T05: Buggy Fruit

Learning Objectives

- More practice identifying functions
- More practice creating new functions
- Understand the value of return values from fruitful functions
- Learn how to capture the output of a function
- Practice some new ways of debugging programs

How to Start

- Paired assignment T05 should be completed with a partner.
- To begin, make a copy of this document by going to File >> Make a Copy...
- Share the copied document with all members of your team. You can share this document by hitting the blue button in the top right of the document, then entering the email addresses of all members in the bottom input field.
- Change the file name of this document to **username1, username2 - T05: Buggy Fruit** (for example, **pearcej, shepherdp - T05: Buggy Fruit**). To do this, click the label in the top left corner of your browser.
- Next, go to the **GitHub Classroom for T05**.
- You will be asked to create a team:
    - One of you will create the team.
    - Then, the other will join the team created by the first partner.

Checkpoint 1: What is the web address for the GitHub repository for your T05?	C1:

- Open PyCharm. If you are not at the Welcome to PyCharm page, close the current project.
- Using the **Get from VCS** button, open your T05 repository.

Roles

Driver[1]:	
Navigator[2]:	
Quality Control[3] (if the class is odd numbered):	

Area of Circle

To begin T05, we will be exploring some code in the online debugging tool from last class. The tool will allow us to see how the program is used in memory. You can access the code and tool here: https://goo.gl/HP2yoe . Use the "**Forward>**" button at the bottom to step through each line of code. In the table below, briefly explain what is happening.

In which step does `r_str` first get a value? Where does this value come from?	1.
In which step does `r_val` first get a value? Where does this value come from?	2.
Describe the difference between the value stored in `r_str` and `r_val` after step 8 of the program.	3.

[1] The driver will be doing the majority of the typing in PyCharm. Your job is to solve the problem given to you by the Navigator.

[2] The navigator will be giving directions to the driver, and helping the driver catch syntax and logic errors as he or she creates the code. The navigator should keep track of time and make sure progress is being made.

[3] The quality control specialist will ensure rules are being followed, both in the code (suggesting places to add comments, watching for misspellings, etc.) and in this document (making sure the questions are being answered at the right times, checking for typos, etc.) In a group of two, everyone is responsible for quality control.

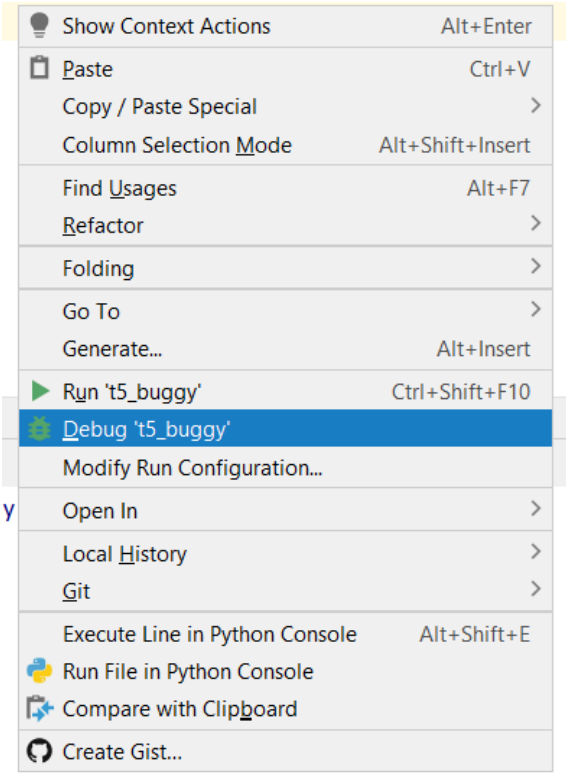
In which steps does `radius` exist as a variable? i.e. Where does this value first come from?	4.
In which steps does `result` exist as a variable? Where does this value first come from?	5.
What happens to `radius` and `result` when the flow of execution returns to the `main()` function?	6.
In which step does `area_val` first get a value? Where does this value come from?	7.

Debugging Code

One of the most common frustrations with programming is seeking out bugs and squashing them. **Debugging** is the process of identifying and eliminating unexpected behavior produced by your code. The IDE understands your frustrations, and has tools to help make debugging code easier for you. Let's first explore some code, then look at some of these tools. Run the following code like you normally do in PyCharm: t05_buggy.py

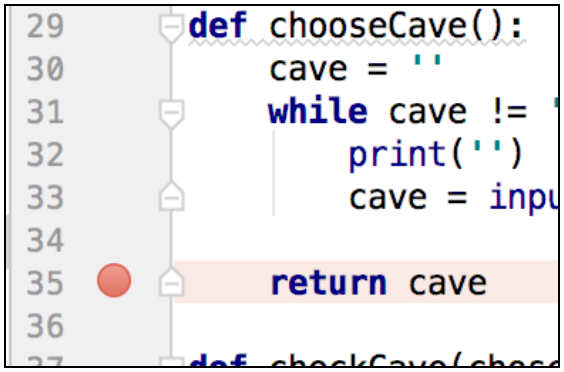
Does the code work as expected? Why not? (at this point, don't fix the code!)	8.

Here are a few of the most important tools you'll need to understand to become a master debugger. First, instead of running the code, you will need to set the debug mode of the IDE as shown on the right.



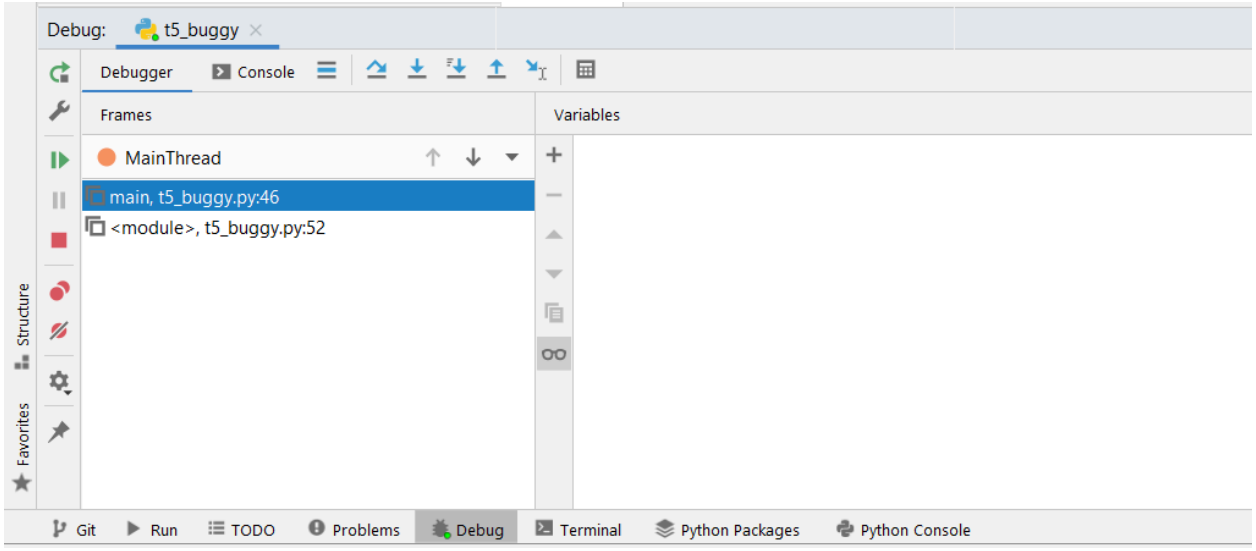
Breakpoints: Debug mode will make it so that the IDE will stop at your breakpoints. Clicking on the button again after reaching a breakpoint will resume running your program until it ends or another breakpoint is reached. Breakpoints are lines in your program that you have indicated where the IDE should stop running the code so that you can see a "snapshot" of what the variable

values are on that line. The figure below shows an example of setting a breakpoint INSIDE a function, which is very useful.

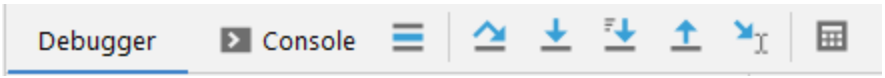


In most IDEs you can set a breakpoint by clicking in the margin near the line number. A red dot will appear as shown above. Note that using breakpoints before returns (as shown) is generally a good idea. When you run the code in debug mode, the IDE will execute up to the first breakpoint. Pressing run again will cause the program to run until the next breakpoint it hits, and so on. However, it would be inconvenient to set breakpoints, for instance, at a large number of consecutive lines. Not to worry! There is a solution for that as well.

The Debugger Console: When you click the Debug button, the debugger console appears, as shown below.

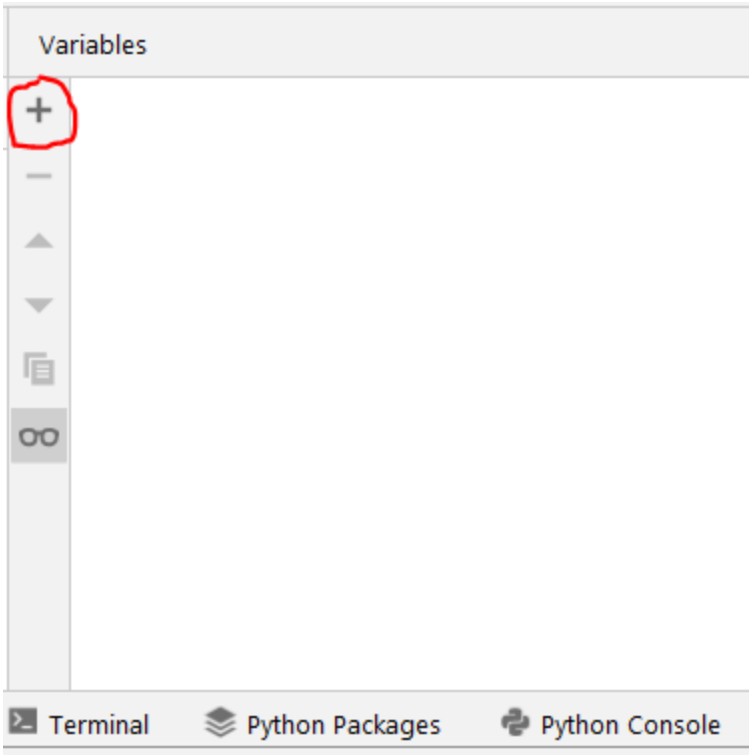


Stepping Over, Stepping Into, Stepping Out: The image below shows the different ways we can step through a program using the debugger. Hold your mouse over each button to see which step operation it does.



Click the Step Over button while in Debug mode. What does it cause the debugger to do?	9.
Click the Step Into My Code button while in Debug mode. What does it cause the debugger to do?	10.
Click the Step Out button while in Debug mode. What does it cause the debugger to do?	11.
As you step through the code, take note when changes occur in the Frames, Variables, and Console sections. Explain what information is being displayed in each section.	12.

The PyCharm Watches tab: In the image below, click the circled button.



Add a watch to the variable `n1`, and then run the debugger. Use a breakpoint early so that you can step through the program one line at a time. Make sure to use "Step Into" when you reach the line that calls the function `display_result`, and step through that one line at a time as well. What does this cause the debugger to do as you step through the program?	13.

Your Turn to Explore Some Bugs

The following are three buggy files, including the penny stealing code from last class:

- t05_buggy_birthday.py
- t05_buggy_circle_area.py
- t04_making_change.py

Use the remaining time in class to debug these codes. **Remember, that the primary objective is to learn to use the debugger effectively, so please do not think that the objective is to just fix the errors--fixing the errors is just a vehicle to using the debugger. You may not get through all three. That is okay.**

Which feature of the debugger did you find the most useful for this assignment? For what error did you find it useful?	14.
How much do you foresee you and your partner using the debugger in future assignments? Explain.	15.
Were there any other things about the code I gave you that made it hard to debug? What ways could you make it easier to debug?	16.

Submission Instructions

1. Edit the **README.md** file in your repository. Replace the lines with the correct information for your name(s), the link the repository, and the link to this document, which you can get via the blue Share button 

in the top right of this window.
2. Check the **Share** settings for this document (top right). Set them to **"Anyone with the link can view"**. That is how we will be able to access this document to grade you:



3. **Add, Commit,** and **Push** your changes to your repository.
    a. Right click any new files in the Project pane of PyCharm (far left), and click **Git >> Add.** If the filename is red, it still needs to be added. If it is green, it has already been added.
    b. Right click the project directory in the Project pane of PyCharm, and click **Git >> Commit Directory**...:
    c. Add a meaningful commit message, and click "**Commit and Push**":
    d. Click the "**Push…**" button on the screen that follows.
4. **Check your repository in Github to ensure everything was submitted.** You can view the updated repository at the link you pasted in Checkpoint 1.