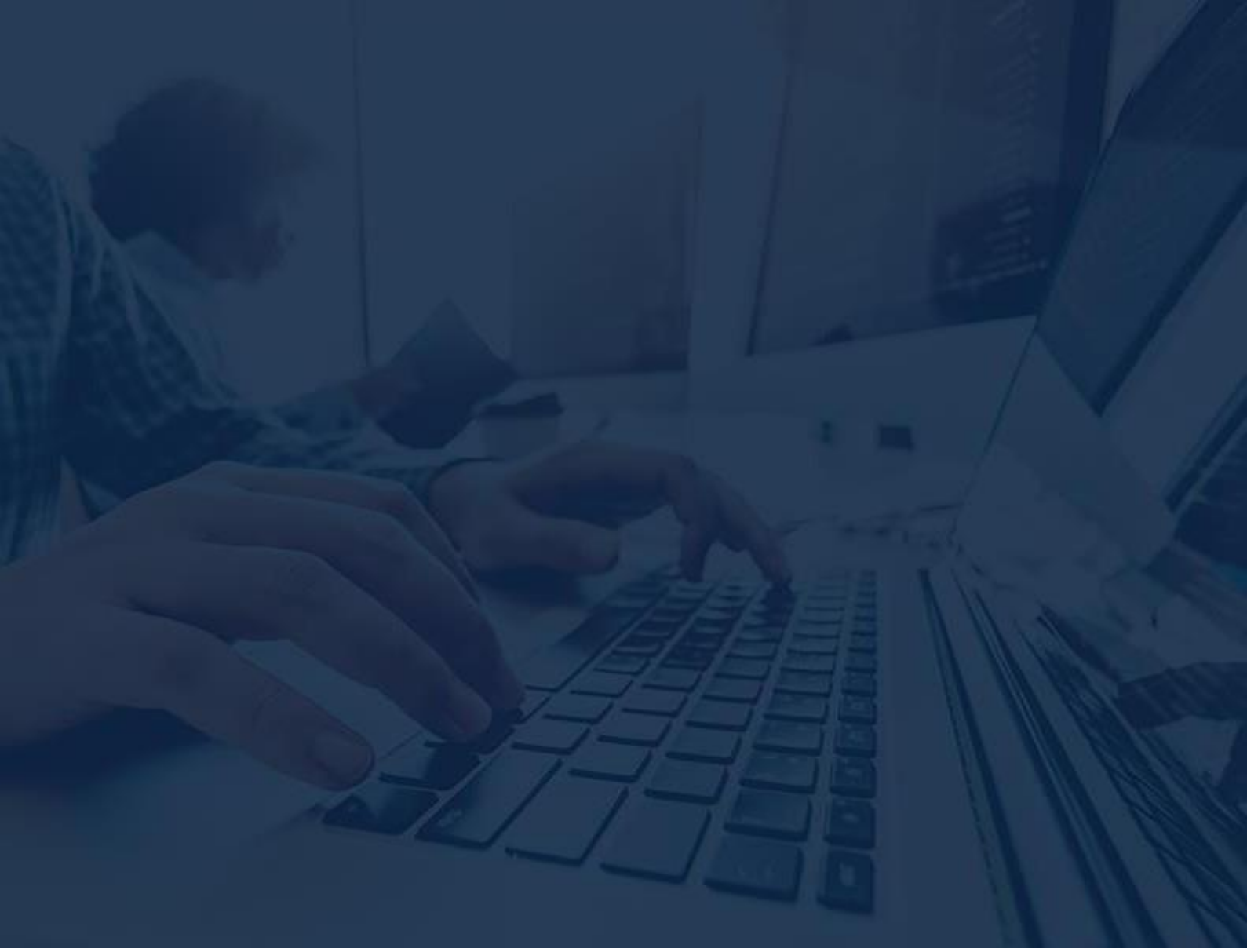

Buenas prácticas de programación en Python

Lección 4: Depuración y reglas de optimización de código

eip ESCUELA INTERNACIONAL POSGRADOS

DEPURACIÓN Y REGLAS DE OPTIMIZACIÓN DE CÓDIGO

Actividad relacionada con la lección 4:

1. **Haciendo uso de comprensión de listas realice un programa que, dado una lista de listas de números enteros, devuelva el máximo de cada lista. Ahora, haciendo uso del pdb, inserte puntos de parada justo en la línea donde ha implementado la compresión de listas. Haga pruebas mostrando el contenido de las variables y continuar con la ejecución línea a línea. ¿Qué conclusiones obtiene?**

   Para poder usar el debugger implementado de Python, no es necesario importarlo en el código ni llamar al método `set_trace()`.

   Podemos ejecutar el script con pydb como módulo y añadir directamente los breakpoints en el propio código con `breakpoint()`

```
PS C:\Users\giber\repos\Buenas prácticas\Tercera Leccion> python -m pdb main.py
```

   Si debuggamos el código y añadimos un breakpoint después de la *list comprehension*, veremos que las variables usadas dentro de la misma no son accesibles, ya que se tratan de variables almacenadas en memoria temporales, sólo durante la duración de la comprensión de lista.

   En Python 3.x, las variables definidas dentro de las list comprehensions no se filtran en su ámbito de cierre. Por lo tanto, la variable no estará disponible después de la list comprehension.

```
> c:\users\giber\repos\buenas prácticas\tercera leccion\main.py(7)getMaxFromIterator()
-> return maximumValues
(Pdb) p subIterator
*** NameError: name 'subIterator' is not defined
(Pdb) p maximumValues
[4, 8, 250]
(Pdb)
```