

Eclipse Python Debugging Tutorial

For Introduction to Programming Using Python By Y. Daniel Liang

This supplement covers the following topics:

- Set Breakpoints
- Start the Debugger
- Control Program Execution
- Examine Variables

0 Introduction

This tutorial introduces how to debug a Python program using Eclipse. The Python debugger utility is integrated in Eclipse. You can pinpoint bugs in your program with the help of the Eclipse debugger without leaving the IDE. The Eclipse debugger enables you to set breakpoints and execute programs line by line. As your program executes, you can watch the values stored in variables, observe which functions are being called, and know what events have occurred in the program.

To demonstrate debugging, let us use Listing 2.7, `ComputeLoan.py`, to demonstrate debugging. Create a new program named `ShowCurrentTime.py` in the `pybook` project.

1 Set Breakpoints

You can execute a program line by line to trace it, but this is time-consuming if you are debugging a large program. Often, you know that some parts of the program work fine. It makes no sense to trace these parts when you only need to trace the lines of code that are likely to have bugs. In cases of this kind, you can use breakpoints.

A *breakpoint* is a stop sign placed on a line of source code that tells the debugger to pause when this line is encountered. The debugger executes every line until it encounters a breakpoint, so you can trace the part of the program at the breakpoint. Using the breakpoint, you can quickly move over the sections you know work correctly and concentrate on the sections causing problems.

There are several ways to set a breakpoint on a line. One quick way is to click the cutter of the line on which you want to put a breakpoint. You will see the line highlighted, as shown in Figure 1. You also can set breakpoints by choosing *Run, Toggle Line Breakpoint*. To remove a breakpoint, simply click the cutter of the line.

As you debug your program, you can set as many breakpoints as you want, and can remove breakpoints at any time during debugging. The project retains the breakpoints you have set when you exit the project. The breakpoints are restored when you reopen it.

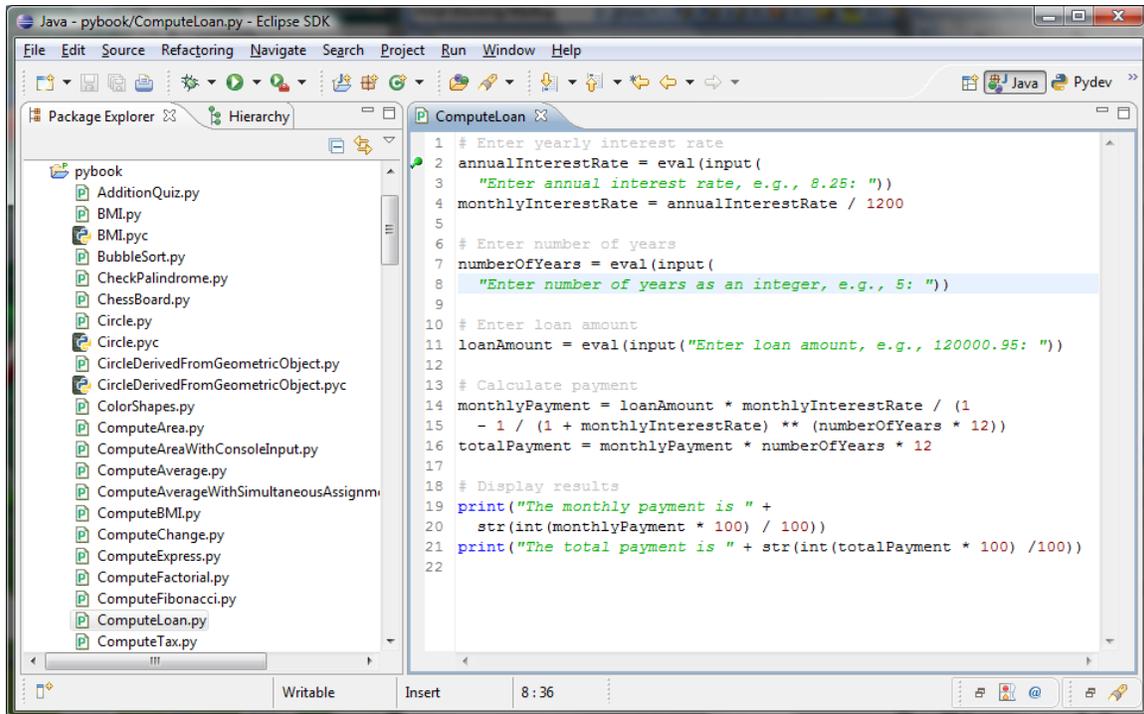


Figure 1

You can set breakpoints in the source code.

2 Start the Debugger

There are several ways to start the debugger. A simple way is shown below:

1. Set a break point at the first statement in the proigram in the Source Editor.
2. Right-click on ComputeLoan.py in the project pane to display a context menu. Choose *Debug As, Python Run* to start debugging. You will first see the Confirm Perspective Switch dialog, as shown in Figure 2. Click Yes to switch to the Debug perspective. The UI for Debug perspective is shown in Figure 3.

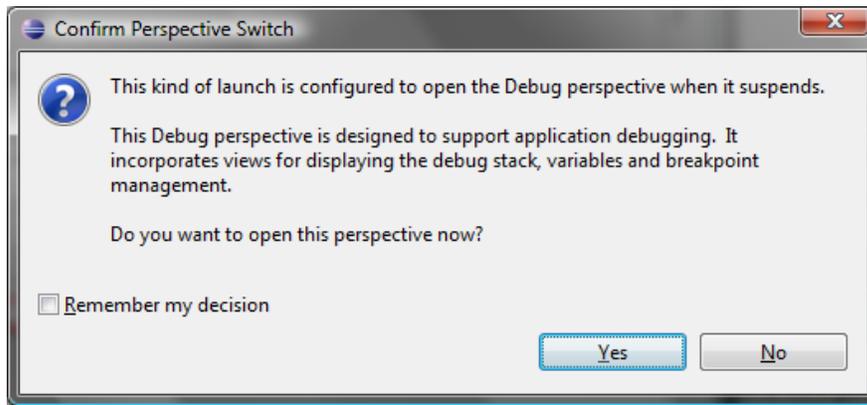


Figure 2

To start debug, Eclipse needs to switch to the Debug perspective.

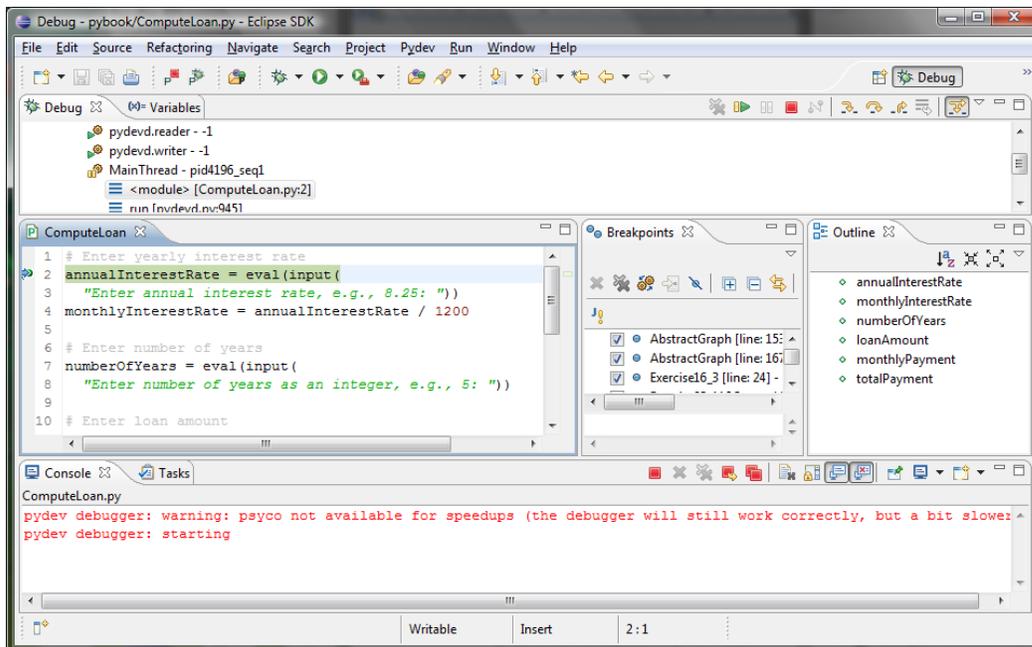


Figure 3

The debugger starts to run ComputeLoan.py.

3 Control Program Execution

The program pauses at the first line in the script. This line, called the *current execution point*, is highlighted in green. The execution point marks the next line of source code to be executed by the debugger.

When the program pauses at the execution point, you can issue debugging commands to control the execution of the program. You also can inspect or modify the values of

variables in the program.

When Eclipse is in the debugging mode, the toolbar buttons for debugging are displayed in the Debug window, as shown in Figure 3. The toolbar button commands also appear in the Run menu (see Figure 4). Here are the commands for controlling program execution:

- **Resume** resumes the execution of a paused program.
- **Suspend** temporarily stops execution of a program.
- **Terminate** ends the current debugging session.
- **Step Into** executes a single statement or steps into a function.
- **Step Over** executes a single statement. If the statement contains a call to a function, the entire function is executed without stepping through it.
- **Step Return** executes all the statements in the current function and returns to its caller.
- **Run to Line** runs the program, starting from the current execution point, and pauses and places the execution point on the line of code containing the cursor, or at a breakpoint.

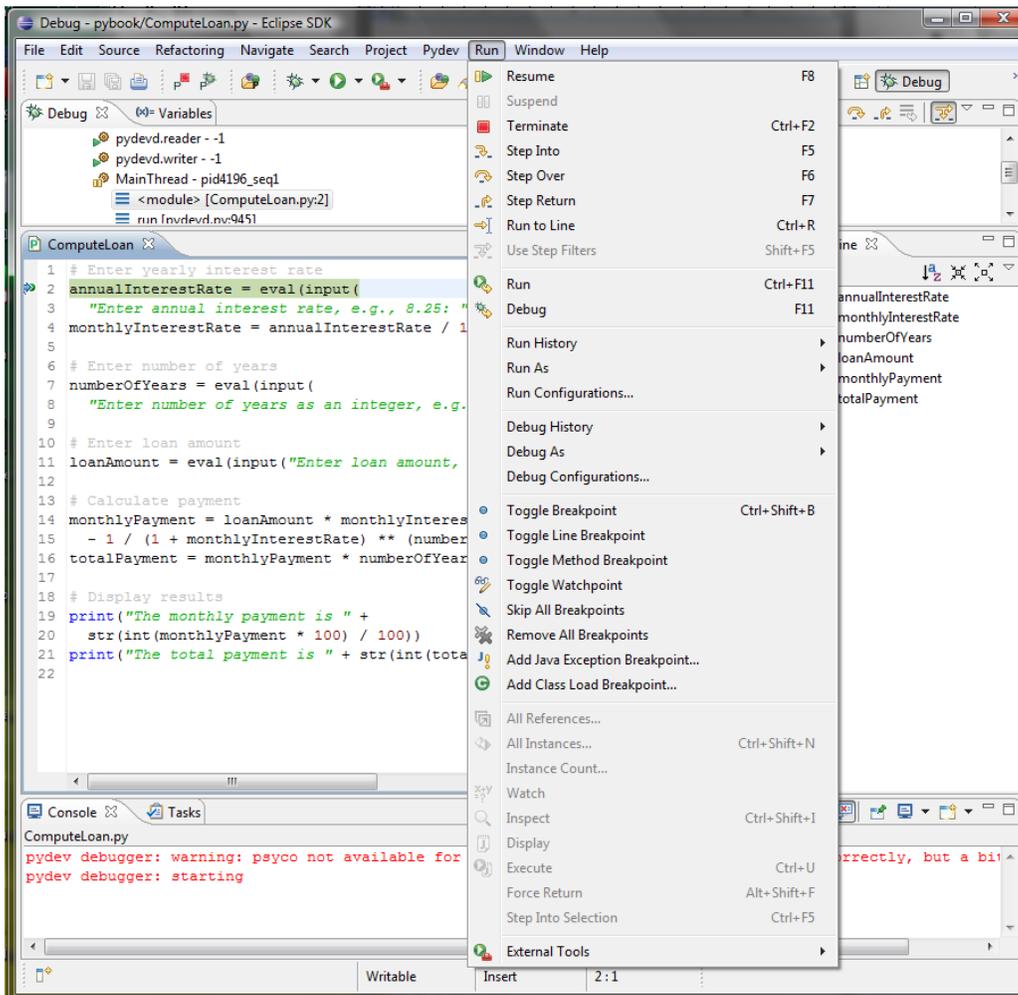


Figure 4

The debugging commands appear under the Debug menu.

4 Examine Variables

Among the most powerful features of an integrated debugger is its capability to examine the values in variables, or the values of the parameters passed in a function call. You also can modify a variable value if you want to try a new value to continue debugging without restarting the program.

To demonstrate it, choose *Run, Step Over* to execute one line in the source code, and you will see the prompting message displayed in the Console, as shown in Figure 5.

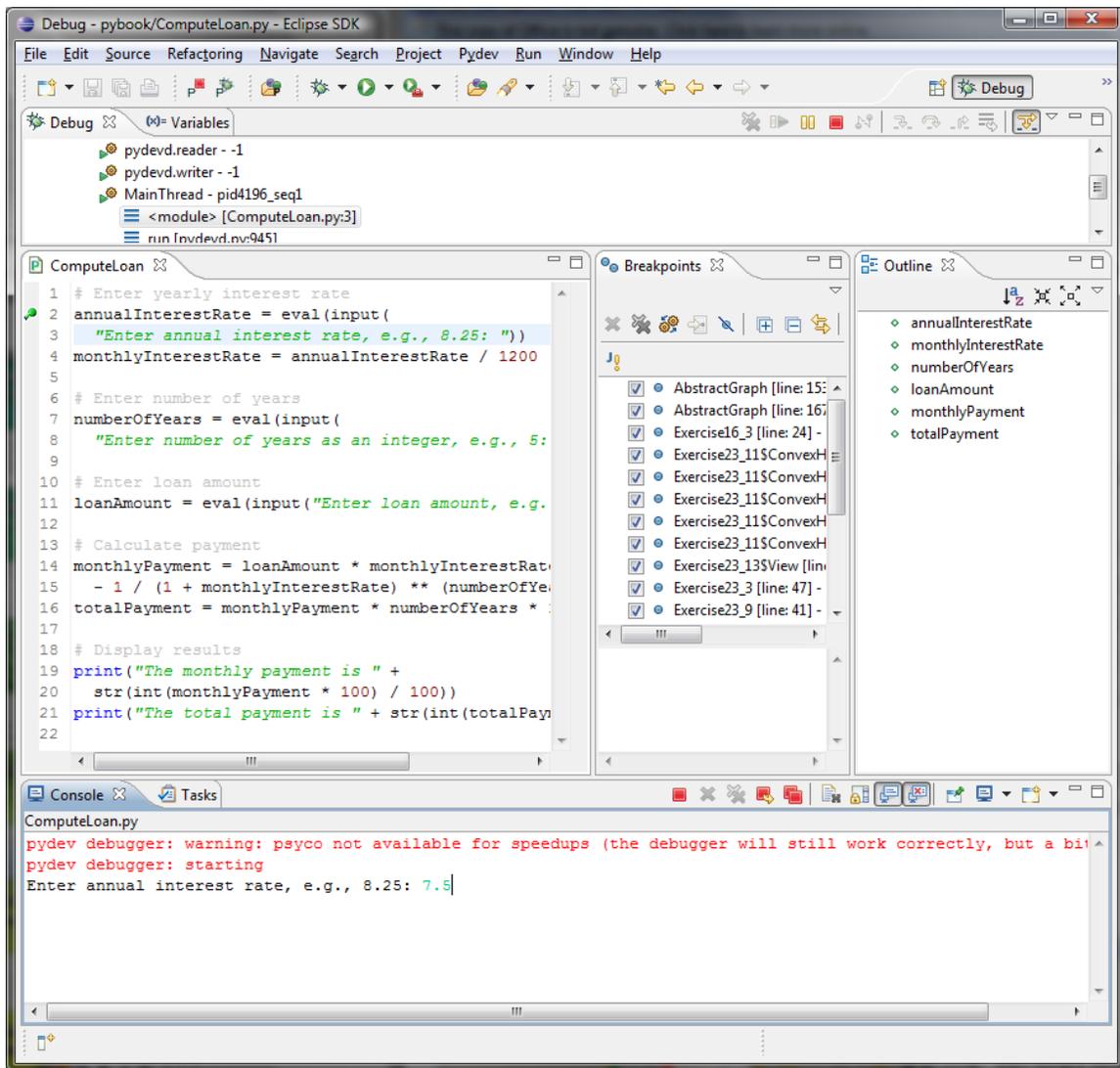


Figure 5

Executing line 2 prompts the user for an input.

Enter a 7.5 and press the Enter key. You will see the current execution point moved to line 4. Now in the Variables tab, you will see the variable `annualInterestRate` with value 7.5, as shown in Figure 6.

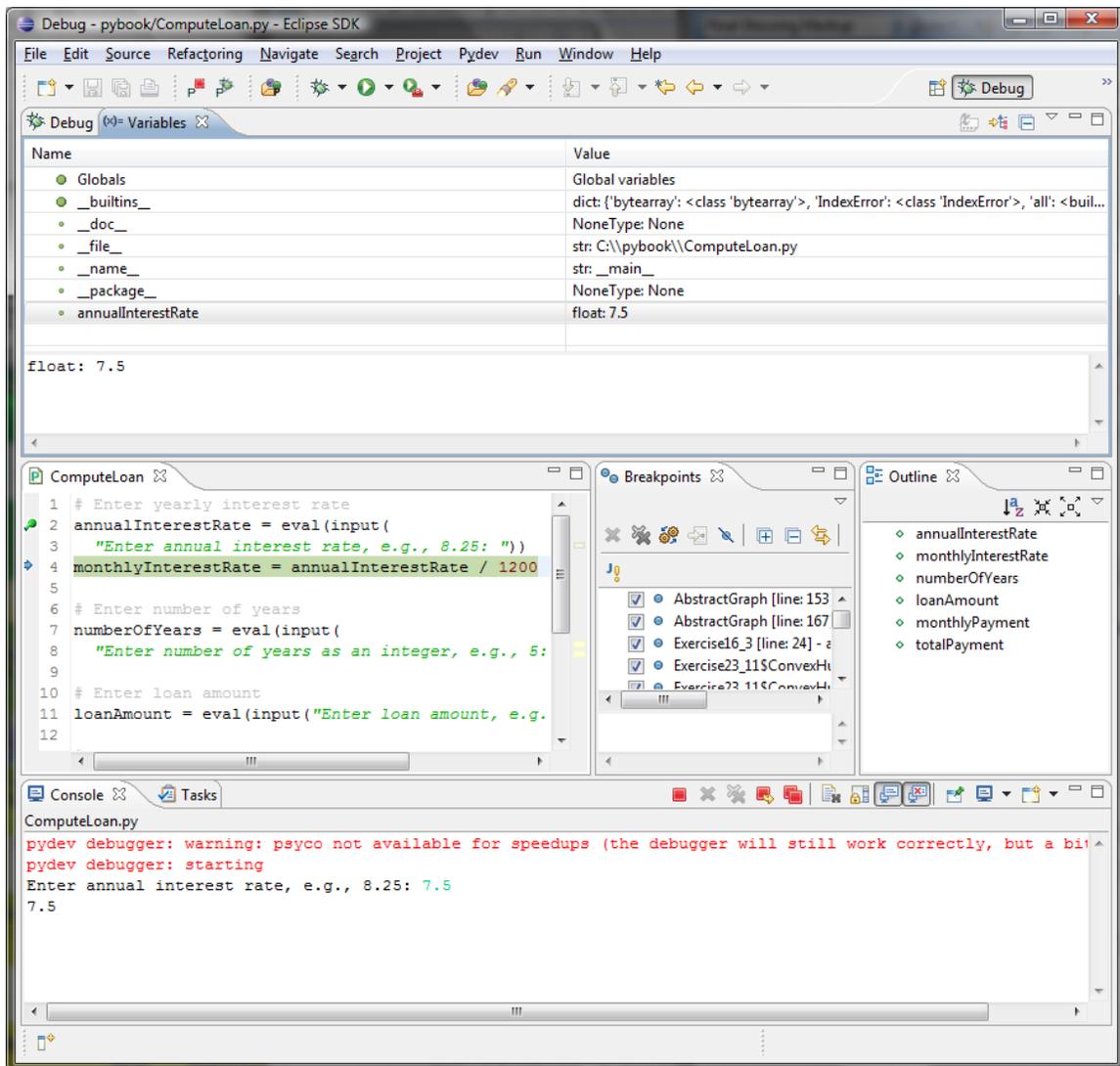


Figure 6

You can examine the variable value from the Variables tab.

Continue to Step Over and enter a value for number of years and enter a value for loan amount, as shown in Figure 7. Now you will see these values in the Variable tab, as shown in Figure 8.

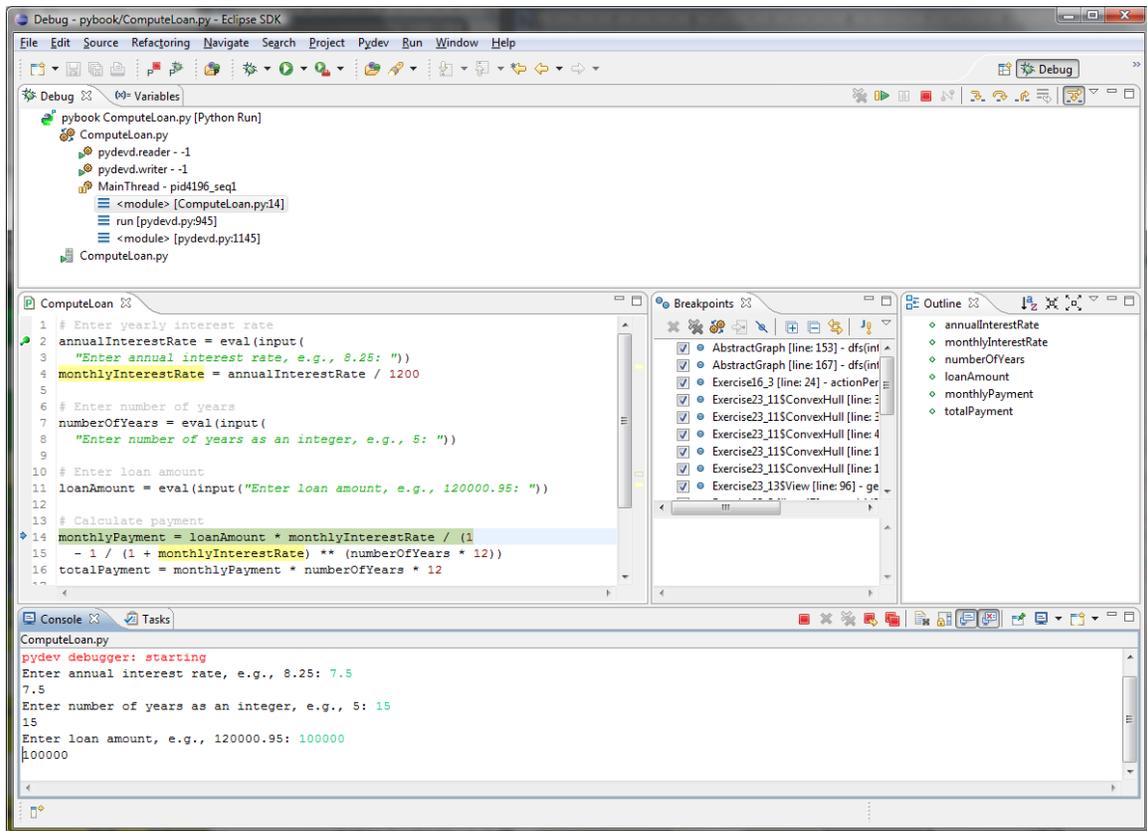


Figure 7

The input values are entered from the Console tab.

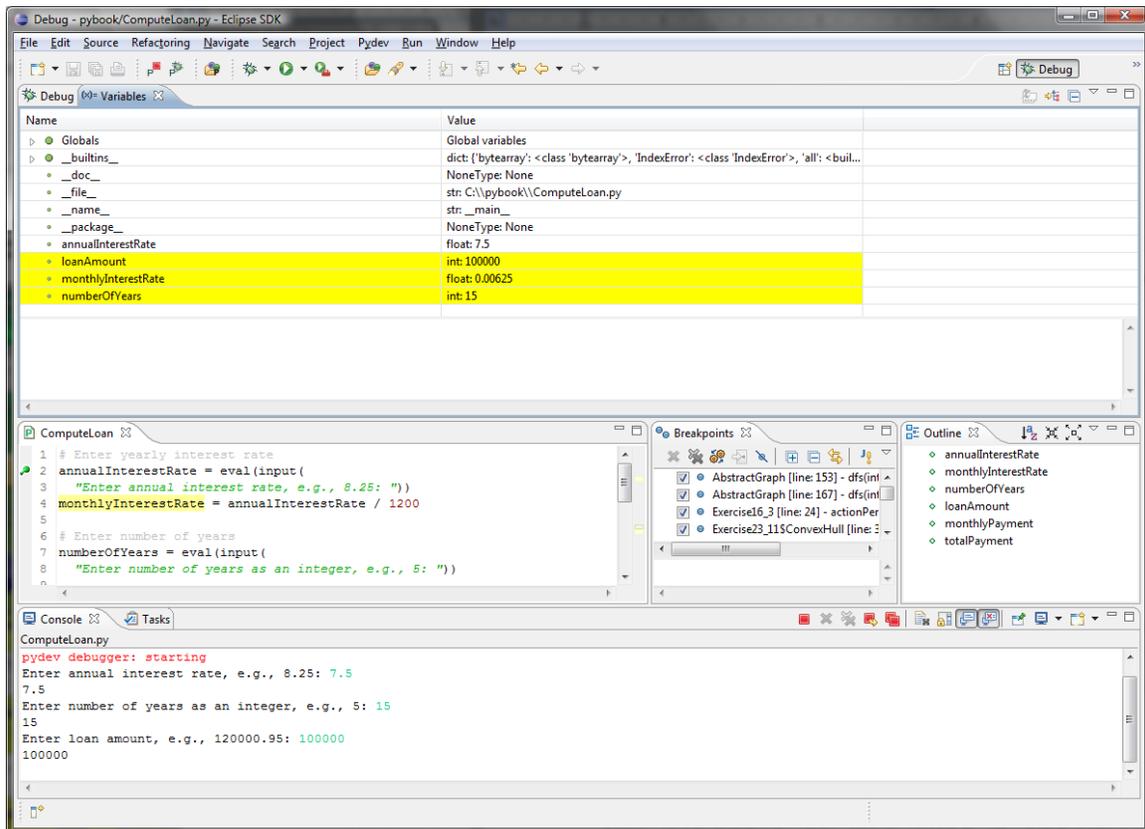


Figure 8

The variables are displayed in the Variables tab.

Continue to Step Over to see the change of variables in the Variable pane.

If you want to stop debugging, click the Terminate button in the Console pane or choose *Run, Terminate*.

To go back to edit the program, choose Java from the Perspective menu, as shown in Figure 9.

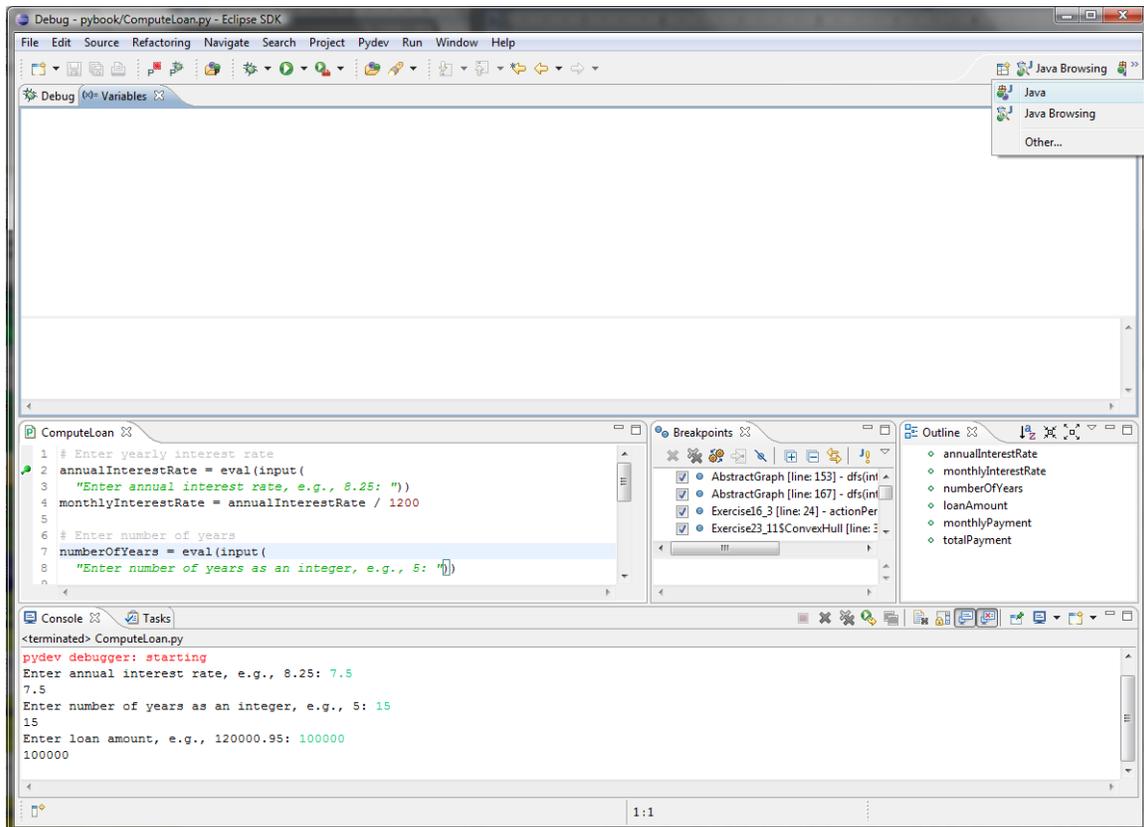


Figure 9

You can choose a layout for workspace in Eclipse from the perspective menu.

TIP:

The debugger is an indispensable, powerful tool that boosts your programming productivity. It may take you some time to become familiar with it, but the effort will pay off in the long run.

Note:

The debugger is not only a valuable tool for finding errors, but it is also a valuable pedagogical tool for learning programming.