Further Readings for Week7

This week we'll learn an important skill that complements programming skills, debugging. ***Debugging*** is the process of identifying and removing bugs or errors from the code. Why is it important? It makes the code we write error-free to obtain the output we expect.

There are many techniques for debugging, some of them being, trace debugging(the most common where we add print statements to print the values of the variable that we want to trace as the code executes), post-mortem debugging(debugging after the program has failed) and interactive debugging(using debugging tools to pause the execution of the program and inspect our code at various points). We'll learn about interactive debugging today.

Most Integrated Development Environments(IDEs) like VS Code, Sublime Text, PyCharm come with built-in debuggers. They allow us to pause the program execution, inspect variable values, and step through the code one line at a time. We'll first familiarize ourselves with common terms used in debuggers and then look at a demo.

Breakpoints

A ***breakpoint*** is a marker that tells the debugger to pause execution at a specific line. This allows us to check the state of the program. Once the program hits a breakpoint, the debugger will pause, and we can inspect the values of variables. Breakpoints are usually indicated by a red dot next to the line numbers on your editor.

Logpoints

A ***logpoint*** is a variant of a breakpoint that does not halt the debugger but instead logs a message to the debug console. It is similar to the print statements we add to trace our variables. The main advantage of logpoints is that they enable debugging without modifying the code to add print statements. They also help save time by not having to add or remove logging statements in our code. Logpoints are indicated by a red diamond shape next to the line numbers.

Debugger tools

When we add a breakpoint, the program execution pauses. Now, there are a few options to choose from:

1. **Continue** - 'Continue' resumes normal program/script execution up until the next breakpoint.

2. **Step Over** - If we add a breakpoint at a line executing a function call, with ‘Step Over’, we can move to the next line of code without going into the functions.
3. **Step Into** - It allows us to enter inside the function to follow its execution line-by-line.
4. **Step Out** - When inside a function, if we are done debugging and want to come out of the function, we can use ‘Step Out’.
5. **Restart** - Used to terminate the current program execution and start debugging again using the updated code or configuration.
6. **Stop** - Used to terminate the current program execution in debugging mode.

Data Inspection

Once the program hits a breakpoint, the debugger will pause, and we can inspect the values of variables. The debugger shows the variable names and their values in both the local scope and global scope. For example, if we are inside a function, we can have different variables that we use inside the function(local variables) which are different from the variables defined outside the function(global variables). While in debugging mode, we can also edit the value of variables to see their impact on the execution of the program.

Further, it supports “watching” a variable. This can be useful in situations where we are interested in inspecting how one or a small set of variables change over time during the execution of the program.

Additionally, it provides a call stack. A call stack is helpful when you have multiple functions in your code where one function calls another function.

Demo

We’ll look at a demo to understand the utilities of a debugger. I have used the debugger in VS Code to show this demo. The layout of these functionalities can change based on the IDE you are using, but the basic idea behind debuggers remains the same.

Suppose we have a program “translation.py” to translate a DNA sequence into a protein sequence. We’ll see how to add breakpoints and logpoints, use the debug toolbar, inspect the local and global variables and call stack, and finally see our log and the final output.

https://gatech.instructure.com/courses/412702/external_tools/17731
(The video is also in the media gallery)

Configuring the debugger on your IDE

VS Code -

VS Code provides detailed documentation and also video tutorials on how to configure the debugger:

1. Generic debugger documentation - https://code.visualstudio.com/docs/editor/debugging
2. Python-specific debugger documentation - https://code.visualstudio.com/docs/python/debugging
3. YouTube tutorial - https://www.youtube.com/watch?v=oCcTiRGPogQ

Sublime Text -

The graphical debugger for Sublime Text can be installed separately and the documentation says that they attempt to match Visual Studio Code's Debugger fairly closely.

1. Instructions to install and setup the debugger can be found here - https://packagecontrol.io/packages/Debugger
2. YouTube tutorial - https://www.youtube.com/watch?v=qACDg_B5DSQ&t=333s

PyCharm -

PyCharm has a built-in debugger and its documentation is a great resource for learning about debugging utilities provided in PyCharm as well as easy-to-follow walkthroughs -

1. Tutorial 1 - https://www.jetbrains.com/help/pycharm/debugging-your-first-python-application.html
2. Tutorial 2 - https://www.jetbrains.com/help/pycharm/part-1-debugging-python-code.html
3. Debug configurations - https://www.jetbrains.com/help/pycharm/run-debug-configuration.html
4. Debug tools - Refer to subsections under https://www.jetbrains.com/help/pycharm/debugging-code.html