



Debugger

- [Before you start](#)
- [What this tutorial is about](#)
- [What this tutorial is not about](#)
- [Refreshing the memories](#)
 - [Run/debug configuration](#)
 - [Breakpoint](#)
- [Preparing an example](#)
- [Setting breakpoints](#)
- [Debugging session](#)
 - [Changing the Debug tool window layout](#)
 - [Adding a watch](#)
 - [Exploring frames](#)
 - [Simple debugging](#)
 - [Stepping through the script](#)
 - [Running to cursor](#)
- [How to invoke the debugging commands](#)
- [Summary](#)

Before you start

- Make sure that you are working with PyCharm version 3.0 or higher. If you still do not have PyCharm, download it from [this page](#). To install PyCharm, follow the instructions, depending on your platform.
- You have at least one Python interpreter installed and configured for your project.
- You have already created a pure Python project.

What this tutorial is about

Here we intend to debug a sample Python script, look how to change layout of the Debug tool window, how the toolbar buttons work. etc.

What this tutorial is not about

The basics of Python programming are out of scope of this tutorial. Please refer to the [Python documentation](#).

Refreshing the memories

For completing this tutorial, you have to refresh your knowledge of two things:

- Run/debug configuration
- Breakpoint

What are they?

Run/debug configuration

Each script or test you wish to run or debug from within PyCharm, needs a special profile that specifies script name, working directory, and other important data required for running or debugging. PyCharm comes with a number of such pre-defined profiles, or run/debug configurations that serve patterns, against which you can create any number of run/debug configurations of your own.

Every time you click the Run or Debug buttons (or choose Run or Debug commands on the context menu), you actually launch the current run/debug configuration in the run or debug mode.

Refer to the [product documentation](#) for details.

Breakpoint

A [breakpoint](#) is a line of the source code, where PyCharm will suspend, when this line is reached. PyCharm discerns several [types of breakpoints](#), each one denoted with its own [icon](#).

For details, refer to the [product documentation](#) and to the [Breakpoints tutorial](#).

Preparing an example

In your project, [create a new Python file](#) with the name ThreadSample.py, and type the following code:

```
import threading
import time

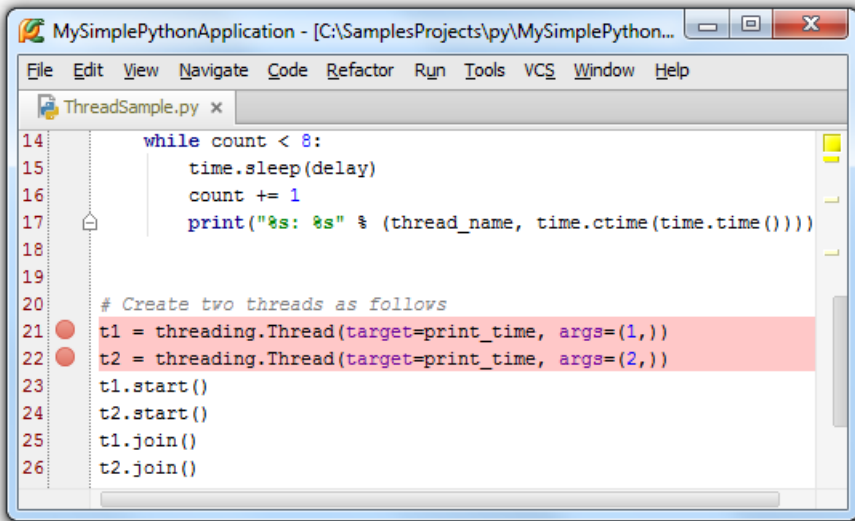
def get_thread_name():
    t = threading.current_thread()
    return t.name

def print_time(delay):
    """Define a function for the thread."""
    thread_name = get_thread_name()
    count = 0
    while count < 8:
        time.sleep(delay)
        count += 1
        print("%s: %s" % (thread_name, time.ctime(time.time())))


# Create two threads as follows
t1 = threading.Thread(target=print_time, args=(1,))
t2 = threading.Thread(target=print_time, args=(2,))
t1.start()
t2.start()
t1.join()
t2.join()
```

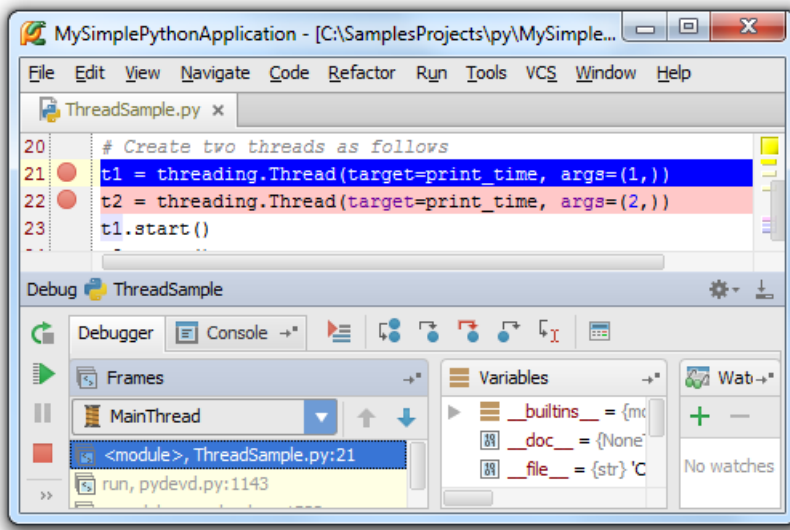
Setting breakpoints

First of all, let's set breakpoints in the source code - this is the major requirement of debugging. To do that, just click the left gutter on the lines you want PyCharm to pause at while debugging:



Debugging session

Select run/debug configuration "ThreadSample", and then press Shift+F9 (or click  on the toolbar). The debugger starts, and pauses at the first breakpoint:



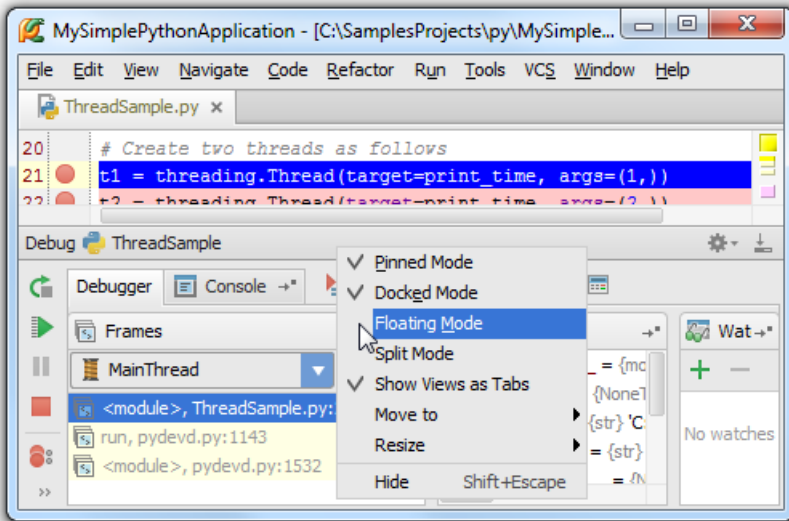
The line is now marked blue. It means that PyCharm hit this line, but not yet executed it.

Changing the Debug tool window layout

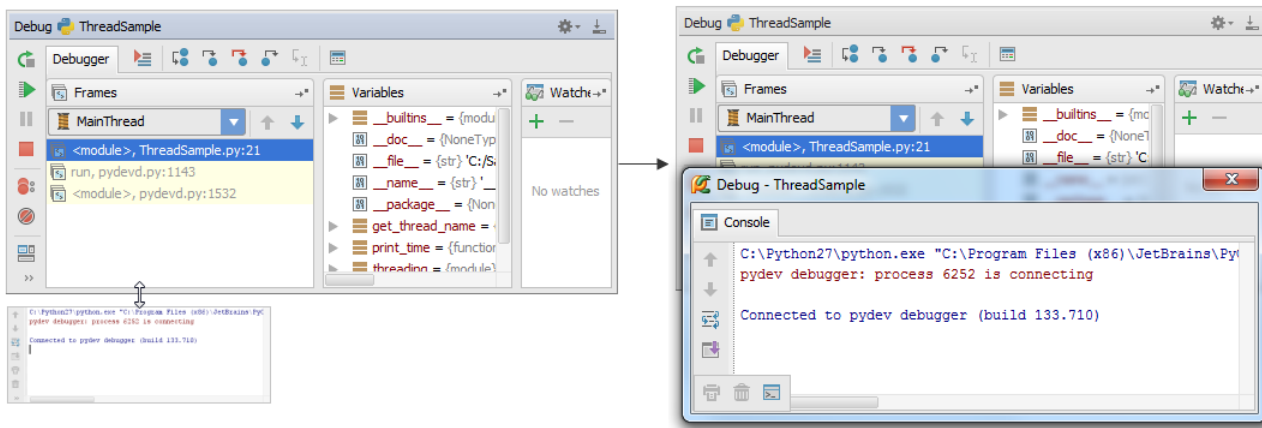
You see that the debugging session actually takes place in the [Debug tool window](#). You already [know how to use this tool window](#).


By the way, if you don't feel quite at ease with the default layout - for example, it seems more handy for you to view the debugger and the debugger console in separate windows - you can always change it.

First, let's make the Debug tool window separate: to do that, just right-click the window title bar, and select the check command **Floating mode**:



Next, let's move the Console tab to separate window. To do that, drag the Console tab and drop it outside of the Debug tool window:




To restore the default layout, click  in the toolbar of the Debug tool window.

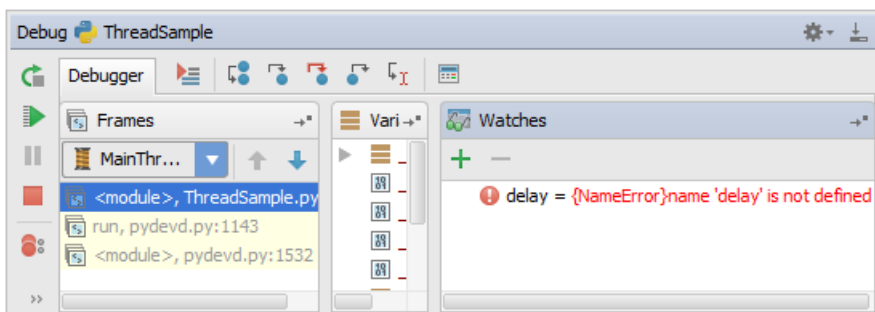
For details, refer to the sections [PyCharm Tool Windows](#) and [Moving tabs and area](#).

Adding a watch

Let's explore how certain variables behave in course of the script execution. To do that, let's put a watch on them.

This how it's done: In the [Watches](#) pane, click , type the desired name, for example, `delay`, and press Enter. You can also do same in the editor: just right-click the parameter `delay` in the function definition, and choose **Add to watches** on the context menu.

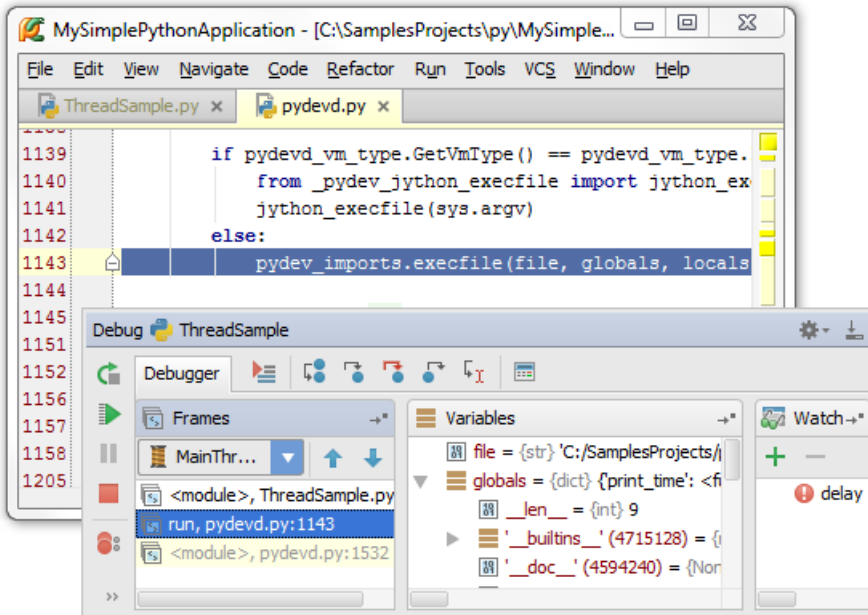
Now, if you look at the Watches pane, you'll see that `delay` is not yet defined:




A little bit later you'll see how this parameter takes values, and how it is reflected in the watch. As an exercise, put the watch on the method `get_thread_name()`.

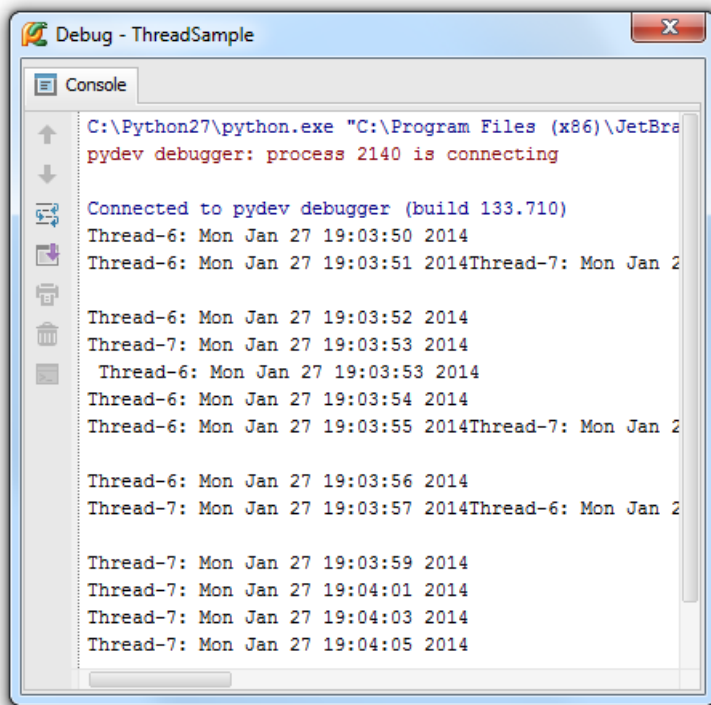
Exploring frames

You see that the thread has the name `MainThread`, and this thread contains three frames. Clicking on each frame displays its variables and opens the corresponding file in the editor, with the line in questions highlighted:





Simple debugging

Resume script execution by clicking  after each breakpoint. Observe the script output in the console:

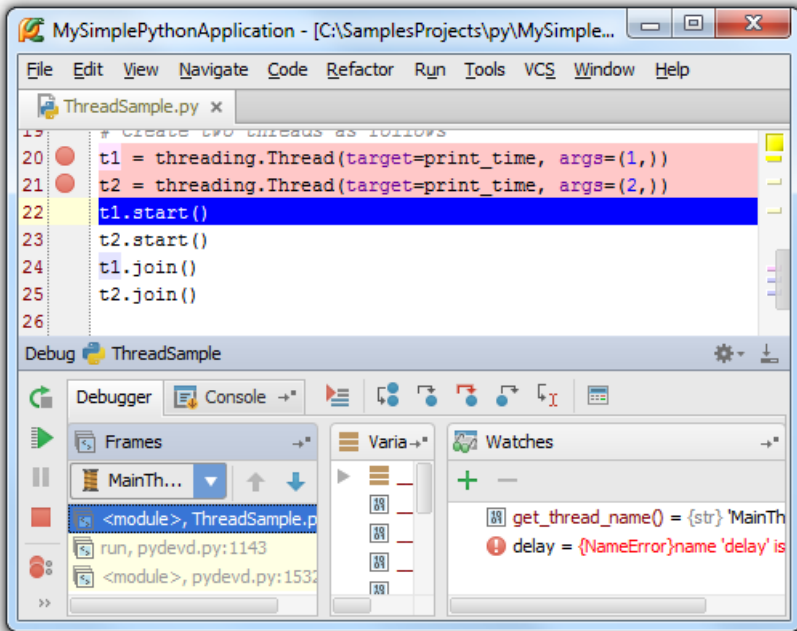



Stepping through the script

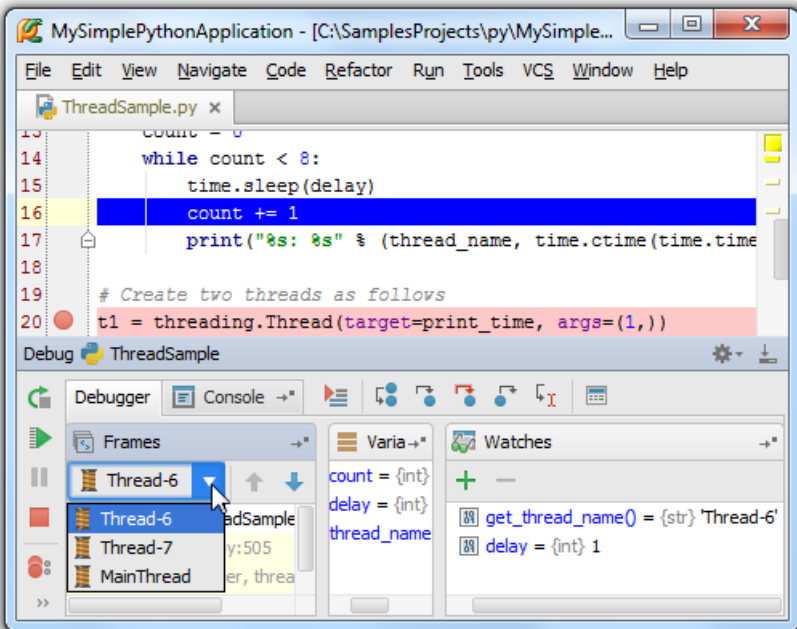
Let's go further and repeat same script once more. To rerun the current debugging session that has come to an end, click the button  in the Debug tool window (if the session has just been suspended, the rerun button looks like ). Again you see the first line with the breakpoint highlighted.

On top of the Debug tool window, you see the toolbar that contains the [stepping buttons](#).

Click the button , or press F8. You see the blue highlight moving and PyCharm suspending at each next line:



Moreover, when you pause the script execution (), you see highlighted line inside the method `print_time()`. You can choose any thread and observe the values in the **Variables** and **Watches** areas changing:

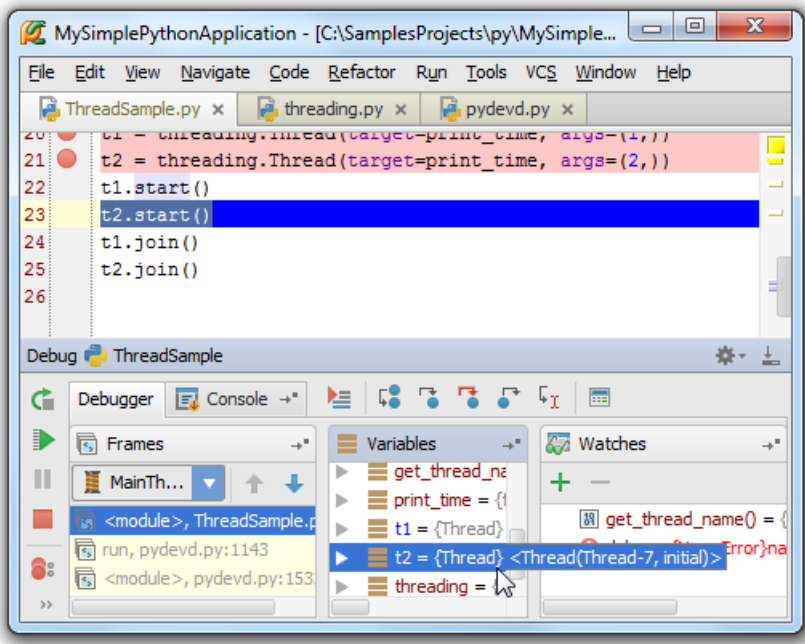


Running to cursor

Suppose, you don't want to add another breakpoint, but you still want to execute your debug session up to a certain line. This is how it's done... Rerun the debug session up to the second breakpoint, and then place the caret at the line








```
t2.start()
```


Then click the button , or press Alt+F9 . The line at caret gets highlight:



How to invoke the debugging commands

Needless to say that all the debugging commands are available not only in the Debug tool window, but also in the Run node of the main menu, and on the context menu of the editor. You can also invoke these commands using the keyboard only. Here is the list of the some useful debugging commands and the corresponding keyboard shortcuts:

Command	Icon	Keyboard shortcut
Toggle breakpoint on the line at caret		Ctrl+F8
View all breakpoints		Ctrl+Shift+F8
Resume debugging session		F9
Rerun debugging session	 or 	Ctrl+F5
Stop		Ctrl+F2
Step over		F8
Step into		F7

Command	Icon	Keyboard shortcut
Run to cursor		Alt+F9

Find full description of the controls on the page [Debug tool window](#).

Summary

Another tutorial is over. Let's summarize what has been done. In this tutorial you have:

- refreshed your memories on the notions of a run/debug configuration and a breakpoint.
- set breakpoints in your code
- launched the debugger
- stepped through the script