

# Math 156

TA: Bumsu Kim

# Today...

- Installing Python and Jupyter Notebook
- Jupyter Notebook Tutorial

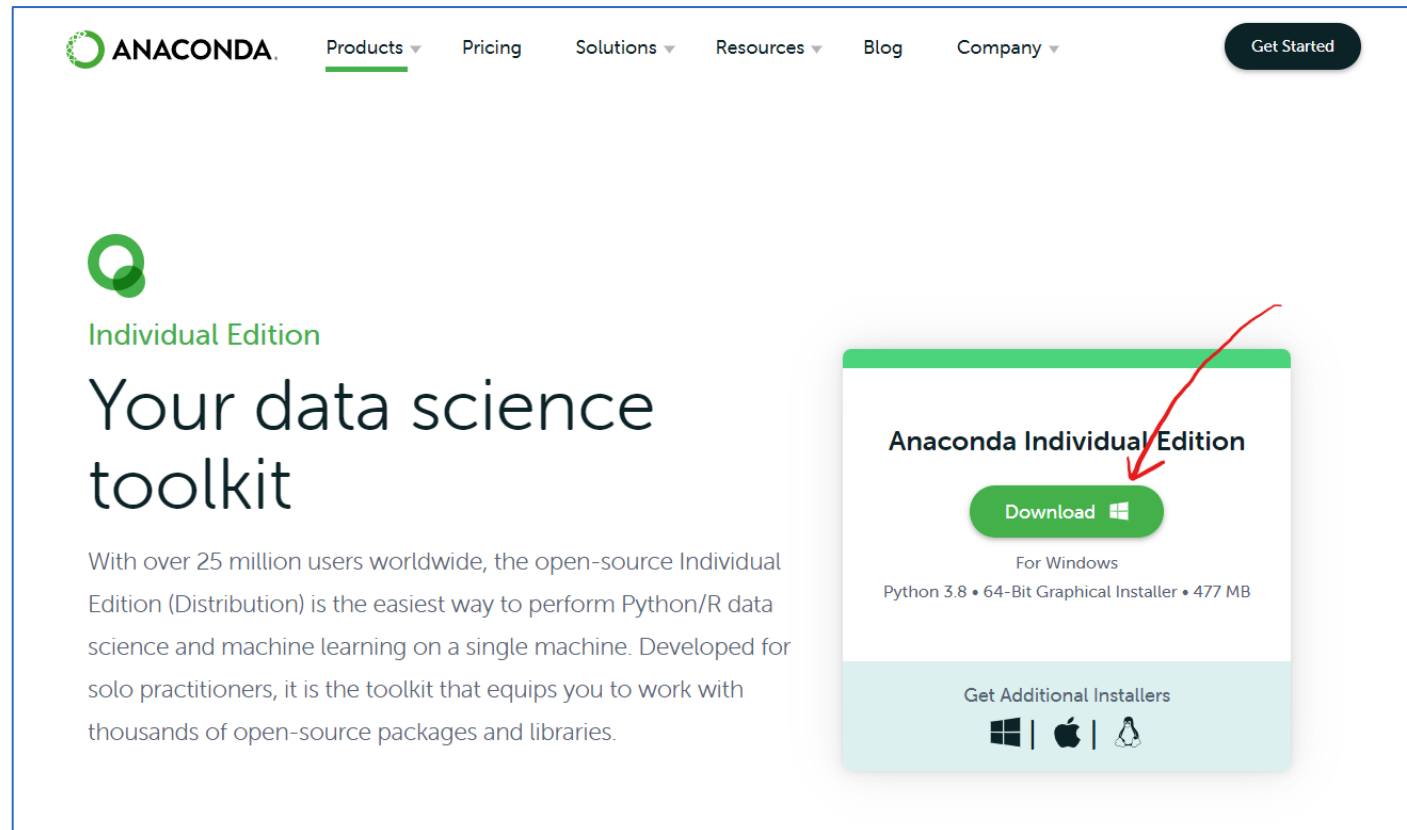
# Introduction

---

- TA: Bumsu Kim (grad student in applied math)
- email: [bumsu@ucla.edu](mailto:bumsu@ucla.edu)
- Slides(if any) and supplementary materials will be uploaded on Canvas
- Office Hours – please complete the survey in the email sent yesterday
- OH Location: Virtually on ZOOM meeting room
  - link will be shared on Canvas when the time is set

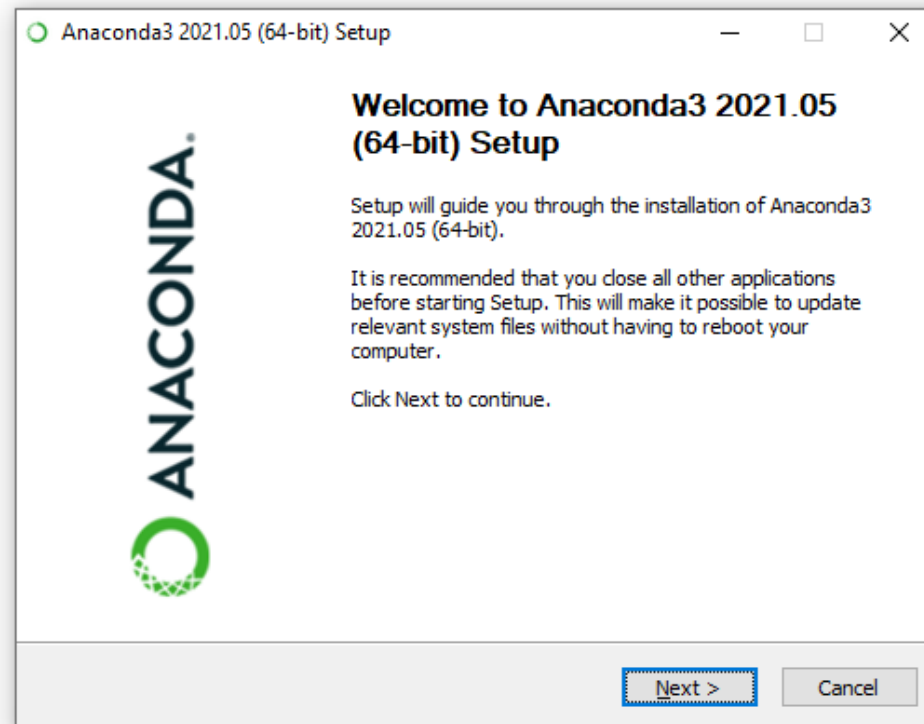
# Anaconda Installation

- We will use Anaconda for installing Python and Jupyter Notebook
- Google “Anaconda” or go to: <https://www.anaconda.com/products/individual>



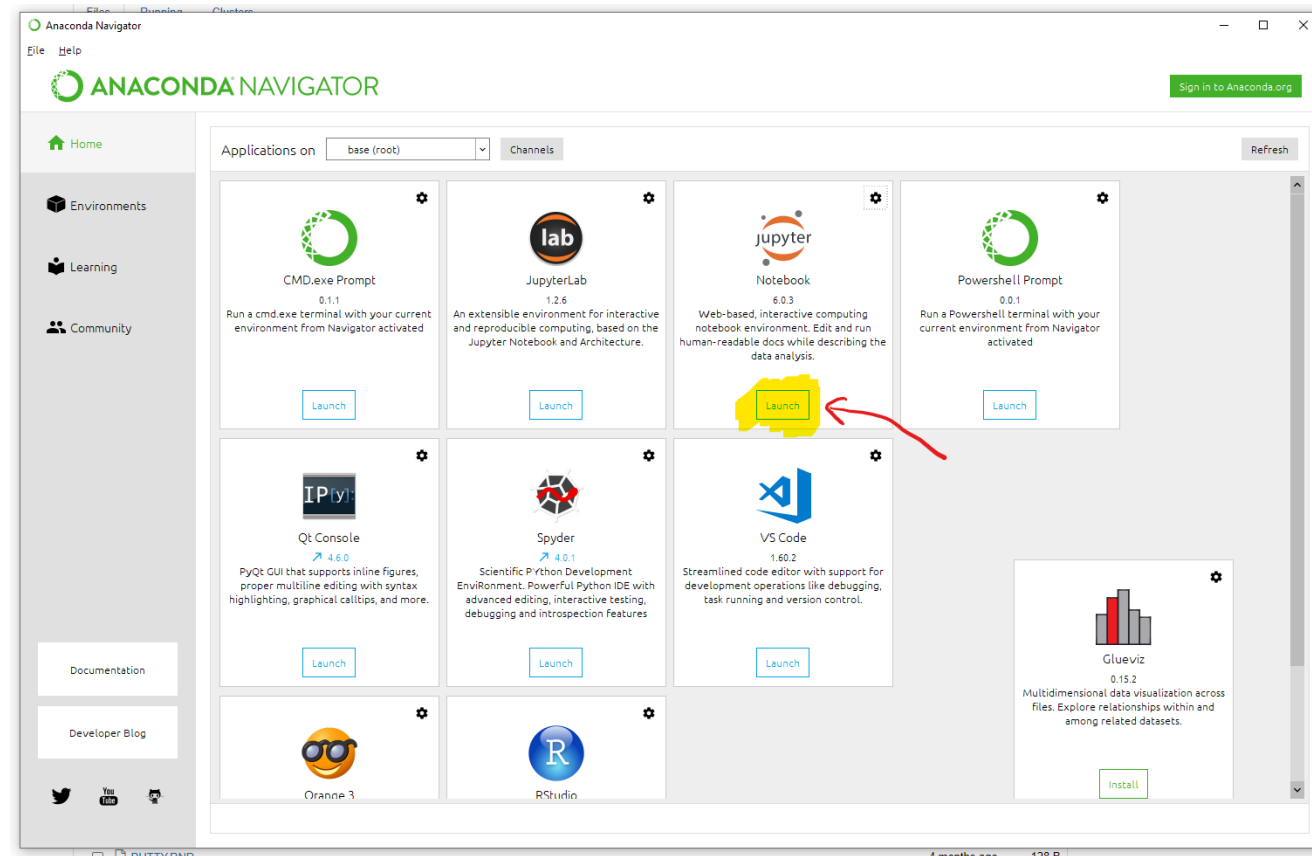
# Anaconda Installation

- And run the Setup file, click Next, Next, Next, ...
- Don't include a white space ( ' ') in the installation path



# Anaconda Installation

- You're done!
- Anaconda installs Python and Jupyter Notebook at once



# Jupyter Notebook Tutorial

- Simplest Program

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

- Type Python commands and [Run this cell]
- Shortcut: [Ctrl]+[Enter], different for Mac computers
- See [Help]-[Keyboard Shortcuts]

- Simple Program

```
In [2]: ▶ x = 1  
        y = 2  
        z = 3  
  
In [3]: ▶ print(f"x = {x}, y = {y}, z = {z}")  
x = 1, y = 2, z = 3
```

## Keyboard shortcuts

The Jupyter Notebook has two different keyboard input modes. **Edit mode** allows you to type code or text into a cell and is indicated by a green cell border. **Command mode** binds the keyboard to notebook level commands and is indicated by a grey cell border with a blue left margin.

### Command Mode (press `Esc` to enable)

[Edit Shortcuts](#)

<code>F</code> : find and replace	<code>Shift-J</code> : extend selected cells below
<code>Ctrl-Shift-F</code> : open the command palette	<code>Ctrl-A</code> : select all cells
<code>Ctrl-Shift-P</code> : open the command palette	<code>A</code> : insert cell above
<code>Enter</code> : enter edit mode	<code>B</code> : insert cell below
<code>P</code> : open the command palette	<code>X</code> : cut selected cells
<code>Shift-Enter</code> : run cell, select below	<code>C</code> : copy selected cells
<code>Ctrl-Enter</code> : run selected cells	<code>Shift-V</code> : paste cells above
<code>Alt-Enter</code> : run cell and insert below	<code>V</code> : paste cells below
<code>Y</code> : change cell to code	<code>Z</code> : undo cell deletion
<code>M</code> : change cell to markdown	<code>D</code> , <code>D</code> : delete selected cells
<code>R</code> : change cell to raw	<code>Shift-M</code> : merge selected cells, or current cell with cell below if only one cell is selected
<code>1</code> : change cell to heading 1	<code>Ctrl-S</code> : Save and Checkpoint
<code>2</code> : change cell to heading 2	<code>S</code> : Save and Checkpoint
<code>3</code> : change cell to heading 3	<code>L</code> : toggle line numbers
<code>4</code> : change cell to heading 4	<code>O</code> : toggle output of selected cells
<code>5</code> : change cell to heading 5	<code>Shift-O</code> : toggle output scrolling of selected cells
<code>6</code> : change cell to heading 6	<code>H</code> : show keyboard shortcuts
<code>K</code> : select cell above	<code>I</code> , <code>I</code> : interrupt the kernel
<code>Up</code> : select cell above	<code>0</code> , <code>0</code> : restart the kernel (with dialog)
<code>Down</code> : select cell below	<code>Esc</code> : close the pager
<code>J</code> : select cell below	<code>Q</code> : close the pager
<code>Shift-K</code> : extend selected cells above	<code>Shift-L</code> : toggles line numbers in all cells, and persist the setting
<code>Shift-Up</code> : extend selected cells above	<code>Shift-Space</code> : scroll notebook up
<code>Shift-Down</code> : extend selected cells below	<code>Space</code> : scroll notebook down

# Jupyter Notebook Tutorial

- **NumPy** is a super useful library adding support for arrays, matrices, and mathematical functions

```
In [10]: import numpy as np

x = [1, 2, 3, 4]
print(f"x = {x}\n Data type of x = {type(x)}\n")
y = np.array(x)
print(f"y = {y}\n Data type of y = {type(y)}\n")

print(f"Sum of the elements in y = {np.sum(y)}")
print(f"Product of the elements in y = {np.prod(y)}")
print(f"Mean of the elements in y = {np.mean(y)}")

x = [1, 2, 3, 4]
Data type of x = <class 'list'>

y = [1 2 3 4]
Data type of y = <class 'numpy.ndarray'>

Sum of the elements in y = 10
Product of the elements in y = 24
Mean of the elements in y = 2.5
```



# Jupyter Notebook Tutorial

- **NumPy** is a super useful library adding support for arrays, matrices, and mathematical functions
- You will later use other cool libraries too!
  - PyTorch, TensorFlow, etc.

```
In [10]: ▶ import numpy as np

x = [1, 2, 3, 4]
print(f"x = {x}\n Data type of x = {type(x)}\n")
y = np.array(x)
print(f"y = {y}\n Data type of y = {type(y)}\n")

print(f"Sum of the elements in y = {np.sum(y)}")
print(f"Product of the elements in y = {np.prod(y)}")
print(f"Mean of the elements in y = {np.mean(y)}")

x = [1, 2, 3, 4]
Data type of x = <class 'list'>

y = [1 2 3 4]
Data type of y = <class 'numpy.ndarray'>

Sum of the elements in y = 10
Product of the elements in y = 24
Mean of the elements in y = 2.5
```

# More Advanced Example

- A simple Convolutional Neural Network for training MNIST dataset →
- Accuracy is usually >99% on images like



```
class Net(nn.Module):
    def __init__(self):
        super(Net,self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=6,
                                kernel_size=5 , stride=1)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=2)
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=16,
                                kernel_size=5, stride=1)
        self.conv3 = nn.Conv2d(in_channels=16, out_channels=120,
                                kernel_size=4, stride=1)
        self.fc_1 = nn.Linear(in_features=120,out_features=84)
        self.fc_2 = nn.Linear(in_features=84, out_features=10)

    def forward(self,u):
        u = self.conv1(u) # apply first convolutional layer
        u = self.relu(u) # apply ReLU activation
        u = self.pool(u) # apply max-pooling
        u = self.conv2(u) # apply second convolutional layer
        u = self.relu(u) # Apply ReLU activation
        u = self.pool(u)
        u = self.conv3(u) # Apply third and final convolutional layer
        u = torch.flatten(u, 1)
        u = self.fc_1(u)
        u = self.relu(u)
        u = self.fc_2(u)
        u = self.relu(u)
        y = F.log_softmax(u, dim=1)
        return y
```