Introduction to Python

Debugging Programs in PyCharm

The PyCharm debugger is an essential tool for observing execution of our programs and "inspecting" the values of our variables. With the debugger we can pause execution at any point, "step through" our code statement-by-statement, and see the value and type of any variable in our code.

The intended purpose of the debugger for this course is to help you determine the **type** and **value** of every variable in your code as it executes. **As you know, you are required to note this information in your code solutions**, so the debugger can be used to complete this requirement**.**

1. Basic Debugging

    a. In the session_00 folder, double-click **hello_vars.py** in the project window. PyCharm loads the code for this file in the main window.

    b. Set a breakpoint. To the left of the code line **b = 10**, between the line number **4** and the code, there is a blank "alley" running from top to bottom. Click in this space at that line. A large red spot appears - this is called a *breakpoint*. When we debug the program, Python will stop at this point.

    c. Run the debugger. Right click in the code window and just below the **Run** selection, select **Debug hello_vars.py**. In the code window, PyCharm highlights the code line with the breakpoint (the red circle). At the bottom, PyCharm opens a **Debugger** window with several new display values and controls.

    d. Note how the code window has changed. The debugger is actually in the process of executing the code, but it has paused.

        i. The program ran to the breakpoint and paused there.

        ii. The highlight at the breakpoint shows that the debugger is ready to execute the highlighted line in the code.

        iii. Above the highlighted line, PyCharm shows us the value of **a**, which is **5**. If this value were to change later in the code, the display would change accordingly.

    e. Examine the Debugger window. Note the following elements of the debugger window:

        i. The run controls. On the far left column of the debugger window, note the following control buttons:

            1) circular green arrow ("Rerun")
            2) green triangle ("Resume program")
            3) red square ("Stop")

            These (and the "step" control button) are used to control the debugger.

ii. The Debugger tab. The Debugger tab is currently selected. This window allows us to inspect our variables.

1) **Frames** and **MainThread** are not needed for our purposes.
2) **Variables** shows the type and value of each variable encountered by the debugger. Note the one variable **a** is identified as an **int** (this means integer) with value **5**.

iii. The "step" control buttons. These are used to control execution of the program. You can see the name of each control by hovering your mouse cursor over each button.

The only step control button we need to consider is **Step into my code** (the 4th from the left, pictured as an arrow with three very short lines).

iv. Click "step into my code". Note three things:

1) The highlight has advanced to the next code line.
2) The value of **b** has appeared next to the previous code line.
3) The value and type of **b** have been added to the **Variables** window below. **b** is also an **int**, with value **10**.

v. Click "step into my code" again. Note that **c** has been added to **Variables** and its value shown in the code window.

vi. Click the green "Resume program" button. The code executes to the end and the variable values disappear.

vii. Click the "Console" tab. This tab shows the output of your program in the same way the **Run** window does. At any point in the debugging process you can check the output of your program as it is running or once it is done running.

2. Using the debugger with the **input()** function. Since **input()** requires that we use the **Run** window, it can be confusing to use it when we also need to attend to the **Debug** window as well.

Before running the below instructions, please make sure you understand how to use the debugger by following the instructions in the above section.

a. In the Session 2 folder, find and open the file **debug_demo.py** in PyCharm. The script should look like this (if you can't find it, copy and paste the below into a new program):

```
a = 5.05
b = 'hello'

ui = input('please enter a value: ')

iui = int(ui)

print(f'the value doubled is {iui * 2}')
```

b. Set a breakpoint at the second line (**b = 'hello'**). Keep in mind that the debugger does not stop at breakpoints at the first line. This seems to be a bug that may be corrected in future versions of PyCharm.

c. Run the script in the debugger. The debugger launches, runs the program up to the breakpoint, and displays the value of **a** in the variable inspector window.

d. Click **Step into My Code** once. This step executes the **b = 'hello'** line. Note the addition of this variable in the variable inspector window. We are now about to execute the **input()** line.

e. Click **Step into My Code** again. This step executes the **input()** line and causes the debug window to go dark. The reason for this is that input is required – see next.

f. Click the **Console** tab next to the **Debugger** tab in the debug window. Here you see the **input()** message and a **>?** prompt.

g. At the **>?** prompt, type some numbers, then hit [Enter]. The window displays a **>>>** console prompt.

h. Click back to the **Debugger** tab next to the **Console** tab. Note that the variable **ui** now has the value that you entered and displays it and its type.

i. Continue stepping through the code as needed. The debugger will now behave as previously. Any encounter with **input()** will require that you click back to the **Console** to enter the input value.

3. Using the debugger when running from the command line with **sys.argv** (Session 9). Since **sys.argv** accepts values from the command line, it is not compatible with the debugger. Although you could enter **sys.argv** values through a PyCharm menu, I recommend instead the following workaround:

```
import sys

sys.argv = ['myscript.py', '1927', 'lowest']
```

The above line sets the program name (you can substitute yours) and two command-line arguments, emulating what you would see in **sys.argv** if you ran the script with the arguments **1927 lowest**. Once you have finished using the debugger and wish to go back to the command line, make sure to comment out the last line above.