# Setting Up the Environment

## Create a working directory on your computer

1. From a terminal,
   - `cd <place_where_my_class_files_are_stored>`
   - `mkdir AE470`
2. Or, using File Explorer, navigate to `<place_where_my_class_files_are_stored>` and create course folder `AE470`
3. Navigate into the class working directory,
   - `cd AE470`

## Install git

We will be using `git` to download the Jupyter notebooks from the course GitHub repository.

1. Download and install the latest version of git from
   - https://git-scm.com/downloads

## Install Anaconda Python (or use Clarkson's anaconda3 in appsAnyWhere)

1. Download Anaconda installer from www.anaconda.com/download

## Create a virtual Python environment for this course

1. Inside an Anaconda Prompt window, navigate into the class working directory,
   - `cd <place_where_my_class_files_are_stored>\AE470`
2. Create a virtual environment by typing,
   - `conda create -n ae470sp25 python=3.12 numpy pandas scipy jupyter matplotlib sympy seaborn`
3. List all virtual Python environments,
   - `conda env list`
4. Activate the class virtual environment,
   - `conda activate ae470sp25`

## Download the course repository from github

1. Inside a terminal, navigate into the class working directory,
   - `cd <place_where_my_class_files_are_stored>\AE470`
2. Clone the course git repository to your computer,
   - `git clone https://github.com/JeffWalton/AE470_Sp25.git`

## Start Jupyter Notebook session

1. Open a Jupyter notebook session in a browser window. Type the following into an Anaconda Prompt window,
   - `jupyter notebook`
2. Using the Jupyter browser, open this notebook, `00_ae470_setup_environment.ipynb`.
3. Read through the notebook and execute cells.

## Explore Jupyter Notebooks

The following is directly from Allen Downey's Modeling and Simulation in Python

### Jupyter

Welcome to *Modeling and Simulation*, welcome to Python, and welcome to Jupyter.

This is a Jupyter notebook, which is a development environment where you can write and run Python code. Each notebook is divided into cells. Each cell contains either text (like this cell) or Python code.

#### Selecting and running cells

To select a cell, click in the left margin next to the cell. You should see a blue frame surrounding the selected cell.

To edit a code cell, click inside the cell. You should see a blue frame around the selected cell, and you should see a cursor inside the cell.

To edit a text cell, double-click inside the cell. Again, you should see a blue frame around the selected cell, and you should see a cursor inside the cell.

To run a cell, hold down SHIFT and press ENTER.

- If you run a text cell, Jupyter formats the text and displays the result.

- If you run a code cell, Jupyter runs the Python code in the cell and displays the result, if any.

To try it out, edit this cell, change some of the text, and then press SHIFT-ENTER to format it.

### Adding and removing cells

You can add and remove cells from a notebook using the buttons in the toolbar and the items in the menu, both of which you should see at the top of this notebook.

Try the following exercises:

1. From the Insert menu select "Insert cell below" to add a cell below this one. By default, you get a code cell, as you can see in the pulldown menu that says "Code".
2. In the new cell, add a print statement like `print('Hello')`, and run it.
3. Add another cell, select the new cell, and then click on the pulldown menu that says "Code" and select "Markdown". This makes the new cell a text cell.
4. In the new cell, type some text, and then run it.
5. Use the arrow buttons in the toolbar to move cells up and down.
6. Use the cut, copy, and paste buttons to delete, add, and move cells.
7. As you make changes, Jupyter saves your notebook automatically, but if you want to make sure, you can press the save button, which looks like a floppy disk from the 1990s.
8. Finally, when you are done with a notebook, select "Close and Shut Down" from the File menu.

### Using the notebooks

The notebooks contain the code along with additional examples, explanatory text, and exercises.

### Installing modules

These notebooks use standard Python modules like NumPy and SciPy. I assume you already have them installed in your environment.

modsim, is a module from Allen Downey's book. We will be using the module for this course.

The following code cells check whether you have these modules already and tries to install them if you don't.

In [ ]:
```
try:
    import pint
```

```
except ImportError:
    !pip install pint
    import pint
```

In [ ]:
```
try:
    from modsim import *
except ImportError:
    print("Download modsim from the course repository at https://github.com/jeffwal
```

You can find out what version of Python, Jupyter, and NumPy you have by running the following cells.

In [ ]:
```
!python --version
```

In [ ]:
```
!jupyter-notebook --version
```

In [ ]:
```
print(f"numpy version: {np.__version__}")
```

### Configuring Jupyter

The following cell:

1. Uses a Jupyter "magic command" to specify whether figures should appear in the notebook, or pop up in a new window.

2. Configures Jupyter to display some values that would otherwise be invisible.

Select the following cell and press SHIFT-ENTER to run it.

In [ ]:
```
# Configure Jupyter so figures appear in the notebook
%matplotlib inline

# Configure Jupyter to display the assigned value after an assignment
%config InteractiveShell.ast_node_interactivity='last_expr_or_assign'
```

## Basic Computations

In [ ]:
```
t = 10
```

In [ ]:
```
a = 9.81
```

In [ ]:
```
v = a * t
```

### Python Error Messages

Let's generate an error to see what happens by multiplying a number with a string variable.

In [ ]:
```
v = a * 'test'
```

The error messages you get from Python are (sometimes) big and scary, but if you read them carefully, they contain a lot of useful information.

1. Start from the bottom and read up.
2. The last line usually tells you what type of error happened, and sometimes additional information.
3. The previous lines are a "traceback" of what was happening when the error occurred. The first section of the traceback shows the code you wrote. The following sections are often from intermediate Python libraries.

In this example, you should get a `TypeError`, which indicates that you have violated a rules of multiplication: you cannot multiply quantities with different types.

Before you go on, you might want to delete the erroneous code so the notebook can run without errors.

## Restart and run all

When you change the contents of a cell, you have to run it again for those changes to have an effect. If you forget to do that, the results can be confusing, because the code you are looking at is not the code you ran.

If you ever lose track of which cells have run, and in what order, you should go to the Kernel menu and select "Restart & Run All". Restarting the kernel means that all of your variables get deleted, and running all the cells means all of your code will run again, in the right order.

**Exercise:** Select "Restart & Run All" now and confirm that it does what you want.

This is a *text* cell.

In [ ]:
```
a = 37
b=2
a*b
```

In [ ]:
```
```

In [ ]:
```
```