vscode_code_editing

August 2, 2023

1 Using vscode to write code

You can use any editor you wish, but we suggest using vscode. vscode is a very commonly used IDE, and provides a great deal of extensions including github copilot, which is powered by openai and chatGPT (vscode, github and in large part openai are all owned by microsoft.)

1.1 How to code in a package

Our code is organized as a Python package. To develop code and complete the assignment, you will need to add or adjust code and/or comments in both the source code files (eg, `assignment.py`) and possibly the test file (`test_assignment.py` in this example).

1.1.1 Using the test runner and interactive debugger

- In the lefthand most side of the vscode screen, there is a vertical list of icons. click on the **flask** icon.
- This will open a menu which stores a list of tests – by default it is collapsed. Go ahead and expand it.
- If you hoover over any one of the tests you'll notice that three icons appear next to it. The first looks like a play button – that will run the selected test. The second is a play button with a little bug – this will run the test in debug mode. More on this shortly. The last is a file icon with an arrow – click this and the test will open in the text editor. Go ahead and click on the `test_draw` file icon to open the corresponding test.
- if you hover your mouse to the left hand side of the numbered lines in the `test_draw.py` which is now open in the text editor, a little read dot somewhat transparent dot will appear. Click and the dot will become solid.
- Now, over in the left hand test explorer, hover over the test_draw test and click the debug button (play button with a bug).
- The code will now execute up to the breakpoint (that red dot) which you set. The execution will halt there and a new control panel will appear in the top center of the text editor panel. This will have four icons:
  - A play button with a bar to the left, which if clicked will execute the rest of the test until it either encounters another breakpoint or reaches the end.
  - A dot with an arrow over the top. This button will let you step through the code line by line
  - A down arrow. If the current line (the highlighted line) is a function, clicking this button will let you 'step into' the function and walk through the code which may be in a different file or even a different package. Sometimes you might have to do this multiple times if there are functions in the function call.

  - An up arrow. If you have stepped into a function, clicking the up arrow will move you back up in the call stack.
  - a green "do over" icon – this will re-run the test. It will halt at any breakpoints
  - And a stop icon, which will exit the test.

You can set a breakpoint anywhere in the test or source code. There are two things you'll find helpful – in the left hand panel, there will be a variable explorer, which will let you see the current state of any variables at that point int he code. You might also find the call stack information useful, which basically gives you a 'map' of the path through the various functions that have been called which have gotten to you to this point in the code.

The most useful feature, though, is the 'debug console'. This may appear by default in the same panel (by default the bottom panel) as your terminal. If it is not there, then you can open it by clicking the three dots at the top of the left hand panel and ensuring that the debug console is checked. In the debug panel, you can execute python. All of the variables, functions and objects listed in the variable explorer on the left will be available. This is a very nice place to test out variations on the code which may be causing a bug.

This is how I typically program in a package with code which is as complete as the code that you encounter here is.

1.1.2 Using a jupyter notebook

Your environment is setup so that you can use a jupyter interactive console (an interactive python session, not something you can save to a file) or a jupyter notebook (something you can save). The console is useful to use to just test out some python code – unlike the debugger, it cannot track the state of the code as it executes. But you can copy/paste functions into the console, and execute code.

The submission code must be written in `assignment.py` and in some rare cases, in `test_assignment.py`. You cannot submit code outside of these two files. But, you can certainly use a jupyter console or notebook to test out code, and you may find it easier to do so in one of these formats than in the debugger as you first get started understanding the code, algorithm and assignment.

Using an interactive window

- To use the console, hit cntrl-p and type: `>jupyter` which will filter down the options in the drop down menu. Select the `>Jupyter: Create an Interactive Window` option
  - At this point, you can code in the console. If you make a change the the source code and wish to import it (eg from probability_intro import draw), you will need to restart the session (see the top of the jupyter console window. Click the restart icon.)

Using a notebook

- Similar to the console, hit cntrl-p and type in `>jupyter` but this time select `>Create: New Jupyter Notebook`. Unlike the console, this is a typical notebook where you can write code or text. A nice new feature of jupyter notebooks is the debugger, which works very similarly to the vscode debugger. You can set breakpoints in the notebook or the source code. **But**,

if you make any changes to the source code, make sure to restart your session so that it is available in the notebook.
  - If you wish to save the notebook, save it in `docsource`. As a general tip, putting your notebook in the docs isn't a bad idea as your EDA can be turned into a tutorial.

1.2 Using the logger

Rather than writing print statements, which is a common debugging practice, you can use the logger. To configure the logger in an interactive session or in a notebook, do this:

- in a interactive session or a notebook, you can configure the logger by setting the log level, and where the log outputs, like this:

```
import logging
import probability_intro

probability_intro.utils.configure_logging(logging.DEBUG)
```

The logging levels are:

```
logging.DEBUG
logging.INFO
logging.WARN
logging.ERROR
```

By setting the logging level, the logger will log anything at that level and above. By default, the logger is set to WARN. Using the logger, rather than print statements, means that you can make your debugging part of your program as you develop.