encontrando-resolviendo-errores

January 28, 2020

1 Encontrar y corregir errores

Todos cometemos errores. Hasta los programadores profesionales. El IDLE es bueno atrapando errores de sintaxis y de ejecucion, pero hay un tercer tipo de errores: lógicos. Los errores lógicos ocurren cuando un programa válido no hace lo esperado del mismo.

Errores logicos causan comportamiento inesperado al cual llamamos **bugs**. La traduccion literal no es favorable: *insectos*, por lo que preferimos llamarlos simplemente **comportamiento no intencionado**. La remoción de estos errores se llama **depuración**, y un **depurador** es una herramienta que nos ayuda a cazar errores y entender porque estan ocurriendo.

Aprendiendo a encontrar y resolver errores es una habilidad muy preciada y de gran utilidad.

En esta sección vamos a: * Aprender usar la ventana del Depurador de IDLE * Depurar una función con errores

1.1 Utilizando el Depurador IDLE

La interfaz principal del depurador IDLE es a travez de la ventana de control del depurador, llamada **Debug Control Window**. De ahora en adelante esta ventana la llamaremos el Depurador.

Abramos el Depurador seleccionando `Debug > Debugger` desde el menu de la ventana interactiva.

Cuando el Depurador esta abierto, podemos ver que el prompt `>>>` dice `[DEBUG ON]`, indicando que el depurador esta abierto.

En esta seccion vamos a aprender como el Depurador esta organizado y como navegamos a traves del codigo, linea por linea. Ademas, vamos a ver como crear puntos de interrupción para acelerar el proceso de depuracion.

1.2 La ventana del Depurador: Vision General

Para que podamos ver como funciona el depurador, vamos a empezar escribiendo un programa simple sin ningun error

[1]:
```
for i in range(1, 4):
    j = i * 2
    print(f"i es {i} y j es {j}")
```

```
i es 1 y j es 2
i es 2 y j es 4
i es 3 y j es 6
```

Cuando guardamos y ejecutamos este script con el Depurador, observamos que la ejecución no llega muy lejos.

El panel contiene la siguiente linea en azul: `> '__main__'.<module>(), line 1: for i in range(1, 4):`

El tablero `stack` nos indica que la linea 1, esta por ejecutarse pero no se ha ejecutado todavia. La parte `'__main__'.module()` nos indica que estamos en la parte principal del script y no adentro de en una funcion.

Abajo del tablero `stack`, esta el tablero de `Locals`, que muestra una lista de algunos terminos extraños como `__annotations__`, `__builtins__`, `__doc__`, entre otros. Estos son unos valores internos que por el momento los podemos ignorar. Mientras el programa se ejecuta, vamos a ver variables declaradas en el codigo mostradas en esta ventana para poder darle seguimiento a su valor.

Existen cinco (5) botonos ubicados en la parte superior izquierda del Depurador: * Go * Step * Over * Out * Quit

1.2.1 Step

Si presionamos el boton *Step* en la esquina superior izquierda del Depurador, la ventana cambia un poco de apariencia.

Hay dos diferencias que hay que destacar. Primero, el mensaje en la ventana Stack cambia a:

`> '__main__'.<module>(), line 2: j = i * 2:`

En este punto, la linea 1 del codigo se ejecuto y el depurador se detuvo justo antes de ejecutar la linea 2.

El segundo cambio es la nueva variable `i` que es asignada al valor `1` en el tablero de `Locals`. Eso es porque el `for` loop en la primera linea de codiga cambia la variable `i` y se la asigna el valor `1`.

Ahora presionemos el botom `Step` para navegar por el codigo linea por linea, observando que ocurre en el depurador. Cuando llegamos a la linea `print()`, podemos ver que el dato de salida es mostrado en la ventana interactiva una pieza a la vez.

Asimismo, podemos darle seguimiento a los valores de `i` y `j` mientras navegamos por el `for` loop. Esto es muy util cuando estamos tratando de ubicar errores en programas. Saber que valor tiene una variable en cada linea de codigo nos permite identificar donde hay algo incorrecto o no deseado.

1.2.2 Puntos de interrupcion y el boton "Go"

En ocasiones sabemos que el error debe estar en una seccion en particular, pero no sabemos el lugar exacto. En vez que estar presionando **Step** todo el dia, podemos crear un punto de interrupcion que le indica al depurador que ejecute todo el codigo antes del punto de interrupcion hasta el punto de interrupcion. La interrupcion se produce en el depurador para la ejecucion llega a una pausa a fin de poder verificar el estado actual del programa.

Para establecer un punto de interrupcion, habra que presionar el boton de la derecha del mouse en la linea de codigo en el script y seleccionar "Set Breakpoint". IDLE resalta esa linea en amarillo para indicar que el punto de interrupcion ha sido configurado. Podemos eliminar el punto en

cualquier momento presionando en boton de la derecha del mouse an la linea que tiene el punto y seleccionando *Clear Breakpoint*.

Presiona el boton *Quit* en la parte superior del Depurador para apagarlo. Esto no cierra la ventana porque lo vamos a utilizar nuevamente en un momento.

Vamos a crear un breakpoint en la linea de codigo con el `print()`.

Ahora ejecutemos el script presionando *F5*. Justo como antes, el tablero *Stack* del Depurador indica que el depurador ha iniciado y esta a la espera de la ejecucion de la linea 1. Esta vez, en vez de presionar el boton *Step*, presionemos el boton *Go*.

El tablero *Stack* ahora nos indica que estamos a la espera de la ejecucion de la linea 3.

Si miramos al table *Locals*, veremos que ambas variables `i` y `j` tienen los valores `1` y `2` respectivamente.

Presionando el boton *Go*, le indicamos al depurador que ejecute el codigo de manera continua hasta llegar al final del programa o an punto de interrupcion. Ahora presionemos *Go*.

El mismo mensaje de antes es mostrado en el tablero *Stack*, indicando que el depurador esta a la espera de ejecutar la linea 3, sin embargo los valores de las variables `i` y `j` son `2` y `4`. La ventana interactiva tambien muestra el dato de salida obtenido por la ejecucion de la linea con `print()` por primera vez.

Cada vez que presionamos *Go*, el depurador ejecuta el codigo de manera continua hasta que el programa llegue al proximo punto de interrupcion. Como ya configuramos el punto de interrupcion en la linea 3, adentro del `for` loop, el depurador se detiene en esta linea cada vez que paga por el loop.

Si presionamos *Go* una tercera vez, ahora `i` y `j` tienen los valores `3` y `6`. Que creemos que pasaría si presionaramos *Go* otra vez? Ya que el `for` loop solo cicla 3 veces, cuando presionamos Go esta vez, el programa termina de ejecutarse.

1.3 "Over" y "Out"

El boton *Over* funciona como una combinacion de *Step* y *Go*. Navega o pasa sobre una funcion o ciclo. En otras palabras, si estamos por utilizar *Step* en una funcion con el depurador, todavia podemos ejecutar el codigo de la funcion sin la necesidad de utilizar *Step* en cada linea del programa. El boton *Over* nos lleva directamente al resultado de ejecutar esa funcion.

Asimismo, si ya estamos adentro de una cuncion o ciclo, el boton *Out* ejecuta el codigo restante de una funcion o ciclo y luego ocasiona una pausa.

1.4 Arregluemos algunos errores

Ahora que hemos utilizado el depurador, veamos un programa que presenta errores.

El siguiente codigo define una funcion `agrega_guion_bajo()` que toma como parametro un objeto string llamado `palabra` y retorna el nuevo string conteniendo una copia del mismo con cada caracter rodeado por guion bajos. Por ejemplo, `agrega_guion_bajo('python')` deberia retornar `"_p_y_t_h_o_n_"`.

[5]:
```
def agrega_guion_bajo(palabra):
    nueva_palabra = "_"
    for i in range(0, len(palabra)):
        nueva_palabra = palabra[i] + '_'
    return nueva_palabra

frase = 'Hola'
print(agrega_guion_bajo(frase))  # deberia imprimir _h_e_l_l_o_
```

```
a_
```

Si ya sabe cual es el problema, no lo arreglue, toda vez que el objetivo de esta seccion es aprender a utilizar depurador IDLE para identificar el problema.

La depuracion radica en la resolucion de un problema. En esta seccion aprenderemos cuatro (4) pasos para empezar el proceso de depuracion.

1. Adivine cual es la seccion del codigo que produce el error.
2. Coloque un punto de interrupcion e inspeccione el codigo utilizando el boton *Step* una seccion a la vez, mientras le damos seguimiento a variables importantes.
3. Identifique la linea de codigo con el error y haga el cambio para resolver el problema.
4. Repita pasos 1 a 3 hasta que el codigo funcione correctamente.

1.4.1 Adivinemos donde esta el error

El codigo esta dividido en dos (2) secciones: 1. La definicion de la funcion 2. Bloque principal de codigo con la variable frase y el `print()`

El punto 2 no parece indicio de algun problema. El problema debe estar en la funcion.

[7]:
```
def agrega_guion_bajo(palabra):
    nueva_palabra = "_"  # creacion de nueva variable
    for i in range(0, len(palabra)):  # el problema debe a partir de aqui
        nueva_palabra = palabra[i] + '_'
    return nueva_palabra
```

1.4.2 Configuremos el punto de interrupcion e inspeccionemos el codigo

Ahora que hemos identificado donde el error debe estar, configuremos un punto de interrupcion al comienzo del `for` loop para que podamos darle seguimiento desde la ventana de control del Depurador.

- Ahora abramos el Depurador y ejecutemos el script.
- Ejecucion se detiene se la primera linea que ve.
- Presionemos el boton *Go* para que el codigo se ejecute hasta el punto de interrupcion.
- En este punto el codigo se detiene justo antes de ingresar al `for` loop. Notemos que existen dos variables `palabra` y `nueva_palabra` en el tablero *Locals*. La variable `palabra` tiene un valor de `'hola'` y la variable `nueva_palabra` tiene un valor de `_`.
- Presionemos el boton *Step* una vez para ingresar al `for` loop.
- La ventana del depurador cambia y vemos una nueva variable `i` con el valor `0`

- `i` es el contador utilizado en el `for` loop y lo podemos utilizar para darle seguimiento al numero de pasos ejecutados.
- Presionemos una vez mas en *Step*
- Observemos que `nueva_palabra` ahora tiene un valor de `'h_'`
- Notemos que originalmente `nueva_palabra` tenia un valor de `'_'` y en la segunda iteracion del ciclo, deberia tener un valor de `'_h_'`
- Observemos que presionando el boton *Step* nuevamente se produce el mismo comortamiento, ahora `nueva_palabra` tiene un valor de `'e_'` y deberia ser `'_h_e_'`

1.4.3 Identifiquemos el error y intentemos arregarlo

La conclusion a la cual podemos llegar es que la variable `nueva_palabra` es sobreescrita en cada iteracion del `for` loop con el proximo caracter del string `'hola'` y el guion bajo. Toda vez que existe solo una linea de codigo en el `for` loop, podemos llegar a la conclusion que el problema radica en la siguiente linea:

```
nueva_palabra = palabra[i] + '_'
```

Si observamos detalladamente, esta linea de codigo le dice a Python que obtenga el proximo caracter de la variable `palabra`, le agrega un guion bajo `_`, y produce la asignacion del nuevo string a la variable `nueva_palabra`.

Sin embargo, para arreglar el problema, necesitamos decirle a Python que concatene el string `palabra[i] + '_'`al valor actual de `nueva_palabra`.

Presione *Quit* en la ventana de control del depurador, pero no cierre esa ventana todavia.

Abra el script y cambiemos la siguiente linea en el `for` loop:

```
nueva_palabra = nueva_palabra + palabra[i] + '_'
```

1.4.4 Repitamos los pasos anteriores hasta que no haya errores

1.4.5 Otras formas de encontrar errores en el codigo

A veces es mas facil identificar errores utilizar la funcion `print()` para mostrar los valores de algunas variables. Por ejemplo, en vez que utilizar el depurador en el script previo, podemos agregar la siguiente linea de codigo al final del `for` loop:

```
print(f"i = {i}; nueva_palabra = {nueva_palabra}")
```

[9]:
```
def agrega_guion_bajo(palabra):
    nueva_palabra = "_"
    for i in range(0, len(palabra)):
        nueva_palabra = palabra[i] + '_'
        print(f"i = {i}; nueva_palabra = {nueva_palabra}")
    return nueva_palabra

frase = 'hola'
print(agrega_guion_bajo(frase))
```

```
i = 0; nueva_palabra = h_
i = 1; nueva_palabra = o_
```

```
i = 2; nueva_palabra = l_
i = 3; nueva_palabra = a_
a_
```

1.5 Resumen

Hemos aprendido sobre el Depurador IDLE. Hemos visto como inspeccionar los valores de variables, ingresar puntos de interrupcion e utilizar los botones *Step*, *Go*, *Over*, y *Out*. Tambien hemos practicado el proceso de depuracion de una funcion que no funcionaba correctamente e utilizamos cinco pasos para lograrlo: 1. Adivinamos donde el error estaba 2. Ingresamos un punto de interrupcion 3. Inspeccionamos el codigo 4. Identificamos el error 5. Intentamos arreglar el error

1.6 Prueba de conocimiento

1. Que es un depurador?
2. Quienes utilizan el depurador?
3. Que es un breakpoint o punto de interrupcion?
4. Que pasos componen el proceso de depuracion?