PUG_BPP_A4

Por

Pablo Ujados García

Actividad 1

Dada la siguiente lista “[2, 4, 1], [1,2,3,4,5,6,7,8], [100,250,43]]” muestra el mayor de cada lista, depura el programa con PCB y saca conclusiones

Programa que hemos desarrollado

PUG_BPP_A4.py

```python
import pdb
pdb.set_trace()

mayor_L1 = 0
mayor_L2 = 0
mayor_L3 = 0
count = 0
LL = [(2,4,1),(1,2,3,4,5,6,7,8),(100,250,43)]

for l1 in LL:
    for l2 in l1:
        if count == 0:
            if mayor_L1 < l2:
                mayor_L1 = l2

        if count == 1:
            if mayor_L2 < l2:
                mayor_L2 = l2

        if count == 2:
            if mayor_L3 < l2:
                mayor_L3 = l2

    count += 1

print(f'El mayor de la lista 1 es {mayor_L1}')
print(f'El mayor de la lista 2 es {mayor_L2}')
print(f'El mayor de la lista 3 es {mayor_L3}')

if __name__ == '__main__':
    pass
```

Se ha importado PDB y se ha utilizado la función set_trace() para poder depurar el programa por consola

Activamos consola de comando y nos ubicamos en la carpeta de del desarrollo y activamos programa y hacemos el listado del programa

Command Prompt - PUG_BPP_A4.py

```
E:\PABLO\Documents\EIP_Escuela_Internacional_Postgrado\Master_paython_8\5_Buenas_Practicas_De_Programacion_Con_Python\Practicas>cd
E:\PABLO\Documents\EIP_Escuela_Internacional_Postgrado\Master_paython_8\5_Buenas_Practicas_De_Programacion_Con_Python\Practicas

E:\PABLO\Documents\EIP_Escuela_Internacional_Postgrado\Master_paython_8\5_Buenas_Practicas_De_Programacion_Con_Python\Practicas>PUG_BPP_A4.py
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(4)<module>()
-> mayor_L1 = 0
(Pdb) list
  1     import pdb
  2     pdb.set_trace()
  3
  4  -> mayor_L1 = 0
  5     mayor_L2 = 0
  6     mayor_L3 = 0
  7     count = 0
  8     LL = [(2,4,1),(1,2,3,4,5,6,7,8),(100,250,43)]
  9
 10     for l1 in LL:
 11         for l2 in l1:
(Pdb) list
 12             if count == 0:
 13                 if mayor_L1 < l2:
 14                     mayor_L1 = l2
 15
 16             if count == 1:
 17                 if mayor_L2 < l2:
 18                     mayor_L2 = l2
 19
 20             if count == 2:
 21                 if mayor_L3 < l2:
 22                     mayor_L3 = l2
(Pdb) list
 23
 24         count += 1
 25
 26     print(f'El mayor de la lista 1 es {mayor_L1}')
 27     print(f'El mayor de la lista 2 es {mayor_L2}')
 28     print(f'El mayor de la lista 3 es {mayor_L3}')
 29
 30     if __name__ == '__main__':
 31         pass
[EOF]
(Pdb)
```

Ponemos los puntos de interrupción en las líneas 12,16 y 20

```
(Pdb) break 12,16,20
Breakpoint 1 at e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py:12
(Pdb) break 16
Breakpoint 2 at e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py:16
(Pdb) break 20
Breakpoint 3 at e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py:20
```

Listamos de nuevo para verificar los puntos de interrupción

```
(Pdb) list 1
  1     import pdb
  2     pdb.set_trace()
  3
  4  -> mayor_L1 = 0
  5     mayor_L2 = 0
  6     mayor_L3 = 0
  7     count = 0
  8     LL = [(2,4,1),(1,2,3,4,5,6,7,8),(100,250,43)]
  9
 10     for l1 in LL:
 11         for l2 in l1:
(Pdb) list
 12 B           if count == 0:
 13                 if mayor_L1 < l2:
 14                     mayor_L1 = l2
 15
 16 B           if count == 1:
 17                 if mayor_L2 < l2:
 18                     mayor_L2 = l2
 19
 20 B           if count == 2:
 21                 if mayor_L3 < l2:
 22                     mayor_L3 = l2
(Pdb) list
 23
 24         count += 1
 25
 26     print(f'El mayor de la lista 1 es {mayor_L1}')
 27     print(f'El mayor de la lista 2 es {mayor_L2}')
 28     print(f'El mayor de la lista 3 es {mayor_L3}')
 29
 30     if __name__ == '__main__':
 31         pass
[EOF]
(Pdb)
```

Hacemos una inspección inicial de todas las variables que necesitamos seguir, viendo que tiene los valores iniciales y correctos.

Count = contador de vuelta por la que va sobre la lista de listas

L1 = sub-lista actual de la lista

L2 = número cogido de la sub-lista actual

Mayor_LX = variable en la que se guardara el mayor de cada sub-lista.

```
(Pdb) p l1
(2, 4, 1)
(Pdb) p count
0
(Pdb) p l2
2
(Pdb) p mayor_L1
0
```

Avanzamos a la siguiente línea de ejecución viendo , viendo como entra en la instrucción de comparación de “mayor_l1” menor que “l2” , al darse esta condición entra y asigna a “mayor_l1” el valor de “l2”

```
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(13)<module>()
-> if mayor_L1 < l2:
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(14)<module>()
-> mayor_L1 = l2
(Pdb) p l1
(2, 4, 1)
(Pdb) p count
0
(Pdb) p l2
2
(Pdb) p mayor_L1
0
```

```
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(16)<module>()
-> if count == 1:
(Pdb) p l1
(2, 4, 1)
(Pdb) p count
0
(Pdb) p l2
2
(Pdb) p mayor_L1
2
(Pdb)
```

Quedando las variables de esta manera

L2 = 2

Mayor_L1 = 2

Continuamos avanzando línea a línea la ejecución, vemos que en la siguiente vuelta del bucle no encontramos que “L2” a cogido el valor 4 y que “mayor_L1” tiene 2

```
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(12)<module>()
-> if count == 0:
(Pdb) p count
0
(Pdb) p l2
4
(Pdb) p mayor_L1
2
(Pdb)
```

Dado este caso vuelve a entrar en la condición dado que “count” = 0 y “l2” es mayor que “mayor_l1”

Y asigna el valor de “l2” a “mayor_l1”

```
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(13)<module>()
-> if mayor_L1 < l2:
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(14)<module>()
-> mayor_L1 = l2
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(16)<module>()
-> if count == 1:
(Pdb) p count
0
(Pdb) p l2
4
(Pdb)
4
(Pdb) p mayor_L1
4
(Pdb)
```

Seguimos avanzando línea a línea la ejecución , hasta encontrarlo en el ultimo numero, en esta ocasión tenemos que “count” = 0 con lo cual eso cumple pero “l2” = 2 y “mayor_L1” = 4, con lo cual no es mayor que “l2”.

```
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(20)<module>()
-> if count == 2:
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(11)<module>()
-> for l2 in l1:
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(12)<module>()
-> if count == 0:
(Pdb) p l2
1
(Pdb) p mayor_L1
4
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(13)<module>()
-> if mayor_L1 < l2:
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(16)<module>()
-> if count == 1:
(Pdb)
```

De esta manera, la variable no se asigna, teniendo ya el mayor de la lista, saliendo del sub-bucle y avanzando en el bucle principal a la siguiente sub-lista.

A partir de aquí se ha puesto la primera iteración, para ver el inicio de los datos y el final de la iteración

```
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(16)<module>()
-> if count == 1:
(Pdb) continue
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(20)<module>()
-> if count == 2:
(Pdb) continue
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(12)<module>()
-> if count == 0:
(Pdb) p count
1
(Pdb) p l1
(1, 2, 3, 4, 5, 6, 7, 8)
(Pdb) p l2
1
(Pdb) p mayor_L2
0
(Pdb)
```

Vemos como avanzando línea a línea ocurre exactamente el mismo proceso que la anterior vez, pero teniendo en cuenta que ahora “count” = 1, y “l2” se compara con “mayor_l2”

```
(Pdb) continue
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(16)<module>()
-> if count == 1:
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(17)<module>()
-> if mayor_L2 < l2:
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(18)<module>()
-> mayor_L2 = l2
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(20)<module>()
-> if count == 2:
(Pdb) p l2
1
(Pdb) p mayor_L2
1
```

De esta manera llegamos a la última iteración, dando la casualidad de que el ultimo número es el mayor, con lo cual, cumpliendo condición entra en los diferentes condicionales y “mayor_l2” termina con valor 8

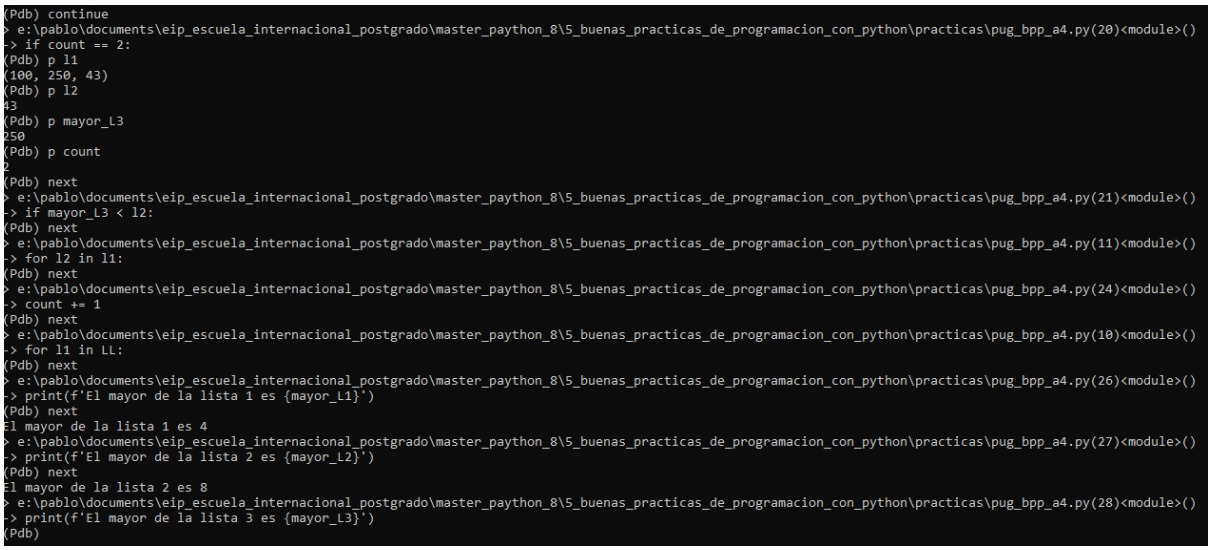
```
(Pdb) continue
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(16)<module>()
-> if count == 1:
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(17)<module>()
-> if mayor_L2 < l2:
(Pdb) p l2
8
(Pdb) p mayor_L2
7
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(18)<module>()
-> mayor_L2 = l2
(Pdb) next
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(20)<module>()
-> if count == 2:
(Pdb) p l2
8
(Pdb) p mayor_L2
8
(Pdb)
```

Por ultimo pasamos a la ultimo sub-lista, “count” pasa a tener valor 2, “l2” = 100 y “mayor_l3” = 0

```
(Pdb) continue
> e:\pablo\documents\eip_escuela_internacional_postgrado\master_paython_8\5_buenas_practicas_de_programacion_con_python\practicas\pug_bpp_a4.py(12)<module>()
-> if count == 0:
(Pdb) p l1
(100, 250, 43)
(Pdb) p l2
100
(Pdb) p count
2
(Pdb) p mayor_L3
0
(Pdb)
```

Dándose la misma casuística que las anteriores, el ejecución va entrado en el condicional con condición “count” = 2 y va viendo que se cumple la condición, hasta encontrarse con la utlima iteración en la cual “l2” = 43 y “mayor_l3” = 250. Siendo “mayor_L3” mayor que “l2” no se cumple condición y no se asigna el valor de “l2” a “mayor_l3”, dado que este ya tiene el mayor de la lista.

El cursor sigue avanzando y vemos como sale del sub-bucle y también es la ultima iteración de bucle principal, saliendo de este también y mostrando por ultimo las correspondientes línea al usuario diciéndole cual es el mayor de cada lista



Conclusiones, hay que tener mucho cuidado tanto con los condicionales que se aplican y los accesos a lista, dado que un mal acceso o una mala condición no pueden llevar a un error de flujo del programa o una lógica errónea.

A nivel del depurado de consola de Python, me ha resultado más fácil de lo que creia, también es cierto que es un código pequeño y habria que ver cómo se maneja esto en un código de producción.