Status	Finished
Started	Saturday, 23 November 2024, 4:50 PM
Completed	Saturday, 23 November 2024, 4:54 PM
Duration	3 mins 54 secs
Marks	5.00/5.00
Grade	**10.00** out of 10.00 (**100**%)

Information

Depuración

Al escribir un programa se pueden cometer errores sintácticos, esto es, errores de escritura que impiden que el programa pueda siquiera ejecutarse. Un error sintáctico es, por ejemplo, no poner dos puntos al final de la cabecera de una función.

Los errores sintácticos son fáciles de arreglar, porque sólo hay que cumplir las reglas del lenguaje. Una vez que un programa está correctamente escrito de acuerdo con dichas reglas, ya se puede ejecutar, pero durante la ejecución pueden aparecer errores de funcionamiento, originados ya sea por una equivocada concepción del programa o por no cumplir alguna regla del lenguaje que solo es factible comprobar en ejecución.

Es habitual que un programa mínimamente complejo funcione bien la mayoría de las veces, pero que, para unos pocos casos, falle dando resultados erróneos, o incluso, aborte (se detenga bruscamente sin terminar su tarea).

El ciclo de desarrollo de un programa pasa por diferentes fases. Las primeras son: análisis de los requerimientos y características del problema a resolver, diseño de la solución y codificación de la misma en un lenguaje de programación. Una vez que disponemos del programa ya ejecutable, se pasa a la fase de prueba, en la que se somete al programa a una batería de casos de prueba (*tests*) con el fin de encontrar casos de fallo a corregir en el código del programa.

Si las pruebas detectan un caso para el que el programa falla, se pasa a la fase de depuración que consiste en analizar el código para buscar, normalmente con ayuda de herramientas específicas, cuál es la causa del fallo: la **depuración** (*debugging*) es el proceso de buscar y solucionar defectos en el código de un programa que provocan que funcione incorrectamente.

Una herramienta de depuración (depurador o *debugger*) tiene que ofrecer funcionalidades para detener y reanudar la ejecución de un programa y, cuando está detenido, examinar el estado del proceso que lleva a cabo, inspeccionando los valores de las variables involucradas, la secuencia de llamadas a funciones realizada, el contexto de la instrucción actual, etc.

El depurador proporciona datos sobre el funcionamiento del programa que el programador tiene que analizar, razonando sobre ellos, para encontrar las causas del mal funcionamiento detectado por las pruebas.

Question 1

Complete

Mark 1.00 out of 1.00

La depuración...

Select one:

- aplica diferentes casos de prueba para comprobar si un programa funciona correctamente
- busca defectos ocultos en un programa que provocan su mal funcionamiento
- requiere del uso de herramientas especializadas que encuentran automáticamente los errores ocultos en un programa

Information

Iniciando la depuración en Python

Python dispone de un módulo, llamado *pdb* (**p**ython **d**e**b**ugger), que sirve como herramienta de depuración. Para iniciar la depuración de un programa, sólo hay que importar ese módulo y ejecutar la instrucción *set_trace()*, tal como se hace en las dos primeras líneas del siguiente ejemplo:

```
 1. import pdb
 2. pdb.set_trace()
 3. import random
 4. def find(numbers, searched):
 5.     found = False
 6.     for number in numbers:
 7.         if number == searched:
 8.             found = True
 9.             break
10.     return found
11.
12. if __name__ == '__main__':
13.     numbers = (19, 12, 35, 21, 12, 41, 37)
14.     number = random.randint(10, 50)
15.
16.     if find(numbers, number):
17.         print("El número", number, "está en ", numbers)
```

Esta instrucción hace que se arranque el depurador y se quede esperando órdenes para continuar la ejecución del programa, que queda detenido justo en la línea siguiente:

```
> /home/p16251/main.py(3)<module>()
-> import random
(Pdb)
```

Obsérvese que justo antes del *prompt* del depurador (Pdb) hay una flecha que indica que el programa está detenido en la línea cuyo contenido es "import random", la línea 3 de nuestro programa de ejemplo. Un comando muy útil es este momento, es *help*, que nos mostraría todos los comandos del depurador:

```
> /home/p13011/main.py(3)<module>()
-> import random
(Pdb) help
Documented commands (type help <topic>):
========================================
EOF    c          d        h         list      q        rv       undisplay
a      cl         debug    help      ll        quit     s        unt
alias  clear      disable  ignore    longlist  r        source   until
args   commands   display  interact  n         restart  step     up
b      condition  down     j         next      return   tbreak   w
break  cont       enable   jump      p         retval   u        whatis
bt     continue   exit     l         pp        run      unalias  where
Miscellaneous help topics:
==========================
exec  pdb
(Pdb)
```

help seguido de un comando concreto, nos da ayuda sobre ese comando en particular.

Question 2

Complete

Mark 1.00 out of 1.00

¿Qué instrucción inicia el modo depuración en Python?

Select one:

- set_pdb()
- start_pdb()
- set_trace()

Information

Breakpoints (puntos de parada)

Una cosa que podemos hacer cuando iniciamos la depuración es establecer **puntos de parada** (*breakpoints*). De esta manera, podemos hacer que la ejecución se vaya deteniendo en distintos puntos del programa, permitiéndonos inspeccionar el estado del proceso en cada momento. En el siguiente ejemplo se ponen dos puntos de parada en las líneas 6 y 14 de nuestro programa de ejemplo:

```
> /home/p17284/main.py(3)<module>()
-> import random
(Pdb) b 6
Breakpoint 1 at /home/p17284/main.py:6
(Pdb) b 14
Breakpoint 2 at /home/p17284/main.py:14
(Pdb)
```

Para poner un punto de parada se puede usar el comando *break*, o su forma abreviada, *b*, como en el ejemplo.

Question 3

Complete

Mark 1.00 out of 1.00

Elija la definición adecuada de *breakpoint*.

Select one:

- Un punto de un programa donde hay un error oculto que causa fallos en su funcionamiento
- Un punto donde el programa se detiene esperando la entrada de datos por parte del usuario
- Un punto donde el programa debe detenerse para permitir su inspección

Information

Avance

Cuando un programa está detenido en un punto de parada, o al inicio de la depuración, se puede **reanudar la ejecución** (hasta el siguiente punto de parada, si existe), usando el comando ***continue***, o sus versiones abreviadas, *cont* o ***c***:

```
> /home/p14373/main.py(3)<module>()
-> import random
(Pdb) b 14
Breakpoint 1 at /home/p14373/main.py:15
(Pdb) c
> /home/p14373/main.py(14)<module>()
-> numbers = (19, 12, 35, 21, 12, 41, 37)
(Pdb)
```

Nótese que, siempre, antes del prompt del depurador, una flecha nos muestra la línea en la que el programa está detenido, línea que aún no se ha ejecutado (justo antes, se muestra también el número de dicha línea).

Una opción alternativa es **avanzar solo una línea**, ejecutando la actual y parando antes de la siguiente, con el comando ***next***, o ***n***:

```
> /home/p16883/main.py(14)<module>()
-> numbers = (19, 12, 35, 21, 12, 41, 37)
(Pdb) n
> /home/p16883/main.py(15)<module>()
-> number = random.randint(10, 50)
(Pdb) n
> /home/p16883/main.py(17)<module>()
-> if find(numbers, number):
(Pdb)
```

Cuando la próxima instrucción a ejecutar contenga una llamada a una función, cabe la posibilidad de usar el comando *next*, que ejecuta la llamada a la función y se para en la siguiente instrucción después de que la función retorne, o el comando ***step***, abreviado ***s***, que se para en la primera instrucción de la función, permitiéndonos continuar la depuración por dentro de la misma:

```
> /home/p17498/main.py(17)<module>()
-> if find(numbers, number):
(Pdb) s
--Call--
> /home/p17498/main.py(5)find()
-> def find(numbers, searched):
(Pdb)
```

Question 4

Complete

Mark 1.00 out of 1.00

Empareje cada acción con el comando que la realiza.

La ejecución del programa avanza hasta el siguiente punto de parada	continue
Ejecuta la instrucción actual y se detiene en la siguiente de la misma secuencia	next
Si la instrucción actual es una llamada a una función, inicia la llamada y se detiene al principio de la función	step

Information

Inspección

Cuando el programa está detenido, podemos inspeccionar el estado del proceso, para averiguar si va bien o ya se ha desviado de la situación esperada, lo que significa que el fallo del programa está antes del punto actual. Una de las cosas que podemos ver es el **contexto de la línea actual**, usando el comando ***list*** o ***l***:

```
> /home/p12875/main.py(17)<module>()
-> if find(numbers, number):
(Pdb) l
 12
 13     if __name__ == '__main__':
 14         numbers = (19, 12, 35, 21, 12, 41, 37)
 15         number = random.randint(10, 50)
 16
 17 B->      if find(numbers, number):
 18             print("El número", number, "está en ", numbers)
 19         else:
 20             print("El número", number, "no está en ", numbers)
(Pdb)
```

Para **inspeccionar el valor de una variable** usamos el comando ***p*** (***print***):

```
> /home/p12608/main.py(17)<module>()
-> if find(numbers, number):
(Pdb) p numbers
(19, 12, 35, 21, 12, 41, 37)
(Pdb) p number
40
(Pdb)
```

En el ejemplo vemos que *numbers* referencia una lista con siete valores enteros y que *number* referencia el valor entero 40.

El comando ***where*** (abreviado ***w***) permite conocer la **secuencia de llamadas** que nos han conducido al punto actual. Esto es útil cuando a una función se la puede llamar desde diferentes puntos de un programa, ya que el camino que se ha seguido para llegar al punto actual puede ser relevante para determinar cómo se produce el fallo:

```
> /home/p15071/main.py(6)find()
-> found = False
(Pdb) w
  /home/p15071/main.py(17)<module>()
-> if find(numbers, number):
> /home/p15071/main.py(6)find()
-> found = False
(Pdb)
```

En el ejemplo, vemos que el programa está detenido en la función *find()*, en la línea 6 del módulo *main.py*, adonde se ha llegado ejecutando la llamada a *find()* que se encuentra en la línea 17 del mismo módulo.

Question 5

Complete

Mark 1.00 out of 1.00

Empareje cada acción con el comando que a realiza.

Muestra el contexto de la instrucción actual	l
Muestra la secuencia de llamadas a funciones que ha llevado al punto actual	w
Muestra el valor referenciado por una variable	p