Fundamentals: Assignment, Input, and Output

You typically write a program so that you or others can execute it whenever necessary to accomplish something (e.g., to calculate your taxes). The program typically requires the user to supply some inputs (e.g., your income, outstanding debt, number of dependents, etc.). From the inputs, it generally computes some intermediate values (e.g., your tax bracket), which it assigns to variables in order that it can then use them in producing some outputs (e.g., amount of taxes to be paid).

This exercise illustrates such a program. Instead of Spyder, we will use an on-line tool, called Python Tutor, which lets you step through an execution of a program to see the effect of each statement.

Part (a): We will start by using Python Tutor to visualize execution of a sequence of assignment statements, intended to solve the following problem.

The Foobar Manufacturing Company makes widgets, which it sells individually or in bundles of 150. It charges $4.95 for widgets when they are purchased individually, and offers a 10% discount on bundles. You want to purchase 1000 widgets. How many bundles should you purchase and what will be the total cost for the 1000 widgets?

Click on the Visualization link in the Artifacts section of this week’s webpage to bring up a Python Tutor visualization that we created for this problem.

Briefly, the visualization depicts:
- the program being executed (in the code window on the left)
- the contents of two areas in memory, labeled
  - Frames, showing how variables reference objects
  - Objects, showing objects that the program can reference
- the statement that will be executed next (red arrow), if there is one
- the statement that was just executed (green arrow), if there is one

Beneath the program, Python Tutor provides a progress bar and four buttons for you to use to in exploring an execution.

Press the Forward button to see how Python Tutor shows the effects of executing the first two assignments.

With your partner, take turns predicting what the effect of each step will be. Then press the Forward button to check your understanding.

---

1 The person executing the program. When you are writing a program, you are the “programmer.” When you are testing it, you are the “user.”
**Part (b) [Input]:** To make this solution more useful, you will modify it to use the number of widgets input by a user instead of always using 1000.

The simplest way to obtain input from a user is with the Python `input` function on the right side of an assignment statement. The `input` function takes a single argument, called a **prompt**, of type `str`. When called, it displays the prompt to the user and waits for the user to type a sequence of characters terminated by the `enter`-character. The `input` function then creates a string from the sequence of characters (up to, but not including the `enter`-character), and returns this string.

For our problem, you need to change the assignment in line 7 to prompt the user for the number of widgets to be purchased. In Python Tutor, press **Edit code**. This takes you to an editor. Replace the 1000 with a call to the `input` function:

```
widgets = input("How many widgets? ")
```

Then press **Visualize Execution**. Step through the code until the visualization reaches line 7.

**Q:** Why doesn't Python Tutor allow you to press the **Forward** button at line 7?

**Q:** Where does Python Tutor display the prompt?

**Q:** In Python Tutor, where do you think the user enters the input?

Enter a number into the text field in the red box below the code window and press **Submit**.

**Q:** In Python Tutor, what does pressing the **Submit** represent?

**Q:** What is the type of variable `widgets`? Why?

**Q:** What happens if you now press **Forward**? Why?

Edit the program so that it assigns to `widget` the `int` represented by the `str` returned by the call to `input`. (Press **Edit code** to return to the editor.)

Step through:
- a visualization in which the user enters 1000
- a visualization in which the user enters a number less than 150, and
- a visualization in which the user enters a multiple 150.

**Note:** To clear the input after a visualization, you need to press **Edit code** and then **Visualize Execution**. This allows you to provide a new input on the next visualization.
**Part (c) [Print]:** The current program does not display any output to the user. To remedy this, you will add some calls to the Python `print` function.

In its simplest form, the `print` function takes any number of arguments, which are separated by commas, and displays its arguments, in order, on the same line, and separated by a single space-character. It does not return any value; a statement calls `print` for the side-effect it produces in displaying its arguments.

For example: `print(5, "+", 5, "=" , 5*5)` displays `5 + 5 = 25`.

Add two print statements to the program in Python Tutor: one to print the number of widgets desired and the number of bundles and individual widgets to be purchased, and another to print the cost, both with text saying what is being displayed. Round the cost to two decimal digits.

For example:

```
For 1000 widgets, purchase 6 bundles and 100 individual widgets.
Total cost: 4504.5
```

**Part (d):** Where Python Tutor provides special windows for input and output, **Spyder** uses the console for both.

To see how this works:
- Create a new folder for week 2 in the CTL folder from the last class meeting.
- Open Spyder and navigate to the new folder to make it the working directory.
- In Spyder, open a new file and save it to a file in the working directory.
- Copy and paste the code from the Python Tutor code editor into the Spyder code editor.
- Run the code in Spyder.

Run the program with the same three inputs that you used for the three visualizations in part (b).