1 Setting Up Your Environment

1.1 Install Anaconda Distribution of Python 3.6

We will be using the **Anaconda** distribution of Python **3.6**, which comes pre-installed with several scientific computing libraries including `NumPy` and `Matplotlib`.

1. Download the Python 3.6 version from `https://www.anaconda.com/download/` for your specific operating system (OS), such as Windows, macOS, or Linux.

2. The detailed installation instruction steps are outlined in `https://docs.anaconda.com/anaconda/install/` for each OS. You do not need to install Microsoft Visual Studio Code when prompted. For Linux, you can skip step 2 (hash check) as it is optional.

1.2 Install PyCharm Community IDE

For our Integrated Development Environment (IDE), we will be using **PyCharm Community**. NOTE: it might be possible to obtain free student license for **PyCharm Professional**. Check `https://www.jetbrains.com/student/` for more details.

1. Download the latest **Community** version from `https://www.jetbrains.com/pycharm/download/` for your specific OS.

2. The installation instruction for PyCharm Community for each OS can be found at `https://www.jetbrains.com/help/pycharm/install-and-set-up-pycharm.html`.

During the Windows Installation: Check off the box for "Download and install JRE x86 by JetBrains", as shown in Figure 1. You can optionally check off to create a desktop shortcut and/or create associations with `.py` files to be opened automatically in PyCharm.

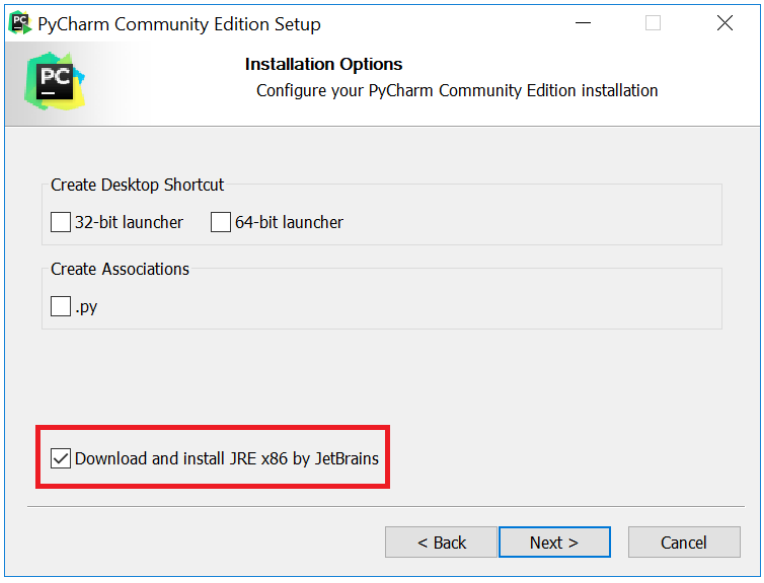

Figure 1: Windows PyCharm installation

For Mac Installation: Simply drag the PyCharm icon over to the "Applications" folder:

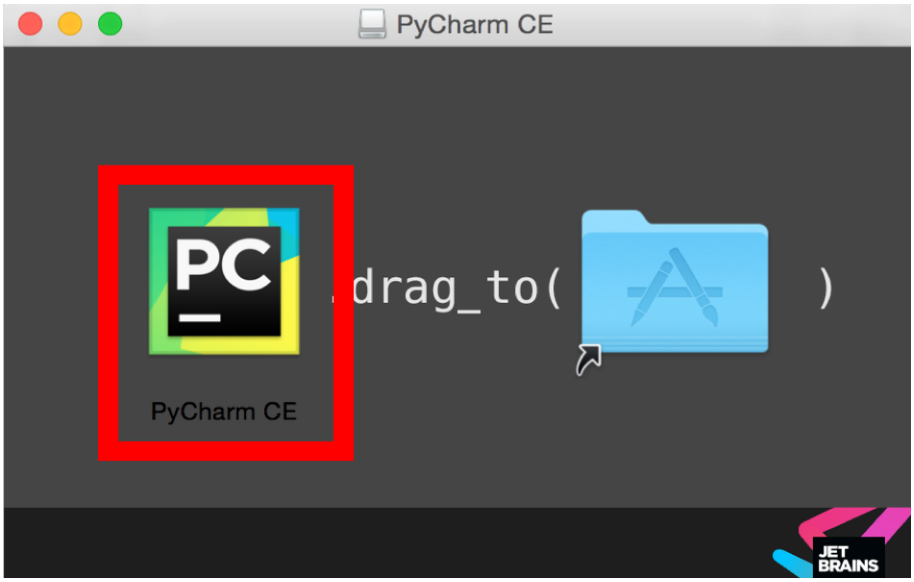

Figure 2: Mac PyCharm installation.

1.3 Setting up Anaconda Python environment

For this course, we will make use of a 'virtual environment' which isolates Python tools and libraries to be the right ones that we specify. You will create a virtual environment using Anaconda called `aps360`, using the 'conda' command. Do the following steps to create a conda virtual environment:

1. Open up a command line terminal. In Windows, you can search for "Command" and open Command Prompt. On Mac and Linux, you can open up the "Terminal" application

2. To create the virtual environment, run the command:

   ```
   conda create -n aps360 python=3.6 anaconda
   ```

   This process may take a while.

3. To test that the environment works, activate the environment by running:

   `source activate aps360` (Mac/Linux)
   `activate aps360` (Windows)

   You should see a `(aps360)` at the beginning of the line.

4. To exit from the environment, you can simply close the window, or run:

   `source deactivate` (Mac/Linux)
   `deactivate` (Windows)

   Then the `(aps360)` should disappear from the beginning of the line.

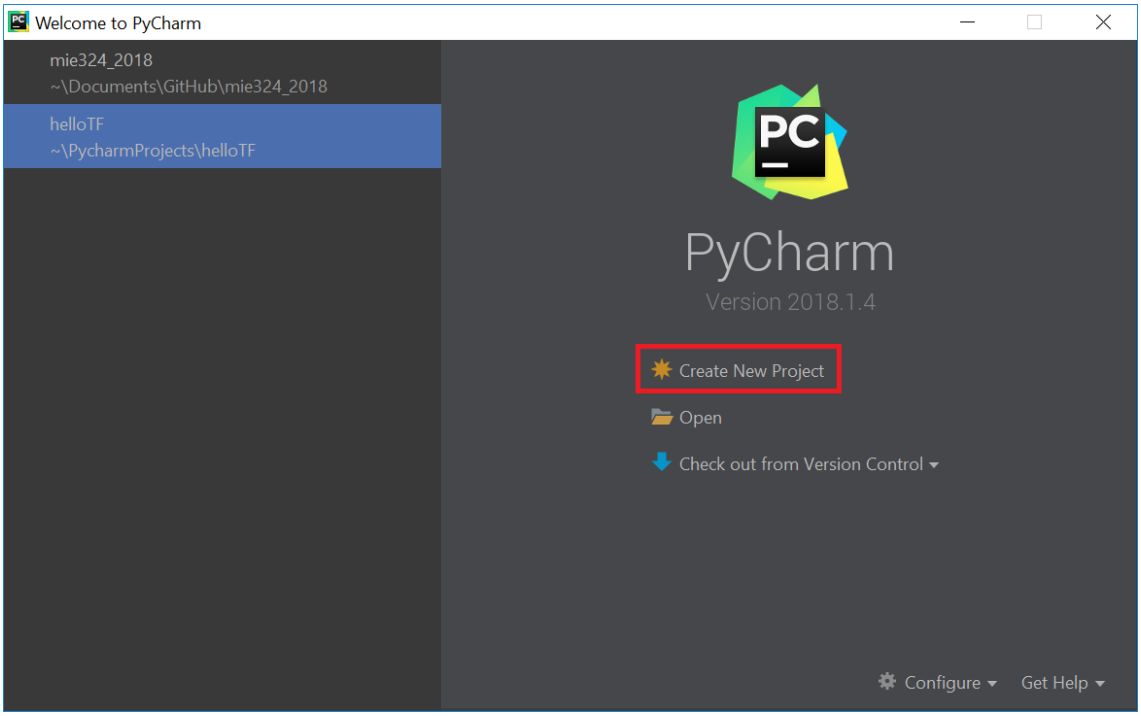

Figure 3: Starting PyCharm

1.4 Setting up PyCharm Community version to use Anaconda Python

Now, we will create a new project in PyCharm and configure it to use the `aps360` conda environment:

1. When starting PyCharm for the first time, click "Create a New Project". See Figure 3.
2. In the dialog box that comes after you click **"Create a New Project"** you should name the project as **"aps360"** as shown in Figure 4. Click on the **"Project Interpreter"** to dropdown and select the **"Existing interpreter"** option (#2 in the figure). By default it will not list your new conda environment in the drop down, so you will need to click on the **"..."** button (#3).
3. In the pop up window, select **"Conda Environment"** in the left panel (#1), as shown in Figure 5. Check off **"Make available to all projects"** (#2). In the Interpreter dropdown box, you should see an option that contains "aps360". Select that one and press **"OK"** (#4).
4. You should now see that the chosen interpreter is **"Python 3.6 (aps360)"**, as in Figure 6. Click **"Create"** to finish creating the project.

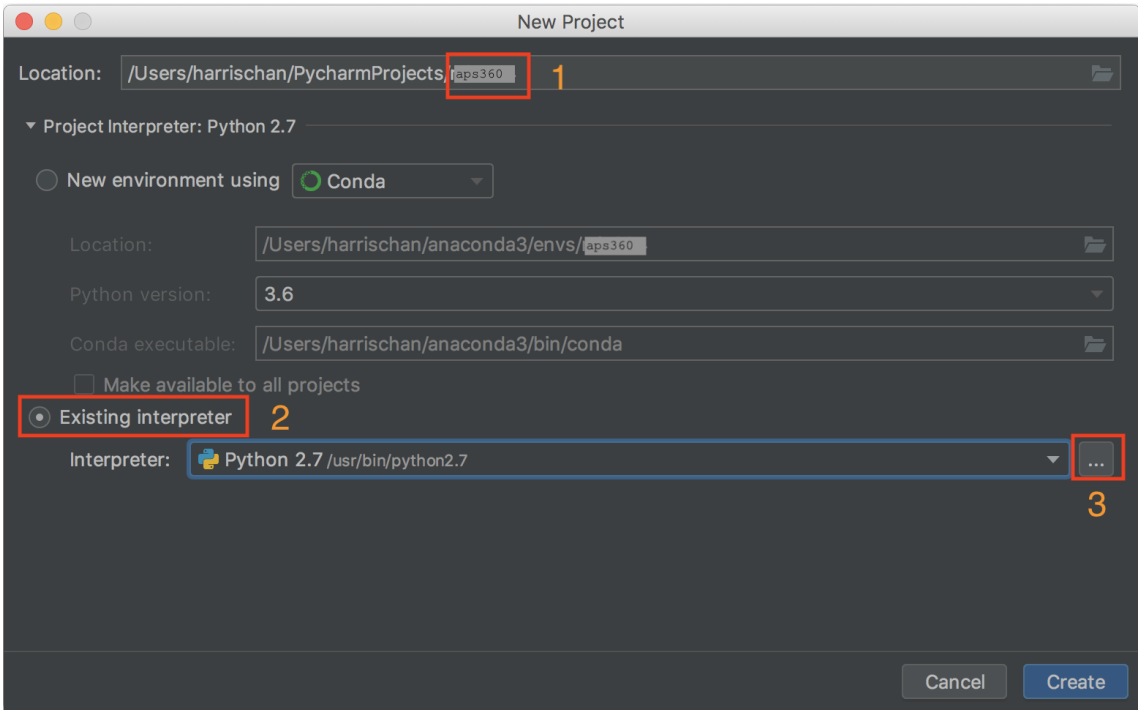

Figure 4: Naming your project and choosing your Python interpreter

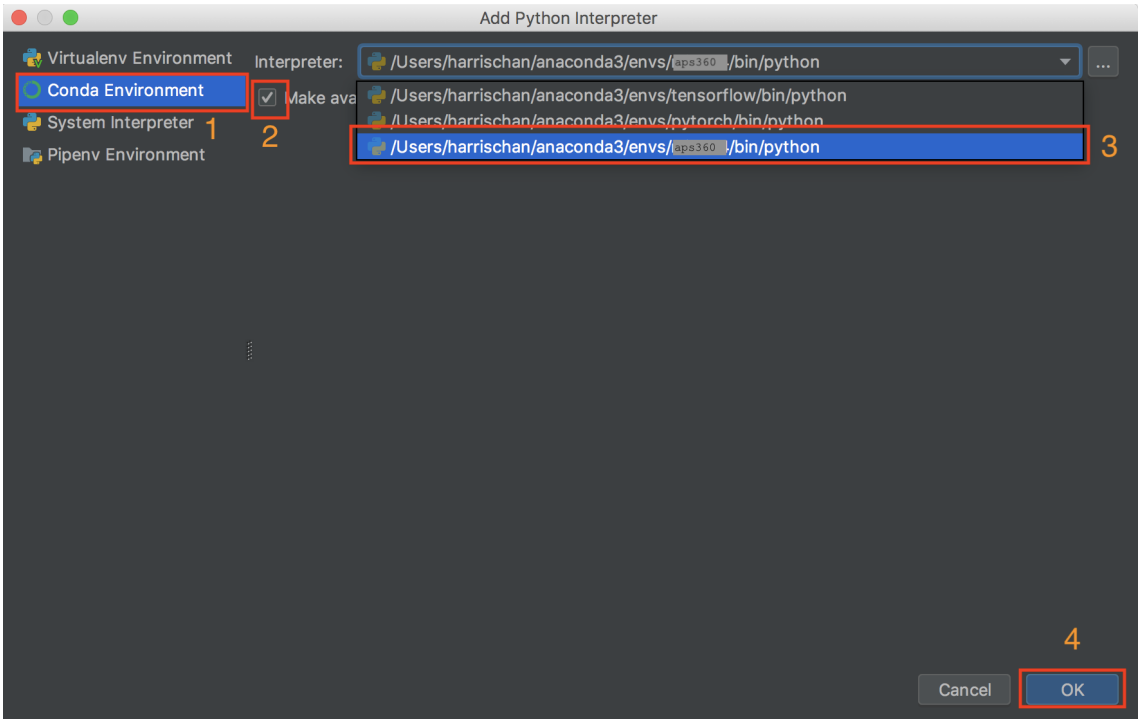

Figure 5: Selecting your aps360 environment for the Python interpreter

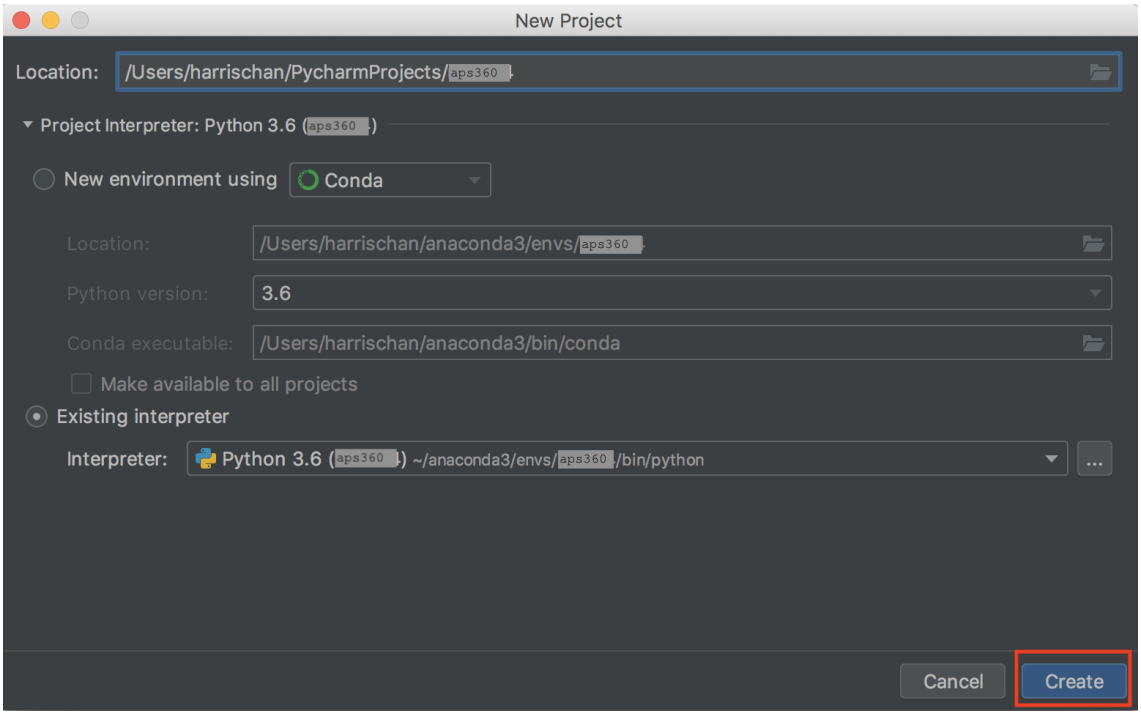

Figure 6: Confirming that you have selected the correct aps360 environment for your project.

1.5 Installing and Using Jupyter Notebook

Jupyter Notebook is a web application for interactive coding. The app is popular in machine learning / data science community because it is easy to perform quick prototyping and visualization using Jupyter Notebook's interactive web interface. In this course, assignments will use Jupyter Notebook.

1. Open up a command line terminal. Find the location of your project (i.e. path for your PyCharm project from Figure 4) and navigate to that directory by typing:

   ```
   cd /Path/to your/Project
   ```

2. Jupyter Notebook comes with Anaconda package. To start the app, activate "aps360" environment (section 1.3), and then type:

   ```
   jupyter notebook
   ```

3. Jupyter Notebook should start up with a message in the terminal similar to what is shown in Figure 7. A browser window should automatically open with Jupyter Notebook. If not, use URL from the terminal message (see red box in Figure 7).

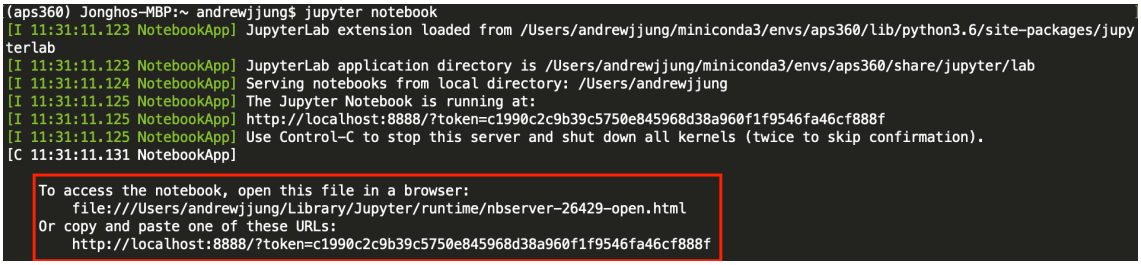

Figure 7: Terminal message after starting Jupyter Notebook

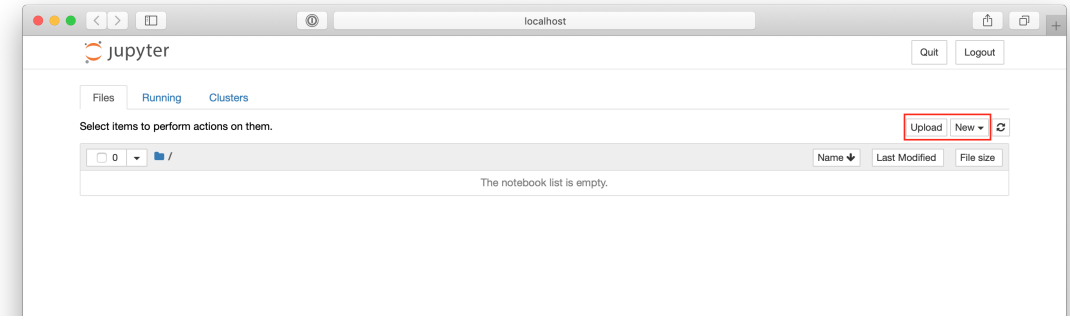

Figure 8: Dashboard screen when Jupyter Notebook starts in your browser

4. Jupyter Notebook's dashboard will show all the directories and files within the location you started Jupyter Notebook in command line terminal. As shown in Figure 8, there is nothing in this project folder yet. You can upload an existing Notebook file (files ending in .ipynb) or create a new one. Let's create a new Python 3 Notebook file (Figure 9)

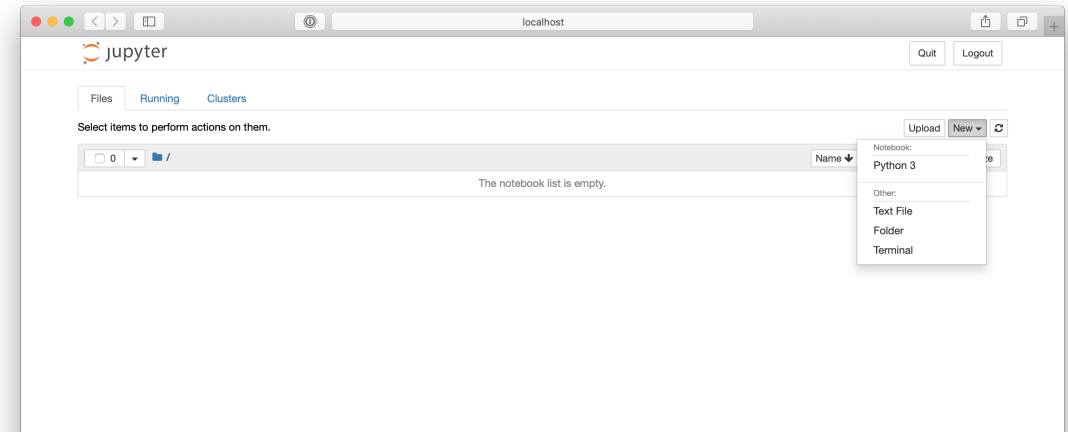

Figure 9: From dashboard, existing Notebook file can be uploaded or a new one can be created

5. A Notebook file is composed of "cells", blocks of code that can be run independently. You can add more cells by clicking the "+" button. You can type code normally as you do in IDE or texteditor. You can run a specific cell by clicking the cell and "Run" button at the top, or run multiple cells from "Cell" menu. See Figure 10 and Figure 11 for an example.

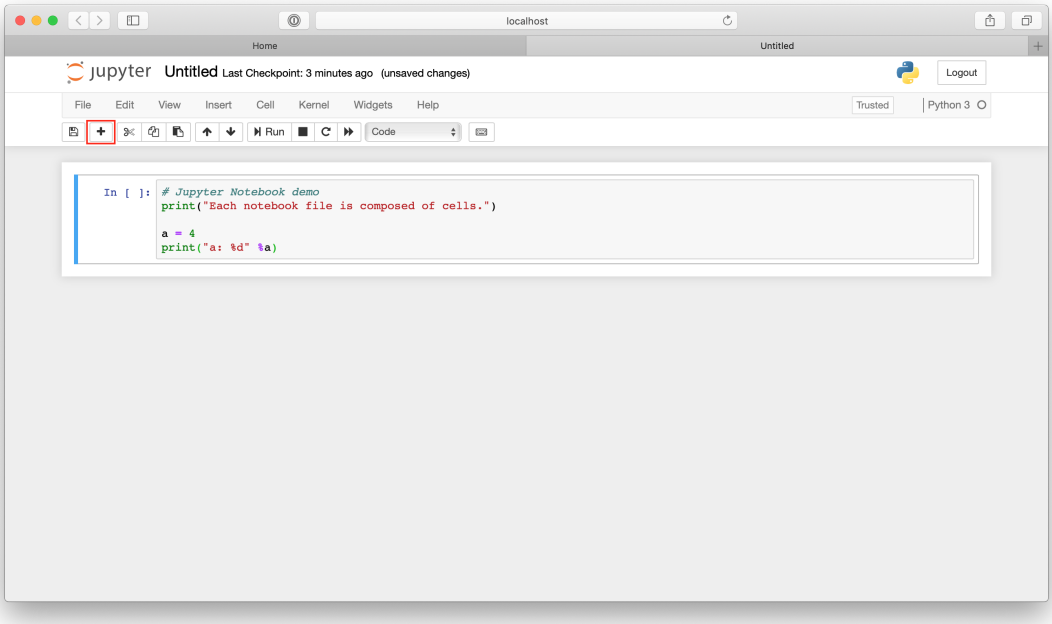

Figure 10: A new Notebook file with a single cell

6. After running both of the shells in Figure 11, results are shown on the bottom of each cells. If the definition of "b" variable needs to be modified, only the second cell needs to be run again (Figure 12). This helps with quick and easy prototyping and data visualization because parts of the code can be easily modified without running everything again.

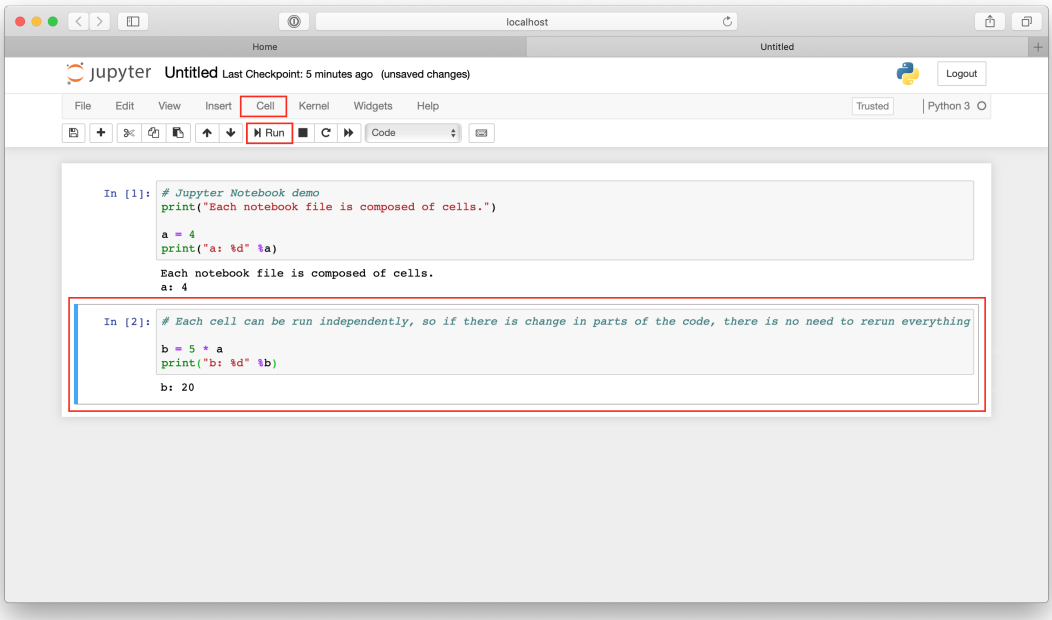

Figure 11: Notebook file can have multiple cells of code and they can be run independently

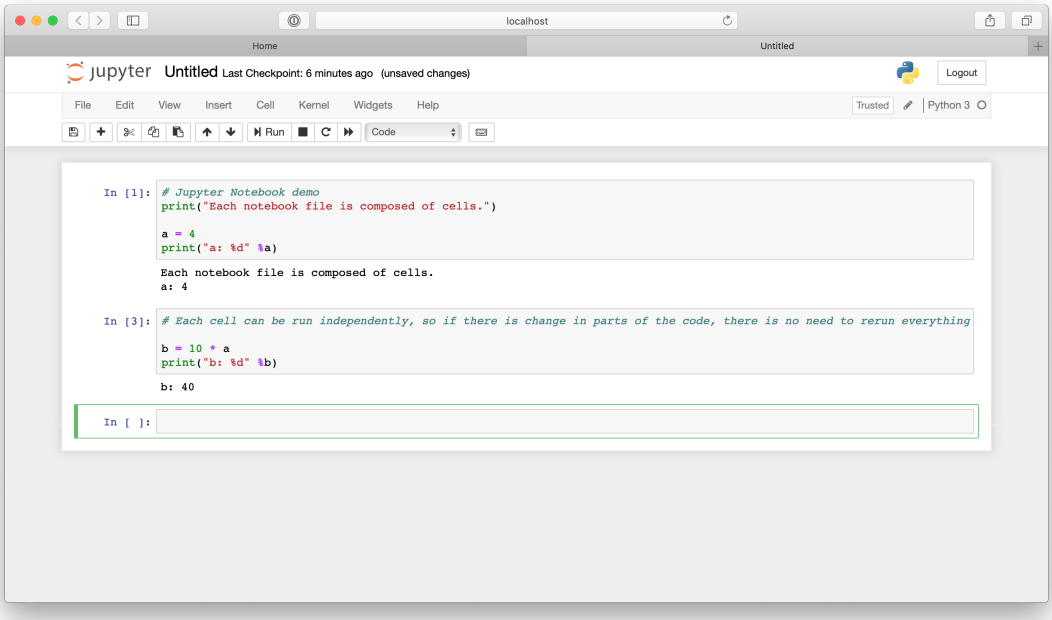

Figure 12: When the second cell is modified, only this needs to be run again.

7. Make sure to decompose your code into blocks of cells, instead of writing everything in a single cell. This modular approach makes it easy to change parts of the code without running everything.

1.6 Installing PyTorch

PyTorch is one of the most popular machine learning library for Python. We will be using PyTorch in this course.

1. Activate "aps360" environment in command line terminal
2. Follow the official download guide for PyTorch from https://pytorch.org/ for your specific operating system.
3. Make sure to choose "Stable" build and "Python 3.6" for language. PyTorch can be easily installed from Anaconda, so select "Conda" for package (choose other options only if you feel more comfortable with them).
4. Chose "None" for CUDA unless you have NVIDIA graphics card that has CUDA support.

   NOTE: it is not necessary to use GPU for this course. However, for those who really want to try it out, but don't have CUDA supported NVIDIA graphics card, consider cloud GPU computing. There are many services available and a few offer free credit for students (Amazon AWS, Google Cloud Computing, and sometimes Microsoft Azure)
5. (Optional) If you have CUDA supported graphics card, choose the correct CUDA library version (you can find which CUDA library is supported from `https://developer.nvidia.com/cuda-gpus`. You also need to install the right CUDA library and NVIDIA has a very good documentation on this (`https://docs.nvidia.com/cuda/`). CUDA installation can get pretty tricky and could consume a bit of time and effort. It is not necessary to GPU for this course, so stick with CPU only PyTorch if it becomes too much hassle. One easy solution for those using Linux is Docker (see the last optional Docker section of this guide)
6. After choosing the right settings, run installation command in your command line terminal (see red box in Figure 13)
7. You can test if PyTorch was installed successfully by bringing up a Python interpreter (e.g. from command line terminal, PyCharm's Python console, or Jupyter Notebook) and running

   ```
   import torch
   ```

   If you have CUDA supported PyCharm installed, you can check the availability of CUDA by running

   ```
   torch.cuda.is_available()
   ```

   If you get "False", it is very likely your NVIDIA graphics card driver or CUDA library is not installed properly. Check by typing "nvidia-smi" and "nvcc –version" in command line terminal. Also, if you are using virtual machine or containers like Docker, you might need something special to make CUDA work in your environment (e.g. for Docker, you need NVIDIA Docker).

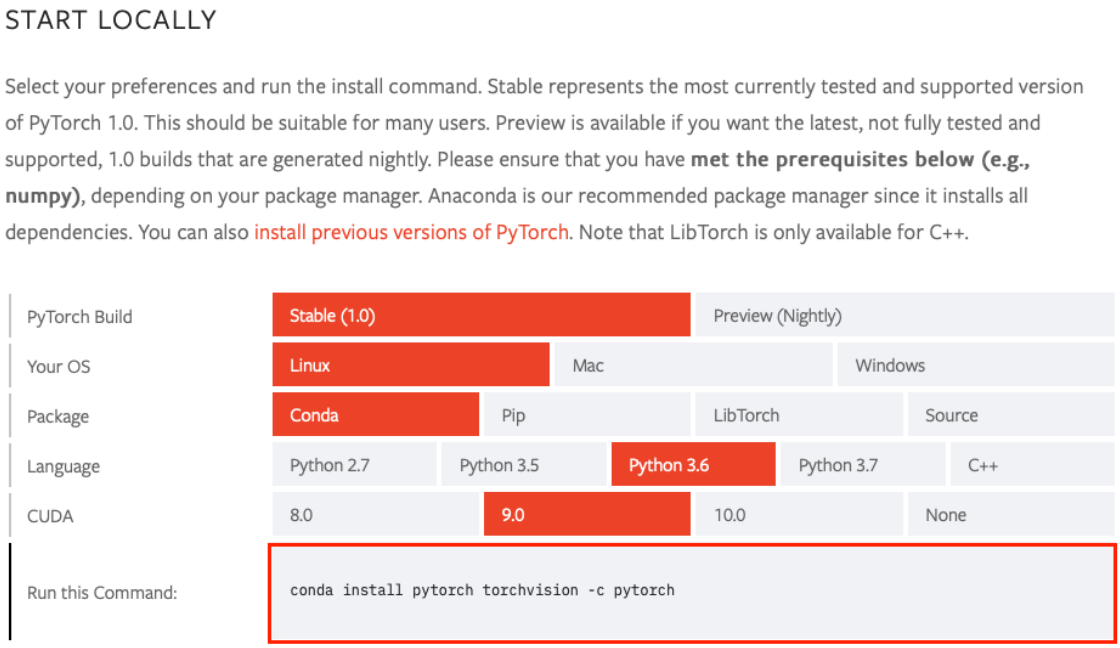

Figure 13: Make sure to choose the right build, OS, language, and CUDA version

2 Preparatory Readings

1. For a concise summary of Python, see: `https://learnxinyminutes.com/docs/python3/`. You only need to read up to (and including) section 6.1 (Inheritance). Focus on the simpler functionalities like for-loops and manipulating lists. A good exercise is to type out the code in command line or pycharm and run it to see what happens, if you do not understand a specific part.

2. See the NumPy and Matplotlib section of the Stanford CS231n course Python Tutorial: `https://cs231n.github.io/python-numpy-tutorial/`. For NumPy, focus on different ways to create and manipulate (i.e. slicing) arrays, as well as vector and matrix mathematics.

3. (Optional) NumPy Tutorial on `https://engineering.ucsb.edu/~shell/che210d/numpy.pdf`.

4. (Optional) Matplotlib Tutorial: `http://www.scipy-lectures.org/intro/matplotlib/matplotlib.html`. Another tutorial that focuses more on the image visualization: `https://matplotlib.org/users/image_tutorial.html`

You may find the following reference (cheat) sheets are useful:

1. `NumPy` cheatsheet: `https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf`

2. `Matplotlib` cheatsheet: `https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_Matplotlib_Cheat_Sheet.pdf`

3 (Optional) Using Docker

Docker is an OS-level virtualization software that can help encapsulating and sharing environments. You can think of Docker a light-weight version of virtual machine. With Docker, I can also set-up an environment (e.g. all the installations we have done so far) and create an exact replica of this environment in another computer using Docker.

Docker has been a popular tool for system administrators and recently it has been getting more attention from machine learning community. One of the reason is that sometimes setting up all the software packages and libraries can be tricky and time consuming, especally if you want to use exactly same versions across different machines. Anaconda helps with easy encapsulation and migration of your Python environment, but for everything outside of Python (e.g. CUDA library), we need something else. With Docker, you can easily swap hardware, share environment and code with collaborators, and scale up very easily.

Let's say you want to train a neural network model on your laptop for quick prototyping, so you setup a Docker image on your laptop with all the necessary libraries and settings. Then when you decide to use more complex model and bigger dataset on powerful machine. You can simply use Docker to duplicate your Docker image on your laptop on this new machine. Let's say you add another machine to run multiple experiments concurrently, and with Docker, you can easily set things up in exactly the same environment without much hassle. Also, let's say you decide to ask your friend to test your code and give some feedback, you can simply share Docker image and your friend can test your code in the exact environment you have been using without any manual setup.

Although it is not necessary to use Docker for this course, if you have time, it is good to know what it is and perhaps start using it. For those who struggle to install CUDA library (for some machines, it can get a bit tricky) or those who would be using multiple CUDA enabled machines, using Docker can make your life easier.

1. Read `https://docs.docker.com/get-started/#containers-and-virtual-machines` and install Docker Community Edition (use stable version).

2. Read part 2 of the Get Started Guide `https://docs.docker.com/get-started/part2/`

3. Download Dockerfile into the "aps360" project directory.

4. (Optional) In order for Docker to use CUDA, NVIDIA Docker needs to be installed as well. Follow installation instructions from `https://github.com/NVIDIA/nvidia-docker`

5. In your command line terminal, navigate to the "aps360" project directory and create a Docker image for this class by running:

   ```
   docker build -t aps360
   ```

   It might take a while to install everything.

6. Create a Docker container using the image we just created by running:

   ```
   docker run -it --rm -p 8888:8888 -v $(pwd):/workspace aps360
   ```

   To use CUDA, enable NVIDIA Docker by adding "–runtime=nvidia":

```
docker run --runtime=nvidia -it --rm -p 8888:8888 -v $(pwd):/workspace aps360
```

7. This should start the Docker container with Jupyter Notebook running