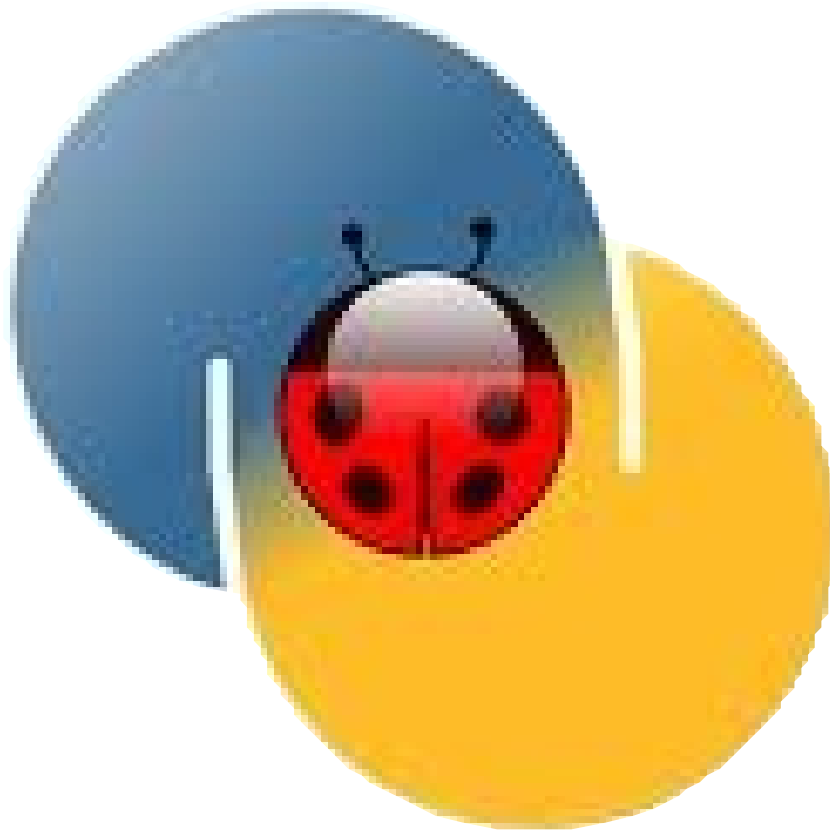


# Tutorial de Winpdb

## Primeros pasos

Matias Gerard  
matias.gerard@gmail.com



Versión adaptada y traducida al español.

Fuente: <http://winpdb.org/tutorial/WinpdbTutorial.html>

Año 2012

Este documento no pretende ser una traducción exacta de la fuente original, sino que debe verse como una guía cuyo objetivo es facilitar a los hablantes hispanos los primeros pasos en el uso de esta herramienta. Existe en la red una amplia oferta de tutoriales sobre esta temática, por lo cual espero poder aportar una opción más a aquellos que estén comenzando su aprendizaje.

Copyright (C) 2012 Matias Gerard.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Este tutorial presenta los conceptos fundamentales acerca de los depuradores simbólicos. Aunque se explicará en particular el uso básico de Winpdb para hacer depuración sobre scripts de Python, los conceptos esbozados son generales para el uso de cualquier depurador. Al finalizar este tutorial, espero haber logrado que se sientan cómodos con el uso de los depuradores simbólicos y que entiendan por qué estas herramientas facilitan y reducen el tiempo de depuración al eliminar la necesidad de tener que modificar constantemente el código durante el proceso.

Contenido:

1. Donde encontrar documentación adicional acerca de Winpdb
2. Primeros pasos en Winpdb con *simple.py*
  - 2.1. La ventana de Winpdb
    - a) Explorador de código
    - b) Consola
    - c) Menú y Controles
    - d) Estado del depurador
  - 2.2. Controles Básicos
    - a) Iniciar
      - Reiniciando la sesión de depuración
    - b) Siguiendo
      - Explorando el Espacio de nombres (Namespace)
      - Terminación del ámbito de la aplicación (Scope Completion)
  - 2.3. Conceptos básicos sobre las pausas (breakpoints)
  - 2.4. Conceptos básicos de la consola
  - 2.5. Resumiendo
3. Profundizando los conceptos con la ayuda de *divisibles.py*
  - 3.1. Filtrado o no filtrado?
  - 3.2. Avanzando por pasos
  - 3.3. Regresando al ámbito de la llamada
  - 3.4. Análisis de excepciones (Exceptional Analyses)
  - 3.5. Pausas condicionales
  - 3.6. Resumiendo
4. Palabras finales
5. Reconocimientos
6. Apéndice A: Código script *simple.py*
7. Apéndice B: Código script *divisibles.py*
8. GNU Free Documentation License

## Donde encontrar documentación adicional acerca de Winpdb

La mayor parte de la documentación para Winpdb está disponible en el sitio web de Winpdb.<sup>1</sup> Pueden acceder en forma sencilla seleccionando la opción “Online Docs” en el menú “Help” de la ventana de Winpdb. Muchos de los comandos de la consola Winpdb/rpdb2 se encuentran documentados en la propia consola. Para acceder a la ayuda sólo deben escribir en la consola de comandos “help”; para obtener ayuda sobre un comando específico deben ingresar “help <command>”.

---

<sup>1</sup> <http://winpdb.org/>

# Primeros pasos en Winpdb con *simple.py*

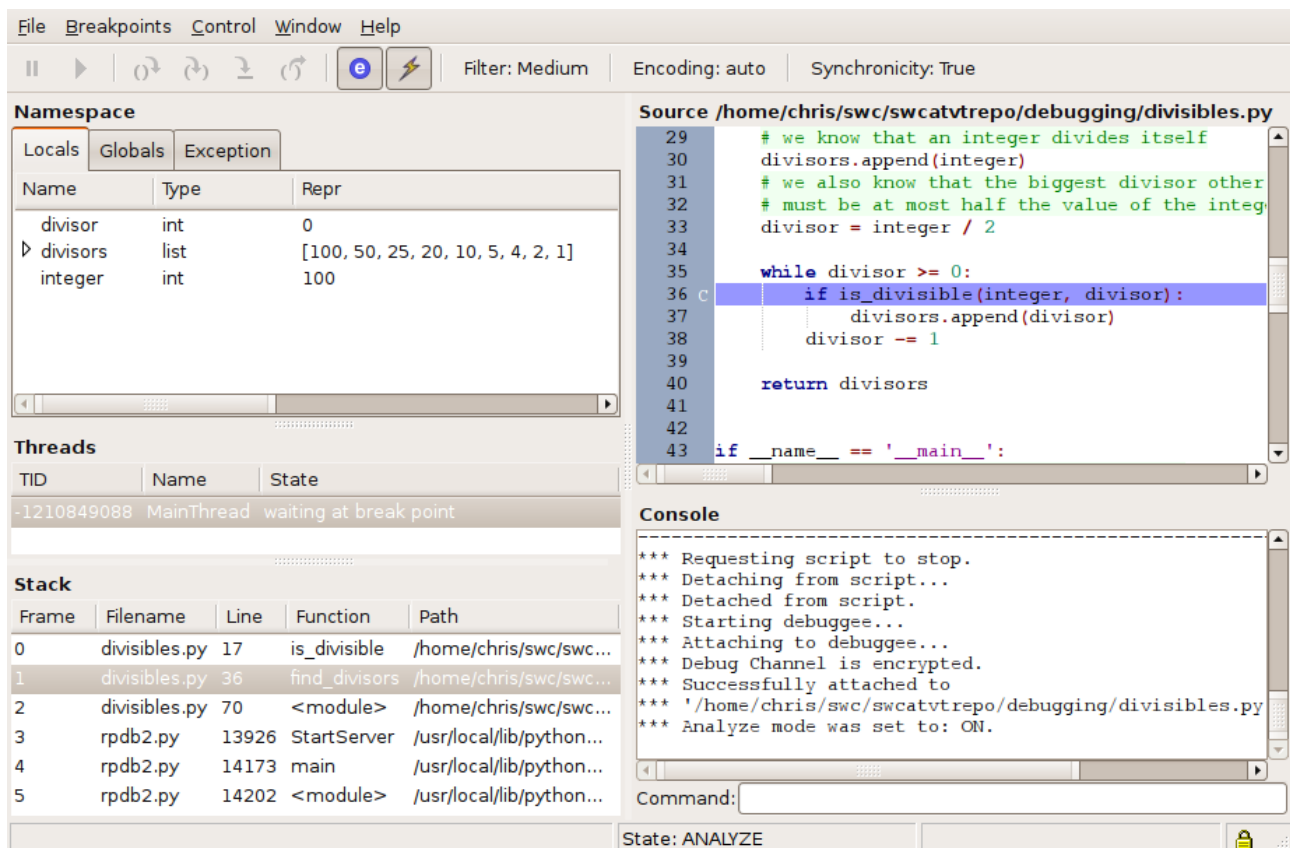
Vamos a comenzar con un script sencillo para familiarizarnos con Winpdb. Para ello descargaremos el archivo *simple.py* de la dirección siguiente:

<http://winpdb.org/cgi-bin/moin.cgi/WinpdbTutorial?action=AttachFile&do=view&target=simple.py>

En el Apéndice A también pueden encontrar una copia completa del código. Abramos el archivo *simple.py* con Winpdb empleando el siguiente comando:

- Para usuarios de Linux/OS X
  - `winpdb simple.py`
- Para usuarios de Windows
  - `%PYTHONHOME%\Scripts\winpdb simple.py`

La ventana de Winpdb se abrirá y se vinculará al script que estamos ejecutando. De ahora en más me referiré a éste código (o el que estemos utilizando) como “código a depurar”. Cuando el depurador se inicie verán una ventana como la que se presenta a continuación, junto a una ventana de terminal que se abrirá en blanco.



Examinemos rápidamente que ocurrió antes de que Winpdb se iniciara. En primer lugar, el depurador (Winpdb) lanzó el script, pero en lugar de permitir que éste se ejecutara hasta su finalización, el depurador detuvo la ejecución del script antes de la primera línea de código pudiera ser ejecutada. En efecto, Winpdb puso en “suspensión animada” a *simple.py*, quedando a la espera de algún comando para continuar. Enseguida veremos como hacer que el script comience a ejecutarse, pero primero exploraremos las diferentes partes que componen la ventana de Winpdb.

## La ventana de Winpdb

La ventana de Winpdb posee diferentes partes, algunas de las cuales nos permitirán interactuar con el código a depurar, y otras nos brindarán información acerca del estado del depurador o del código mismo.

### Explorador de código

Probablemente, la parte más intuitiva en la ventana de Winpdb sea el explorador de código, que aparece en la posición superior derecha de la ventana.

```
Source /home/chris/swc/swcatvtrepo/debugging/
simple.py
1 #!/usr/bin/env python
2
3 c """A simple script."""
4
5 a = 1
6 b = 2
7
8 c = a + b
9
10 print c
11
```

Este cuadro muestra el código fuente del script o módulo que estemos depurando. Más adelante veremos que el navegador de código se actualiza dinámicamente para mostrar el código que se está ejecutando. También veremos como interactuar con el depurador a través del explorador de código; sin embargo, su mayor utilidad será la de ver el código mientras el depurador lo ejecuta.

Pueden notar que el explorador también provee resaltado de sintaxis para mejorar la legibilidad del código, e indica el número de línea en la columna ubicada a la izquierda. La línea que se está analizando se resalta en azul. En este punto es MUY importante remarcar que la línea resaltada será la siguiente instrucción que será ejecutada y que, por lo tanto, aun está pendiente de realizar su función.

El estatus del depurador se encuentra indicado mediante un caracter que se sitúa entre el número de línea actual y el código. Por ahora veremos el caracter “C”. Más adelante discutiremos el significado específico de los caracteres que toma este indicador. También veremos más adelante cómo podemos poner y remover pausas empleando el explorador de código.

### Consola

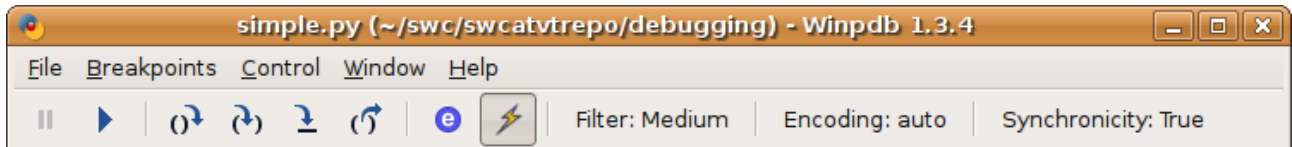
La consola se encuentra justo debajo del cuadro del explorador de código.

```
Console
*** Password has been set to a random password.
*** Spawning debuggee...
*** Attaching to debuggee...
*** Debug Channel is NOT encrypted.
*** Successfully attached to
*** '/home/chris/swc/swcatvtrepo/debugging/simple
*** Debuggee is waiting at break point for further
Command: 
```

Esta nos permite interactuar con el depurador en forma directa, lo que proporciona una gran capacidad de control. También veremos como emplear la consola un poco más adelante.

## Menú y Controles

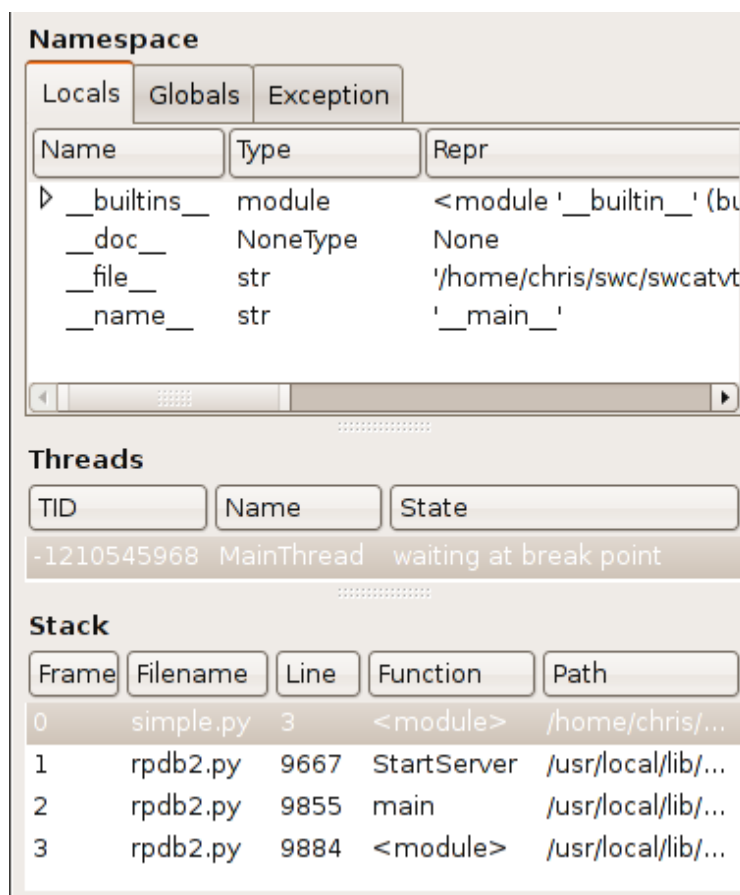
El menú de Winpdb y los controles se encuentran situados en la parte superior de la ventana.



Siéntase libres de explorar las opciones del menú por su cuenta. Un poco más adelante veremos en detalle los controles de los que disponemos.

## Estado del depurador

Estas secciones presentan información del monitoreo del estado del código a depurar y se encuentran ubicadas en el lado izquierdo de la ventana de Winpdb.

The image displays three panels from the Winpdb interface. The top panel is titled "Namespace" and has three tabs: "Locals", "Globals", and "Exception". The "Locals" tab is selected, showing a table with columns "Name", "Type", and "Repr". The table contains entries for "\_\_builtins\_\_" (module), "\_\_doc\_\_" (NoneType), "\_\_file\_\_" (str), and "\_\_name\_\_" (str). The middle panel is titled "Threads" and has columns "TID", "Name", and "State". It shows a single thread with TID "-1210545968", Name "MainThread", and State "waiting at break point". The bottom panel is titled "Stack" and has columns "Frame", "Filename", "Line", "Function", and "Path". It shows four frames, with the top one being the current frame at line 3 of simple.py.

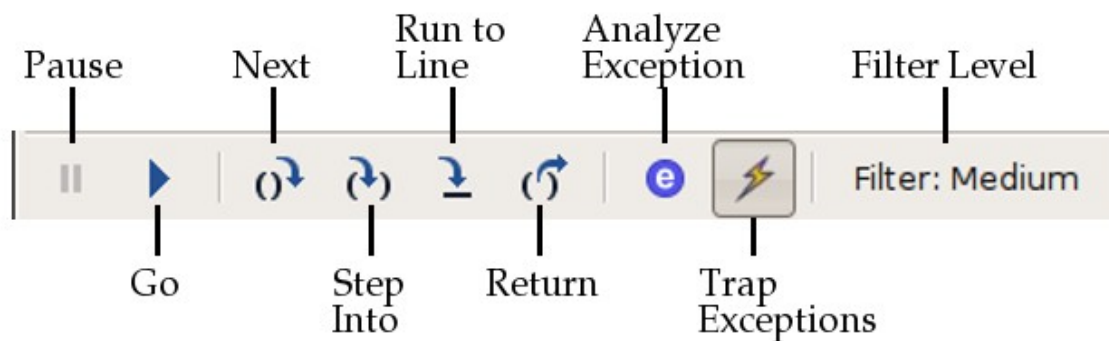
El cuadro **Namespace** (Espacio de Nombres) mantiene un listado en tiempo real de las variables y los valores que éstas toman mientras Winpdb va ejecutando el código a depurar. Para ésto divide los datos en tres categorías: *Locals*, que corresponde a variables de ámbito local; *Globals*, variables de ámbito global; y *Exception*, para examinar los datos asociados a excepciones. El cuadro **Threads** provee información acerca de los hilos que se encuentra ejecutando el código a depurar. No

entraremos en detalles acerca de éste tema ya que escapa a los objetivos perseguidos en este tutorial. La ventana **Stack** (Pila) muestra las llamadas a la pila, ubicando en la parte superior del cuadro la llamada más reciente. *rpdb2.py* siempre ocupará la posición inferior en la pila, ya que Winpdb requiere mantenerla en memoria para realizar la depuración.

- **NOTA:** Aquí debemos hacer una diferencia para mantener la claridad del asunto: cuando hablemos de **stack** (en minúscula), haremos referencia a la estructura de la pila dentro del cuadro de llamadas, que no debe ser confundida con la ventana de llamadas a la Pila dentro de la ventana de Winpdb. Siempre que sea posible me referiré a la ventana de llamadas a la Pila usando mayúscula (Pila) y usaré minúscula cuando me refiera a la estructura de la pila de llamadas.

## Conceptos básicos de control

Bueno, ahora que no hemos orientado un poco con la ventana de Winpdb, vamos a comenzar a aprender como usar el depurador. Para ello utilizaremos los controles de depuración. Vamos a pegarle una mirada de cerca al panel de control, ubicado en la ventana de Menú y Control. Voy a enfocarme en la mitad izquierda del panel de control. Los botones de la mitad derecha quedarán para que los exploren por su cuenta :)



### Go

El botón **Go**, que aparece en el lado izquierdo del panel de control permite que el código a depurar se ejecute hasta que se cumple una de tres situaciones: se encuentra una pausa (breakpoint, que se explicará más adelante) en el código, una excepción o el final del programa. Vamos a ver como Winpdb ejecuta *simple.py*. Si no se encuentran pausas o excepciones durante la ejecución, el código a depurar se ejecutará hasta el final como lo haría si el script se ejecutara en un intérprete de Python.

Comencemos entonces presionando el botón **Go**. El depurador se moverá a través de cada línea y descenderá a través de la pila de llamadas. El explorador de código mostrará a *rpdb2.setbreak()* como la línea actual. La ventana de la Pila sólo mostrará un elemento en la estructura de la pila, que estará identificado como "Frame 0" y que pertenece a *rpdb2.py*. Si miramos en la ventana de la terminal abierta por Winpdb, veremos el valor 3, que pertenece a la sentencia `print c` de *simple.py*.

En este punto, el depurador ha dejado de ejecutar el código. Esto a sido una experiencia un poco "corta", y realmente creo que no tenemos idea de qué es lo que ocurrió. Por lo tanto, emplear el resto del ejercicio para aprender a controlar mejor la ejecución de nuestro código.

### Reiniciando la sesión de depuración

Ahora que ya hemos recorrido cada línea de nuestro código, debemos reiniciar la sesión de



depuración. Podemos hacer esto de dos formas:

1. Forma poco elegante

Cerrar la ventana de Winpdb (debería cerrarse automáticamente la ventana de la terminal), y luego abrir nuevamente el depurador con el comando `winpdb.py`.

2. Forma correcta

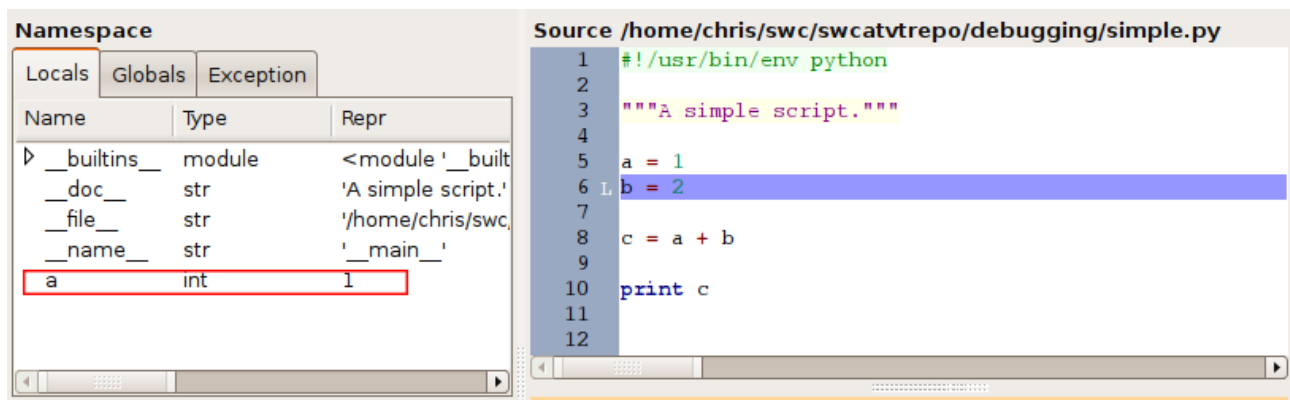
Seleccionar la opción “Restart” en el menú “File”. Esta opción hará que Winpdb reinicie la sesión de depuración.

## Siguiente

El botón **Next** está ubicado en la parte media del panel de control. Éste permite ejecutar la sentencia actual y avanzar directamente a la siguiente sentencia dentro del ámbito donde nos encontremos. Como mencionamos anteriormente, la ventana del navegador posee un caracter que indica el estatus; en este momento debería aparecer con la letra “C”. Éste caracter (“C”) indica que el depurador acaba de ingresar al ámbito local del código. Hasta el momento, el depurador no está realmente listo para ejecutar la línea actual. Vamos a hacer que lo esté. Para ello presionemos **Next**. Veremos que el caracter de estatus cambia de “C” a “L”, donde “L” indica que el depurador ahora si se encuentra preparado para ejecutar la línea actual. Presionemos **Next** nuevamente y ahora veremos que el depurador se mueve a la siguiente línea, que contiene la expresión `a = 1`. Una vez más presionamos **Next** para que el depurador ejecute esta sentencia.

## Explorando el espacio de nombres

El depurador ahora podrá seguir a la variable `a` y brindarnos información acerca del valor que posea la misma. Exploremos ahora el cuadro asociado a los espacios de nombres.<sup>1</sup> Podemos ver que la variable `a` se encuentra en la parte inferior de la lista mostrada en la pestaña *Locals* (podría ser que haga falta que se desplacen hacia abajo para verlo), que es del tipo `int` y que tiene asignado valor `1`.



The screenshot shows the Winpdb interface. On the left, the 'Namespace' panel is open to the 'Locals' tab. It displays a table of local variables:

Name	Type	Repr
▷ __builtins__	module	<module '__builtins__' from ...>
__doc__	str	'A simple script.'
__file__	str	'/home/chris/swc/swcatvtrepo/debugging/simple.py'
__name__	str	'__main__'
a	int	1

On the right, the source code editor shows the following Python code:

```
1 #!/usr/bin/env python
2
3 """A simple script."""
4
5 a = 1
6 L b = 2
7
8 c = a + b
9
10 print c
11
12
```

Presionemos **Next** nuevamente para ejecutar la sentencia `b = 2`. Deberían ver que el cuadro asociado al Espacio de Nombres ahora también contiene a esta variable. Si presionan la pestaña “Globals”, podrán ver que las variables `a` y `b` se encuentra allí. ¿Por qué? Sigamos adelante y presionemos **Next** una vez más. Ahora la variable `c` también aparece en la pestaña *Namespace*. ¿Cuál es su valor?

## Terminación del ámbito de la aplicación

En este punto deberíamos estar posicionados en la sentencia `print c`. Presionemos **Next**. Observen el cambio en el indicador de estado, que pasa de “L” a “R”, indicando que las tareas en el ámbito actual están completas y que el depurador está listo para volver. En este contexto, cuando hable de “volver” me referiré a la forma en que las funciones devuelven los valores cuando han completado su tarea. Más adelante veremos un mejor ejemplo de esto. Por el momento, presionamos **Next** nuevamente para permitir el retorno. Ahora pueden ver como el depurador comienza a moverse hacia atrás en la pila. `rpdb2.py` será el archivo actual de la pila y la función de llamada será “StartServer”. También veremos que el explorador de código se ha actualizado para mostrar el código de la nueva línea actual, que ahora se encuentra en `rpdb2.py`. El caracter de estatus nuevamente es “R”, indicando que está preparado para salir del ámbito de la función.

The screenshot displays a Python debugger interface with several panels:

- Namespace:** Shows local variables. The `f` variable is highlighted as a `frame` object at `0x844237`.
- Threads:** Shows a single thread named `MainThread` with state `waiting at break point`.
- Stack:** Shows the call stack with three frames. The top frame (index 0) is `StartServer` at line 13926 in `rpdb2.py`, which is highlighted with a red box.
- Source:** Shows the code for `rpdb2.py`. Line 13926, `imp.load_source('__main__', _path)`, is highlighted with a red box and has an 'R' status indicator.
- Console:** Shows the output of the program, including messages like `*** NEW: Use CTRL-N for auto completion in the eval and exec.` and `*** Debuggee is waiting at break point for furt`.

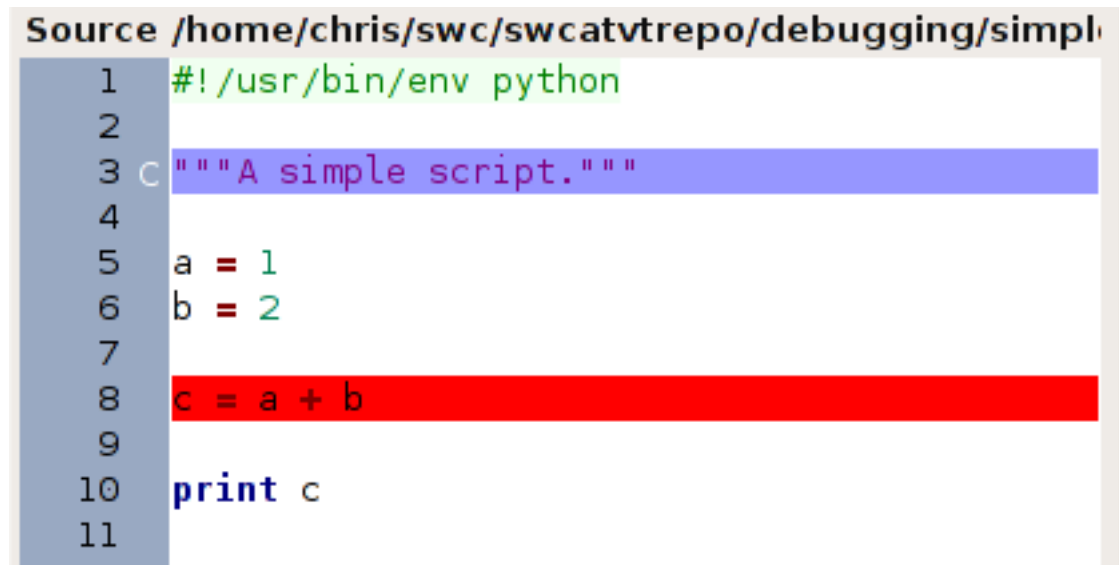
At the bottom, the state is indicated as `State: WAITING AT BREAK PO`.

Presionemos **Next** (si, un poco repetitivo, ¿no?) hasta alcanzar la línea con que contiene `rpdb2.setbreak()`. ¿Les resulta familiar esta línea? Debería, ya que ésta es la misma línea en la que el depurador terminó cuando presionamos el botón **Go** varios minutos atrás. Lo que hemos hecho es decirle al depurador que avance paso a paso ejecutando cada línea del código de `simple.py`. Frecuentemente sólo estaremos interesados en ejecutar una pequeña porción del código a depurar, pero permitiendo que el resto del código se ejecute normalmente. En la próxima sección veremos como hacer esto utilizando las pausas.

## Conceptos básicos sobre pausas

Una pausa (breakpoint) es una instrucción que le indica al depurador que debe pausar la ejecución del código cuando se alcanza esa línea. Qué mejor forma de explicar esto que poniendo nuestra primera pausa!! Supongamos que nos gustaría examinar los valores de `a` y `b` para determinar cual podría ser el valor resultante para `c`. Por lo tanto, vamos a poner una pausa en la línea `c = a + b`, lo que significa que `c` será definido en esta línea pero que aun no se ha calculado.

Reinicien la sesión de depuración si aun no lo han hecho. Dentro de la ventana del explorador de código, ubiquen el puntero del ratón entre el número de línea y la sentencia `c = a + b` y realicen un clic con el botón izquierdo. La línea debería quedar resaltada en rojo indicando que hemos definido una pausa en esta línea.



```
Source /home/chris/swc/swcatvtrepo/debugging/simpli
1  #!/usr/bin/env python
2
3  c """A simple script."""
4
5  a = 1
6  b = 2
7
8  c = a + b
9
10 print c
11
```

Con nuestra pausa puesta en su lugar, podemos ejecutar el código directamente hasta este punto, sin la necesidad de tener que hacer clic repetidamente en **Next**. Presionemos entonces el botón **Go**. Deberíamos ver que el código se ejecuta normalmente y se detiene en la línea donde hemos situado la pausa. Aquí podemos inspeccionar los valores de `a` y `b`, hacer una conjetura de cuánto valdrá `c` y presionar **Next** para ejecutar la sentencia y verificar nuestra suposición. Cuando se sientan preparados, presionen **Go** para finalizar con la depuración del código.

Para eliminar la pausa, solamente debemos realizar un clic sobre la línea que la contiene. Veremos entonces que la línea deja de estar resaltada en rojo y pasa a ser de color blanco. Para la mayoría de las pausas, eso es todo!!! ¿Bastante simple, no? De esta forma podemos poner tantas pausas como queramos!! Más adelante veremos como poner pausas un poco más sofisticadas.

## Conceptos básicos acerca de la consola

La consola provee una poderosa herramienta para interactuar con el depurador. De hecho, tiene todos los comandos que se encuentran disponibles en el panel de control y muchos otros que no se encuentran disponibles en éste o en los menús. Comencemos con un ejemplo sencillo: dado que necesitamos reiniciar nuestra sesión de depuración, hacemos un clic dentro de nuestra consola de comandos, escribimos `restart` y presionamos “Enter”. Al igual que si hubiéramos seleccionado la opción “Restart” dentro del menú “File”, ésto reiniciará nuestra sesión de depuración.

De la misma manera podemos ejecutar la línea de código en la que estamos parados ingresando `next` en la consola y presionando “Enter”. Los comando de consola usados más frecuentemente poseen atajos o métodos abreviados. Por ejemplo, el atajo para `next` es `n` (escribir en la consola `n` en lugar de `next`). Para probarlo, ingresen `n` en la consola y presionen “Enter”. Con ésto debería ejecutarse la segunda sentencia.

En este momento deberíamos estar situados en la sentencia `c = a + b`. Echemos un vistazo a un comando muy poderoso: `exec`. El método abreviado para este comando es `x`. Este comando sólo requiere que se le pase una sentencia como argumento, la cual ejecuta dentro del ámbito donde nos encontremos (discutiremos el concepto de ámbito un poco más adelante.) Por ahora, la variable `a`

tiene asignado el valor 1. (¿Dónde deberíamos fijarnos para confirmar esto?) Vamos a cambiar su valor por 3. Para hacer esto, ingresamos la siguiente sentencia en la caja de comandos:

```
exec a = 3
```

Esto asignará a la variable `a` el valor 3. Retomando donde nos quedamos, en deberíamos seguir ubicados en la sentencia `c = a + b`. Presionemos el botón **Next** o ingresemos `next` en la consola para ejecutar esta sentencia. ¿Se mantuvo igual o se modificó el valor de `c`? ¿Por qué lo hizo o no lo hizo? El comando `exec` incluso nos permite crear nuevas variables en el ámbito donde nos encontramos. Intentemos con el siguiente comando:

```
x d = 4
```

¿Se creó una nueva variable `d` a la que se le asignó el valor 4? ¿Cómo podemos comprobar esto? Examinemos ahora otro comando muy poderoso: `eval`. El método abreviado para este comando es `v`. Levemente diferente de `exec`, `eval` toma una expresión sencilla como argumento y devuelve por consola el resultado de la expresión. Intentemos lo siguiente:

```
eval a + b
```

¿Qué número devuelve? Incluso podemos realizar una evaluación empleando la variable `d` creada en nuestra sesión. ¿Qué devuelve la siguiente línea?

```
v b + d
```

**A diferencia de `exec`, `eval` no afecta el estado del código a depurar ya que sólo evalúa la expresión en el contexto del ámbito en el que nos encontramos.** Por ejemplo, probemos el siguiente comando:

```
v a = 1
```

¿Qué ocurre? ¿Toma ahora el valor 1 la variable `a`? ¿Por qué sí o por qué no? (Ayudita: Piensen en la diferencia entre *sentencia* y *expresión*.)

Vamos a aprender un último comando de consola y luego terminaremos nuestro estudio con el script `simple.py`. El comando `jump` nos permite saltar a una posición previa o posterior a la línea actual de código<sup>2</sup> dentro del ámbito en el que nos encontramos. El atajo para éste comando es `j`. Este comando sólo requiere un argumento: un número entero que indique la línea a la cual queremos movernos. Supongamos que nos gustaría hacer que la variable `a` retorne a su valor original. Esto podemos hacerlo saltando hacia atrás hasta volver a la sentencia `a=1`. Ingreseemos lo siguiente en la consola:

```
jump 5
```

Veremos que la línea 5 se convierte en nuestra nueva línea actual, y que el caracter de estatus indicará que el depurador está listo para correr la sentencia. Ejecutemos la sentencia con **Next** o `next`. ¿Cambió el valor de `a`? Ahora, nos gustaría resetear el valor de `c`, el cual depende de `a`, al valor que tenía previamente. Dado que no hemos modificado el valor de `b`, podemos entonces elegir volver hacia atrás a través del código hasta la sentencia que queremos ejecutar nuevamente, que en este caso es: `c = a + b`. Para ello ingresamos en la consola:

```
jump 8
```

Ejecutemos ahora esta línea. ¿Cuál es el valor de `c` ahora?

## Resumiendo

Con esto concluye el trabajo con `simple.py`. En este punto deberíamos estar familiarizados con los

siguientes conceptos:

- Qué información presenta cada cuadro de la ventana de Winpdb y qué nos permite controlar.
- Cómo ejecutar un script completo y cómo ejecutar una única línea.
- Cómo reiniciar una sesión de depuración.
- Cómo poner una pausa simple.
- Cómo manipular variables y evaluar expresiones que se encuentran en la sesión actual.

## Profundizando los conceptos con la ayuda de *divisibles.py*

Vamos ahora a avanzar a un escenario un poco más complejo: un script con una función y un bucle, que nos permitirá explorar algunas propiedades interesantes de los depuradores simbólicos. Para comenzar, primero cierren la ventana de Winpdb si aun no lo han hecho. Nuestro código a depurar ahora será *divisibles.py*. Pueden obtener una copia en la siguiente dirección:

<http://winpdb.org/cgi-bin/moin.cgi/WinpdbTutorial?action=AttachFile&do=view&target=divisibles.py>

En el Apéndice B también pueden encontrar una copia completa del código. Este script toma un entero positivo como argumento e imprime en pantalla que los números enteros son divisores del mismo (Diremos que un entero  $m$  es divisible por un entero  $n$  si el resto  $r$  de la operación  $m/n$  es cero). Bueno, vamos a hacer una prueba:

```
python divisibles.py 100
```

¿Qué resultó? ¿Parece esto un resultado razonable? En otras palabras: ¿Todos esos números son divisores de 100? Si respondieron que “Si”, por favor háganse un favor apagando la computadora y tómense una siesta — creo que en este momento no están en condiciones de depurar nada... :)

Si respondieron “No”: ¿Cuál sería el siguiente paso? Las opciones son:

- “Deberían emplear más números como casos de estudio.”
- “Deberían mirar el código fuente.”
- “Deberían abrir el depurador y averiguar qué está ocurriendo.”
- “Eso está bien pero: ¿Cómo vamos a inflar el elefante con una bomba de pie?”

Éstos son los puntajes de acuerdo a la respuesta que escogieron:

- 30 puntos: Buena idea, más casos nunca hacen mal.
- 50,250 puntos: Es una buena idea, considerando principalmente que el 90% de los errores conocidos pueden ser descubiertos si se realiza una inspección rigurosa del código fuente.<sup>3</sup> Para seguir con el argumento, supongamos que ya han inspeccionado el código y que aun no han encontrado el error.
- 1,380,296 puntos: Van por el camino correcto.
- $-6.0221415 \times 10^{23}$  puntos: ¿Qué se fumaron? Por favor!!! Sólo consuman productos legales mientras leen este tutorial :p

Muy bien, vamos a depurar el script. Como siempre:

- Para usuarios de Linux/OS X:
  - `winpdb divisibles.py 100`

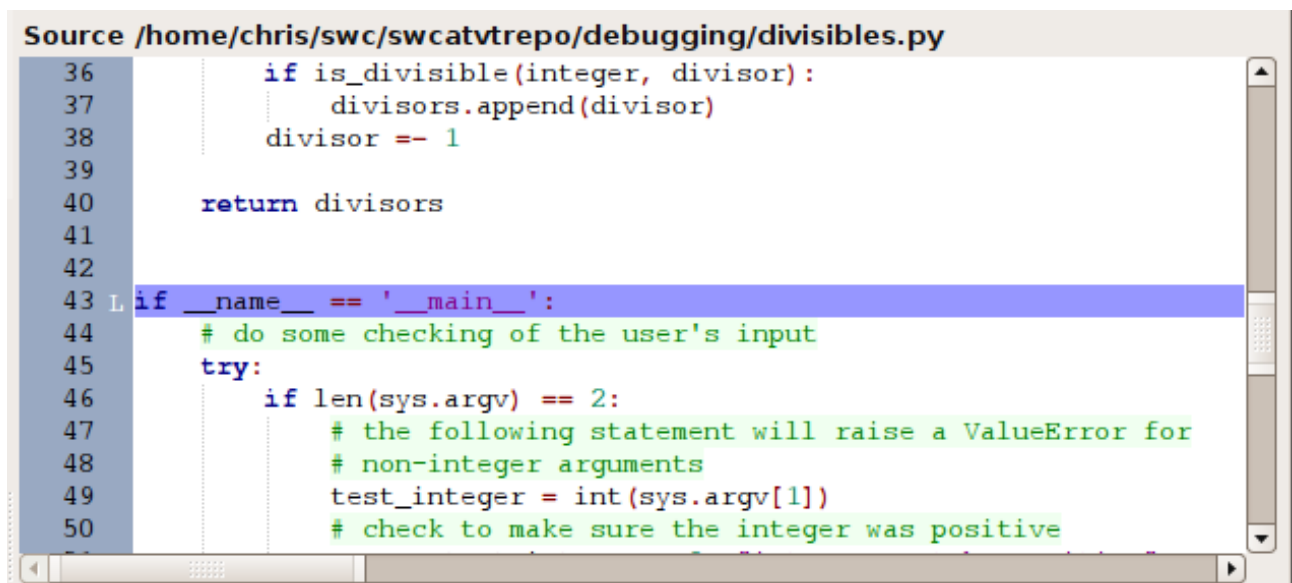
- Para usuarios de windows:
  - %PYTHONHOME%\Scripts\winpdb divisibles.py 100

Si se fijan en el navegador de código verán que este código es un poco más elaborado que el anterior. Sin embargo, al igual que en el caso anterior, podemos recorrerlo sin problemas. Presionemos **Next** para ingresar al código. Luego de la sentencia `import sys` se encuentra la definición de la función `is_divisible`. Presionemos **Next** nuevamente y notaremos que el depurador pasa por sobre el código de la función, debido a que el código no se ejecuta hasta que se realiza la llamada a la función. Presionemos **Next** una vez más para saltarnos la definición de la función `find_divisors`.

En este momento deberíamos encontrarnos encima de la sentencia:

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

la cual verifica si el script a sido llamado directamente desde la consola.



Presionemos **Next** dos veces para ingresar en la sentencia `try` y luego en la sentencia `if len(sys.argv) == 2:`. Ejecuten las sentencias para confirmar que a `test_integer` se le asigna valor 100. Fíjense que si el depurador salta estas sentencias y avanza directamente a la línea 52, es probable que se hayan olvidado de pasar algún argumento a `divisibles.py` al momento de invocar el depurador. Presionemos **Next** en la sentencia `assert`. Noten cómo el depurador avanza hasta finalizar el resto de las sentencias lógicas. Como ejercicio, les recomiendo que exploren el script pasando diferentes valores como argumento y analicen la otra ruta lógica posible. Por el momento, nos enfocaremos en el problemas que tenemos delante nuestro: ¿Por qué no obtenemos todos los divisores para el número entero que hemos ingresado como argumento?

```
Source /home/chris/swc/swcatvrepo/debugging/divisibles.py
62     # alert the user to provide a valid argument
63     print "Please provide a positive integer!"
64     sys.exit(2)
65     # catch the error if too many arguments were provided
66     except SyntaxError:
67         print "Please provide only one argument!"
68         sys.exit(2)
69
70 L   divisors = find_divisors(test_integer)
71     # print the results
72     print "The divisors of %d are:" % test_integer
73     for divisor in divisors:
74         print divisor
75
76
```

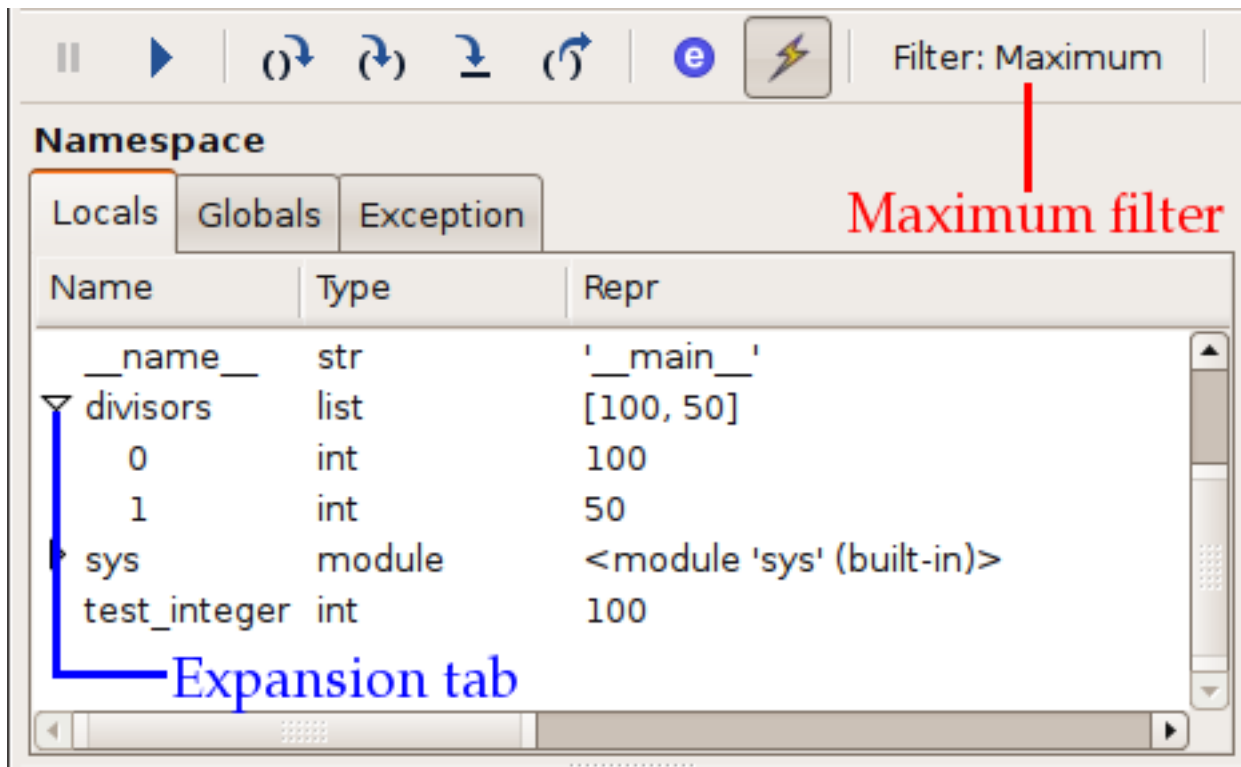
Ahora deberían estar ubicados en la sentencia `divisors=find_divisors(test_integer)`. Presionemos **Next** una vez más para ejecutar esta sentencia, que llama a la función `find_divisors`. Ahora necesitamos inspeccionar el resultado de la llamada a la función, que se almacena en `divisors`.

## ¿Filtrado o no filtrado?

A estas alturas podrán notar que nuestra ventana *Namespace* está comenzando a llenarse. El botón de “Filtro”, el tercero desde la derecha, controla las variables que serán presentadas en esta ventana. Por defecto, el nivel de detalle está activado en nivel Medio. Sin embargo, pueden hacer clic en el botón “Filtro” y seleccionar el nivel máximo de información o desactivarlo completamente. ¿Qué variables son filtradas y cuales no? Pongan el filtro al máximo. ¿Qué ocurre?

Ahora que ya hemos limpiado un poco nuestro Espacio de Nombres, vamos a inspeccionar el resultado que nos interesa: `divisors`. Tengan en cuenta que Winpdb presenta los resultados como una lista entre corchetes bajo la columna *Repr*, de la misma manera que si hubiéramos impreso los resultados de `divisors`, sólo que en lugar de imprimirlos por pantalla podemos verlos en el contexto del programa junto a otros valores mientras el programa se está ejecutando!!! Adiós imprimir los resultados de las sentencias. Bienvenidos Depuradores!!!

Tal vez hayan notado una flecha que aparece al lado de algunas variables en el cuadro *Namespace*. Éstas son las lengüetas de expansión! Hagan clic en la lengüeta de expansión de `divisores`.



Fíjense cómo ahora se presenta la información detallada de cada elemento que figura en la lista. En nuestro caso, la lista de los divisores es pequeña. Sin embargo, para grandes estructuras esto puede ser muy útil. Hagan clic nuevamente en la pestaña de la expansión para contraer la información.

Bueno, hemos encontrado que los resultados de nuestra llamada a `find_divisors` dio un decepcionante conjunto de dos resultados. Por lo tanto, podemos limitar el ámbito de nuestro error de código a esa función o a las llamadas que se hicieron a partir de la misma. Sin embargo, noten que aunque la función sea llamada y se recoja el valor de retorno cuando hacen clic en **Next**, en realidad no hemos visto el código en la función que ejecutamos. Sabemos que el error se encuentra en alguna parte por allí, pero... ¿Cómo lo encontramos? Ingreseemos el comando `step`.

## Avanzando por pasos

Dado que hemos ejecutamos nuestro script más allá del punto que queríamos para inspeccionarlo, vamos a reiniciar nuestra sesión de depuración. Pueden hacerlo desde el menú o la consola, como prefieran. En este punto, dependiendo de cuánto café hayan tomado, puede ser que se estén preguntando, “¿Qué pasó con el argumento 100 que le pasamos al script cuando arrancamos el depurador? ¿Está todavía ahí o desapareció?” Para responder a esa pregunta, continúen la ejecución del script hasta que se ejecute la sentencia `import sys`, que se encarga de cargar el módulo del sistema, y luego examinen `sys.argv`. ¿Cuál es el resultado? (Ayudita: deberían mirar el valor de `sys.argv[1]`). En lugar de utilizar `eval` (aunque es más rápido), podrían desactivar el botón “Filtro” y ampliar las lengüetas de `sys` y `argv` para inspeccionar sus valores. ¿Ya se sienten tranquilos de que se conservó el argumento? Muy bien, procedamos entonces a la parte del programa que más nos interesa. Piensen en las herramientas que hemos aprendido usando `simple.py`, que nos permiten ejecutar gran parte del código y detenernos en la línea que nos interesa. ¿Se acuerdan? Si no es así, tómense un momento y releen lo que vimos con ese ejemplo porque será muy importante tener claros los conceptos para lo que sigue.

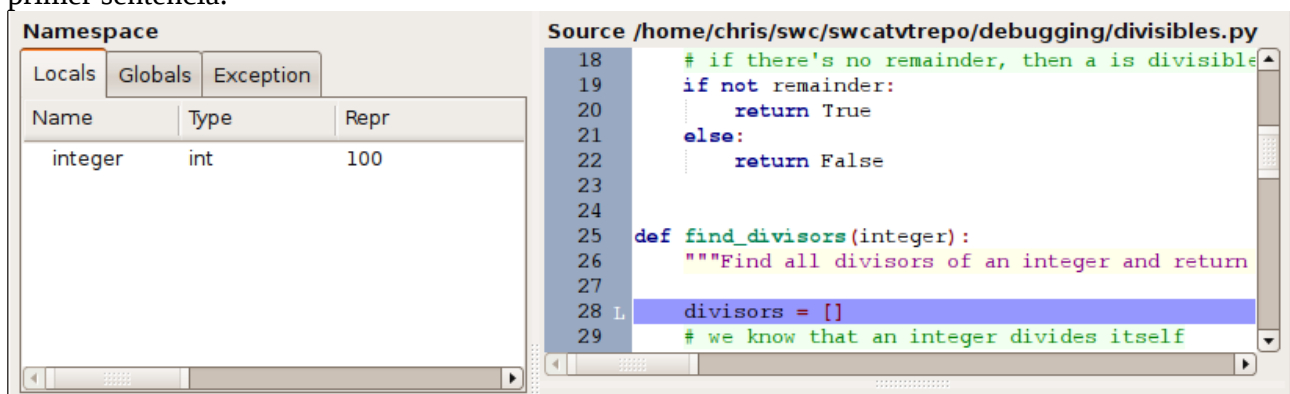


Vamos pues a introducir una pausa en la línea donde se realiza la llamada a `find_divisors`. Luego, avancemos hasta llegar a ella presionando el botón **Go** o ingresando el comando `go` (cuyo atajo es `g`) en la consola.

```
66     except SyntaxError:
67         print "Please provide only one argument!"
68         sys.exit(2)
69
70     divisors = find_divisors(test_integer)
71     # print the results
72     print "The divisors of %d are:" % test_integer
73     for divisor in divisors:
74         print divisor
```

Ahora necesitamos alguna manera de ingresar en la función `find_divisors` cuando se realiza la llamada. El comando `step` nos permite realizar ésta tarea. El botón **Step**, o el comando `step` (cuyo atajo correspondiente es `s`), nos permite ingresar en el código asociado al ámbito de la llamada (por ejemplo, una función o método) y navegar a través de sus líneas. Esto funciona incluso para las llamadas al código en otros módulos.

Ahora presionemos **Step** (o ingresen el comando) para acceder a la función `find_divisors`. Fíjense que el caracter de estatus es "C". ¿Recuerdan qué significa? Estamos a punto de ingresar en un nuevo ámbito, el de la función `find_divisors`. Presten atención también a la ventana *Namespace*. Échenle un vistazo a la pestaña *Locals* y compárenla con la pestaña *Globals*. ¿Son diferentes? ¿Qué aparece ahora en *Locals* que antes no lo hacía? ¿Qué permaneció igual y qué cambió? Presionemos **Next** o **Step** para terminar de entrar al nuevo ámbito y procedamos a la primer sentencia.



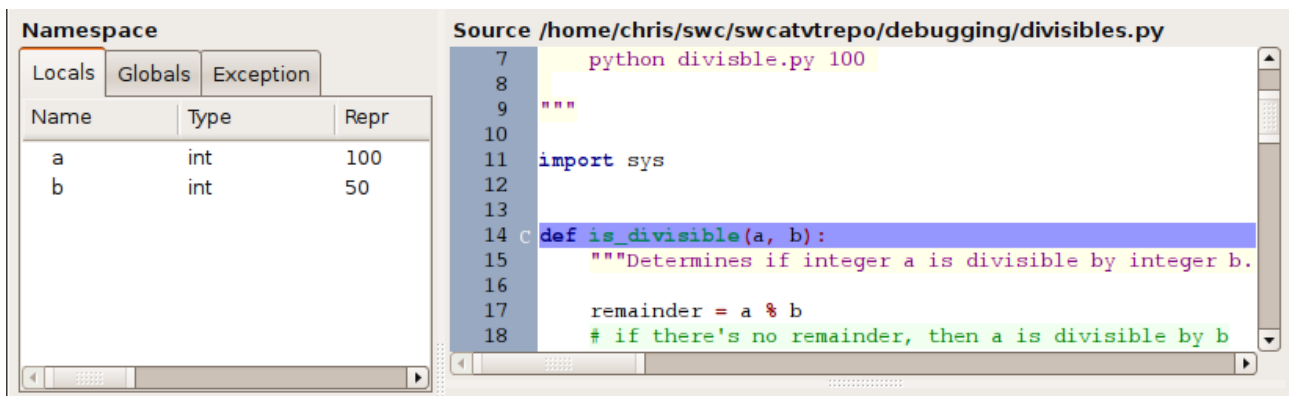
The screenshot shows a Python IDE interface. On the left, the **Namespace** window is open, with the **Locals** tab selected. It displays a table with one entry: `integer` of type `int` with a value of `100`. On the right, the **Source** window shows the code for `find_divisors.py`. The code is as follows:

```
18     # if there's no remainder, then a is divisible
19     if not remainder:
20         return True
21     else:
22         return False
23
24
25 def find_divisors(integer):
26     """Find all divisors of an integer and return
27
28 L  divisors = []
29     # we know that an integer divides itself
```

Presionemos **Next** o **Step** nuevamente para ejecutar la primer sentencia de `find_divisors`. ¿Han notado que una nueva variable a ingresado en el cuadro de espacio de nombres, en la pestaña *Locals*? ¿Este cambio también se refleja en la pestaña *Globals*? ¿Por qué si o por qué no? Ejecuten las dos declaraciones siguientes y observen los cambios en *Namespace*. En este momento deberían encontrarse dentro del bucle `while` de la función `find_divisors`. Presionemos **Next** o **Step** para ingresar en el bucle. Fíjense que ambos comandos permiten ingresar en el bucle, pero únicamente **Step** realiza un cambio de ámbito (igual que cómo lo hace la llamada a una función). En este momento deberían estar ubicados en la sentencia:

```
if is_divisible(integer, divisor):
```

Deben notar que se está realizando una llamada a la función `is_divisible` pasando como argumentos las variables `integer` y `divisor`, y que su resultado es evaluado por la sentencia `if`. Ingrese en esta función. Tomen nota de los valores de `integer` y `divisor` y luego presionen **Step**. Deberían ser transportados a la declaración de la función `is_divisible`, y el estatus debería ser “C”.



Pueden ver que los valores de las dos variables pasadas como argumentos, `a` y `b` ya se encuentran definidas, incluso a pesar de que todavía aun no se ha terminado de ingresar en el nuevo ámbito. Comparen estos valores con los valores de `integer` y `divisor` que fueron pasados a la función. Sigamos avanzando y ejecutemos la sentencia `remainder = a % b` (¿Qué hace el operador “%” en Python? Si no lo recuerdan, les recomiendo mirar el tutorial [Python para todos](#) o la [Documentación de Python](#)) ¿Cuál es el valor de `remainder`? Hagan una conjetura acerca de cual de las dos ramas del programa se ejecutará y luego ejecuten las sentencias. Cuando alcancen la sentencia de retorno, encontrarán que el caracter indicador de estatus será una “R”. ¿Recuerdan que significa esto? Procedamos a regresar al ámbito de la llamada. Fíjense que las variables seguidas en la pestaña *Locals* son actualizadas al salir del ámbito anterior. En este punto deberían encontrarse dentro de la sentencia `if is_divisible:`

```
divisors.append(divisor)
```

¿Qué creen que indica esta sentencia? (Pista: Piensen en el nombre de la función llamada, `is_divisible`) Ejecuten esta sentencia. ¿Se dieron cuenta que se actualizó un valor en *Namespace*? Para asegurarnos que vamos bien, deberíamos estar ubicados en la sentencia:

```
divisor -= 1
```

¿Qué piensan que hace esta sentencia? Ok, vamos a ejecutarla y ver que hace. Deberían haber vuelto nuevamente a la parte superior del bucle `while`. Pero... Un momento!!! Miren muy detenidamente el *Namespace*. ¿Notan algo sospechoso? Deberían darse cuenta de en qué línea se está produciendo el error. De hecho, puede ser que hayan descubierto el verdadero origen del error en esa línea. ¿Aun no lo ven? Échenle otro vistazo. Piensen en algún cambio inesperado que le ocurrió a “`divisor`”... ¿Lo tienen?

Claramente, el autor del script intenta disminuir en una unidad el valor de `divisor`. Sin embargo, el resultado de la sentencia no es ese precisamente. En Python, el espacio en blanco entre operadores es generalmente opcional. Esto significa que las expresiones

```
divisor -= 1 ; divisor=-1 ; divisor -=1 ; divisor = -1
```

son equivalentes debido a que en realidad el operador es el signo “=”. De esta manera, la sentencia está asignando el valor “-1” a `divisor`, en lugar de disminuir su valor en una unidad. El orden correcto de los símbolos que debería haber sido “-” en primer lugar, seguido de “=”. Así, el

operador adecuado que deberíamos haber empleado es “- =”. Abramos “*divisibles.py*” en nuestra IDE o editor de texto favorito y modifiquemos la línea:

```
divisor -= 1
```

por

```
divisor -= 1
```

Correcto!! Felicitaciones depuradores natos!!! Pero, antes de dejarnos llevar por nuestras habilidades de élite de depuradores de código, vamos a asegurarnos de que esto haya arreglado el código. No hace falta que cerremos Winpdb para probar el script editado. Podemos reiniciar la sesión de depuración y Winpdb se encargará de ejecutar nuevamente nuestro código editado. Incluso conservará las pausas que hayamos definido o los argumentos pasados por línea de comandos! ¿Qué les parece?

Reinicien la sesión de depuración, confirmen que aun mantienen la pausa que han definido con anterioridad llamando a la función `find_divisors` y presionen **Go**, para avanzar hasta llegar a este punto. Presionen **Step** e ingresen al ámbito de `find_divisors`. Usen el botón **Step** para proceder en forma secuencial dentro del bucle `while` y de la función `is_divisible`, que forma parte de la sentencia `if is_divisible(integer, divisor):`.

Den un paso para asegurarse que todo está en orden. La sentencia debería devolver `True`, ya que 50 es divisor de 100. Regresen al ámbito de `find_divisors` y avancen hasta que alcancen nuestra sentencia modificada `divisor -= 1`. Avancen un paso más para ejecutar esta sentencia. ¿Disminuye su valor en forma correcta?

Ahora avancen una iteración más en el bucle `while`. Ingresen a la siguiente llamada a `is_divisible`. Sabemos que 100 no es divisible por 49, así que avancemos a través de toda la secuencia para asegurarnos que retorna `False`. Salgan del ámbito de `find_divisors` y confirmen que **NO** se agrega el número 49 a la lista de divisores y que `divisors` disminuye correctamente en 1. Avancen hasta la siguiente llamada a `is_divisible`, la cual ahora debería ser llamada con los valores 100 y 48.

## Regresando al ámbito de la llamada

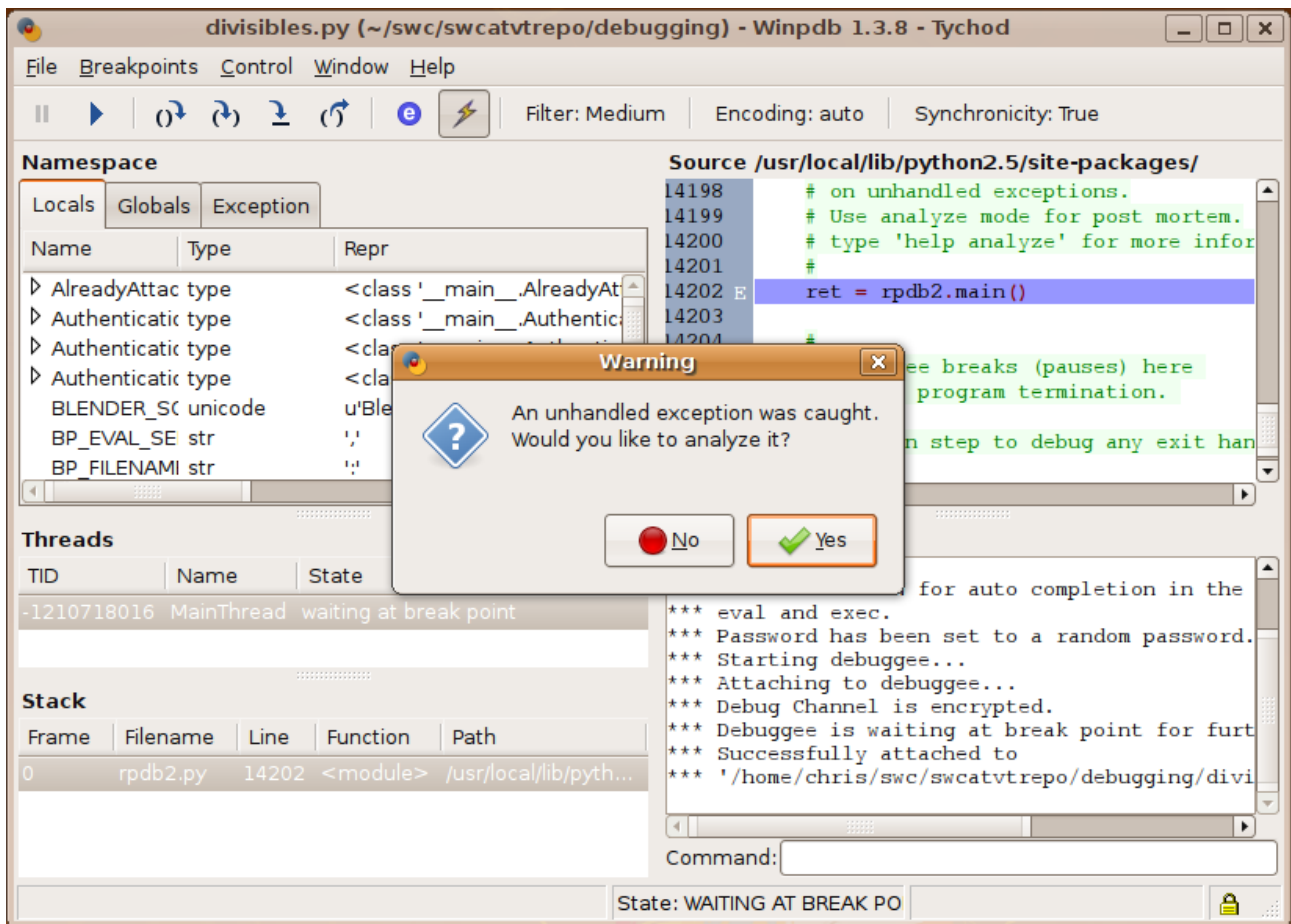
En este momento deberían encontrarse en la declaración `is_divisible` con el estatus “C”. A estas alturas usar **Step** se ha vuelto obsoleto. Sería más conveniente usar **Next** para, al menos, ejecutar completamente y sin supervisión el código de `is_divisible`, ya que estamos bastante seguros de que está trabajando correctamente.

```
Source /home/chris/swc/swcatvtrepo/debugging/divisibles.py
13
14 def is_divisible(a, b):
15     """Determines if integer a is divisible by int
16
17     remainder = a % b
18     # if there's no remainder, then a is divisible
19     if not remainder:
20         return True
21     else:
22 R   return False
23
24
25 def find_divisors(integer):
```

Muy bien, ya lo hemos ejecutado y ahora debemos salir del ámbito de `is_divisible`. Podrían hacer clic en **Step** o **Next** un par de veces para llegar allí, pero... qué pasaría si la función fuera muy larga? Este método sería inviable!! Debería haber una mejor manera, y afortunadamente la hay!!! El botón **Return**, o su comando de consola equivalente `return` (cuyo atajo es `r`), ejecuta el código completo del ámbito actual y nos regresa al ámbito donde se realizó la llamada. Sigán adelante haciendo clic en **Return** o ingresando el comando.

Deberían ser llevados a la sentencia `return False`, con el caracter de estatus "C". Presionen **Next** para regresar al ámbito de la llamada de `find_divisors`.

Continúen presionando **Next** para recorrer algunas veces más el ciclo `while`, pero evitando salir del código de `is_divisible`. Una vez que estén seguros de que el código funciona apropiadamente, presionen **Go** para permitir que el código se ejecute completamente y se finalice la ejecución. Muy bien, si no lo hicieron, presionen **Go**.



Oh no!! ¿Qué ocurrió?!? Encontramos una excepción que detuvo la ejecución del código!!!

## Análisis de excepciones

Amablemente, Winpdb nos informa que capturó una excepción no controlada durante el proceso de depuración, y nos pregunta si nos gustaría analizarla. Sigamos adelante y presionemos “Yes”, lo que le indicará a Winpdb que ingrese en el modo de “Análisis de Excepciones”. Esto se evidenciará de dos maneras. Por un lado, se activará el botón **Analyze Exception**, que se encuentra del lado derecho del panel de control. Por otra parte, en la consola se observará el mensaje “Analyze mode was set to ON (“se ha activado el modo de Análisis”). Este modo nos lleva directamente a la sentencia donde se produjo y capturó la excepción.

Namespace		
Locals	Globals	Exception
Name	Type	Repr
traceback	traceback	<traceback object at 0xb7c0cbe4>
▶ type	classobj	<class exceptions.ZeroDivisionError at 0xb7df7a1c>
value	str	'integer division or modulo by zero'

Ahora vemos que alguna línea en nuestro código arroja una excepción del tipo `ZeroDivisionError`, lo que significa que la sentencia intenta realizar una división por cero o intenta realizar una operación módulo\* usando el cero. Ahora que sabemos de qué tipo de excepción se trata, vamos a analizar que estaba ocurriendo con `divisibles.py` cuando se produjo `ZeroDivisionError`. Para ello, utilizaremos los datos de la pila de llamadas de manera de averiguar cuál fue la causa de la excepción.

Stack				
Frame	Filename	Line	Function	Path
0	divisibles.py	17	is_divisible	/home/chri...
1	divisibles.py	36	find_divisors	/home/chri...
2	divisibles.py	70	?	/home/chri...
3	rpdb2.py	9287	StartServer	/usr/local/li...
4	rpdb2.py	9474	main	/usr/local/li...
5	rndb2.py	9503	?	/usr/local/li...

**Winpdb ordena las llamadas dentro de la pila en orden descendente, ubicando la llamada más reciente en la parte superior y debajo de ella las demás, de la más reciente a la que lo es menos.** Sin embargo, en términos del número en el cuadro asociado a la pila, la llamada más reciente tendrá el menor número (0) y las demás incrementarán su valor a medida que nos alejemos de la misma.

También presten atención al hecho que para cada elemento de la pila se muestra el nombre del archivo al que está asociado. Esto resulta de mucha utilidad ya que, de un vistazo podemos ver que sólo debemos inspeccionar tres elementos de la pila para encontrar el evento que causó la excepción `ZeroDivisionError` (dado que sólo los tres primeros elementos pertenecen a `divisibles.py`).

Si hacemos clic en el primer elemento del cuadro `Stack`, veremos que la línea infractora se encuentra en la función `is_divisible` en la operación módulo para calcular `remainder`.

\* [http://en.wikipedia.org/wiki/Modulo\\_operation](http://en.wikipedia.org/wiki/Modulo_operation)

The screenshot shows a Python IDE with the following components:

- Namespaces:**

Name	Type	Repr
a	int	100
b	int	0
- Stack:**

Frame	Filename	Line	Function	Path
0	divisibles.py	17	is_divisible	/home/chris/swc/swc...
1	divisibles.py	36	find_divisors	/home/chris/swc/swc...
2	divisibles.py	70	<module>	/home/chris/swc/swc...
3	rpdb2.py	13926	StartServer	/usr/local/lib/python...
4	rpdb2.py	14173	main	/usr/local/lib/python...
5	rpdb2.py	14202	<module>	/usr/local/lib/python...
- Source (/home/chris/swc/swcatvtrepo/debugging/divisibles.py):**

```

13
14 def is_divisible(a, b):
15     """Determines if integer a is divisible by in
16
17 E remainder = a % b
18     # if there's no remainder, then a is divisibl
19     if not remainder:
20         return True
21     else:
22         return False
23
24
25 def find_divisors(integer):
26     """Find all divisors of an integer and return
27

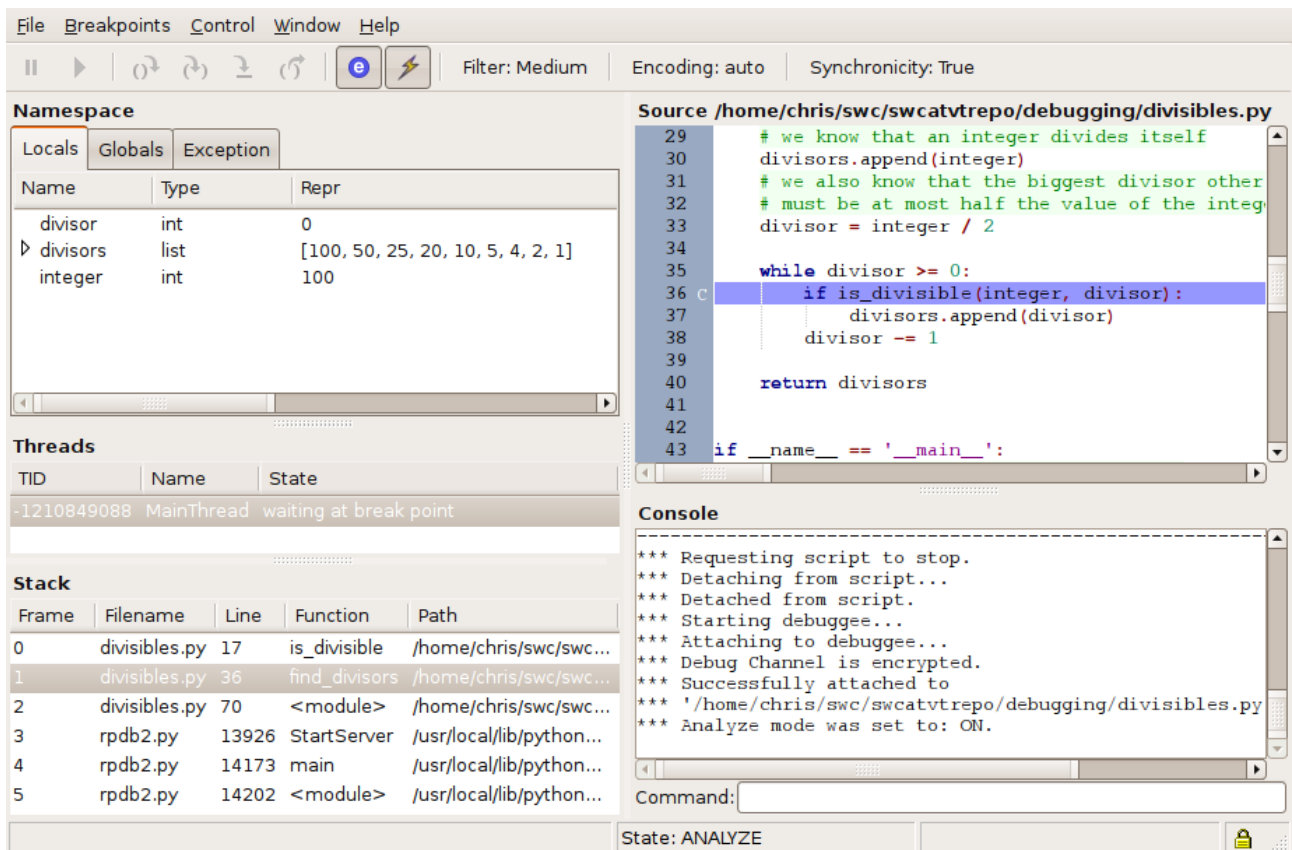
```
- Console:**

```

*** Requesting script to stop.
*** Detaching from script...
*** Detached from script.
*** Starting debuggee...
*** Attaching to debuggee...
*** Debug Channel is encrypted.
*** Successfully attached to
*** '/home/chris/swc/swcatvtrepo/debugging/divisibles.py
*** Analyze mode was set to: ON.

```

El caracter de estatus “E” al lado de la sentencia nos indica que en ese punto se originó la excepción `ZeroDivisionError`. Esto significa que el “0” debió originarse en un elemento más antiguo en la pila de llamada (con un valor de “Frame” más grande). Entonces, ahora que ya conocemos el “Qué” y el “Dónde”, sólo nos falta descubrir el “Por qué”. Podemos ver que en la pestaña *Locals* del *Namespace*, la variable `b` vale 0, y resulta obvio que emplear 0 en una operación de cálculo de módulo produce un error. Pero... ¿Por qué fue que `b` tomó ese valor en primer lugar? Recuerden que `b` es pasado como argumento cuando se realiza la llamada a `is_divisible`. Echemos un vistazo a `find_divisors`. Hagamos clic en la ventana de la Pila para ver la pila de llamadas y analizar el estado justo antes de la excepción.



Podemos ver la sentencia que llama a `is_divisible` desde dentro de `find_divisors` y el pasaje de los argumento `integer` y `divisor`. También podemos ver en la pestaña *Locals*, dentro de la ventana *Namespace*, que a `divisor` se le asignó valor 0 cuando fue pasado como argumento de `is_divisible`. Esto resulta extraño debido a que ya hemos arreglado el problema que habíamos encontrado con la variable `divisor` (cuando corregimos el problema de los símbolos invertidos). Sin embargo, sabemos que el error debe estar en alguna parte dentro de `find_divisors`, ya que la variable “`divisor`” es local a este ámbito. Tal vez esto fue sólo una casualidad. Vamos a ver si podemos repetir el error. Quizás con esto podamos encontrar el error subyacente.

## Pausas condicionales

Seguro que estamos comenzando a perder la paciencia y la concentración con todo esto de la depuración... Necesitamos un método para llegar hasta el error en forma rápida y sencilla. Esto significa que necesitamos una pausa, pero sospechamos que nuestro error se encuentra oculto dentro de un bucle, con lo cual la ejecución se detendrá en cada iteración. Entonces lo que necesitamos realmente es un tipo de pausa que sólo detenga la ejecución del código cuando nosotros queramos que ocurra. Aquí es donde aparecen las pausas condicionales.

Una pausa condicional permanece inactiva (esto significa que el código continua su ejecución aunque se alcanza la línea que contiene la pausa) hasta que alguna condición pre-especificada es alcanzada. En ese punto, la pausa se vuelve activa y la ejecución se detiene en la línea donde la pausa se encuentra definida. Dado que especificar una condición de detención implica definir una expresión, únicamente podremos definir estas pausas elaboradas mediante el uso de la consola. El comando `bp` nos permite especificar pausas; para definir una pausa condicional sólo debemos ingresar por consola



```
bp <line number>, <expression>
```

substituyendo <line\_number> y <expression> por la línea donde deseamos introducir la pausa y la expresión que deseamos evaluar para que se produzca la pausa en el script.

En nuestro caso, queremos ver que pasa con `find_divisors` cuando `divisor` toma valor 1. ¿Continuará disminuyendo hasta llegar a 0? Vamos a definir una pausa condicional para detener el bucle `while` cuando `divisor` toma valor 1, de modo que a partir de éste punto podamos movernos en forma manual a través del código. Si no lo has hecho, reinicia la sesión de depuración. A continuación, debemos poner nuestra pausa condicional. Buscamos el número de línea de la sentencia del bucle `while`:

```
while divisor <= 0:
```

y ponemos una pausa condicional para el instante en que `divisor` alcanza el valor 1.

```
bp 35, divisor == 1
```

The screenshot shows a Python IDE window titled "Source /home/chris/swc/swcatvtrepo/debugging/divisibles.py". The code is as follows:

```
31 # we also know that the biggest divisor other
32 # must be at most half the value of the integer
33 divisor = integer / 2
34
35 while divisor >= 0:
36     if is_divisible(integer, divisor):
37         divisors.append(divisor)
38         divisor -= 1
39
40 return divisors
41
42
```

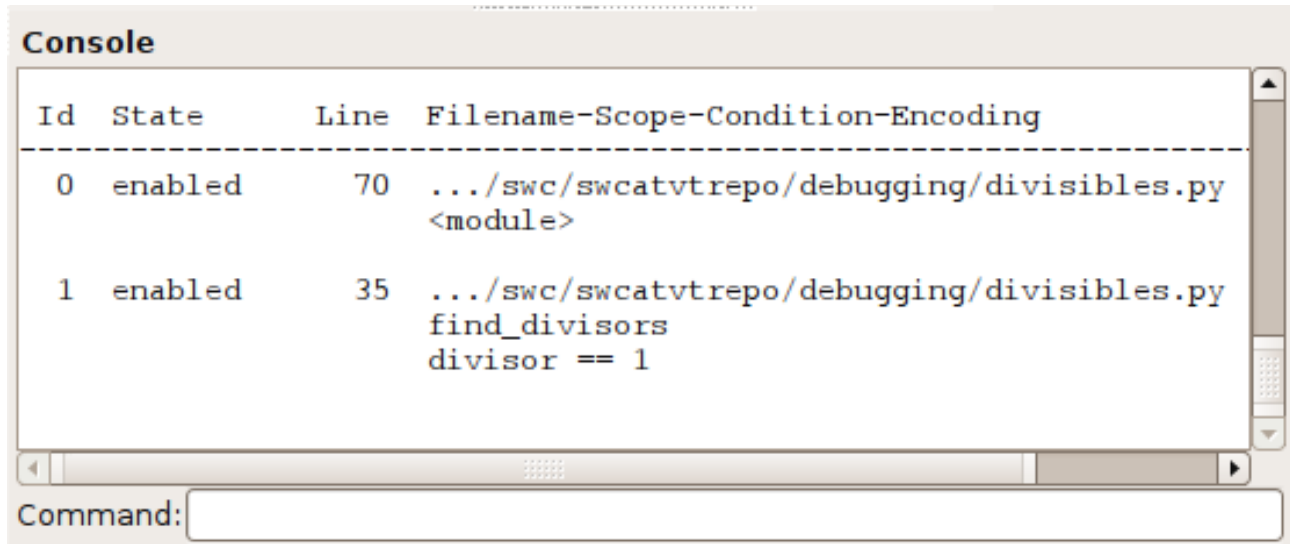
The line 35 is highlighted in red, indicating a breakpoint. Below the source editor is a "Console" window with the following output:

```
*** Requesting script to stop.
*** Detaching from script...
*** Detached from script.
*** Starting debuggee...
*** Attaching to debuggee...
*** Debug Channel is encrypted.
*** Successfully attached to
*** '/home/chris/swc/swcatvtrepo/debugging/divisibles.py'

> bp 35, divisor == 1
```

At the bottom of the IDE, there is a "Command:" input field.

Mientras estamos en ello, vamos a quitar la otra pausa que pusimos varios re inicios atrás. Podemos hacer esto desplazándonos hasta el mismo y realizando un clic sobre la línea que contiene la pausa o introduciendo un comando en la consola. Para hacer esto último, primero debemos obtener el ID (código identificador de la pausa) usando el comando `bl` (que corresponde a “breackpoints list”), que enumera los puntos donde donde se sitúan las pausas.



The screenshot shows a 'Console' window with a table of breakpoints. The table has columns for Id, State, Line, and Filename-Scope-Condition-Encoding. There are two rows of data.

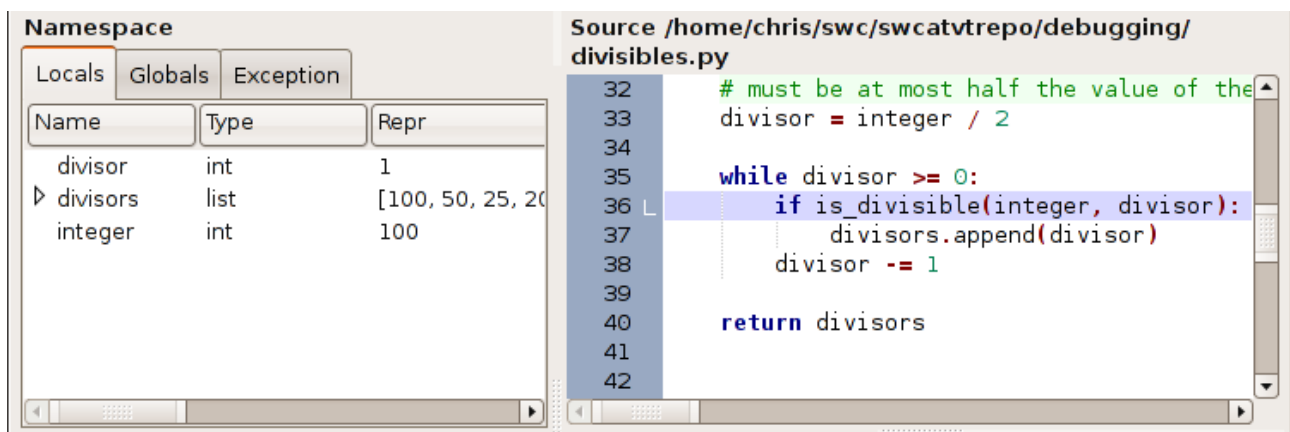
Id	State	Line	Filename-Scope-Condition-Encoding
0	enabled	70	.../swc/swcatvtrepo/debugging/divisibles.py <module>
1	enabled	35	.../swc/swcatvtrepo/debugging/divisibles.py find_divisors divisor == 1

Below the table is a 'Command:' input field.

Podemos ver que nuestra primera pausa, posicionada en la línea 70, tiene un ID igual a 0. Podemos eliminar esta pausa con el comando `bc`, que toma como argumento el ID de la pausa que se desea remover. Para eliminar nuestra primera pausa, utilizamos el comando:

```
bc 0
```

Ahora que hemos puesto y sacado convenientemente nuestras pausas, ejecutemos el programa. Para ello, presionamos el botón **Go**. En este momento deberíamos encontrarnos en la sentencia `if` que se encuentra inmediatamente a continuación luego de ingresar al bucle `while`, y la variable `divisor` debería valer 1.



The screenshot shows two panels. The left panel is titled 'Namespace' and has tabs for 'Locals', 'Globals', and 'Exception'. The 'Locals' tab is active, showing a table of local variables.

Name	Type	Repr
divisor	int	1
divisors	list	[100, 50, 25, 20]
integer	int	100

The right panel is titled 'Source /home/chris/swc/swcatvtrepo/debugging/divisibles.py' and shows the source code. Line 36 is highlighted, which is the start of an `if` statement inside a `while` loop.

```
32 # must be at most half the value of the
33 divisor = integer / 2
34
35 while divisor >= 0:
36     if is_divisible(integer, divisor):
37         divisors.append(divisor)
38         divisor -= 1
39
40     return divisors
41
42
```

Avancemos de a un paso por vez, para asegurarnos de que todo está funcionando como esperamos que lo haga. Como es de esperar, podemos ver que `is_divisible` devuelve `True` ya que 100 es divisible por 1. Si continuamos veremos que `divisor` disminuye su valor en 1 y regresa de nuevo al inicio del bucle `while`, lo cual debería llevar al bucle a su fin, pero... un momento!!!

Presionemos **Step** para ejecutar la evaluación de la condición del bucle `while`. Veremos que el bucle se ejecuta una vez de más!! Se suponía que iba a parar una vez que `divisor` alcanzara el valor 0, pero en cambio este continúa!!! ¿Por qué cuando “divisor” pasa igual la evaluación cuando toma valor 0? Esperen... ¿También vieron eso?... Miremos de cerca la sentencia

```
while divisor >= 0
```

Creímos que la evaluación sería `False` una vez que `divisor` alcanzara el valor 0, ya que sabemos que 1 será siempre el último número entero que puede dividir a un número entero positivo. Después de todo, no podemos dividir un entero por 0.<sup>4</sup> Sin embargo, lo que pensamos que vimos y lo que en realidad estaba allí fueron dos cosas diferentes... En realidad la evaluación será cierta para cualquier número mayor o igual que cero.<sup>5</sup>

Vamos a corregir este pequeño traspié abriendo el script con nuestro editor de texto favorito y vamos a reemplazar la sentencia

```
while divisor >= 0
```

por

```
while divisor > 0
```

Esto demuestra un fenómeno muy importante que ocurre en el desarrollo de software: los errores tienden a estar agrupados.<sup>6</sup> Sin embargo, vamos a suponer que en este punto ya hemos encontrado todos los errores. Si por algún motivo *divisibles.py* no funciona, sería mejor que apagáramos la computadora porque seguro se nos ha secado el cerebro :) Sólo por diversión vamos a comprobar si efectivamente hemos localizado y corregido todos los errores de nuestro script.

Reiniciar la sesión de depuración una última vez, eliminar todas las pausas y presionar **Go**. Veremos entonces que el programa se ejecuta exitosamente hasta el final. Si miramos la terminal, veremos la siguiente salida:

```
Los divisores de 100 son:  
100  
50  
25  
20  
10  
5  
4  
2  
1
```

Bueno, podemos respirar tranquilos!!!

## Resumiendo

Después de haber trabajado con *divisibles.py*, deberían estar familiarizados con los siguientes conceptos:

- Depurar códigos que presentan errores.
- Filtrar el espacio de nombres.
- Entrar y salir del ámbito de una función.
- Analizar excepciones.
- Leer y navegar la pila.

- Poner y sacar pausas empleando la consola.
- Poner pausas basadas en condiciones.

## Palabras finales

Si han llegado a este punto y han completado exitosamente este tutorial, déjenme decirles que han utilizado y aprovechado todas las herramientas que puede proporcionarles un depurador simbólico. Ojalá que hayan encontrado fácil y entretenido el aprendizaje de los conceptos a través del uso de Winpdb. Espero que puedan tomar los conceptos que hemos presentado aquí y puedan aplicarlos a cualquier otro idioma que ofrezca soporte para depuración.

## Reconocimiento

Éste tutorial fue escrito originalmente por [ChrisLasher](#) como parte de su curso [Software Carpentry at Virginia Tech](#).

1. Por cierto, si perdieron la referencia del título, intenten una búsqueda para "[More Cowbell](#)"... (1)
2. [ChrisLasher](#) descubrió un error importante en Python mismo que impedía saltar de la primera sentencia al ámbito local, y se la presentó ante el [seguidor de problemas oficial de Python](#), y el error fue arreglado en el [reporte de error correspondiente](#). Aquellos que usen Python 2.5+ no van a experimentar este problema. (2)
3. Glass, R. 2000. Facts and Fallacies of Software Engineering. Addison-Wesley. pp. 104-105. (3)
4. Bueno, *nosotros no podemos*, pero [Chuck Norris si puede](#). (4)
5. Por cierto, para todos aquellos que se dieron cuenta de ésto hace una hora, gracias por mantener la boca cerrada y han continuado sólo por sus ganas de aprender. (5)
6. Glass, R. 2000. Facts and Fallacies of Software Engineering. Addison-Wesley. pp. 136-137. (6)

## Apéndice A

*simple.py*

```
#!/usr/bin/env python
"""A simple script."""
a = 1
b = 2
c = a + b
print c
```

## Apéndice B

*divisibles.py*

```
#!/usr/bin/env python
"""
Takes a positive integer as an argument and prints all of its divisors.
example usage:
    python divisibles.py 100
"""
import sys
def is_divisible(a, b):
    """Determines if integer a is divisible by integer b."""

    remainder = a % b
    # if there's no remainder, then a is divisible by b
    if not remainder:
        return True
    else:
        return False
def find_divisors(integer):
    """Find all divisors of an integer and return them as a list."""
    divisors = []
    # we know that an integer divides itself
    divisors.append(integer)
    # we also know that the biggest divisor other than the integer itself
    # must be at most half the value of the integer (think about it)
    divisor = integer / 2
    while divisor >= 0:
        if is_divisible(integer, divisor):
            divisors.append(divisor)
        divisor -= 1
    return divisors
if __name__ == '__main__':
    # do some checking of the user's input
    try:
        if len(sys.argv) == 2:
            # the following statement will raise a ValueError for
            # non-integer arguments
            test_integer = int(sys.argv[1])
            # check to make sure the integer was positive
            if test_integer <= 0:
                raise ValueError("integer must be positive")
        elif len(sys.argv) == 1:
            # print the usage if there are no arguments and exit
            print __doc__
            sys.exit(0)
        else:
            # alert the user they did not provide the correct number of
            # arguments
            raise ValueError("too many arguments")
    # catch the errors raised if sys.argv[1] is not a positive integer
    except ValueError, e:
        # alert the user to provide a valid argument
        print "Error: please provide one positive integer as an argument."
        sys.exit(2)
    divisors = find_divisors(test_integer)
    # print the results
    print "The divisors of %d are:" % test_integer
    for divisor in divisors:
```

print divisor

Copyright (C) 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.  
<<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

## 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

## 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a



section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or

control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

### 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition.

Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material.

If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

### 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section.  
You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice.

These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same

cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number.

Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

## 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit.

When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form.

Otherwise they must appear on printed covers that bracket the whole aggregate.

## 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4.

Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

## 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number.

If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

## 11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

### ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.