COSC 130 – Homework 00 Instructions

A. Introduction

The purpose of this assignment is help you to become familiar with the basics of working in Jupyter Notebooks, as well as the basics of Python. We will explore the concepts discussed here in more depth later in the course. For now, the goal is to provide you with just enough information to help get up and running.

Working Locally or in the Cloud

You can complete your work for this course by working in a Python environment set up locally on your personal computer or laptop, or you can complete your work using a cloud service called CoCalc. I recommend that you be familiar with both modes. The bullet points below provide some more information about the two options:

- If possible, I recommend doing most of your work locally. There are a number of reasons for this recommendation. First, working on CoCalc requires you to have an active connection to the internet, whereas you can work on your assignments locally without an internet connection. If you are using CoCalc during a time when their server is particularly busy, then you may encounter slowdowns. Knowing how to set up and work in a local environment is an important skill for anyone who wants to write code.

- There are some reasons why you might want to work entirely in CoCalc. If you don't have access to a computer that you can install software on, then CoCalc might be a better solution for you, as it requires no installation to use. CoCalc will also be convenient if you want to have access to your assignments from several computers without having to bother with transferring them using a USB drive.

- Everyone in this course will expected to use CoCalc in some compacity. In particular, I will ask that you submit the Jupyter notebooks for each of your assignments to CoCalc. This will allow me to add feedback directly to the notebook when I grade your submissions. Additionally, if you ever need assistance with an assignment, put can upload your notebook to CoCalc, and I can look at your work and provide feedback within the notebook. This will be often be the most effective way for me to provide you with assistance with your work.

For this assignment, I would recommend that you install Jupyter on your local device (following the instructions that have been posted on Canvas). When you are finished, you will still need to submit your assignment through CoCalc. If intend on taking this approach, please proceed to Part B of these instructions. You will then skip Part C.

If you would prefer to work entirely within CoCalc when completing this assignment, then you can skip Part B and move on to Part C.

You should have already received an email invitation to join a Project in CoCalc. If you have not received such an email, please reach out to your instructor. Once you have accepted the invitation, you can access CoCalc by clicking on the following link: https://cocalc.com/

B. Creating a Notebook Locally

Create a course folder named **COSC_130** somewhere on your computer, preferably on your Desktop, in the Documents folder, or in a subfolder of one of these two locations. Within your course folder, create three folders named "Homework", "Projects", and "Lessons".

Launch Jupyter Notebook. You can find instructions for doing this on the Canvas page titled "**Launching Jupyter Notebook**".

After launching Jupyter, navigate to the directory that contains your course folder. I placed mine on the Desktop. Upon navigating to the Desktop, I see my course folder in the directory listing, as shown below. If you have other folders stored at this location, you will see those listed here as well.

Click on the name of your course folder. This will open the folder within Jupyter, displaying its contents, as shown below.


Now click on the "Homework" directory to enter that folder. You should see that it does not currently contain any files. We will now create our first notebook. On the right side of your Jupyter interface, toward the top, you should see a dropdown menu named "New". Click on this. This should show you the options displayed to the right.

Select "Python 3". This will create a Jupyter Notebook capable of executing code written in Python Version 3. Technically, we will be working with Python Version 3.7. The new notebook will be opened in a new browser tab. In the new tab, you will see an interface similar to the one displayed below.
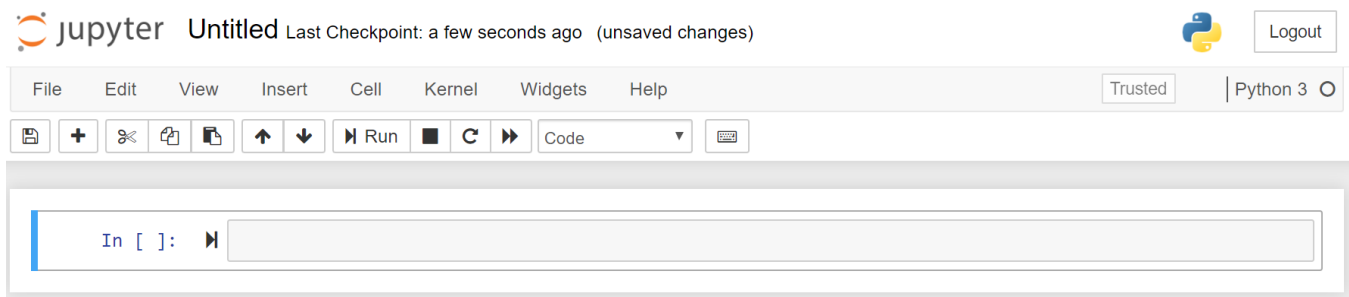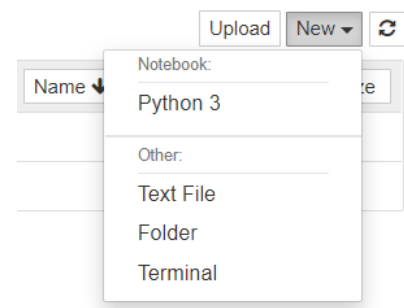
There are several options and controls available to us within the Jupyter Notebook interface. We will discuss the purpose of most of these tools later in the course. For now, notice that the name of the new notebook is listed as "**Untitled**" at the top of the page. Click on this name. This will open a dialog window titled "Rename Notebook". Change the name of the notebook to "**HW_00_YourLastName**", replacing the string "YourLastName" with your actual last name.

If you briefly switch back to the original tab showing the Jupyter file browser, you should see your new notebook file listed inside of the Homework directory. Notice that the extension for the file is .ipynb. This stands for "IPython Notebook". IPython is the original name for Jupyter Notebooks. Switch back to the browser tab displaying your new notebook.
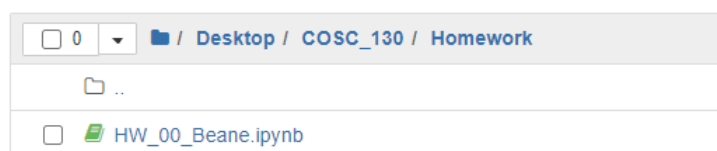
C. Creating a Notebook on CoCalc

You may skip this section and proceed to Part D if you are working locally and followed the instructions in Part B (recommended). If you would prefer to work entirely on CoCalc, however, please follow the instructions in Part C to create a new Jupyter Notebook.

First, log in to CoCalc using your Maryville email address and the password you set up when you accepted your CoCalc inviation. You should see a screen that looks similar to the one shown below.
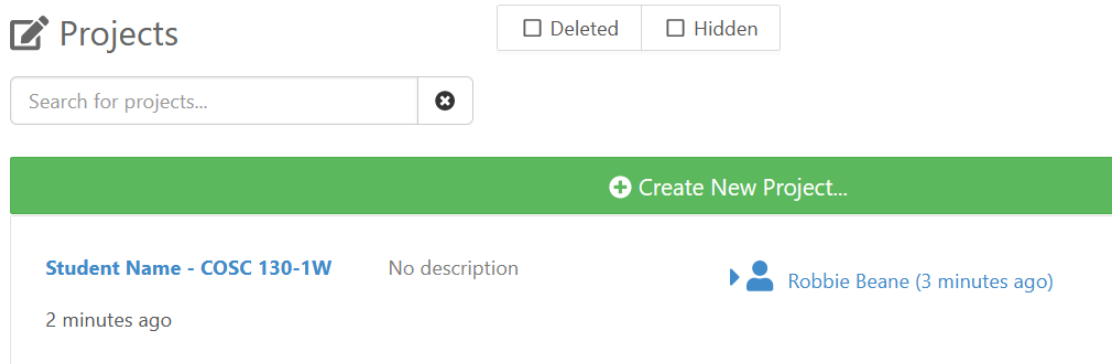
Click on the project that has the same course name and section as the course in which you are enrolled. You should see then see a folder titled "**Homework**". Click on that.


Within that folder, you will see another folder named "**HW 00**". Click on that as well.

The **HW 00** folder contains a pdf copy of these instructions, but no other files. You need to create a Jupyter Notebook file to complete this assignment.

Type "**HW_00_YourLastName**" in the text box near the top of the screen (replacing the text string "YourLastName" with your actual last name). Click on the arrow next to the New button as shown below. Make sure to click on the arrow, and **not** the New button itself. Then select "**Jupyer Notebook (.ipynb)**".
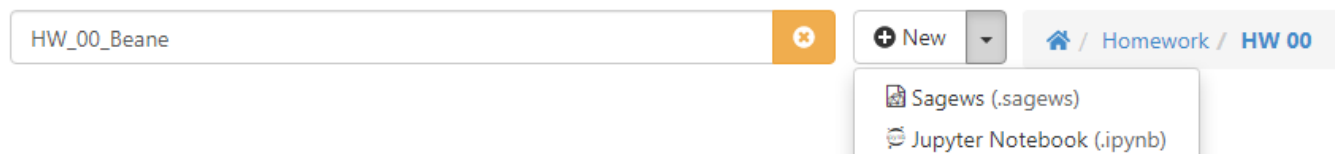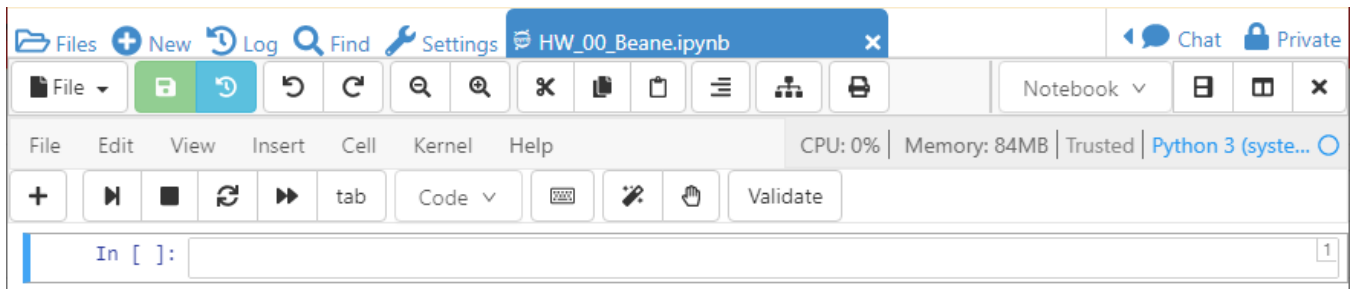

You will be taken to a screen that asks you to "**Select a Kernel**". Under "**Suggested Kernels**" click on "**Python 3 (system-wide)**".

This will open the notebook in a new tab. You should see a screen that looks similar to what is shown in the image below.

D. Working within Jupyter Notebook

This section will constitute the bulk of this assignment. The following instructions will walk you through several steps intended to get you familiar with working within the Jupyter Notebook programming environment. Don't worry if you don't understand all of the code just yet. We will cover that material later. The primary purpose of these assignment is to get you up and running with Jupyter.

Using Code Cells

At the top of your notebook, beneath the toolbar, you see the blue-bordered region shown in the image immediately below. This is a code cell. We can write Python commands inside of the text field within this cell, and then execute those commands within the notebook. The text reading "**In [ ]:**" indicates that we can use this cell to input Python code.


The blue border around the cell indicates that the cell is selected and is "command mode". You are not able to type Python code inside of a cell while it is in command mode. To enter code into the cell, you must first click on the text field inside of the cell. This will cause a cursor to appear inside the cell, and the cell border will change to green. The cell is now in "edit mode". If you click inside of the cell somewhere to the left of the text field, this will put the cell back into command mode.
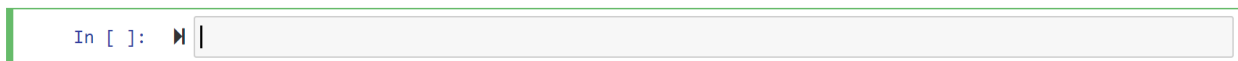
With the cell in edit mode, type the following Python command: `print(3 + 8)`
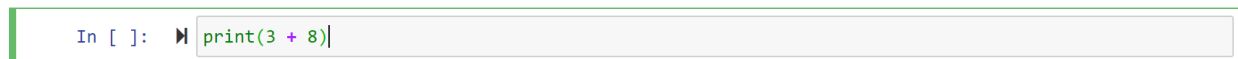
With the cell still in edit mode (showing a green border), click **CTRL+ENTER**. This will run the cell, executing any Python code contained inside of it. When this cell runs, Python will add 3 and 8, and will then print the result. You should see the printed output displayed beneath the text field, as shown in the image below. Also notice that the text on the left side of the cell has changed from "**In [ ]:**" to "**In [1]:**". This tell us that the cell has been ran, and that it was the first cell executed in our notebook.
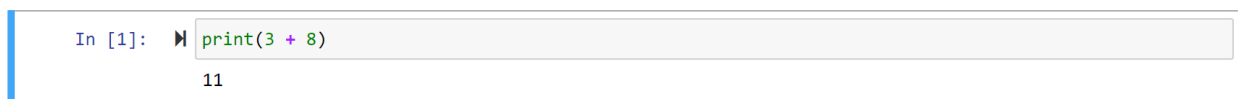
Code cells will occasionally contain computationally intensive instructions that are slow to execute. When such a cell is in the process of executing its instructions, the text on the left of the cell will read: "**In [*]:**".

Adding New Cells

We need to add several more cells to our notebook. If your cell is not currently in command mode, then click in the left part of the cell, outside of the text field. This should turn the border blue. Once the cell is in command mode, press the **B** key on your keyboard. This will insert a new cell beneath the current cell. Press **B** seven more times, to create a total of eight new cells. The contents of your notebook should now appear as shown below.

```
In [1]:   print(3 + 8)

          11
```

```
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
```

Deleting Cells

The first cell we created was simply to illustrate how a code cell works. We don't need this cell in the final version of the notebook, so we will delete it. Click on this cell somewhere to the left of the text field so that it is selected in command mode (with a blue border). Press the **D** key on your keyboard **twice** to delete this cell.

Creating Markdown Cells

Select the top cell in command mode (blue border). Now press the **M** key on your keyboard. Notice that the "**In [ ]:**" to the left of the cell has disappeared. This indicates that the cell has been converted into a **markdown cell**. Markdown cells are used to embed formatted text inside a notebook. Such text might be used to explain code contained in code cells, to provide interpretation of the results of the code, or for other purposes. The top two cells of your notebook appear as in the image below.

```
In [ ]:
```

Click inside of the text box for the markdown cell. Type the following text within this cell. Replace the text "Your Name" with your first and last name. Replacing the due date with the actual due date for this assignment, as indicated on Canvas.

```
COSC 130 – Homework 0

Your Name

Due Wednesday, 5/06
```

We can render the contents of markdown cells by pressing **CTRL+ENTER**. After rendering the markdown cell, you should see the following:

COSC 130 - Homework 0

Robbie Beane

In [ ]: ▶|

One benefit of working with markdown cells is that we can add formatting to the text contained in these cells. Double-click on the markdown cell to put it back into edit mode, and then make the following changes to the contents of the cell:

1. Add one pound signs and a space (#) to the beginning of the first line in the cell. This will covert this text into a level 2 header.
2. Then Add two asterisks (**) in front of your name, and two immediately after your name, with no spaces between. This will cause your name to be displayed in bold.

After making these changes, the content of your markdown cell (still in edit mode) will be as follows:

COSC 130 - Homework 0

Robbie Beane|

In [ ]: ▶|

Press **CTRL+ENTER** to render this markdown cell. Your results should appear as shown in the image below.

COSC 130 - Homework 0

Robbie Beane

In [ ]: ▶|

Convert the second cell of your notebook to a markdown cell (by selecting it and pressing **M**). Enter the following text into the cell, exactly as it appears here.

Part 1: Arithemetic Operations

In this section of the assignment, we will demonstrate the following concepts:
* Addition
* Subtraction
* Multiplication
* Division
* Exponentiation|

Press **CTRL+ENTER** to render this cell. The results will appear as shown below.

Part 1: Arithemetic Operations

In this section of the assignment, we will demonstrate the following concepts:

- Addition
- Subtraction
- Multiplication
- Division
- Exponentiation

Arithmetic Operations

In the first five of the six remaining code cells, enter the Python commands shown below. Enter one command per code cell, and execute each cell using **CTRL+ENTER**. You should get the results shown in the image below.

```
print(7 + 2)
print(7 - 2)
print(7 * 2)
print(7 / 2)
print(7 ** 2)
```

These commands calculate the appropriate arithmetic operations, and then print the results. Note that one peculiarity of Python is that it uses a double asterisk (**) to represent exponentiation.

```
In [2]:  ▶  print(7 + 2)

            9

In [3]:  ▶  print(7 - 2)

            5

In [4]:  ▶  print(7 * 2)

            14

In [5]:  ▶  print(7 / 2)

            3.5

In [6]:  ▶  print(7 ** 2)

            49

In [ ]:  ▶
```

Error Messages

You should have one empty code cell left. If not, then please create one at the bottom of your notebook. Then enter the following code into the cell, and run the cell: `print(7 / 0)`

The output you should get is shown in the image below. Unsurprisingly, Python does not allow us to divide by zero. If we attempt to, we will generate an error. Notice that the last line of the output reads:

```
ZeroDivisionError: division by zero
```

This message tells us what type of error Python encountered. There are many types of errors that can be generated in Python. You should try to become familiar with the various error messages, as well as the types of errors that can cause them. This can help you to identify the source of an error when working with a complex set of Python instructions.

```
In [7]:  ▶  print(7 / 0)

            -------------------------------------------------------------------------
            ZeroDivisionError                         Traceback (most recent call last)
            <ipython-input-7-3ac1fa743a27> in <module>
            ----> 1 print(7 / 0)

            ZeroDivisionError: division by zero
```

We do not want to have any errors in our notebook, so please delete this last cell by selecting it in command mode (blue border) and then pressing the **D** key twice.

Variables

Now that we are moderately familiar with the basics of working in Jupyter Notebooks, we will get a (basic) introduction to variables, which are a very import concept in any programming language.

Please add six new cells at the bottom of your notebook. Convert the first new cell to a markdown cell, and enter text into it to generate the results shown below. You can accomplish this by modifying the markdown code you used for Part 1.

Part 2: Variables

In this section of the assignment, we will demonstrate the following concepts:

- Creating variables
- Printing the value of variables
- Operations with variables
- Nonlinear execution in notebooks

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

In the first empty code cell, enter the code: **x** = **42**. This creates a new variable, named **x**, and containing the value of 42.

```
In [8]:   x = 42
```

We can use variables we have created in any cells within our notebook. In the next code cell, enter the command `print(x)`. Run this cell to confirm that the variable **x** is evaluated as the number **42**.

```
In [9]:   print(x)

          42
```

In the next two cells, create a variable **y**, set to be equal to 7. The print the value of **y**.

```
In [10]:   y = 7
```

```
In [11]:   print(y)

           7
```

You can use variable in arithmetic operations. When you do, the variables are treated as being equal to the values that they store. In the next cell, print the result obtained from dividing **x** by **y**.

```
In [12]:   print(x / y)

           6.0
```

Nonlinear Execution

It is natural to view the commands contained in the code cells of a Jupyter notebook as being in order from top to bottom. However, we are free to run the code cells in any order that we choose. If the order in which we run the cells differs from the natural top-to-bottom order, then we could get unexpected results. We illustrate this concept with an example.

Select the cell containing the command **x = 42** by clicking on the left of the cell to stay in command mode. Press the **A** key on your keyboard to insert a new cell ABOVE the selected cell. You should see the following:

```
In [ ]:   ▶│

In [8]:   ▶│  x = 42

In [9]:   ▶│  print(x)
              42
```

In the newly created cell, enter the command: **x = 91**. Run this cell. This will overwrite the current value of **x**, which is **42**, with a new value of **91**. To confirm that the value of **x** is now **91**, re-run the cell with the command **print(x)**.

```
In [12]:   ▶│  x = 91

In [8]:    ▶│  x = 42

In [13]:   ▶│  print(x)
               91
```
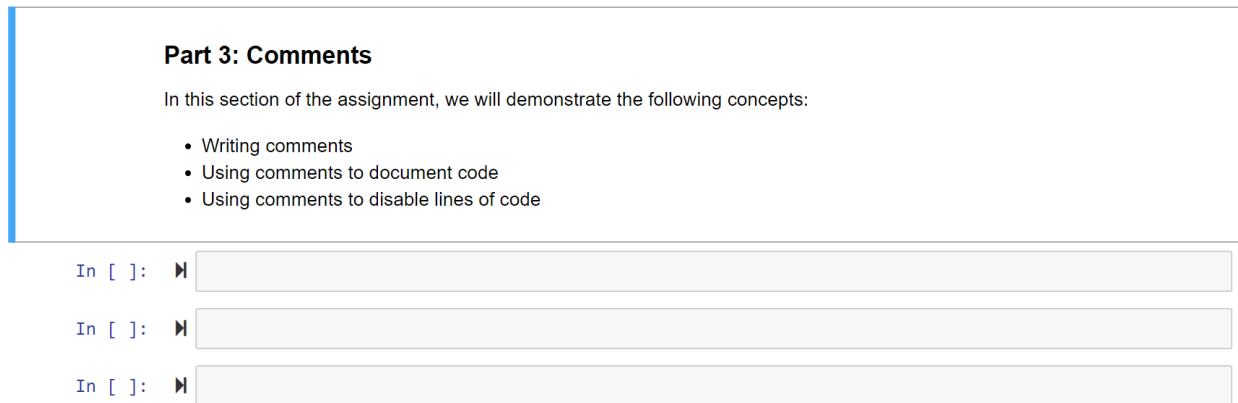
Next, re-run the cell with the command **print(x / y)**. You should see that this cell will use the new value store in **x** to perform its calculation.

```
In [12]:   ▶│  x = 91

In [8]:    ▶│  x = 42

In [13]:   ▶│  print(x)
               91

In [10]:   ▶│  y = 7

In [11]:   ▶│  print(y)
               7

In [14]:   ▶│  print(x / y)
               13.0
```

Notice that anyone reading the commands in the image above from top to bottom would likely be confused about how the results shown were obtained. For this reason, it is generally not advisable to run code cells out of order. It is, at times, convenient to be able to do so, but you should take care to make sure that your code performs the tasks you intend for it to when ran from top to bottom. We will discuss this idea again later in the lecture. For now, leave the code in the section as-is.

Comments

A comment in Python is a piece of text contained inside a code cell, but that is not intended to be executed by Python. We will discuss comments in Part 3 of this assignment. First, add four new cells to your notebook. Then convert the first one to a markdown cell that renders as shown in the image below.

Part 3: Comments

In this section of the assignment, we will demonstrate the following concepts:

- Writing comments
- Using comments to document code
- Using comments to disable lines of code

```
In [ ]:  ▶|
```

```
In [ ]:  ▶|
```

```
In [ ]:  ▶|
```

Type the text shown in the image below, exactly how it appears, in the first empty cell in your notebook.

```
In [ ]:  ▶|  This is a comment
             print(2 + 5)
```
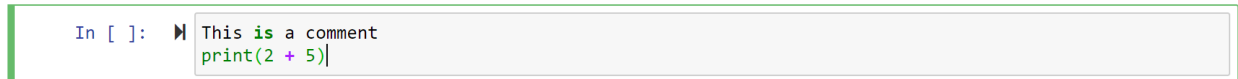
Now run this cell. When you do, you will generate a `SyntaxError`. This means that Python encountered a command that it did not understand. In this case, Python tried to interpret the first line of the cell (`This is a comment`) as executable code. This is not valid Python code, and so we get an error.

```
In [15]:  ▶|  This is a comment
              print(2 + 5)

                File "<ipython-input-15-feecefb68d55>", line 1
                  This is a comment
                                  ^
              SyntaxError: invalid syntax
```

Add a pound sign and a space to the beginning of the first line in the cell pictured above. Notice that this causes the first line to turn green. The pound sign at the beginning of this line signals to Python that this line represents a comment rather than executable code. As a result, Python will simply ignore this line when the cell is run. To see that, run the cell now. You should see that the first line is skipped, and the second line is executed.

```
In [16]:  ▶|  # This is a comment
              print(2 + 5)

              7
```
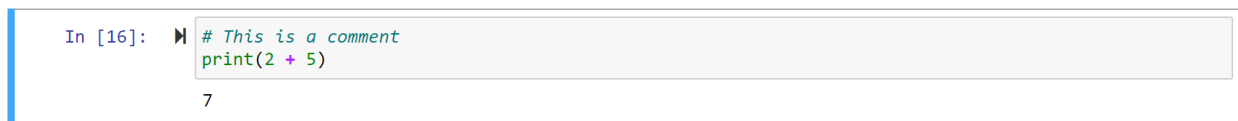
One use of comments is to document or annotate code. In other words, we can include comments in our code to explain to anyone reading our code (including ourselves) what purpose the code is intended to serve.

Copy the text contained in the image below into the next available cell, and then run that cell. Notice that the first line of the cell is a comment that explains the purpose of the cell. Notice also that lines 2 – 4 contain comments, as well as executable code. Any time a pound sign appears in a line of code, everything after that symbol will be interpreted as a comment. Anything that appears on the line prior to the pound sign will still be treated as executable code.

```
In [17]:  ▶| # In this cell, we will calculate sales revenue.
          sales = 173              # Monthly widget sales
          price = 16.98            # Price per widget
          revenue = sales * price  # Monthly revenue
          print(revenue)

          2937.54
```

Another common use of comments is to "turn off" lines of code that we do not wish to run, but that we don't want to delete. This can be a useful tool for debugging your programs. To give a (simple) illustration of this concept, please copy the code that appears in the cell below into the next empty code cell in your notebook. Then execute the cell.

```
In [18]:  ▶| print(12 / 6)
          print(12 / 0)
          print(12 / 4)

          2.0

          ---------------------------------------------------------------------
          ZeroDivisionError                           Traceback (most recent call last)
          <ipython-input-18-a84015234320> in <module>
                1 print(12 / 6)
          ----> 2 print(12 / 0)
                3 print(12 / 4)

          ZeroDivisionError: division by zero
```

Notice that the first line of code in this cell executed, printing a result of 2.0, which is correct. However, we attempt to divide by zero in the second line of code, resulting in an error. When Python encounters an error, it will stop executing commands. As a result, the third line of code in this cell is never executed.

Since the second line generates an error, we need to delete it or disable it. Assume that, for whatever reason, we do not yet wish to delete this line. Then we can instead "comment it out" by placing a pound sign at the start of the line. Please do that, and then run the cell again. This time, you should see that the first and third lines of code are executed perfectly, and no error is generated.

```
In [19]:  ▶| print(12 / 6)
          # print(12 / 0)
          print(12 / 4)

          2.0
          3.0
```

Strings

So far, we have worked only with numerical information in Python. We will occasionally need to work with text information as well. A string in Python is a piece of data that represents text-based information. We will close out this assignment with a discussion of strings.

Please add 5 new cells at the end of your notebook. Convert the first cell to a markdown cell, and then add content to this cell in order to generate the result shown below. Render this cell.

Part 4: Strings

In this section of the assignment, we will demonstrate the following concepts:

- Creating strings
- Storing strings in variables
- Printing multiple items

```
In [ ]:  ▶|
```

```
In [ ]:  ▶|
```

```
In [ ]:  ▶|
```

```
In [ ]:  ▶|
```

Suppose that we wish to write a Python program that outputs a message reading "Hello, world!" to the user. Attempt this by copying the content of the cell shown below into the first empty code cell of your notebook, exactly as it appears here. Then run this cell.

```
In [20]:  ▶|  print(Hello, world!)

            File "<ipython-input-20-d14e6f3d0ea2>", line 1
              print(Hello, world!)
                                  ^
          SyntaxError: invalid syntax
```

Notice that this generates a SyntaxError. Python attempts to process the message Hello, world! as executable code, and then gets confused. We can fix this error by converting this message to a string. To convert this message to a string, simply surround the message with quotation marks, as shown in the cell below. Then re-run the cell.

```
In [21]:  ▶|  print("Hello, world!")

          Hello, world!
```

Notice that the cell executes without error this time. The quotation marks tell Python that the contents of the message are not to be treated as code and are instead inert text data. Notice that a string is different from a comment. When Python encounters a comment, it simply ignores it. When Python encounters a string, it can still perform actions on it, but will treat it as inert text.

We can store strings into variables. In the next available cell, create a variable called name and set it equal to a string that contains your first and last name. In the same cell, add a command to print this variable.

```
In [22]:  ▶|  name = "Robbie Beane"
              print(name)

          Robbie Beane
```

Printing Multiple Items

We have used the print() function several times to display the values of variables and the results of calculations. So far, we have only ever used print() to display a single value at a time. As it turns out, we can ask a single print() statement to display multiple pieces of information at once. We simply have to include all of the pieces of information inside the

parentheses of the `print()` function, separated by commas. This can be useful for printing results that combine static messages with values stored in variables.

Reproduce the code in the image below within the next available cell in your notebook. Then execute the cell.

```
In [23]:  ▶| print("Hello, my name is", name)

          Hello, my name is Robbie Beane
```

Suppose that we would like to display a period at the end of the sentence above. We can do so by adding a third item to be printed. This third item will be a string containing a single character, `"."`. Use the last cell of your notebook to make this modification.

```
In [24]:  ▶| print("Hello, my name is", name, ".")

          Hello, my name is Robbie Beane .
```

Notice that the result is somewhat strange. We end up with a space between the last word of the sentence and the period. That is because when we use `print()` to display multiple items, the function automatically inserts a space between each item. We can change this behavior by including the code `sep=""` inside of the `print()` function. This tells `print()` to use a string with no characters (called an empty string) as the separator, rather than using a space, which is the default separator. Notice that to keep pieces from running together, we now need to add a space to the end of the string `"Hello, my name is"`.

Modify your last code cell as indicated in the image below, and then re-run this cell.

```
In [25]:  ▶| print("Hello, my name is ", name, ".", sep="")

          Hello, my name is Robbie Beane.
```

E. Submitting your Assignment

We will now walk through the process of submitting your assignment. You will use the same process for every homework assignment and project in this course.

Running the Notebook

When you are finished with a notebook, it is important to restart Python and then run the cells in the notebook in order from top to bottom. Jupyter provides us with a convenient method of doing this. Click on **Kernel** in the menu bar at the top of the screen. Then select **Restart & Run All**. It might take a few seconds, but this will restart Python and then run the entire notebook. Carefully review the results within your notebook. If you didn't encounter any errors, and you got the output that you expected, then you are ready to submit your assignment!
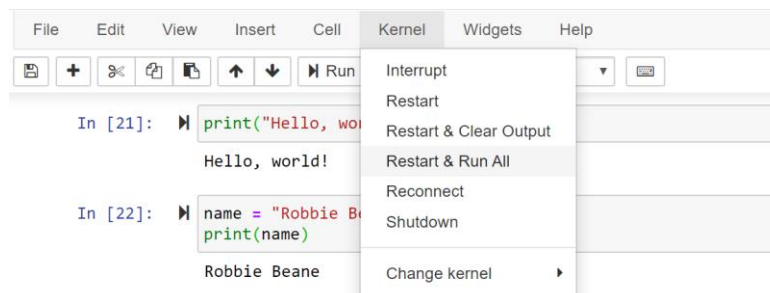
Generating HTML Render of Notebook

We can render the contents of a Jupyter Notebook as a static html file. This can be useful for sharing the results of our analysis with other individuals who might not have Jupyter installed, and who might not need to be able to execute the code. In this class, I will ask you to submit BOTH an html version and an ipynb version of every assignment.

To render your notebook as HTML, please click **File > Download as > HTML (.html)**. You will be prompted to provide a location to save the new file to. I recommend saving the .html file to the directory that already contains your .ipynb file. If you are working within CoCalc, you will need to download the resulting HTML file to your local machine.

Uploading your Assignment Files

When you are finished with this assignment, please upload the .ipynb notebook file to CoCalc and upload the .html file to Canvas.

If you completed the assignment on your local machine, please log in to CoCalc, navigate inside the folder associated with the assignment you are submitting (which would be "**/Homework/HW 00**" in this case), and upload your notebook file to this folder. You can either use the **Upload** button that you see in CoCalc, or you can simply drag and drop the file into the browser window displaying the contents of your folder. Make sure to also upload your .html output to Canvas by clicking on the relevant assignment within Canvas.

If you completed the assignment on CoCalc, then you do not need to do anything special with your .ipynb notebook file (it is already in CoCalc). You will, however, need to download your html output from CoCalc and then upload it to Canvas. You can find the upload option by clicking on the relevant assignment within Canvas.

Please let me know if you have any questions!

S