pdb) n
> /Users/marci/Documents/Postgrados/Python/Cursos/5.Buenas practicas/Tareas/Asignacion_4/Asignacion_2.py(90)<module>()
-> primes, msg2 = actividad_2()
(Pdb) p result
[4, 8, 250]
(Pdb) c

================================================================================
Actividad #2: encontrar los números primos en la lista suministrada

--------------------------------------------------------------------
Nota: se debe separar los elementos de la lista con espacio
--------------------------------------------------------------------

ingrese los elementos de la lista:
```

Figure 14: Depuración paso 13

Actividad #2

A continuación se muestra el código implementado para la función `actividad_2()`

```
def actividad_2():
    print('\n===============================================')
    print('Actividad #2: encontrar los números primos en la lista
       suministrada')

    print('\n-----------------------------------------------')
    print('Nota: se debe separar los elementos de la lista con espacio'
       )
    print('-----------------------------------------------\n')

    try:
        test_case_2 = list(map(int,input(f'ingrese los elementos de la
           lista: ').split()))

    except ValueError as e:
        print('\n-----------------------------------------')
        print('ERROR:')
        msg = 'Los valores de cada elemento en la lista deben ser
           enteros\n'
        print(msg,e)
        print('-----------------------------------------')
        print('\nGeneramos el resumen')
        return False, msg

    else:
        # Creación del resultado de la Actividad #2
        return list(filter(is_prime, test_case_2)), None
```

Para seleccionar los números primos se hará uso de la función `is_prime` contenida dentro del paquete `pickprimes`

```
def is_prime(n):
    if n == 1:
        return False
    elif n <= 0:
        return False
    else:
        cont = True
        for i in range(2, int(n**0.5)+1):
            if n%i == 0:
                cont = False
                break
        return cont
```