```
In [ ]: # Initialize Otter
        import otter
        grader = otter.Notebook("lab1.ipynb")
```

Table of Contents

1 Lab 1

Welcome to the first lab of PSTAT 134! This lab is meant to help you familiarize yourself with JupyterHub, review Python and NumPy.

1.1 Part 1: Using Jupyter Notebook

1.1.1 Running Cells and Displaying Output

Run the following cell. If you are unfamiliar with Jupyter Notebooks, skim this tutorial or selecting **Help –> JupyterLab Reference** in the menu bar above.

```
In [1]: print("Hello World!")
```

```
Hello World!
```

In Jupyter notebooks, all print statements are displayed below the cell. Furthermore, the output of the last line is displayed following the cell upon execution.

```
In [2]: "Will this line be displayed?"

        print("Hello" + ",", "world!")

        5 + 3
```

```
Hello, world!
```

```
Out[2]: 8
```

1.1.2 Viewing Documentation

To output the documentation for a function, use the `help` function.

```
In [3]: help(print)
```

```
Help on built-in function print in module builtins:

print(…)
    print(value, …, sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

You can also use Jupyter to view function documentation inside your notebook. The function must already be defined in the kernel for this to work.

Below, click your mouse anywhere on `print()` and use `Shift` + `Tab` to view the function's documentation.

```
In [4]: print('Welcome to PSTAT 134.')
```

```
Welcome to PSTAT 134.
```

1.1.3 Importing Libraries and Magic Commands

In PSTAT 134, we will be using common Python libraries to help us process data. By convention, we import all libraries at the very top of the notebook. There are also a set of standard aliases that are used to shorten the library names. Below are some of the libraries that you may encounter throughout the course, along with their respective aliases.

```
In [5]: import pandas as pd
        import numpy as np
```

Another useful magic command is `%%time`, which times the execution of that cell. You can use this by writing it as the first line of a cell. (Note that `%%` is used for *cell magic commands* that apply to the entire cell, whereas `%` is used for *line magic commands* that only apply to a single line.)

```
In [6]: %%time

        lst = []
        for i in range(100):
            lst.append(i)
```

```
CPU times: user 8 µs, sys: 12 µs, total: 20 µs
Wall time: 24.3 µs
```

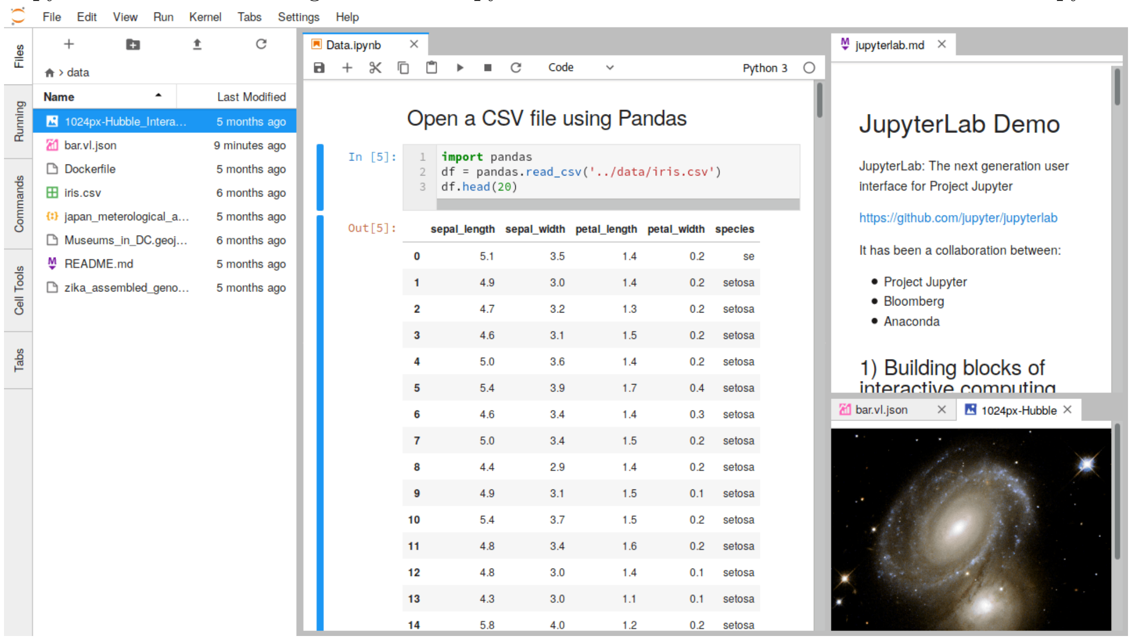
1.1.4 Keyboard Shortcuts

Even if you are familiar with Jupyter, we strongly encourage you to become proficient with keyboard shortcuts (this will save you time in the future). To learn about keyboard shortcuts, go to **Help –> Keyboard Shortcuts** in the menu above.

Here are a few that we like: 1. `Ctrl` + `Return` : *Evaluate the current cell* 1. `Shift` + `Return`: *Evaluate the current cell and move to the next* 1. `ESC` : *command mode* (may need to press before using any of the commands below) 1. `a` : *create a cell above* 1. `b` : *create a cell below* 1. `dd` : *delete a cell* 1. `z` : *undo the last cell operation* 1. `m` : *convert a cell to markdown* 1. `y` : *convert a cell to code*

1.1.5 Jupyter Lab

JupyterLab is the next generation Jupyter environment that includes more than Jupyter Notebook.



Name	Last Modified
1024px-Hubble_Intera...	5 months ago
bar.vl.json	9 minutes ago
Dockerfile	5 months ago
iris.csv	6 months ago
japan_meterological_a...	5 months ago
Museums_in_DC.geoj...	6 months ago
README.md	5 months ago
zika_assembled_geno...	5 months ago

Open a CSV file using Pandas

```
In [5]:  import pandas
         df = pandas.read_csv('../data/iris.csv')
         df.head(20)
```

Out[5]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	se
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa
10	5.4	3.7	1.5	0.2	setosa
11	4.8	3.4	1.6	0.2	setosa
12	4.8	3.0	1.4	0.1	setosa
13	4.3	3.0	1.1	0.1	setosa
14	5.8	4.0	1.2	0.2	setosa

JupyterLab Demo

JupyterLab: The next generation user interface for Project Jupyter

https://github.com/jupyter/jupyterlab

It has been a collaboration between:

- Project Jupyter
- Bloomberg
- Anaconda

1) Building blocks of interactive computing

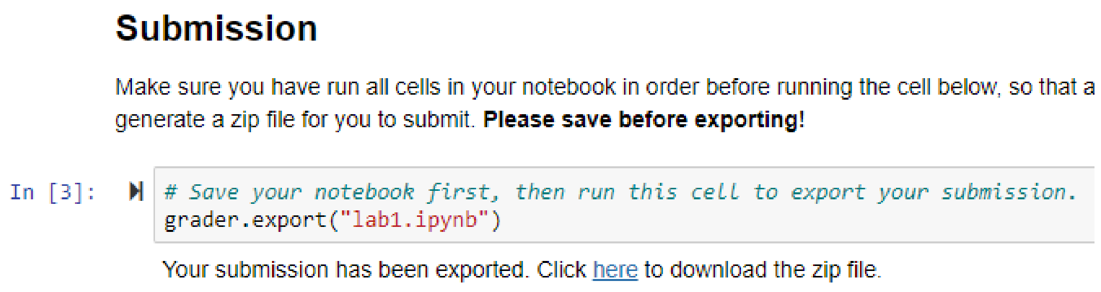
To use Jupyter Lab, visit http://pstat134.lsit.ucsb.edu/lab. In this class, using Jupyter Lab is optional.

1.2 Part 2: Prerequisites

1.2.1 Formatting for autograding

You will be asked to upload your work directly to Gradescope.

At the bottom of the notebook, you can generate the zip file you will upload to Gradescope.



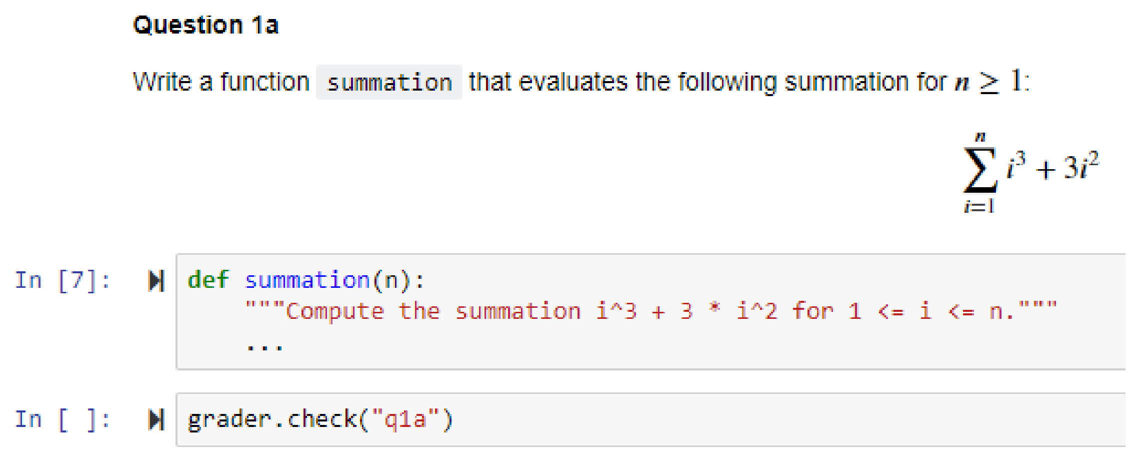
Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that a generate a zip file for you to submit. **Please save before exporting!**

```
In [3]:  # Save your notebook first, then run this cell to export your submission.
         grader.export("lab1.ipynb")
```

Your submission has been exported. Click here to download the zip file.

Gradescope will grade it using automated scripts, which require a specific structure in your notebooks. Following is a screenshot of the question from below:



Question 1a

Write a function `summation` that evaluates the following summation for $n \geq 1$:

$$\sum_{i=1}^{n} i^3 + 3i^2$$

```
In [7]:  def summation(n):
             """Compute the summation i^3 + 3 * i^2 for 1 <= i <= n."""
             ...
```

```
In [ ]:  grader.check("q1a")
```

Instructions The software being used is called otter grader. Here are the basics:

- **Each question is marked by a code chunk: e.g.**
  `<!--     BEGIN QUESTION     name: q1a     -->`
- **Each question has a response cell *directly* below it.**
  This is where you answer the question. If you have extra content in this cell, that's fine.
- **If a question has tests, grading cell follows the response cell.**
  Most response cells are followed by a test cell that runs automated tests to check your work: e.g. `grader.check("q1a");` Test results are meant to give you some useful feedback, but it's your responsibility to answer the question.
- **Keep Question-Response-Grading cell intact**
  Please don't delete questions, response cells, or test cells. You won't get credit for your work if you do.
- **Important: test cells don't always confirm that your response is correct.**
  There may be other tests that we run when scoring your notebooks. We **strongly recommend** that you check your solutions yourself rather than just relying on the test cells.
- **Double check your submission**
  Make sure the correct file is submitted

Please refer to this documentation for how you can use it more effectively: https://otter-grader.readthedocs.io/en/latest/otter_check.html

1.2.2 Python

Python is the main programming language we'll use in the course. We expect that you've taken a programming class, and you are able to pick up on Python syntax as needed. We will not be covering general

Python syntax. If any of the below exercises are challenging (or if you would like to improve your Python knowledge), please review one or more of the following materials.

- **Software Carpentry Python**

Question 1a Write a function `summation` that evaluates the following summation for $n \geq 1$:

$$\sum_{i=1}^{n} i^3 + 3i^2$$

```
In [7]: def summation(n):
            """Compute the summation i^3 + 3 * i^2 for 1 <= i <= n."""
            # BEGIN SOLUTION
            if n < 1:
                raise ValueError("n must be greater than or equal to 1")
            return sum([i ** 3 + 3 * i ** 2 for i in range(1, n + 1)])
            # END SOLUTION
```

```
In [ ]: grader.check("q1a")
```

Question 1b Write a function `list_sum` that computes the square of each value in `list_1`, the cube of each value in `list_2`, then returns a list containing the element-wise sum of these results. Assume that `list_1` and `list_2` have the same number of elements.

```
In [11]: def list_sum(list_1, list_2):
             """Compute x^2 + y^3 for each x, y in list_1, list_2.

             Assume list_1 and list_2 have the same length.
             """
             assert len(list_1) == len(list_2), "both args must have the same number of elements"
             # BEGIN SOLUTION
             return [x ** 2 + y ** 3 for x, y in zip(list_1, list_2)]
             # END SOLUTION
```

```
In [ ]: grader.check("q1b")
```

Question 1c Write a function `average` that takes a number and returns the average of all inputs on which it has ever been called. *Challenge:* Can you do it without any global names besides `average`?

```
In [18]: def average(n):
             """Return the average of all arguments ever passed to the average function.

             >>> average(1)
             1.0
             >>> average(3)
             2.0
             >>> average(8)
             4.0
             >>> average(0)
             3.0
             """
             # BEGIN SOLUTION
             average.total += n
             average.k += 1
             return average.total / average.k
         average.total, average.k = 0, 0
             # END SOLUTION
         # Please don't call average here, or you'll confuse the automated tests.
```

```
In [ ]: grader.check("q1c")
```

1.2.3 NumPy

NumPy is the numerical computing module introduced in Data 8, which is a prerequisite for this course. Here's a quick recap of NumPy. For more review, read the following materials.

- **NumPy Quick Start Tutorial**
- **Stanford CS231n NumPy Tutorial**

Question 2 The core of NumPy is the array. Like Python lists, arrays store data; however, they store data in a more efficient manner. In many cases, this allows for faster computation and data manipulation.

In Data 8, we used `make_array` from the `datascience` module, but that's not the most typical way. Instead, use `np.array` to create an array. It takes a sequence, such as a list or range.

Below, create an array `arr` containing the values 1, 2, 3, 4, and 5 (in that order).

```
In [23]: arr = np.array([1, 2, 3, 4, 5]) # SOLUTION
```

```
In [ ]: grader.check("q2")
```

In addition to values in the array, we can access attributes such as shape and data type. A full list of attributes can be found here.

```
In [26]: arr[3]
```

```
Out[26]: 4
```

```
In [27]: arr[2:4]
```

```
Out[27]: array([3, 4])
```

```
In [28]: arr.shape
```

```
Out[28]: (5,)
```

```
In [29]: arr.dtype
```

```
Out[29]: dtype('int64')
```

Arrays, unlike Python lists, cannot store items of different data types.

```
In [30]: # A regular Python list can store items of different data types
         [1, '3']
```

```
Out[30]: [1, '3']
```

```
In [31]: # Arrays will convert everything to the same data type
         np.array([1, '3'])
```

```
Out[31]: array(['1', '3'], dtype='<U21')
```

```
In [32]: # Another example of array type conversion
         np.array([5, 8.3])
```

```
Out[32]: array([5. , 8.3])
```

Arrays are also useful in performing *vectorized operations*. Given two or more arrays of equal length, arithmetic will perform element-wise computations across the arrays.

For example, observe the following:

```
In [33]: # Python list addition will concatenate the two lists
         [1, 2, 3] + [4, 5, 6]
```

```
Out[33]: [1, 2, 3, 4, 5, 6]
```

```
In [34]: # NumPy array addition will add them element-wise
         np.array([1, 2, 3]) + np.array([4, 5, 6])
```

```
Out[34]: array([5, 7, 9])
```

Question 3a Given the array `random_arr`, assign `valid_values` to an array containing all values x such that $2x^4 > 1$.

```
In [35]: np.random.seed(42)
         random_arr = np.random.rand(60)
         valid_values = random_arr[(2 * random_arr  ** 4 > 1)] # SOLUTION
```

```
In [ ]: grader.check("q3a")
```

Question 3b Use NumPy to recreate your answer to Question 1b. The input parameters will both be lists, so you will need to convert the lists into arrays before performing your operations.

Hint: Use the NumPy documentation. If you're stuck, try a search engine! Searching the web for examples of how to use modules is very common in data science.

```
In [37]: def array_sum(list_1, list_2):
             """Compute x^2 + y^3 for each x, y in list_1, list_2.

             Assume list_1 and list_2 have the same length.
```

```
            Return a NumPy array.
            """
            assert len(list_1) == len(list_2), "both args must have the same number of elements"
            # BEGIN SOLUTION
            # Solution 1
            return np.square(list_1) + np.power(list_2, 3)
            # Solution 2
            return np.array(list_1) ** 2 + np.array(list_2) ** 3
            # END SOLUTION
```

```
In [ ]: grader.check("q3b")
```

Question 4: This is manually graded problem. This question will be exported to a PDF file. Try changing the solution cell below and inspect the pdf.

Type your answer here, replacing this text.

SOLUTION: Answer goes here

1.2.4 Formatting for autograding, continued

Note what happens when you re-run all tests again. The command to run all tests again is

```
grader.check_all()
```

In particular, `q1c` is now wrong. Looking at the values, what do you think happened? You want to be extremely careful that the *all* tests run correctly.

To double-check your work, the cell below will rerun all of the autograder tests.

```
In [ ]: grader.check_all()
```

1.3 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

```
In [ ]: # Save your notebook first, then run this cell to export your submission.
        grader.export()
```