1 Python 101

This assignment is intended to help you get accustomed to programming in Python, debugging in VS Code (or your preferred IDE), and submitting assignments to Gradescope.

1.1 Template Contents

You'll find three Python files in the project template:

1. **latency_calculator.py** contains several function stubs you'll need to complete in order to calculate the total RTT. Each of these functions contains a TODO comment with more information. Make sure that you implement each function, rather than computing everything in a single function. Otherwise, your submission won't be able to pass each of the tests.
2. **network_config.py** contains a single variable that will store a dictionary containing the parameters defining a given network. Normally you would probably just include this inside of latency_calculator.py, but we're breaking it out as a separate file to give you some experience including methods and variables from multiple files. You'll need to import the variable from this file into latency_calculator.py to pass the tests (see the comment at the head of the latency_calculator.py file)
3. **debug_me.py** contains a simple application to calculate the price of a meal including tax and tips. There are several errors in the code that you will need to fix. This is intended as an exercise to give you experience using VS Code's debugger, as this will be invaluable when you are working on more complex projects later in the semester. This will also expose you to several common mistakes I've seen students make in the past, which will hopefully help you identify them more quickly if you make them yourself. You need to fix each of the bugs in order to get the code to work and pass the associated test.

1.2 Writing a Program to Compute Round Trip Time Within a Network

In the first part of this assignment, you'll be writing code to compute the total round trip time (RTT) for a packet moving through a given network. As we've discussed in class, there are four major sources of latency:

1. Transmission delay: the amount of time required to push all of the bits of the packet "onto the wire". This is influenced by both the length of the packet and the bandwidth available to a given router.

$$delay_{transmission} = \frac{packetlength}{bandwidth}$$

2. Propagation delay: the amount of time required for the information to physically travel through the network. This is influenced by the distance the information needs to travel and by the speed at which the information travels through the transmission medium (typically copper wire, fiber, or radio waves).

$$delay_{propagation} = \frac{distance}{speed}$$

3. Processing delay: overhead incurred at the router when a packet is received. This is typically a negligible amount (a few milliseconds at most). For the purposes of this course, a value for processing delay will be provided to you in problems involving latency.
4. Queuing delay: a variable amount of time based on how much other traffic is in the network at a given moment. Delay is proportional to the amount of incoming data divided by the total bandwidth of a router. So long as the amount of data coming in does not get too close to the total bandwidth, queuing delay will typically be very small. However, as the amount of incoming data approaches the outgoing bandwidth, delay increases exponentially.

$$delay_{queuing} \sim \frac{packetlength * arrivalrate}{bandwidth}$$

   Note: this equation assumes that all packets are of the same length, which allows you to calculate the amount of incoming data by multiplying the length by the arrival rate. In a real network, the equation would be more complex due to packets of varying sizes.

latency_calculation.py and network_config.py need to be completed in order to pass the tests associated with this portion of the project. Comments are included in these templates that provide you with more specific instructions.

1.3 Debugging Python Code

You'll need to get the debug_me.py function to work properly in order to pass the test associated with this code. I recommend using VS Code's debugger to help identify each of the bugs in this code. Each of the bugs will throw an exception. When you encounter an unfamiliar exception, the first thing you should do (besides reading it) is to google the error message. This will often directly lead you to a solution.

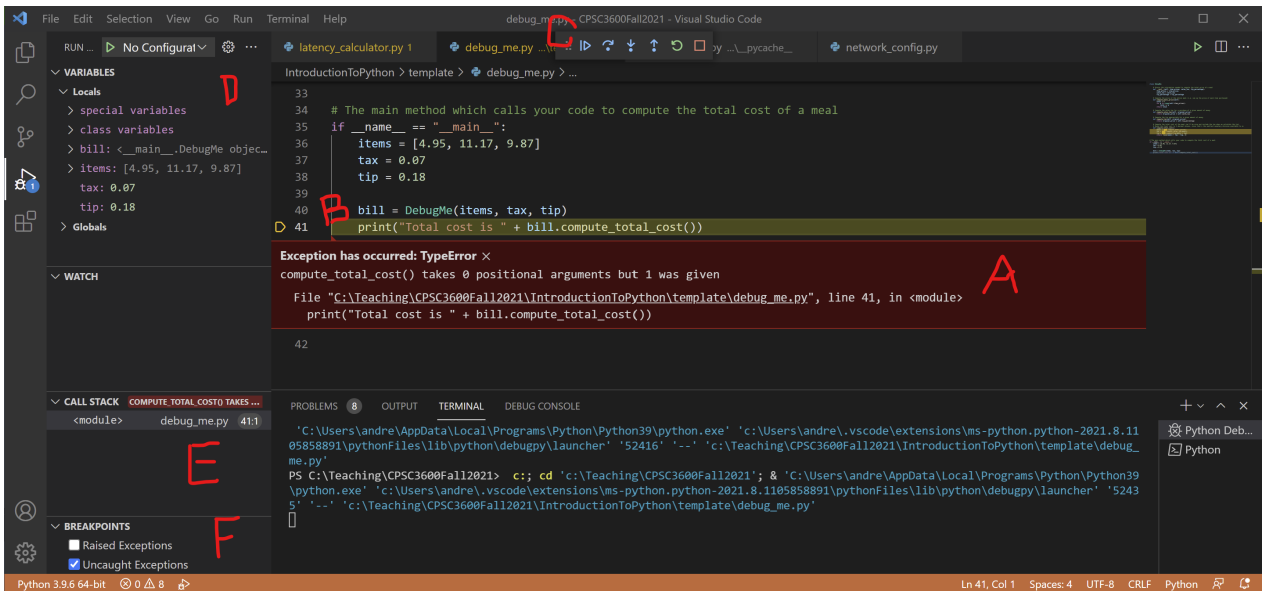

Figure 1: The first exception you should receive on running the code

The above image shows a typical exception being thrown. I've labeled the following parts of the image

A This is the actual message thrown by the exception. Search for this message online will often help you find what is causing the error.

B This is where the exception occurred. More generally, the line highlighted in red is where the code has "broken", i.e. paused execution. This is also indicated by the yellow arrow to the left of the red line. You can also add break points to your code by clicking to the left of the line number (the same column where the yellow arrow is). Breakpoints lets you arbitrarily pause the execution of your code at a specific point, even if an error is not thrown there. Break points will be indicated by a red circle.

C These buttons will allow you to step through your code line by line to see how it executes in action. From left to right, the buttons are:

(a) **Resume execution** exits the debugger and resumes normal execution of the code

(b) **Step over the current line of code** executes the highlighted line of code and moves on to the next line of code. If the current line of code contains a function, the contents of the function will be executed but will not be debugged.

(c) **Step into the current line of code** same as above, except that the code will step into functions and allow you to debug their execution

(d) **Step out of the current block of code** completes the current function and moves on to the next line of code that will be called after the function

(e) **Restart the program** resumes execution of the code from the first line

(f) **Stop executing the program** exits the debugger and stops running the code

D This window pane allows you to inspect the values of the different variables that are currently in scope. This includes simple variables, like *tax* that stores a float value, and more complex values like *bill* that stores a class instance. *This pane is an incredibly useful tool to watch how the state of your program changes while it executes.*

E This window pane shows the current state of the call stack, allowing you to get a sense for what functions have been called in order to reach this state in the program.

F This window pane allows you to control which exceptions the debugger will break for. By default, all uncaught exceptions (i.e. exceptions your code doesn't know how to handle) will cause the code to break. You may also sometimes find it helpful to break on all raised exceptions, even those your code knows how to handle successfully. If so, check the *raised exceptions* box here.

2 Submitting your code

You can submit your project through Gradescope (which can be accessed via Canvas). You'll see the Python 101 assignment listed on your dashboard. Click on it and a window will appear where you can drag your code files (either directly as files, or as a zipped submission containing the files). A batch of tests will begin running once you submit your code. These should complete fairly quickly, after which you'll see what tests you passed and failed.
Some tests will only become visible once the assignment is finally due, however these tests will always build on tests visible during development. If you pass the visible tests, you can be fairly confident that you will also pass any hidden tests.