In this lesson you will learn how to download an install Anaconda and Jupyter notebooks on your mac.

What is Anaconda?

- Anaconda is a distribution software that provides everything a user would need to start Python development.
- It includes:
  - Python language
  - Libraries
  - Editors(Such as Jupyter Notebooks)
  - Package manager

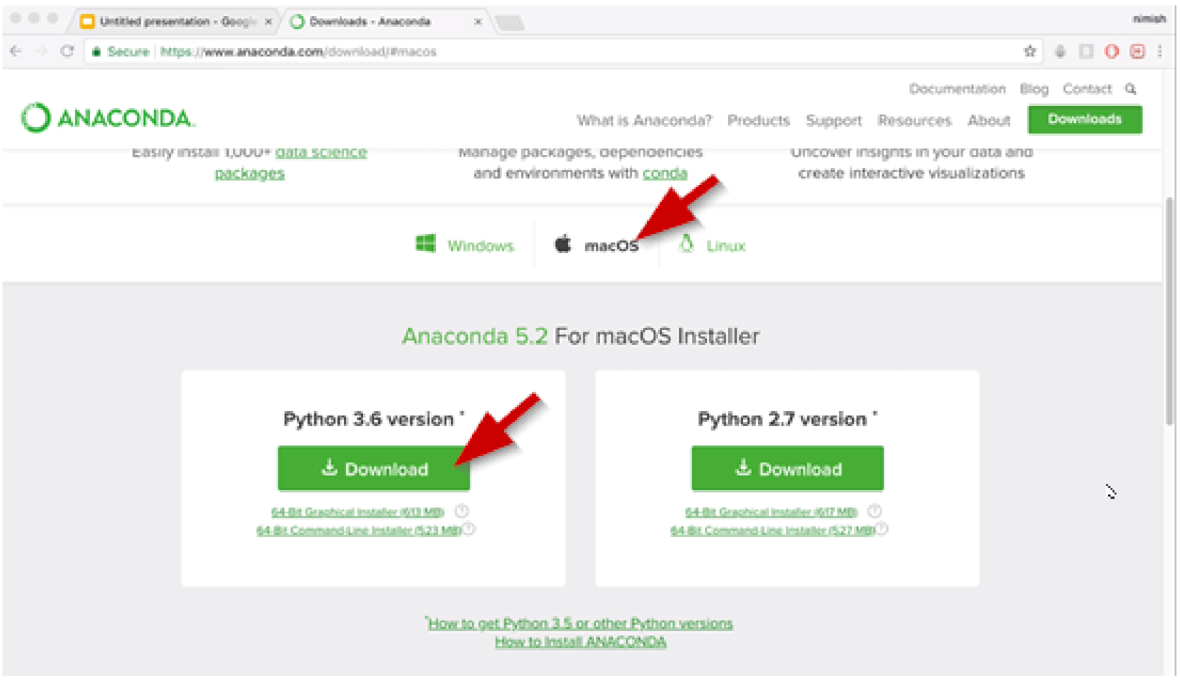
What are Jupyter Notebooks?

- A **Jupyter Notebook** is an **open source web application that allows users to share documents with text, live code, images, and more.**
- We will **use this to write code** as it provides an interactive and easy to use interface.

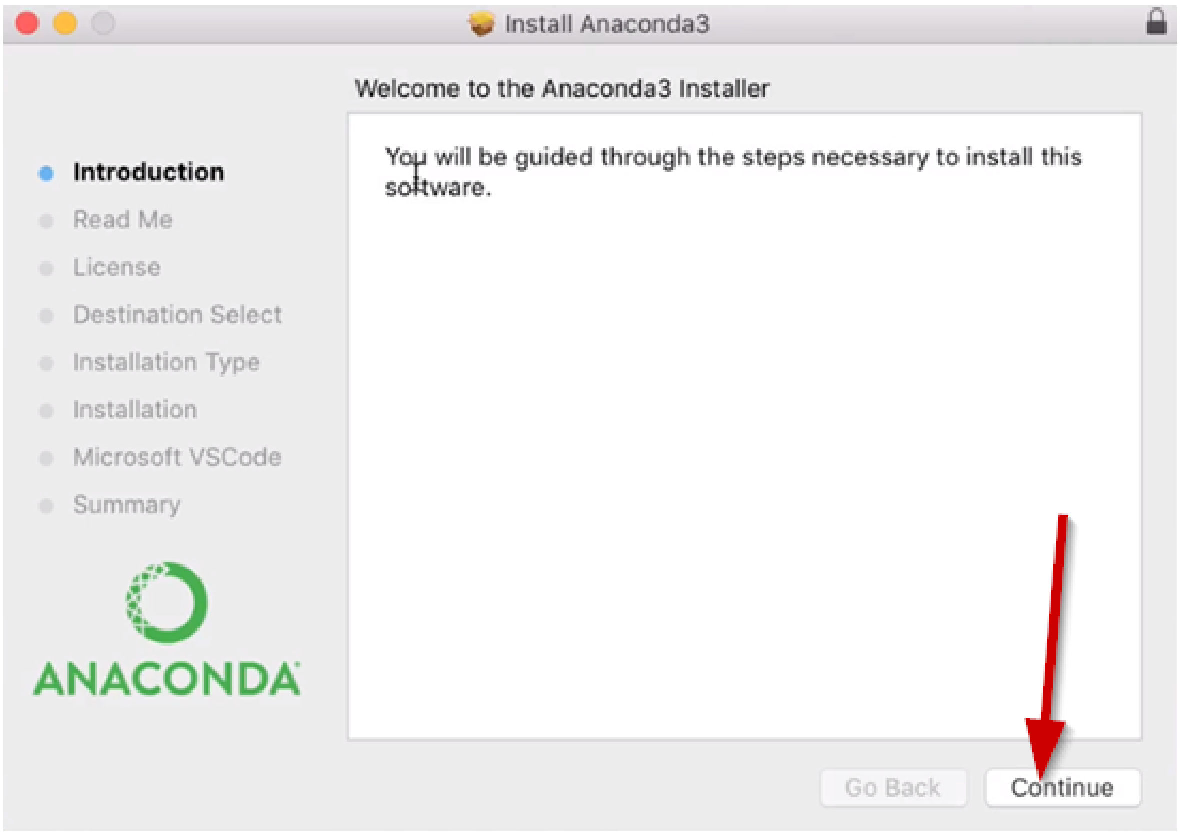
Download and Install Anaconda

Anaconda is available for download from the following link: https://www.anaconda.com/distribution/#download-section

Here is the Direct Link: [Anaconda Download MAC]


Untitled presentation - Google
Downloads - Anaconda
Secure https://www.anaconda.com/download/#macos
Documentation Blog Contact
ANACONDA.
What is Anaconda? Products Support Resources About
Downloads
Easily install 1,000+ data science packages
Manage packages, dependencies and environments with conda
Uncover insights in your data and create interactive visualizations
Windows
macOS
Linux
Anaconda 5.2 For macOS Installer
Python 3.6 version *
Download
64-Bit Graphical Installer (613 MB)
64-Bit Command-Line Installer (523 MB)
Python 2.7 version *
Download
64-Bit Graphical Installer (617 MB)
64-Bit Command-Line Installer (527 MB)
*How to get Python 3.5 or other Python versions
How to Install ANACONDA

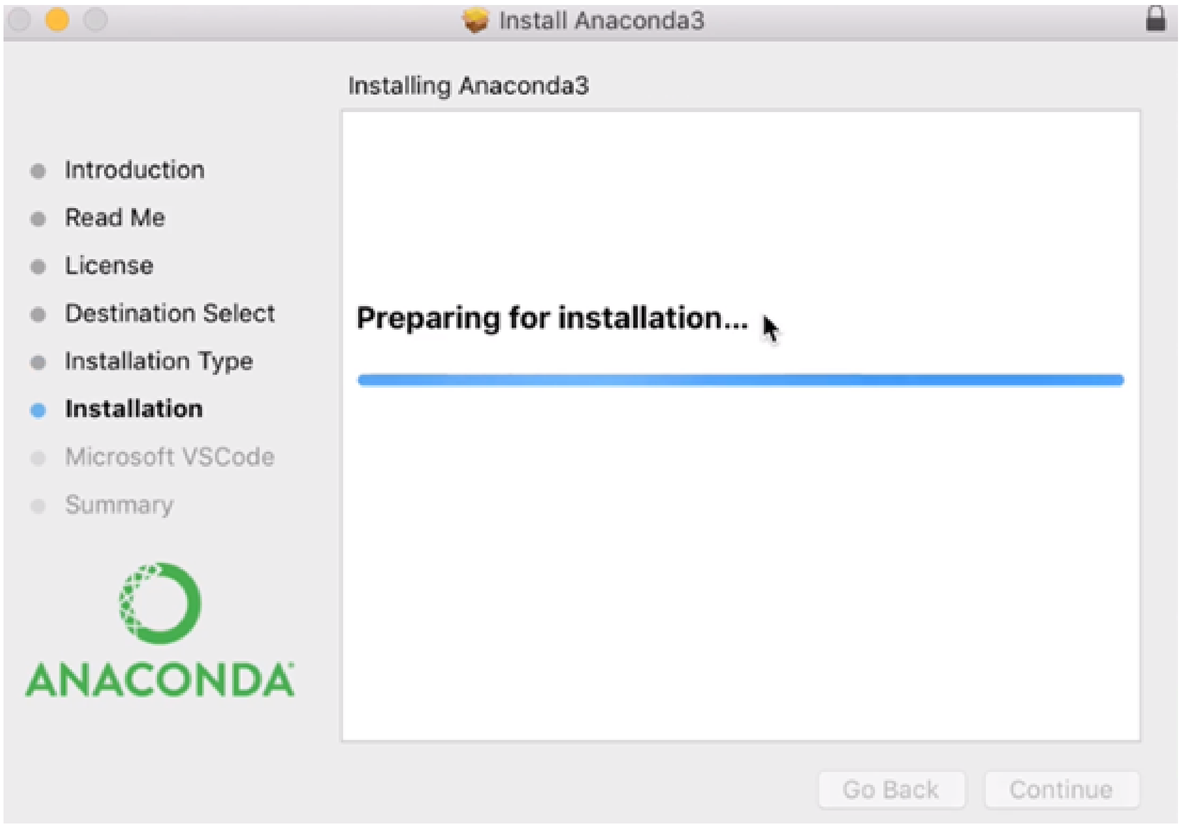
Once you **download the file** go ahead and open it, and **follow the instructions in the installation wizard.**



Welcome to the Anaconda3 Installer

You will be guided through the steps necessary to install this software.

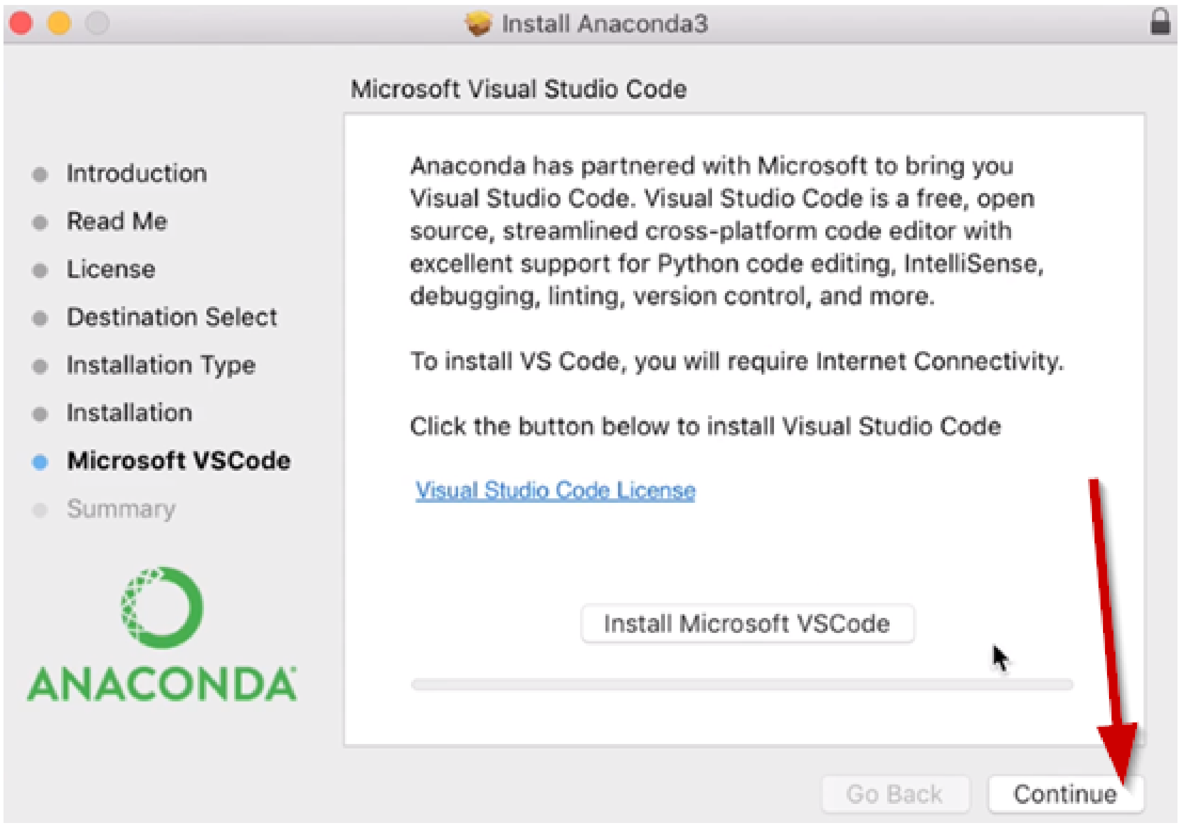
Just follow through the prompts and choose where you want to install Anaconda on your system.



Installing Anaconda3

Preparing for installation...

You **do not need to install Microsoft Visual Code.** So just **hit the Continue button** at that prompt.



Install Anaconda3

Microsoft Visual Studio Code

- Introduction
- Read Me
- License
- Destination Select
- Installation Type
- Installation
- **Microsoft VSCode**
- Summary

ANACONDA

Anaconda has partnered with Microsoft to bring you Visual Studio Code. Visual Studio Code is a free, open source, streamlined cross-platform code editor with excellent support for Python code editing, IntelliSense, debugging, linting, version control, and more.

To install VS Code, you will require Internet Connectivity.

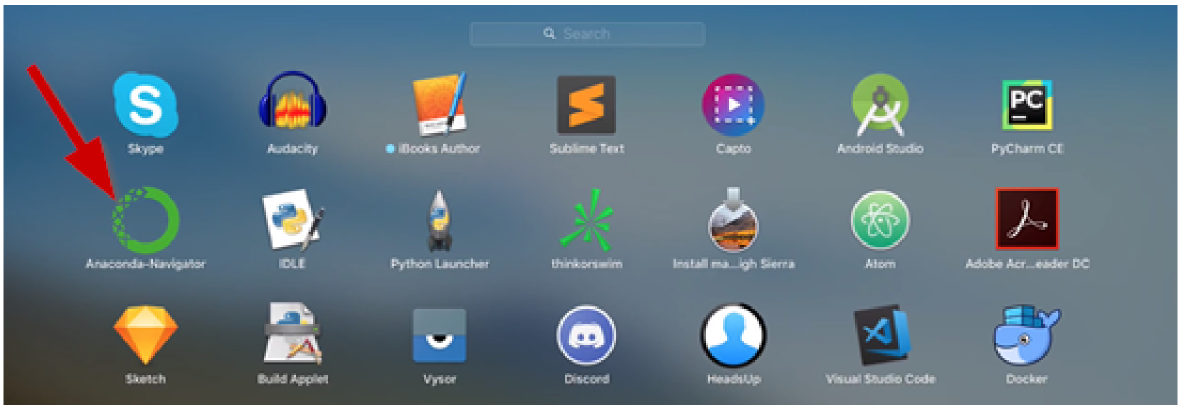
Click the button below to install Visual Studio Code

Visual Studio Code License

Install Microsoft VSCode

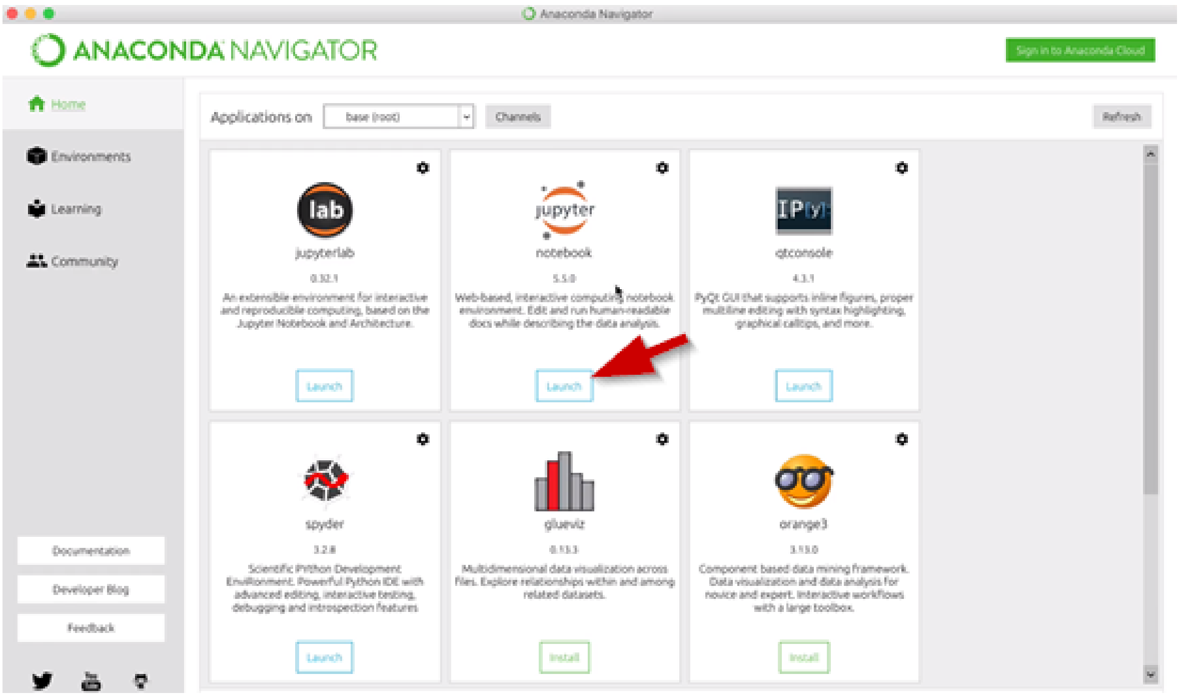
Go Back Continue

Once Anaconda is installed on your Mac system go ahead and **open the Anaconda-Navigator** from your applications menu.

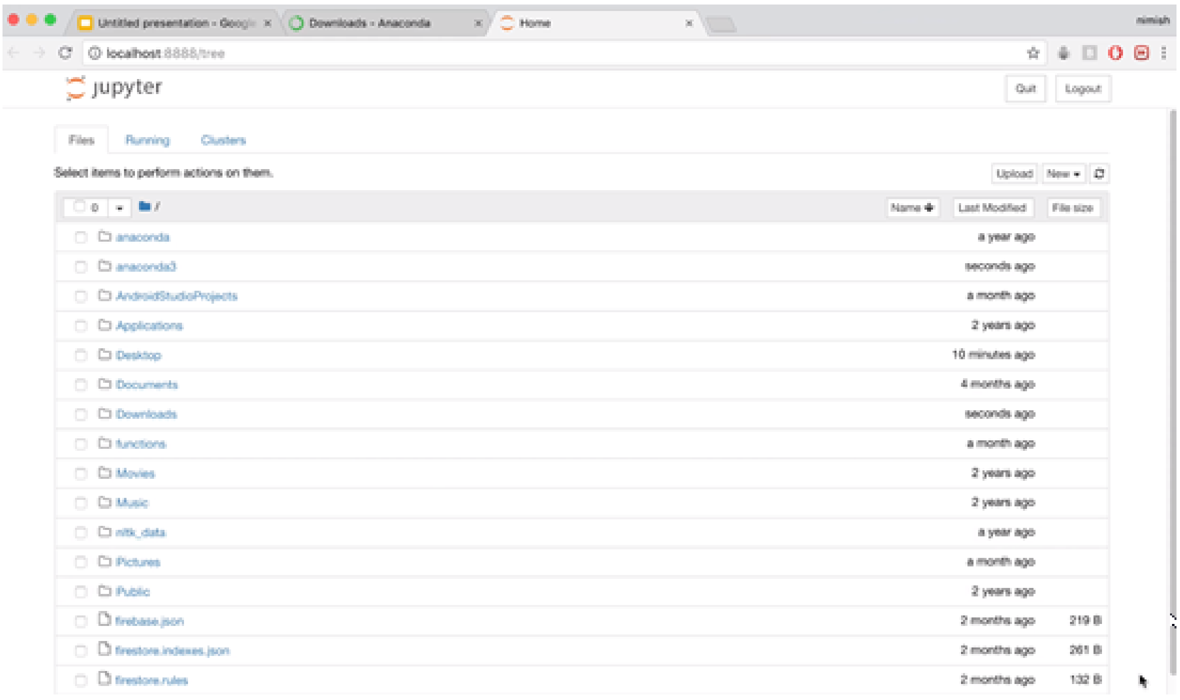


Once Anaconda is opened it may take a view moments to initialize.

Once its open, you will see a few options to choose from, but we will be using the **Jupyter Notebook launch button**. So go ahead and **select the Launch button.**


Anaconda Navigator
ANACONDA NAVIGATOR
Sign in to Anaconda Cloud
Home
Environments
Learning
Community
Applications on
base (root)
Channels
Refresh
jupyterlab
0.32.1
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
Launch
notebook
5.5.0
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch
qtconsole
4.3.1
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Launch
spyder
3.2.8
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features
Launch
glueviz
0.13.3
Multidimensional data visualization across files. Explore relationships within and among related datasets.
Install
orange3
3.13.0
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.
Install
Documentation
Developer Blog
Feedback

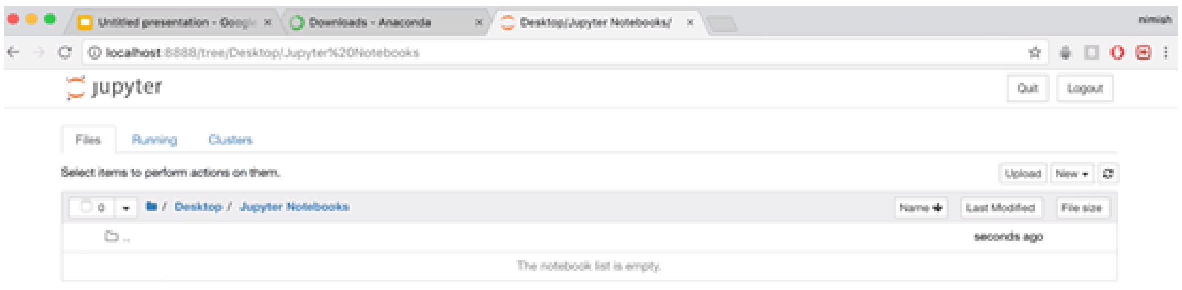
Once the Juptyer Notebook loads it should look like this:



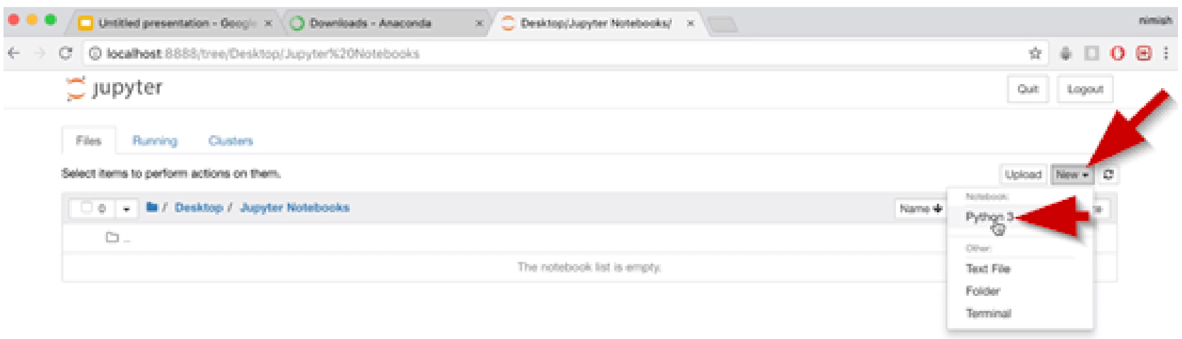
Select items to perform actions on them.

Name	Last Modified	File size
anaconda	a year ago	
anaconda3	seconds ago	
AndroidStudioProjects	a month ago	
Applications	2 years ago	
Desktop	10 minutes ago	
Documents	4 months ago	
Downloads	seconds ago	
functions	a month ago	
Movies	2 years ago	
Music	2 years ago	
nltk_data	a year ago	
Pictures	a month ago	
Public	2 years ago	
firebase.json	2 months ago	219 B
firestore.indexes.json	2 months ago	261 B
firestore.rules	2 months ago	132 B

There will be a list of files and directories, and you can go to the Desktop then Jupyter Notebooks.

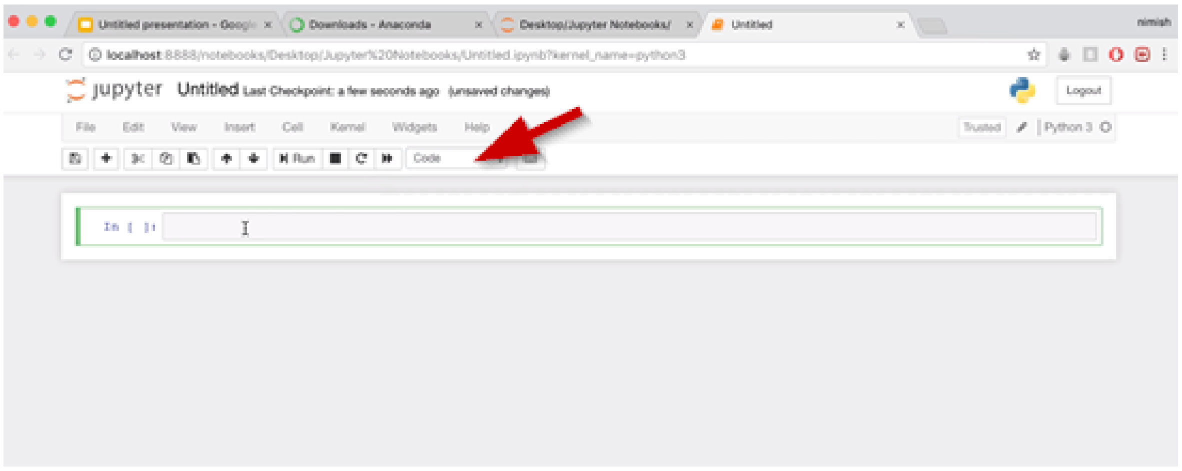

jupyter
Quit
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
Desktop / Jupyter Notebooks
Name
Last Modified
File size
seconds ago
The notebook list is empty.

From here **navigate to the New button and select Python 3.**


jupyter
Quit
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
Desktop / Jupyter Notebooks
Notebook:
Python 3
Other:
Text File
Folder
Terminal
The notebook list is empty.

This will open up a new Jupyter notebook and in the cells, you can write in the code.

These cells can contain text, or code. So we will choose **code** from the drop down menu.



So we now have Anaconda downloaded and installed for MAC.

In this lesson you will learn how to download an install Anaconda and Jupyter notebooks on Windows.

What is Anaconda?

- Anaconda is a distribution software that provides everything a user would need to start Python development.
- It includes:
    - Python language
    - Libraries
    - Editors(Such as Jupyter Notebooks)
    - Package manager

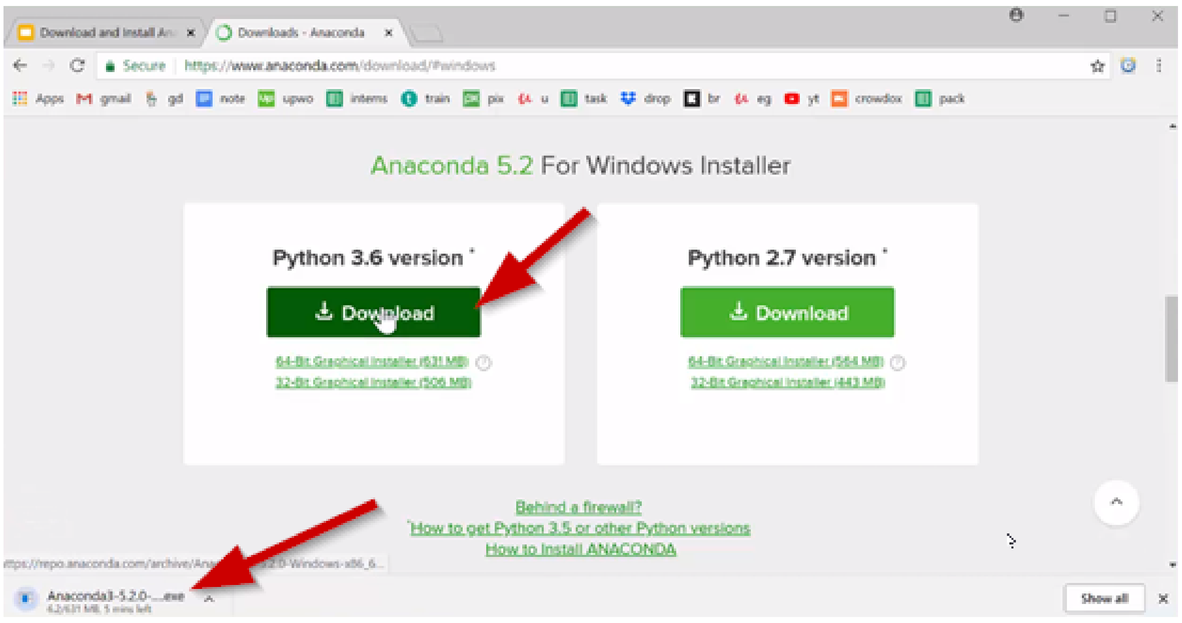
What are Jupyter Notebooks?

- A **Jupyter Notebook** is an **open source web application that allows users to share documents with text, live code, images, and more.**
- We will **use this to write code** as it provides an interactive and easy to use interface.

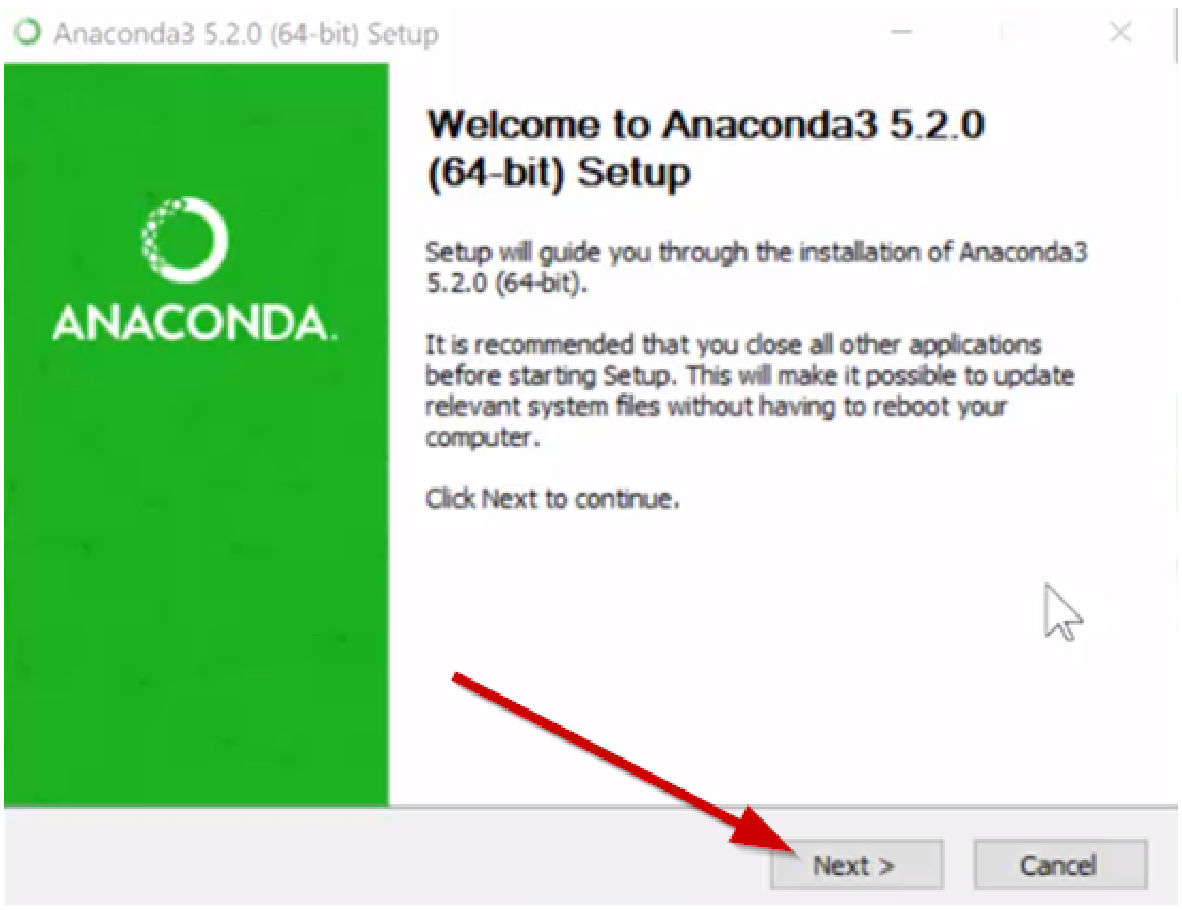
Download and Install Anaconda

Anaconda is available for download from the following link: https://www.anaconda.com/distribution/#download-section

Here is the Direct Link: Anaconda Download Windows


Download and Install An
Downloads - Anaconda
Secure
https://www.anaconda.com/download/#windows
Anaconda 5.2 For Windows Installer
Python 3.6 version *
Download
64-Bit Graphical Installer (631 MB)
32-Bit Graphical Installer (506 MB)
Python 2.7 version *
Download
64-Bit Graphical Installer (564 MB)
32-Bit Graphical Installer (443 MB)
Behind a firewall?
How to get Python 3.5 or other Python versions
How to Install ANACONDA
Anaconda3-5.2.0-....exe
Show all

Once you **download the file** go ahead and open it, and **follow the instructions in the installation wizard.**



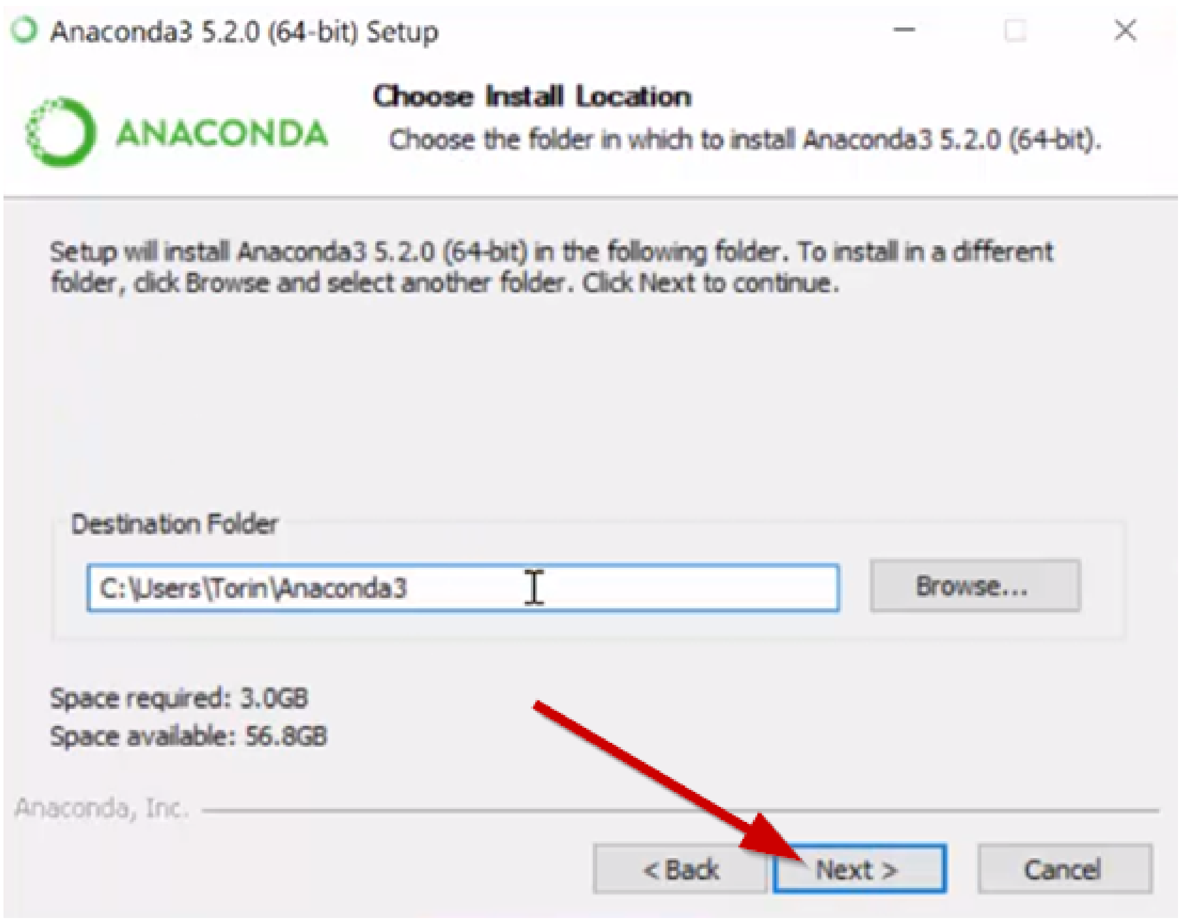
Welcome to Anaconda3 5.2.0 (64-bit) Setup

Setup will guide you through the installation of Anaconda3 5.2.0 (64-bit).

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Just follow through the prompts and choose where you want to install Anaconda on your system.



Choose Install Location

Choose the folder in which to install Anaconda3 5.2.0 (64-bit).

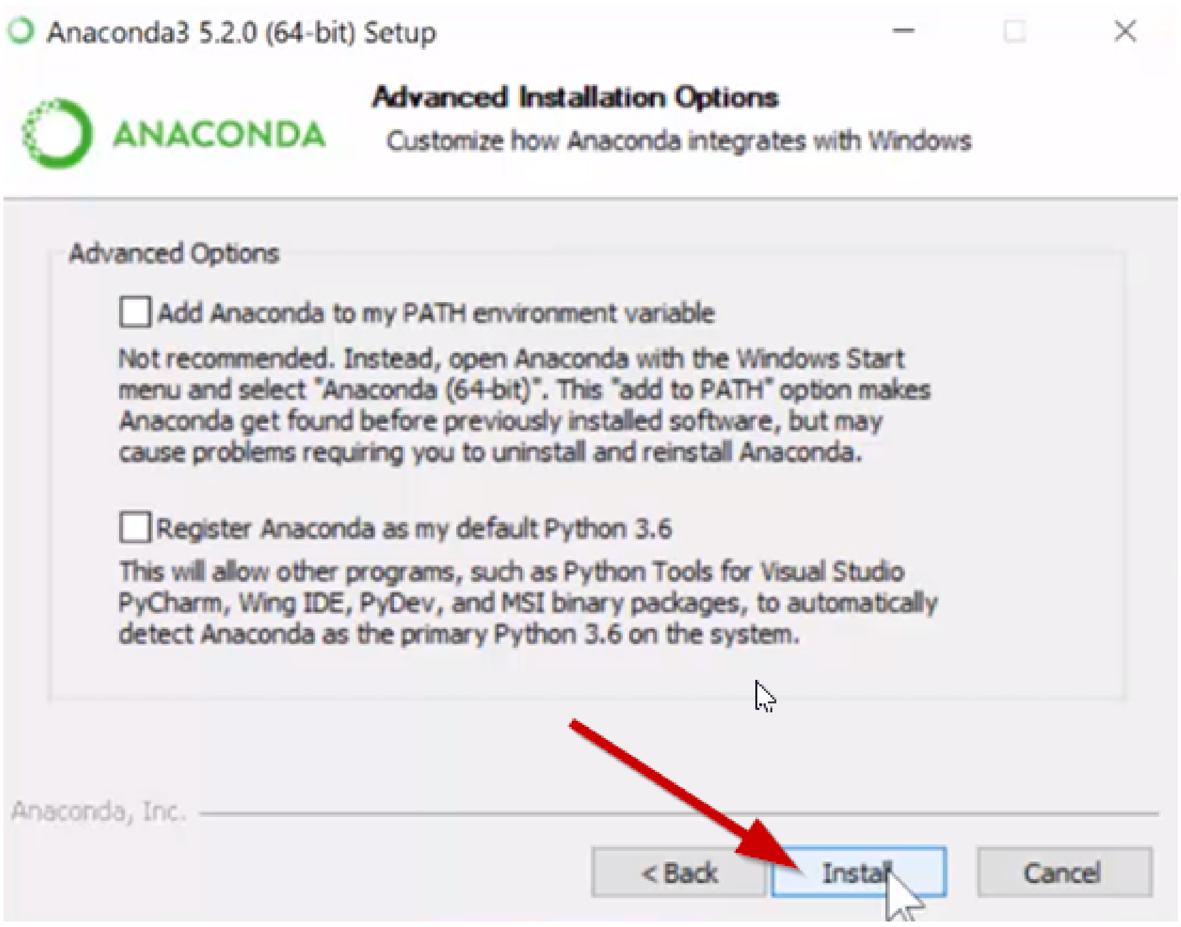
Setup will install Anaconda3 5.2.0 (64-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder

C:\Users\Torin\Anaconda3

Space required: 3.0GB
Space available: 56.8GB

You do not need to select any of the Advanced Options. So just select the Install button, but do not check any of the advanced options.



Anaconda3 5.2.0 (64-bit) Setup

Advanced Installation Options
Customize how Anaconda integrates with Windows

Advanced Options

☐ Add Anaconda to my PATH environment variable

Not recommended. Instead, open Anaconda with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.

☐ Register Anaconda as my default Python 3.6

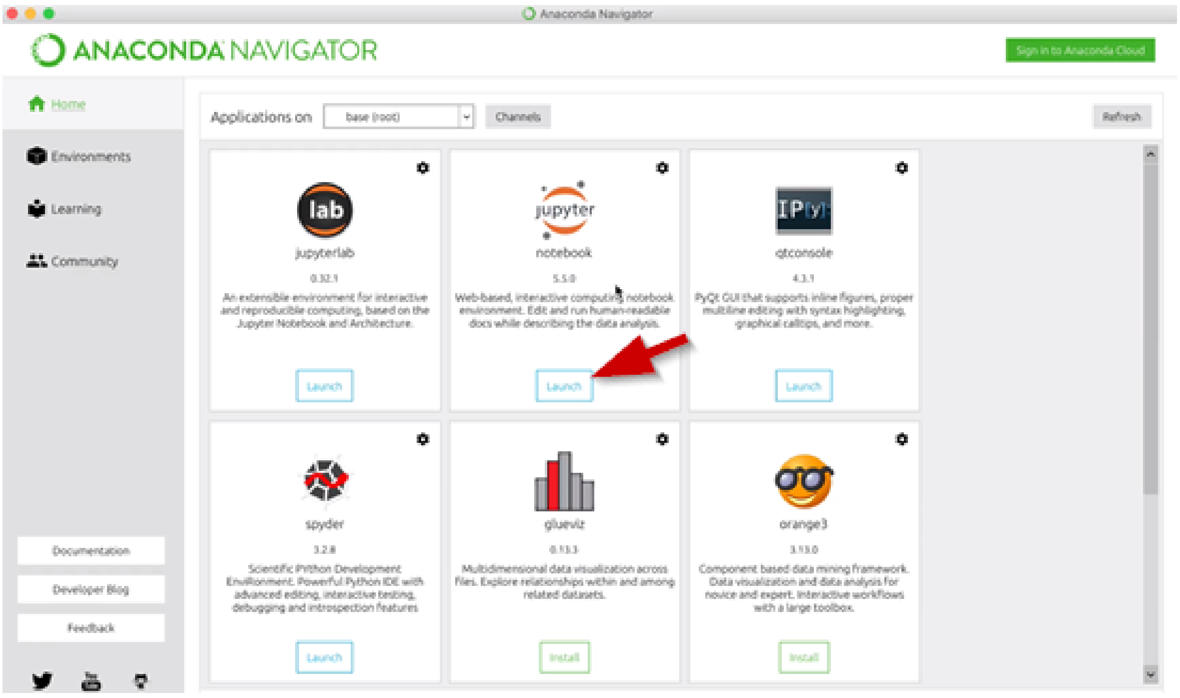
This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.6 on the system.

You can Skip the install of Microsoft VSCode.

Once Anaconda is installed on your Windows system go ahead and **open the Anaconda-Navigator.**

Once Anaconda is opened it may take a view moments to initialize.

Once its open, you will see a few options to choose from, but we will be using the **Jupyter Notebook launch button**. So go ahead and **select the Launch button.**



jupyterlab
0.32.1
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

notebook
5.5.0
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

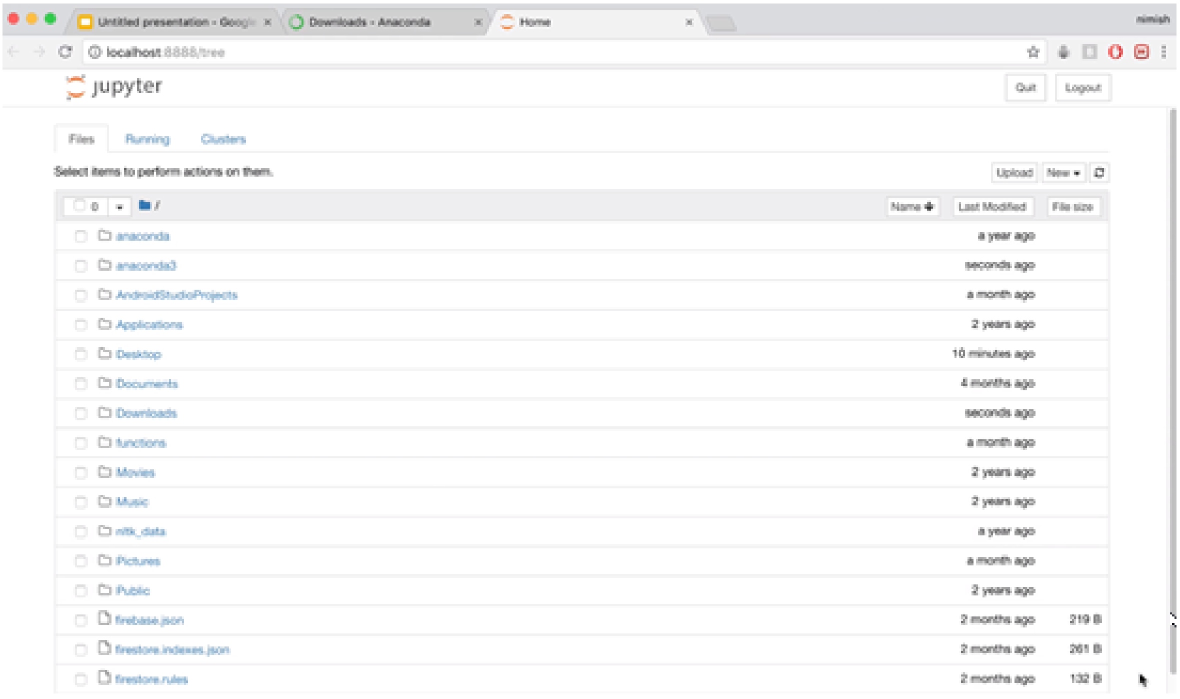
qtconsole
4.3.1
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

spyder
3.2.8
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

glueviz
0.13.3
Multidimensional data visualization across files. Explore relationships within and among related datasets.

orange3
3.13.0
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

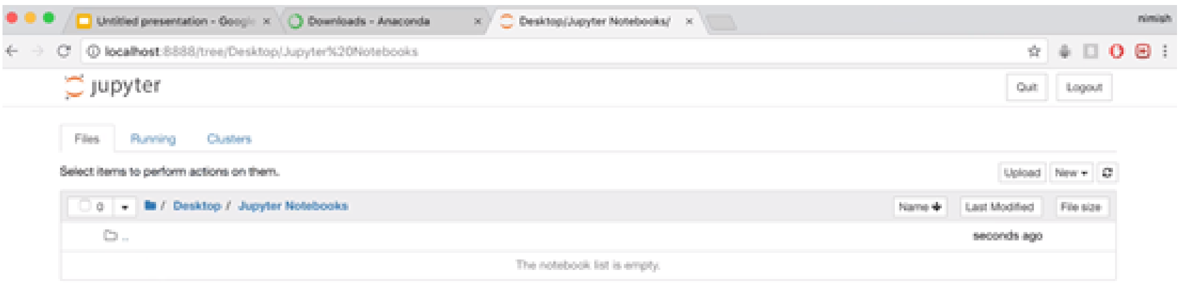
Once the Juptyer Notebook loads it should look like this:



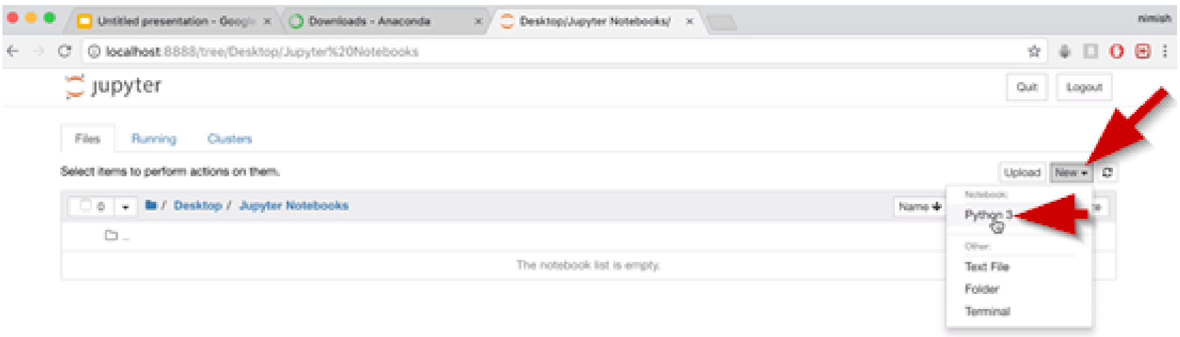
Select items to perform actions on them.

Name	Last Modified	File size
anaconda	a year ago	
anaconda3	seconds ago	
AndroidStudioProjects	a month ago	
Applications	2 years ago	
Desktop	10 minutes ago	
Documents	4 months ago	
Downloads	seconds ago	
functions	a month ago	
Movies	2 years ago	
Music	2 years ago	
nltk_data	a year ago	
Pictures	a month ago	
Public	2 years ago	
firebase.json	2 months ago	219 B
firestore.indexes.json	2 months ago	261 B
firestore.rules	2 months ago	132 B

There will be a list of files and directories, and you can go to the Desktop then Jupyter Notebooks.

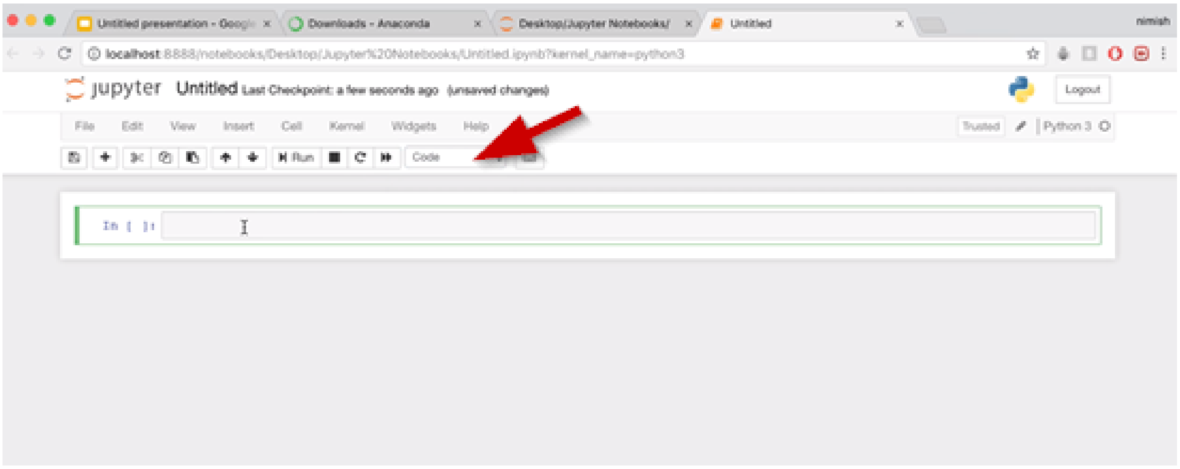


From here **navigate to the New button and select Python 3.**



This will open up a new Jupyter notebook and in the cells, you can write in the code.

These cells can contain text, or code. So we will choose **code** from the drop down menu.


jupyter Untitled Last Checkpoint: a few seconds ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
In [ ]:

So we now have Anaconda downloaded and installed for Windows.

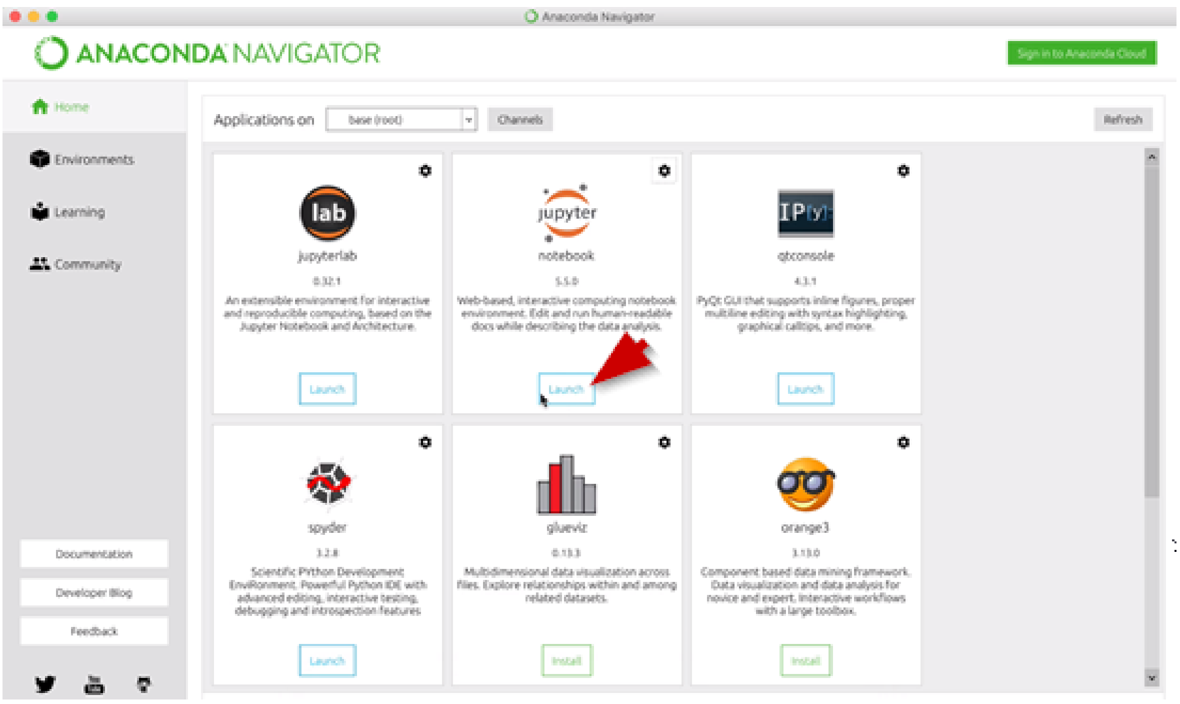
What are Numpy Arrays?

- Numpy arrays are like lists with extra functionality
- Often used in data science and machine learning
- Very powerful with many built in functions written in maximum efficiency

The different ways in Which Numpy Arrays can be Started:

- Pre-assigned values
- Zeroes
- Ones
- Range of vlaues
- Linspace

Open Jupyter Notebook in the Anaconda Navigator menu by **selecting the Launch button.**



jupyterlab
0.32.1
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

notebook
5.5.0
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

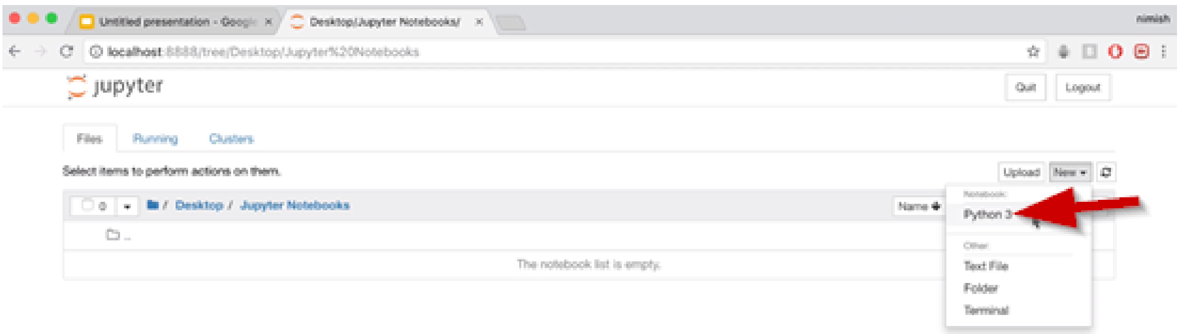
qtconsole
4.3.1
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

spyder
3.2.8
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

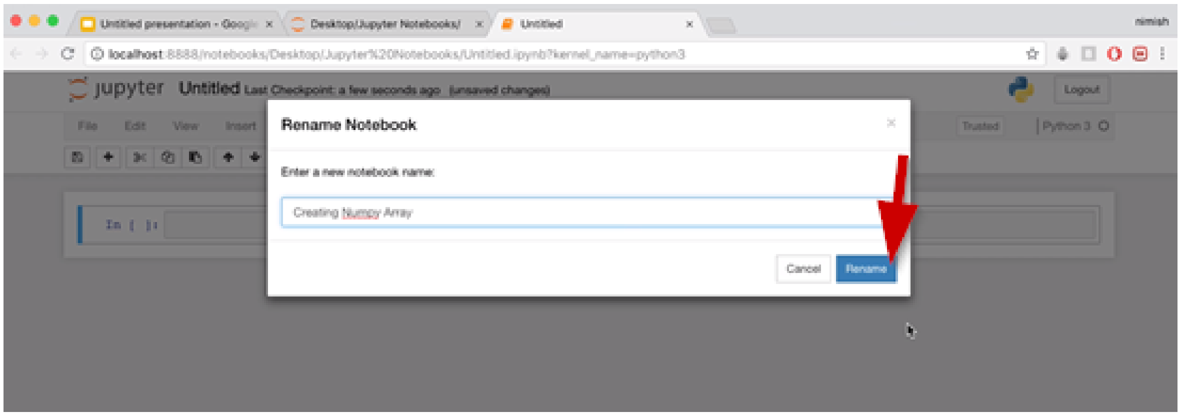
glueviz
0.13.3
Multidimensional data visualization across files. Explore relationships within and among related datasets.

orange3
3.13.0
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

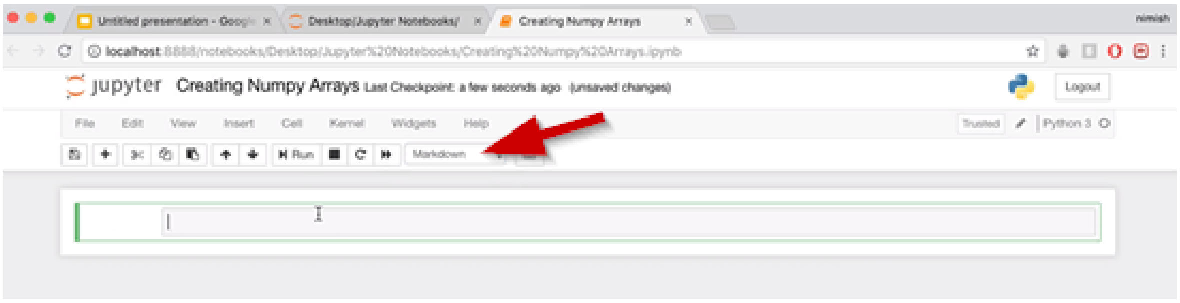
Start a new Jupyter Notebook using Python 3.


jupyter
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
Desktop / Jupyter Notebooks
The notebook list is empty.
Notebook:
Python 3
Other:
Text File
Folder
Terminal

Rename this to “Creating Numpy Array.”


jupyter Untitled Last Checkpoint: a few seconds ago (unsaved changes)
Rename Notebook
Enter a new notebook name:
Creating Numpy Array
Cancel
Rename

We will start with the **first cell being a Header so select Markdown from the drop down menu:**


jupyter Creating Numpy Arrays Last Checkpoint: a few seconds ago (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Run
Markdown

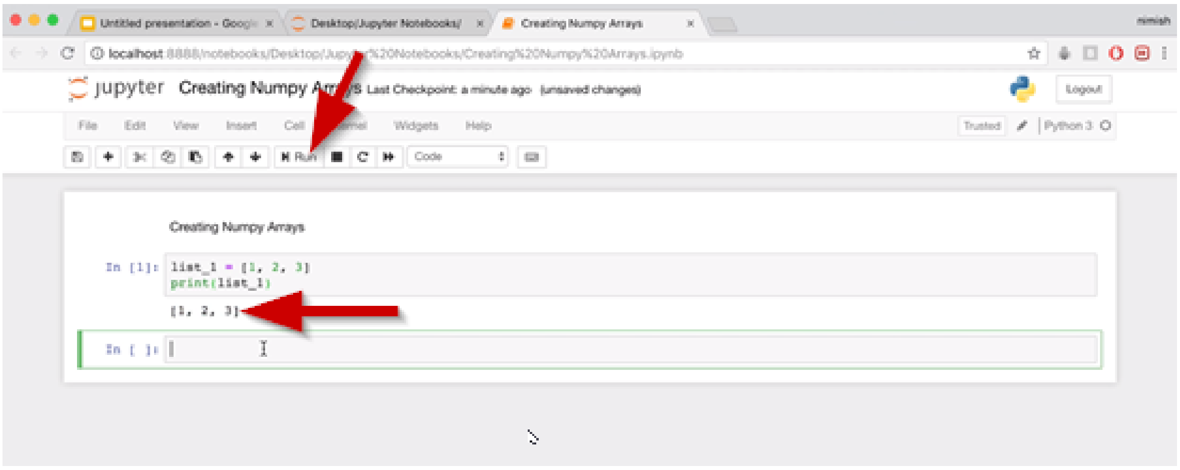
The Header will say **“Creating Numpy Arrays”**

So for starters you may have seen lists fin Python before. So in this example we will create **“list1.”**

See the code below and follow along:

```
list_1 = [1, 2, 3]
print(list_1)
```

So we will **print this list1 and then select Run from the menu:**


jupyter Creating Numpy Arrays Last Checkpoint: a minute ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

Creating Numpy Arrays

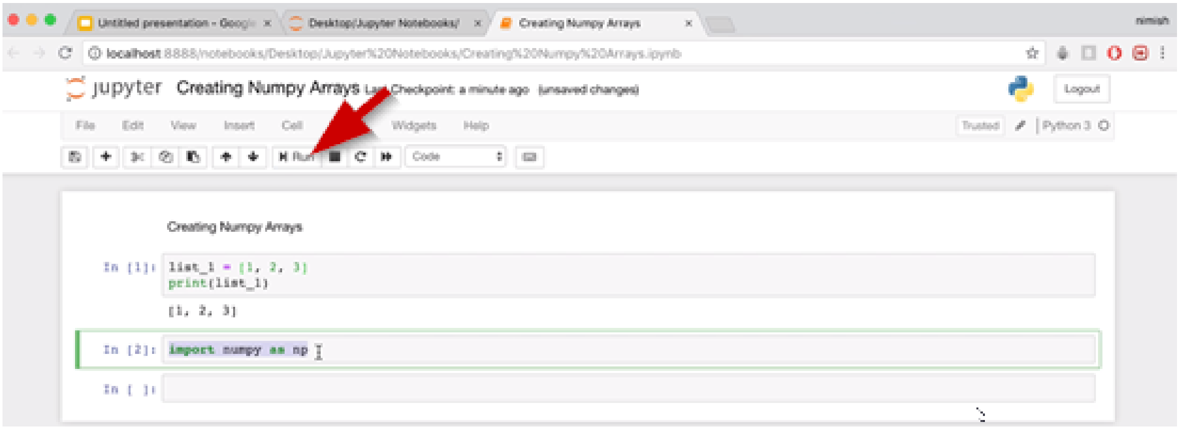
```
In [1]: list_1 = [1, 2, 3]
        print(list_1)

        [1, 2, 3]

In [ ]:
```

Numpy arrays are setup not so differently. Numpy arrays will take in a list as an argument and we'll convert that list into a numpy specific array. This attaches a bunch of extra functionality to it. However before we can actually do anything we'll want to add numpy as an import. The reason we need to do this is because numpy isn't actually built into the native Python library. So we need to simply add the import of the numpy to cell two.

See the code below and follow along:

```
import numpy as np
```

Make sure you **run cell 2.**


jupyter Creating Numpy Arrays Last Checkpoint: a minute ago (unsaved changes)
Logout
File Edit View Insert Cell Widgets Help
Trusted
Python 3
Run
Code

Creating Numpy Arrays

```
In [1]: list_1 = [1, 2, 3]
        print(list_1)

        [1, 2, 3]

In [2]: import numpy as np

In [ ]:
```

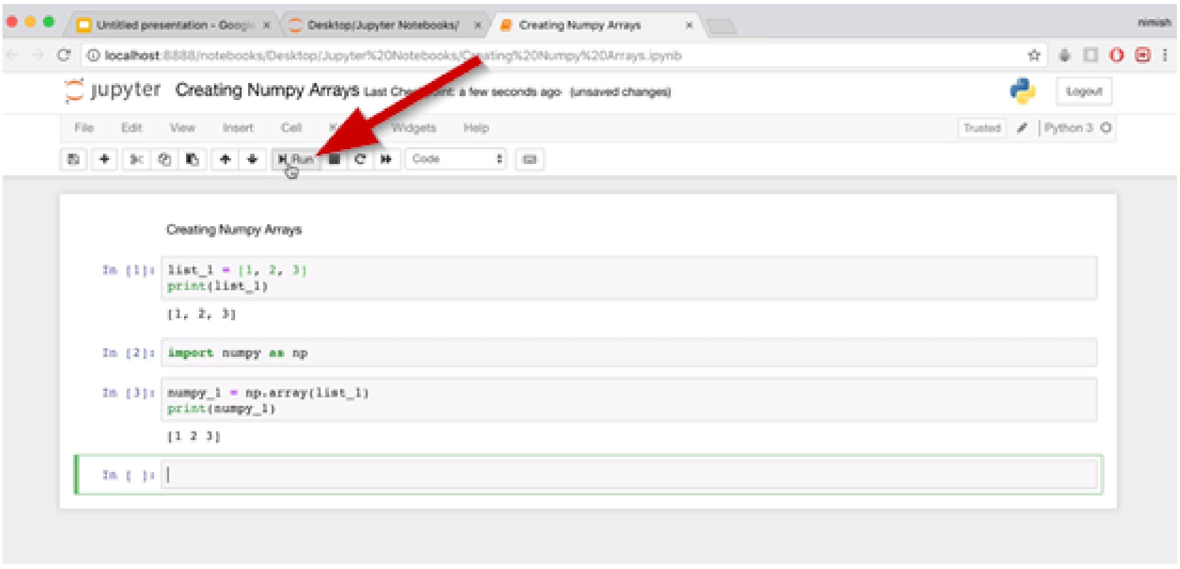
This gives us access to that library.

So let's say we want to c**reate a numpy based on list1 that we created in cell 1.**

See the code below and follow along:

```
numpy_1 = np.array(list_1)
print(numpy_1)
```

Run the cell.


Untitled presentation - Google
Desktop/Jupyter Notebooks/
Creating Numpy Arrays
localhost:8888/notebooks/Desktop/Jupyter%20Notebooks/Creating%20Numpy%20Arrays.ipynb
jupyter Creating Numpy Arrays Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

Creating Numpy Arrays

```
In [1]: list_1 = [1, 2, 3]
        print(list_1)

        [1, 2, 3]

In [2]: import numpy as np

In [3]: numpy_1 = np.array(list_1)
        print(numpy_1)

        [1 2 3]

In [ ]: 
```

We again got 1, 2, 3 printed out just like before.

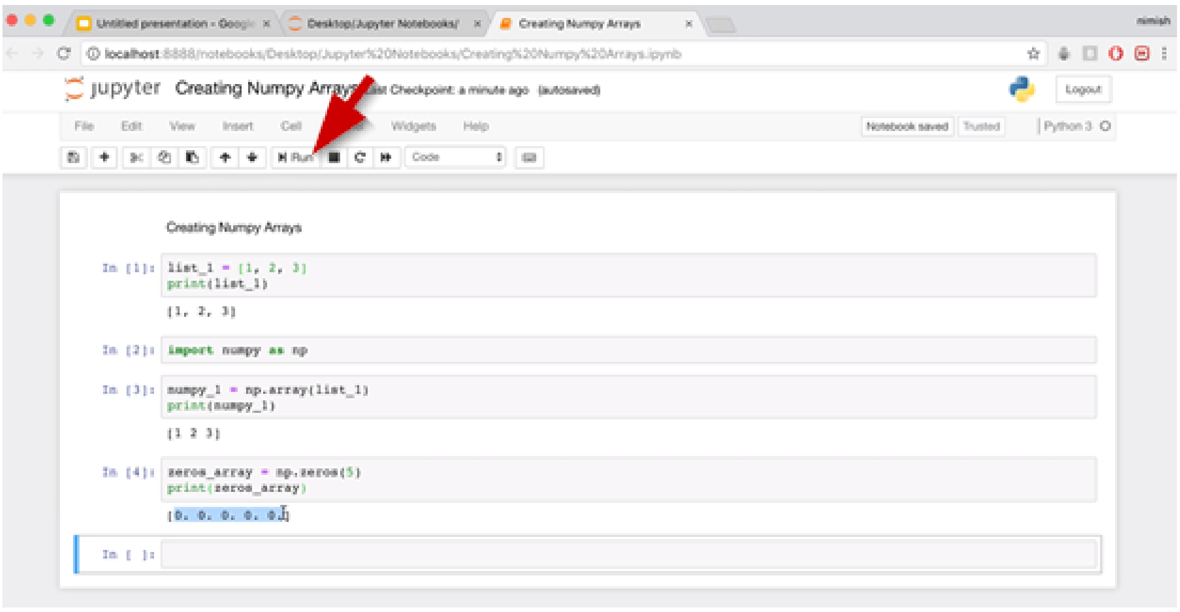
You will notice the format is a little different.

The first cell was a list but the third cell was a numpy array. Earlier, it was mentioned that we can create arrays of zeroes or ones. This is very often a starting point for such topics as machine learning. Like if you were to build machine learning software, and we know that we want an array maybe of size 20. So we want 20 elements in an array. We might initialize the array to be 20-zeroes or 20-ones. Then we can go and modify each element.

So the way in which we do this is by calling the zeroes or the ones operator. So we could create for example a zeroes array. See the code below and follow along:

```
zeros_array = np.zeros(5)
print(zeros_array)
```

Run the cell.


Untitled presentation - Google
Desktop/Jupyter Notebooks/
Creating Numpy Arrays
nimish
localhost:8888/notebooks/Desktop/Jupyter%20Notebooks/Creating%20Numpy%20Arrays.ipynb
jupyter
Creating Numpy Arrays Last Checkpoint: a minute ago (autosaved)
Logout
File
Edit
View
Insert
Cell
Widgets
Help
Notebook saved
Trusted
Python 3
Run
Code

Creating Numpy Arrays

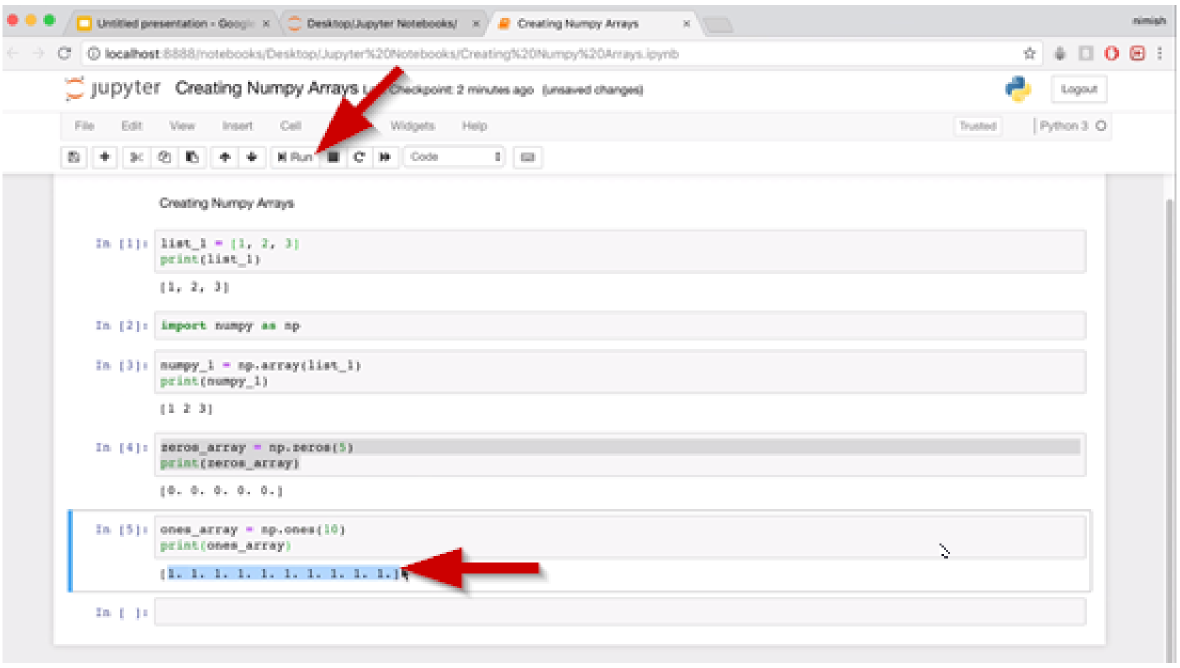
```
In [1]: list_1 = [1, 2, 3]
        print(list_1)

        [1, 2, 3]

In [2]: import numpy as np

In [3]: numpy_1 = np.array(list_1)
        print(numpy_1)

        [1 2 3]

In [4]: zeros_array = np.zeros(5)
        print(zeros_array)

        [0. 0. 0. 0. 0.]

In [ ]:
```

So we get an array of five zeroes.

We can do the same thing for ones. See the code below and follow along:

```
ones_array = np.ones(10)
print(ones_array)
```

Run the cell.


Untitled presentation - Google
Desktop/Jupyter Notebooks/
Creating Numpy Arrays
nimish
localhost:8888/notebooks/Desktop/Jupyter%20Notebooks/Creating%20Numpy%20Arrays.ipynb
jupyter
Creating Numpy Arrays Last Checkpoint: 2 minutes ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Widgets
Help
Trusted
Python 3
Run
Code

Creating Numpy Arrays

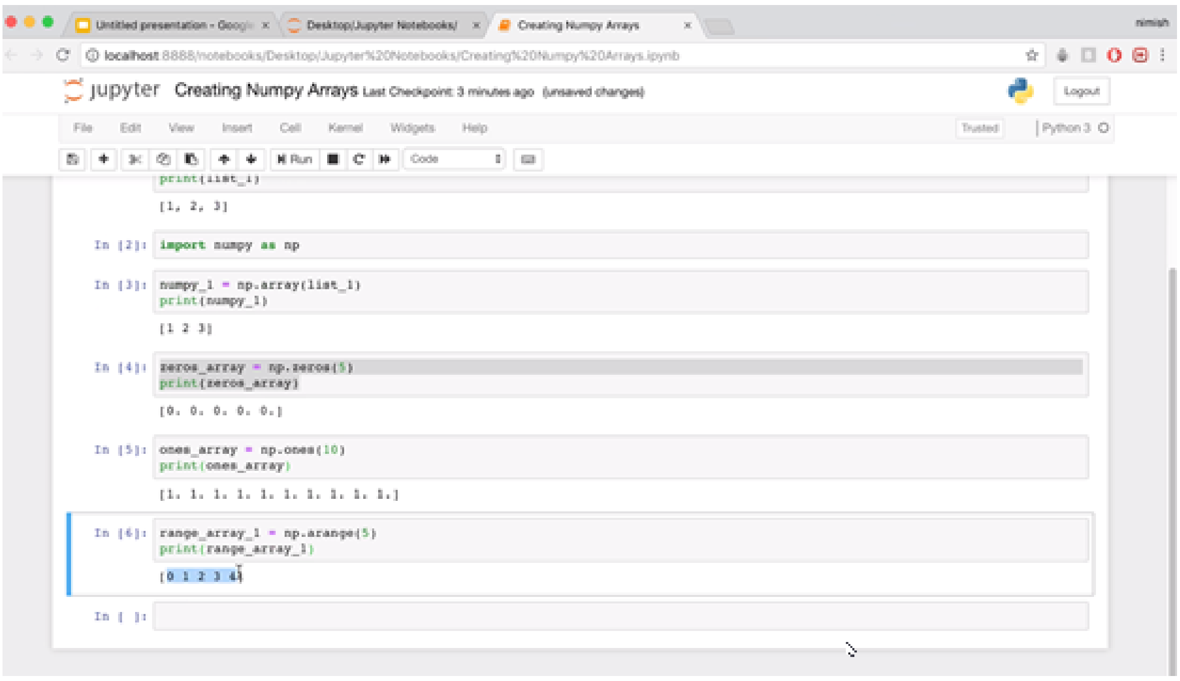
```
In [1]: list_1 = [1, 2, 3]
        print(list_1)

        [1, 2, 3]

In [2]: import numpy as np

In [3]: numpy_1 = np.array(list_1)
        print(numpy_1)

        [1 2 3]

In [4]: zeros_array = np.zeros(5)
        print(zeros_array)

        [0. 0. 0. 0. 0.]

In [5]: ones_array = np.ones(10)
        print(ones_array)

        [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

In [ ]:
```

So we have the array of ones that was printed out for us.

So if we wanted maybe a start point and an end point, and add a step interval. See the code below and follow along:

```
range_array_1 = np.arange(5)
print(range_array_1)
```

Run the cell.


Untitled presentation - Google
Desktop/Jupyter Notebooks/
Creating Numpy Arrays
localhost:8888/notebooks/Desktop/Jupyter%20Notebooks/Creating%20Numpy%20Arrays.ipynb
jupyter Creating Numpy Arrays Last Checkpoint: 3 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

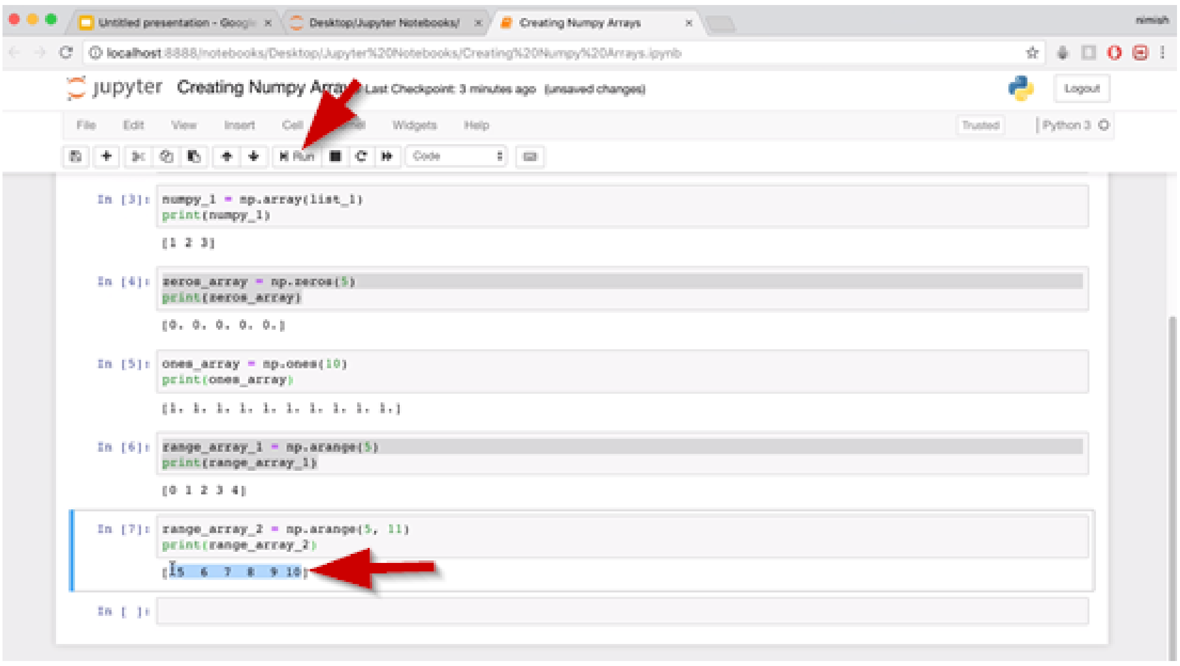
```
print(list_1)
[1, 2, 3]

In [2]: import numpy as np

In [3]: numpy_1 = np.array(list_1)
        print(numpy_1)
[1 2 3]

In [4]: zeros_array = np.zeros(5)
        print(zeros_array)
[0. 0. 0. 0. 0.]

In [5]: ones_array = np.ones(10)
        print(ones_array)
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

In [6]: range_array_1 = np.arange(5)
        print(range_array_1)
[0 1 2 3 4]

In [ ]:
```

So we get five elements printed out, but this might not be what we expected, because we put in a "5" with the "arange" operator this is the upper bound. So this is saying its going to stop before it reaches five. We were only specifying the end point.

So if we want to specify a start point as well, let's say we want the elements five to 10. See the code below and follow along:

```
range_array_2 = np.arange(5, 11)
print(range_array_2)
```

Run the cell.

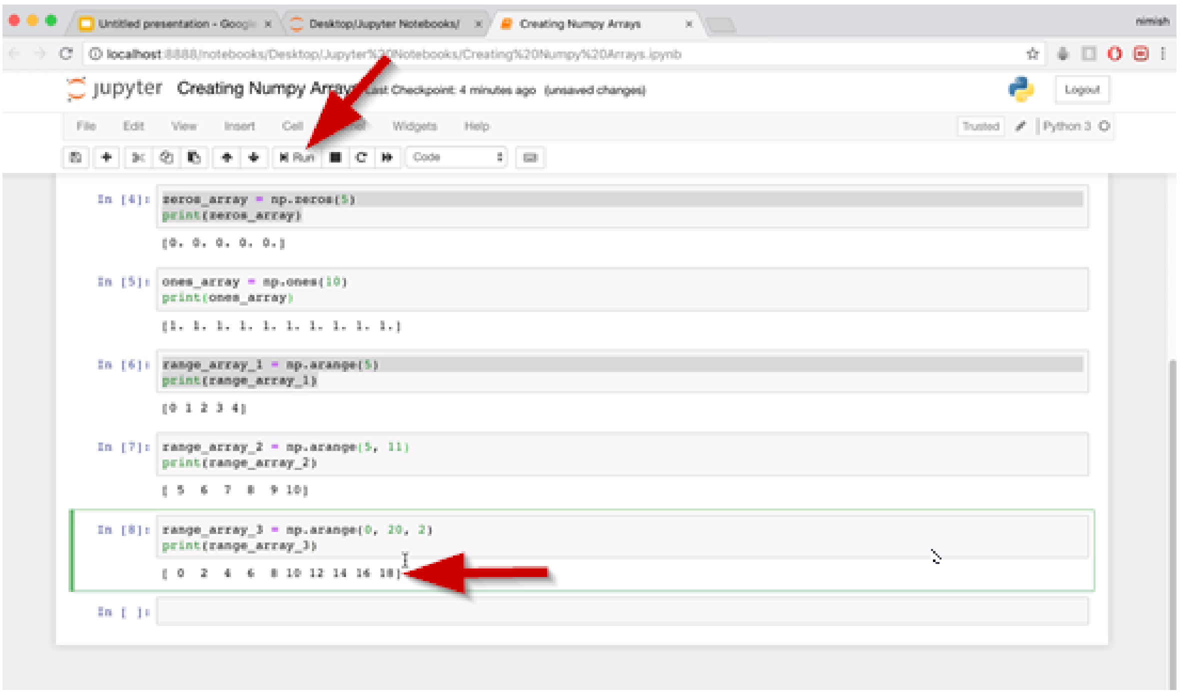


```
In [3]: numpy_1 = np.array(list_1)
        print(numpy_1)
        [1 2 3]
In [4]: zeros_array = np.zeros(5)
        print(zeros_array)
        [0. 0. 0. 0. 0.]
In [5]: ones_array = np.ones(10)
        print(ones_array)
        [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
In [6]: range_array_1 = np.arange(5)
        print(range_array_1)
        [0 1 2 3 4]
In [7]: range_array_2 = np.arange(5, 11)
        print(range_array_2)
        [ 5  6  7  8  9 10]
In [ ]:
```

Now let’s say we want to step by twos. So maybe we want to start at zero go until 20 and we want to skip every other element. We can do that be specifying a third parameter being the step. See the code below and follow along:

```
range_array_3 = np.arange(0, 20, 2)
print(range_array_3)
```

Run the cell.

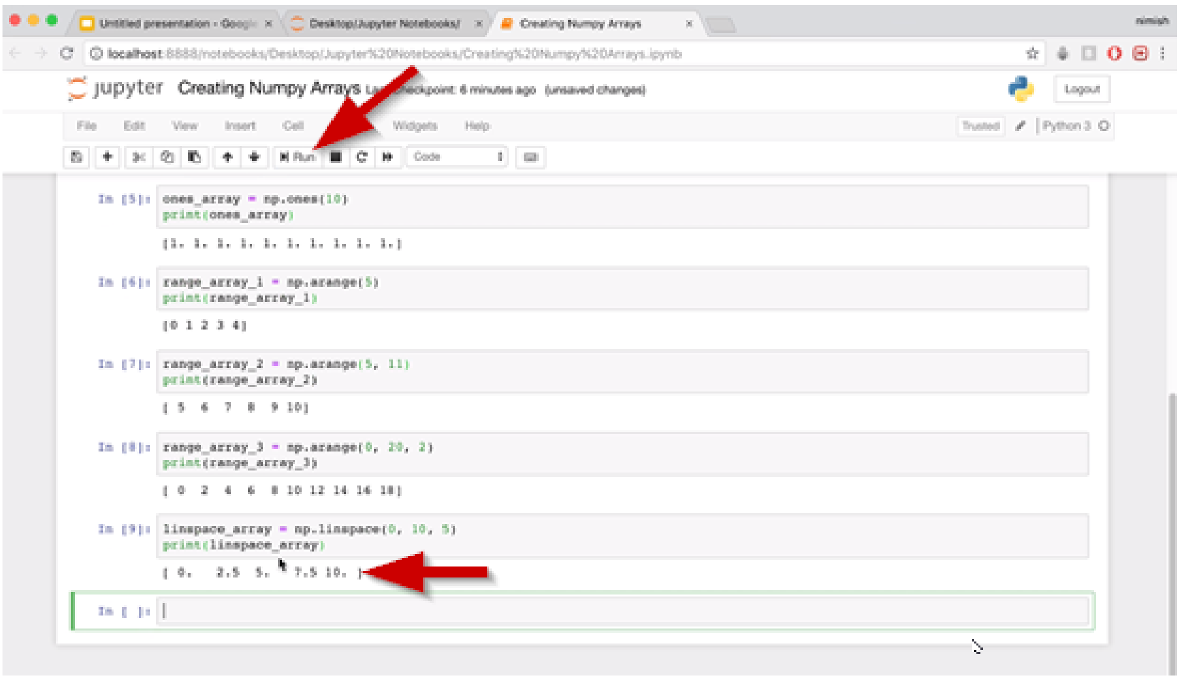


```
In [4]: zeros_array = np.zeros(5)
        print(zeros_array)
        [0. 0. 0. 0. 0.]
In [5]: ones_array = np.ones(10)
        print(ones_array)
        [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
In [6]: range_array_1 = np.arange(5)
        print(range_array_1)
        [0 1 2 3 4]
In [7]: range_array_2 = np.arange(5, 11)
        print(range_array_2)
        [ 5  6  7  8  9 10]
In [8]: range_array_3 = np.arange(0, 20, 2)
        print(range_array_3)
        [ 0  2  4  6  8 10 12 14 16 18]
In [ ]:
```

The final thing we will cover is the **linspace operator**. **Linspace is a conjunction for linear space.** With the linspace we basically take a lower bound an upper bound and the number of elements we want it to populate between the upper bound and lower bound. See the code below and follow along:

```
linspace_array = np.linspace(0, 10, 5)
print(linspace_array)
```

Run the cell.


Untitled presentation - Google
Desktop/Jupyter Notebooks/
Creating Numpy Arrays
localhost:8888/notebooks/Desktop/Jupyter%20Notebooks/Creating%20Numpy%20Arrays.ipynb
jupyter
Creating Numpy Arrays Last Checkpoint: 6 minutes ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Widgets
Help
Trusted
Python 3
Run
Code

```
In [5]: ones_array = np.ones(10)
        print(ones_array)

        [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

In [6]: range_array_1 = np.arange(5)
        print(range_array_1)

        [0 1 2 3 4]

In [7]: range_array_2 = np.arange(5, 11)
        print(range_array_2)

        [ 5  6  7  8  9 10]

In [8]: range_array_3 = np.arange(0, 20, 2)
        print(range_array_3)

        [ 0  2  4  6  8 10 12 14 16 18]

In [9]: linspace_array = np.linspace(0, 10, 5)
        print(linspace_array)

        [ 0.   2.5  5.   7.5 10. ]

In [ ]: 
```

So what this has done is created an array of five elements evenly spaced between zero and ten. Again ten inclusive, zero inclusive.

This will be really useful to us when it comes to graphing values. Because we can just specify and upper bound and a lower bound and then how many points we want in between those two.

Numpy fetch functions, this will be a series of functions that will help us to get an element, multiple elements, or various properties of an array.

Get Elements & Properties of Arrays

- Get elements based on index
- Slice arrays
- Access ranges
- Get length of arrrays
- Get max, min, average of arrrays

Create a new Jupyter Notebook and call it "Numpy Fetch Operations"

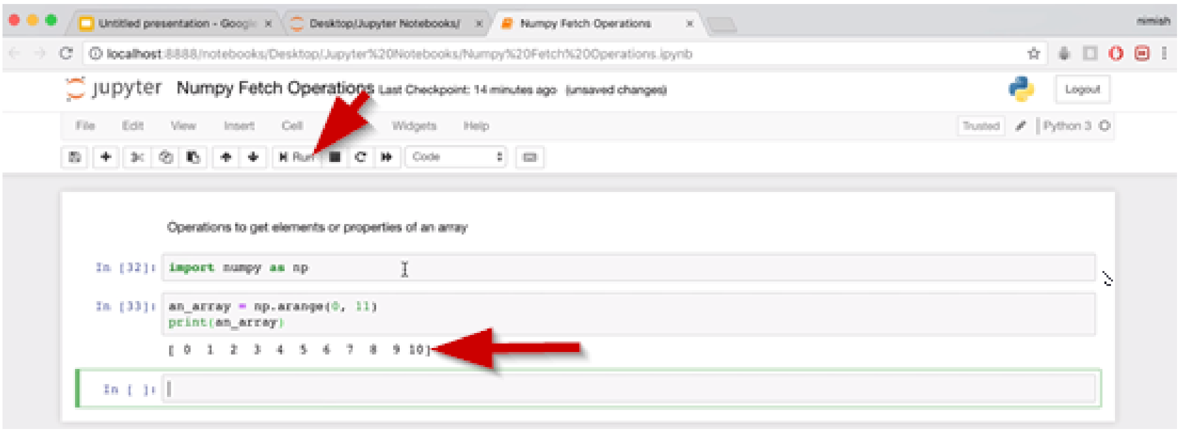
Create the header and name it "Operations to get elements or properties of an array"

Make sure to **import numpy using the import statement:**

```
import numpy as np
```

The first thing we will do is **create a numpy array.** See the code below and follow along:

```
an_array = np.arange(0, 11)
print(an_array)
```


Untitled presentation - Google
Desktop/Jupyter Notebooks/
Numpy Fetch Operations
localhost:8888/notebooks/Desktop/Jupyter%20Notebooks/Numpy%20Fetch%20Operations.ipynb
jupyter Numpy Fetch Operations Last Checkpoint: 14 minutes ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Widgets
Help
Trusted
Python 3
Run
Code

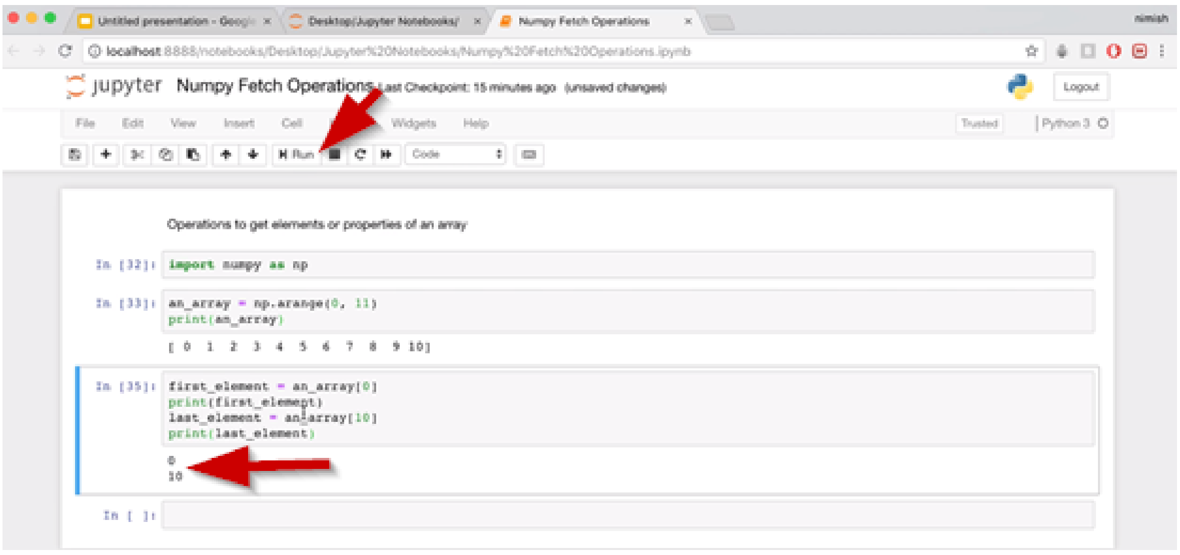
Operations to get elements or properties of an array

```
In [32]: import numpy as np

In [33]: an_array = np.arange(0, 11)
         print(an_array)

         [ 0  1  2  3  4  5  6  7  8  9 10]

In [ ]:
```

For the next cell see the code below and follow along:

```
first_element = an_array[0]
print(first_element)
last_element = an_array[10]
print(last_element)
```

Run the cell.


jupyter Numpy Fetch Operations Last Checkpoint: 15 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
Trusted
Python 3
Logout

Operations to get elements or properties of an array

```
In [32]: import numpy as np

In [33]: an_array = np.arange(0, 11)
         print(an_array)

         [ 0  1  2  3  4  5  6  7  8  9 10]

In [35]: first_element = an_array[0]
         print(first_element)
         last_element = an_array[10]
         print(last_element)

         0
         10

In [ ]:
```

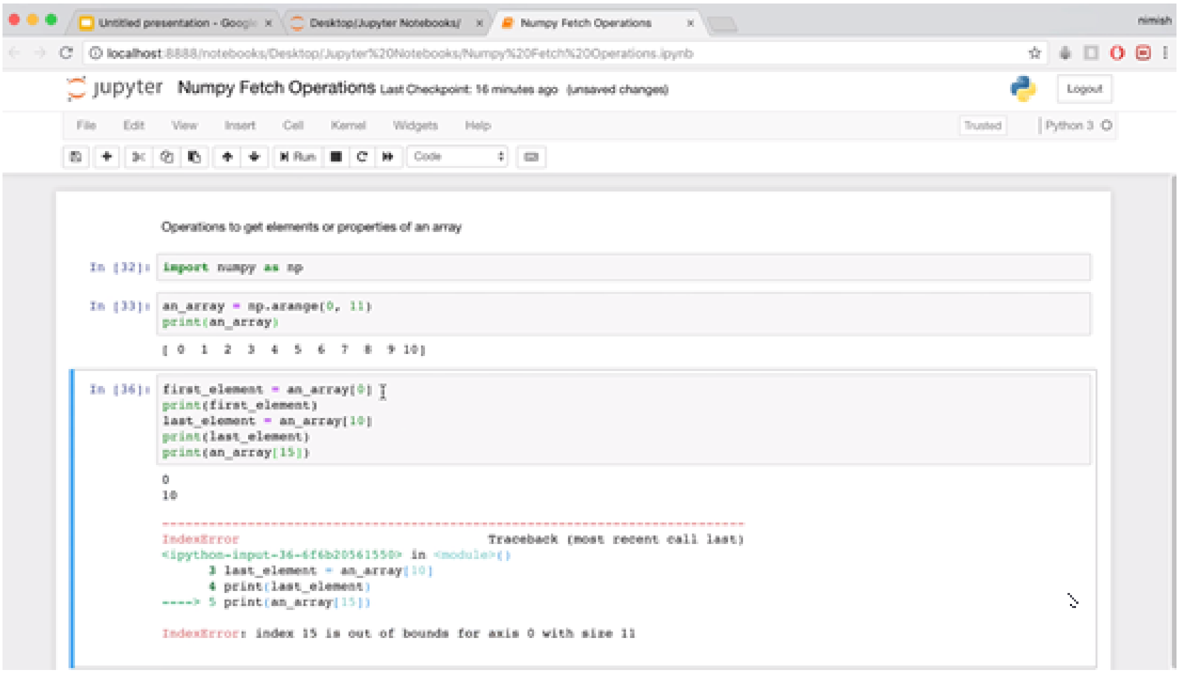
Now what numpy doesn’t do is check for array out of bounds exceptions, these are still going to be thrown if we try to access an element that doesn’t access an index that doesn’t exist.

See the code below and follow along:

```
first_element = an_array[0]
print(first_element)
last_element = an_array[10]
print(last_element)
print(an_array[15])
```

Run the cell.

You will see the error:


jupyter Numpy Fetch Operations Last Checkpoint: 16 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
Trusted
Python 3
Logout

Operations to get elements or properties of an array

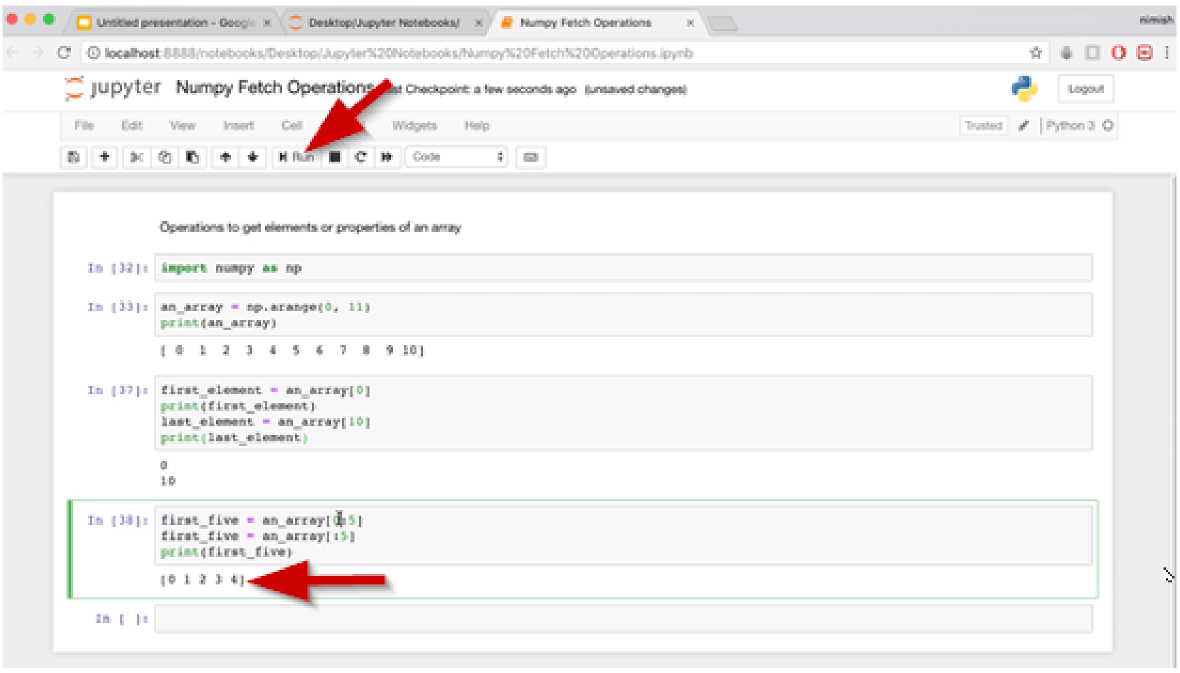
```
In [32]: import numpy as np

In [33]: an_array = np.arange(0, 11)
         print(an_array)

         [ 0  1  2  3  4  5  6  7  8  9 10]

In [36]: first_element = an_array[0]
         print(first_element)
         last_element = an_array[10]
         print(last_element)
         print(an_array[15])

         0
         10

         ---------------------------------------------------------------------------
         IndexError                                Traceback (most recent call last)
         <ipython-input-36-4f6b20561550> in <module>()
               3 last_element = an_array[10]
               4 print(last_element)
         ----> 5 print(an_array[15])

         IndexError: index 15 is out of bounds for axis 0 with size 11
```

What we will want to do in order to make sure we are never out of bounds is always check, so as long as, again, the index that we're trying to access is less than the length of the array then we are good to go.

If we want to access multiple elements, let's say we want to access the very first five elements, so zero through four. For that we will need a range or a slice. See the code below and follow along:

```
first_five = an_array[0:5]
first_five = an_array[:5]
print(first_five)
```

Run the cell.


jupyter Numpy Fetch Operations Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

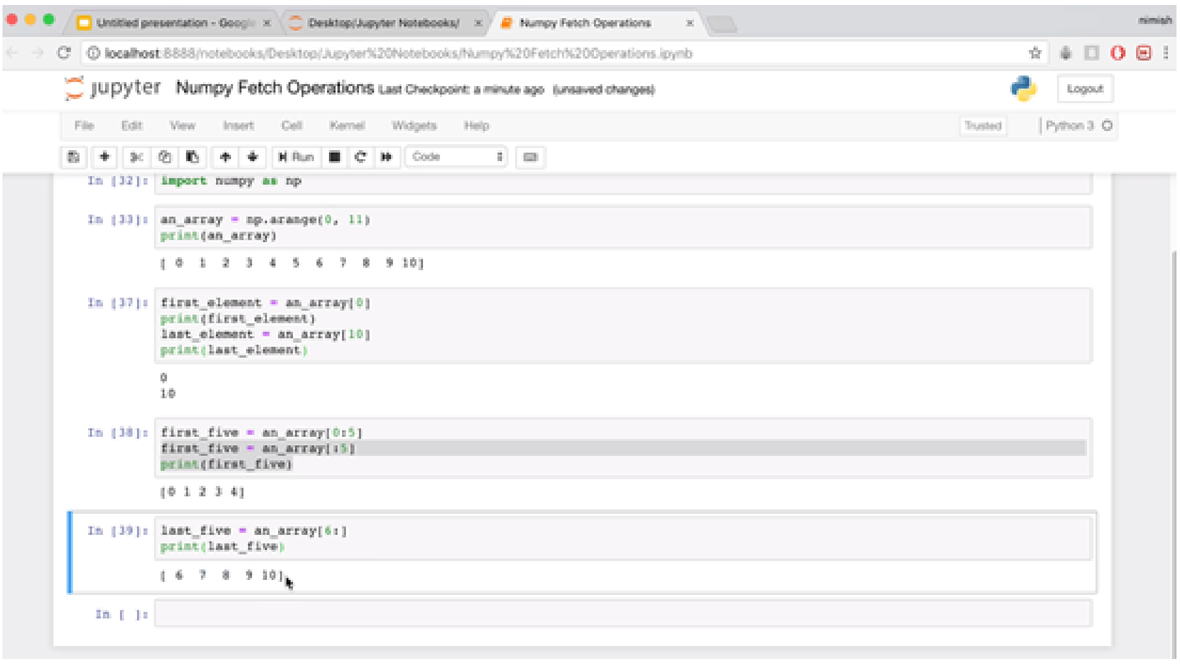
Operations to get elements or properties of an array

```
In [32]: import numpy as np

In [33]: an_array = np.arange(0, 11)
         print(an_array)

         [ 0  1  2  3  4  5  6  7  8  9 10]

In [37]: first_element = an_array[0]
         print(first_element)
         last_element = an_array[10]
         print(last_element)

         0
         10

In [38]: first_five = an_array[0:5]
         first_five = an_array[:5]
         print(first_five)

         [0 1 2 3 4]

In [ ]:
```

See the code below and follow along for the next cell:

```
last_five = an_array[6:]
print(last_five)
```

Run the cell.

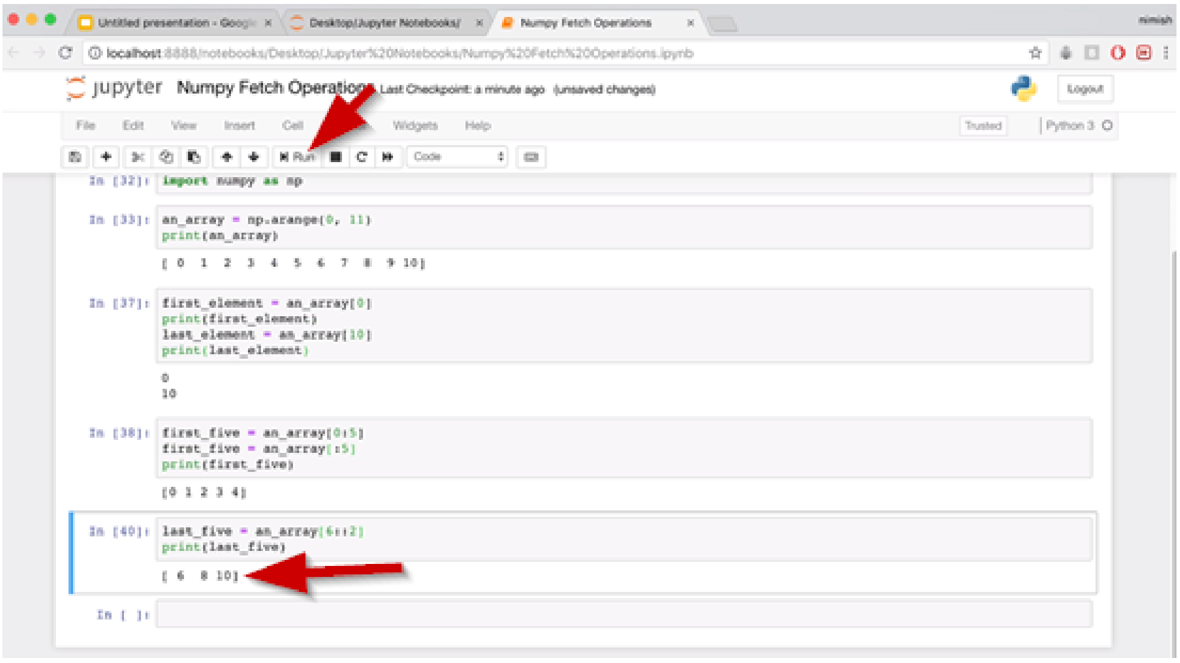

localhost:8888/notebooks/Desktop/Jupyter%20Notebooks/Numpy%20Fetch%20Operations.ipynb
jupyter Numpy Fetch Operations Last Checkpoint: a minute ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [32]: import numpy as np

In [33]: an_array = np.arange(0, 11)
         print(an_array)

         [ 0  1  2  3  4  5  6  7  8  9 10]

In [37]: first_element = an_array[0]
         print(first_element)
         last_element = an_array[10]
         print(last_element)

         0
         10

In [38]: first_five = an_array[0:5]
         first_five = an_array[:5]
         print(first_five)

         [0 1 2 3 4]

In [39]: last_five = an_array[6:]
         print(last_five)

         [ 6  7  8  9 10]

In [ ]:
```

We can also provide a **stride length.** So let’s say we wanted the last five elements, but we want to stride by two. See the code below and follow along:

```
last_five = an_array[6::2]
print(last_five)
```

Run the cell.


localhost:8888/notebooks/Desktop/Jupyter%20Notebooks/Numpy%20Fetch%20Operations.ipynb
jupyter Numpy Fetch Operations Last Checkpoint: a minute ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

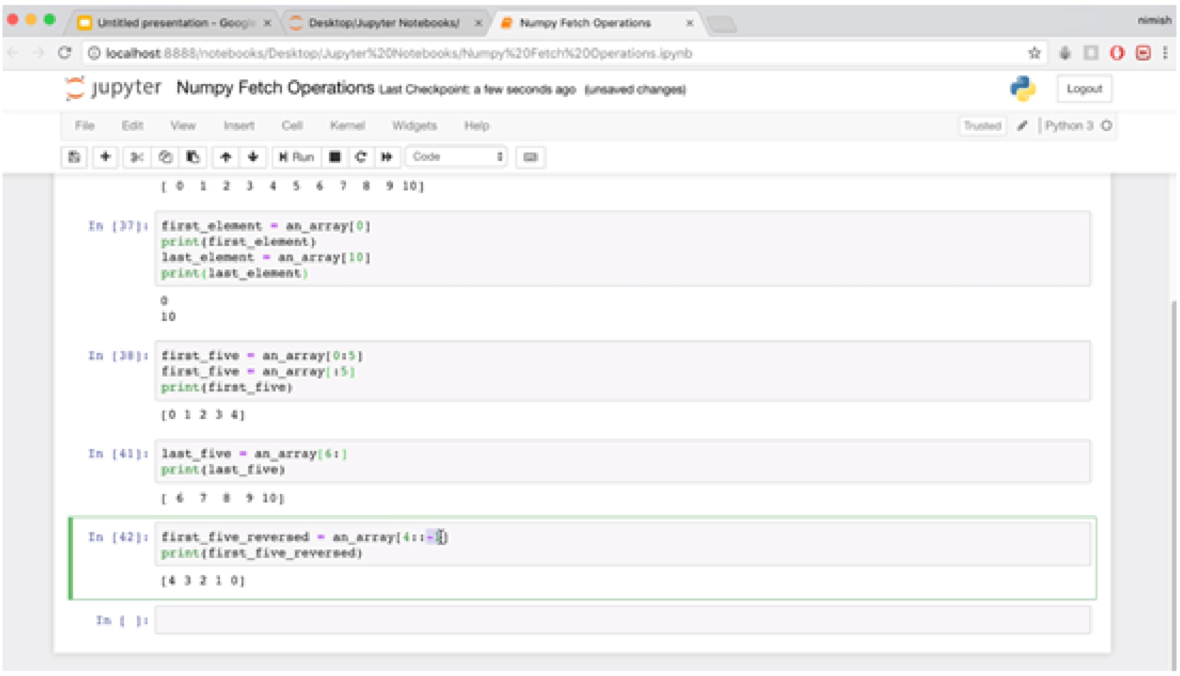
```
In [32]: import numpy as np

In [33]: an_array = np.arange(0, 11)
         print(an_array)

         [ 0  1  2  3  4  5  6  7  8  9 10]

In [37]: first_element = an_array[0]
         print(first_element)
         last_element = an_array[10]
         print(last_element)

         0
         10

In [38]: first_five = an_array[0:5]
         first_five = an_array[:5]
         print(first_five)

         [0 1 2 3 4]

In [40]: last_five = an_array[6::2]
         print(last_five)

         [ 6  8 10]

In [ ]:
```

We are striding by two there and skipping every other element.

Let’s say we wanted to count down, all we have to do is specify stride length of negative one. See the code below and follow along:

```
first_five_reversed = an_array[4::-1]
print(first_five_reversed)
```

Run the cell.



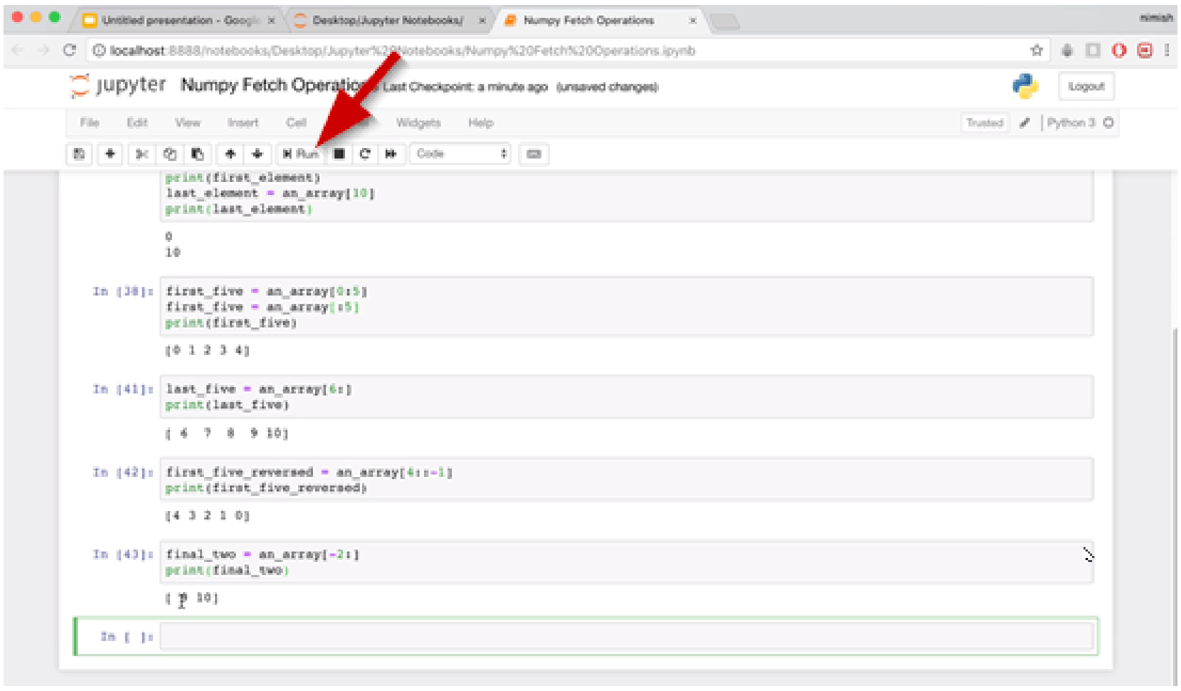
jupyter Numpy Fetch Operations Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
[ 0  1  2  3  4  5  6  7  8  9 10]

In [37]: first_element = an_array[0]
         print(first_element)
         last_element = an_array[10]
         print(last_element)

0
10

In [38]: first_five = an_array[0:5]
         first_five = an_array[:5]
         print(first_five)

[0 1 2 3 4]

In [41]: last_five = an_array[6:]
         print(last_five)

[ 6  7  8  9 10]

In [42]: first_five_reversed = an_array[4::-1]
         print(first_five_reversed)

[4 3 2 1 0]

In [ ]:
```

See the code below and follow along for the next cell:

```
final_two = an_array[-2:]
print(final_two)
```

Run the cell.

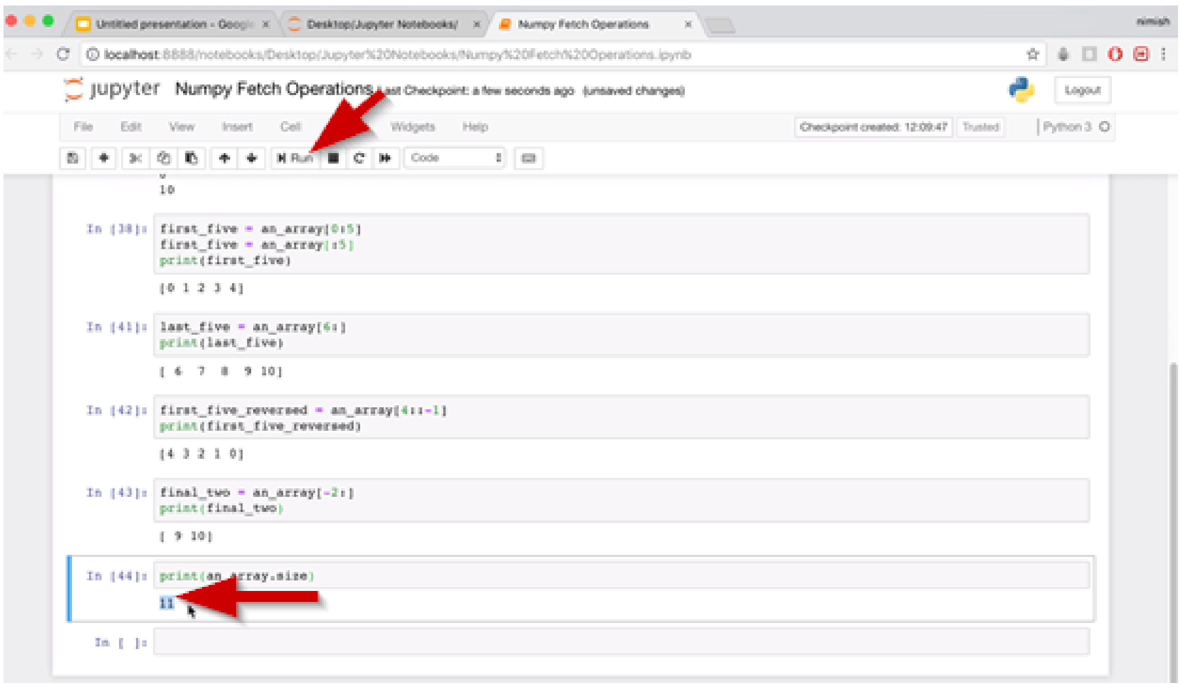

Untitled presentation - Googl
Desktop/Jupyter Notebooks/
Numpy Fetch Operations
localhost:8888/notebooks/Desktop/Jupyter%20Notebooks/Numpy%20Fetch%20Operations.ipynb
jupyter Numpy Fetch Operations Last Checkpoint: a minute ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
         print(first_element)
         last_element = an_array[10]
         print(last_element)

         0
         10

In [38]: first_five = an_array[0:5]
         first_five = an_array[:5]
         print(first_five)

         [0 1 2 3 4]

In [41]: last_five = an_array[6:]
         print(last_five)

         [ 6  7  8  9 10]

In [42]: first_five_reversed = an_array[4::-1]
         print(first_five_reversed)

         [4 3 2 1 0]

In [43]: final_two = an_array[-2:]
         print(final_two)

         [ 9 10]

In [ ]:
```

The **length of an array is calculated based on our array name and then we use .size.** See the code below and follow along:

```
print(an_array.size)
```

Run the cell.

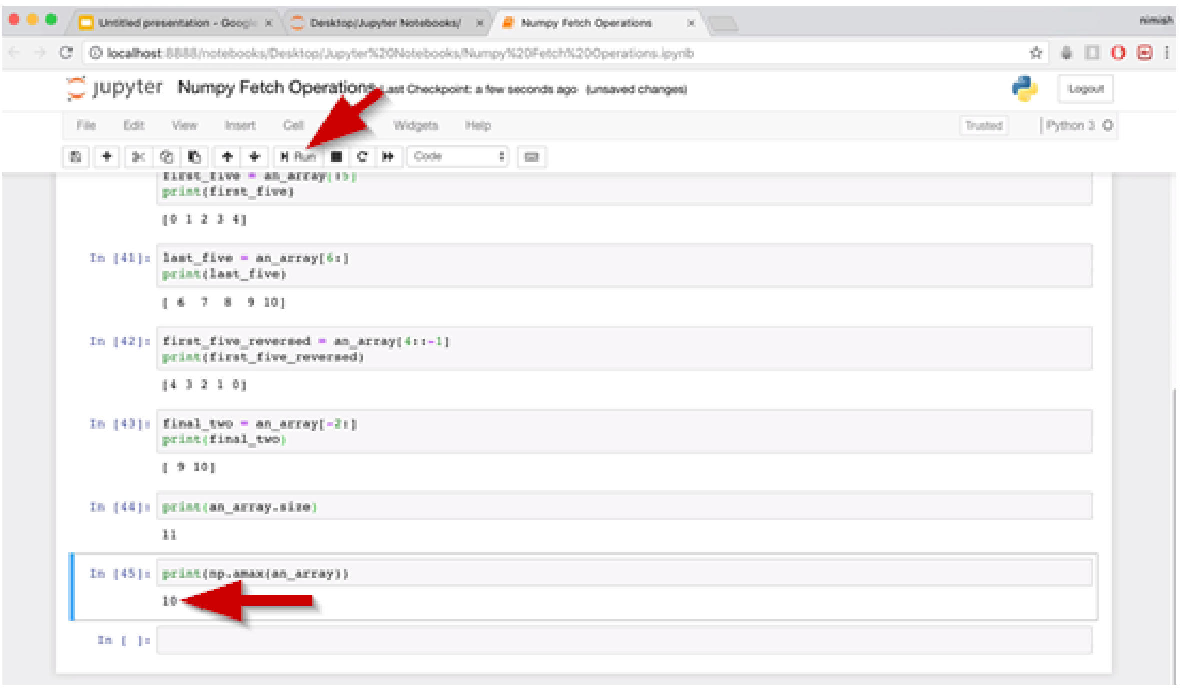

Untitled presentation - Googl
Desktop/Jupyter Notebooks/
Numpy Fetch Operations
localhost:8888/notebooks/Desktop/Jupyter%20Notebooks/Numpy%20Fetch%20Operations.ipynb
jupyter Numpy Fetch Operations Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Checkpoint created: 12:09:47
Trusted
Python 3
Run
Code

```
         10

In [38]: first_five = an_array[0:5]
         first_five = an_array[:5]
         print(first_five)

         [0 1 2 3 4]

In [41]: last_five = an_array[6:]
         print(last_five)

         [ 6  7  8  9 10]

In [42]: first_five_reversed = an_array[4::-1]
         print(first_five_reversed)

         [4 3 2 1 0]

In [43]: final_two = an_array[-2:]
         print(final_two)

         [ 9 10]

In [44]: print(an_array.size)

         11

In [ ]:
```

For the maximum, minimum, and average we have to **use np functions.** See the code below and follow along:

```
print(np.amax(an_array))
```

Run the cell.

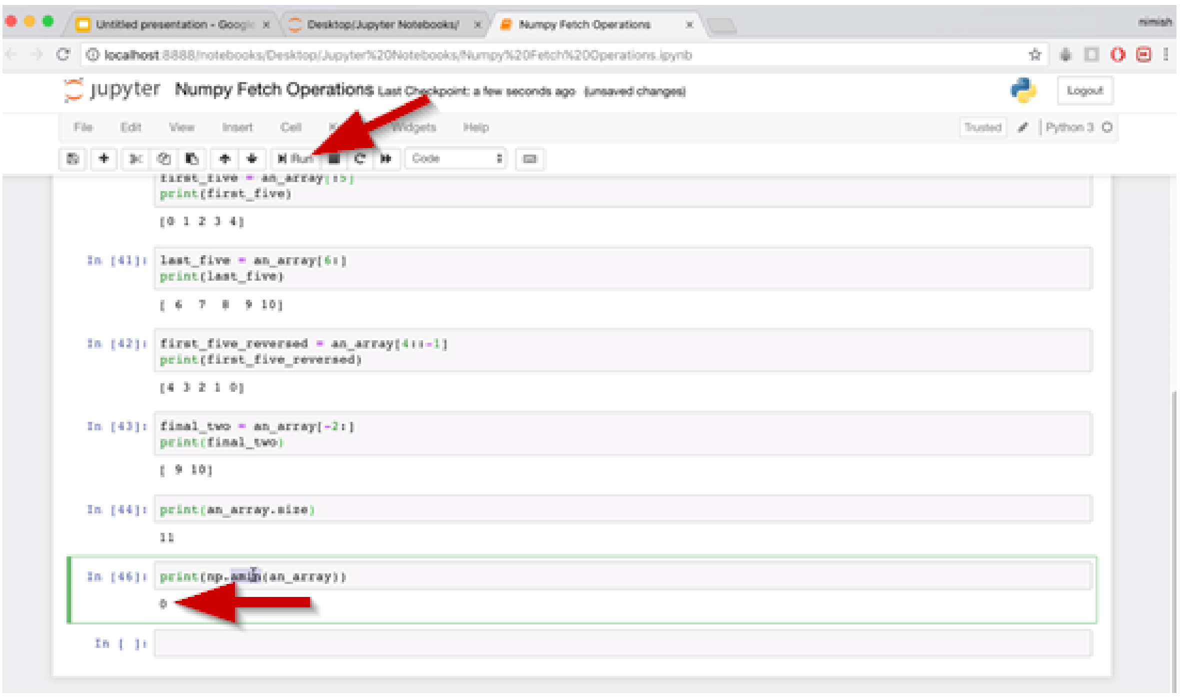

Untitled presentation - Google
Desktop/Jupyter Notebooks/
Numpy Fetch Operations
localhost:8888/notebooks/Desktop/Jupyter%20Notebooks/Numpy%20Fetch%20Operations.ipynb
jupyter Numpy Fetch Operations Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File Edit View Insert Cell Widgets Help
Trusted
Python 3
Run
Code

```
       first_five = an_array[:5]
       print(first_five)

       [0 1 2 3 4]

In [41]: last_five = an_array[6:]
         print(last_five)

         [ 6  7  8  9 10]

In [42]: first_five_reversed = an_array[4::-1]
         print(first_five_reversed)

         [4 3 2 1 0]

In [43]: final_two = an_array[-2:]
         print(final_two)

         [ 9 10]

In [44]: print(an_array.size)

         11

In [45]: print(np.amax(an_array))

         10

In [ ]:
```

See the code below:

```
print(np.amin(an_array))
```

Run the cell.

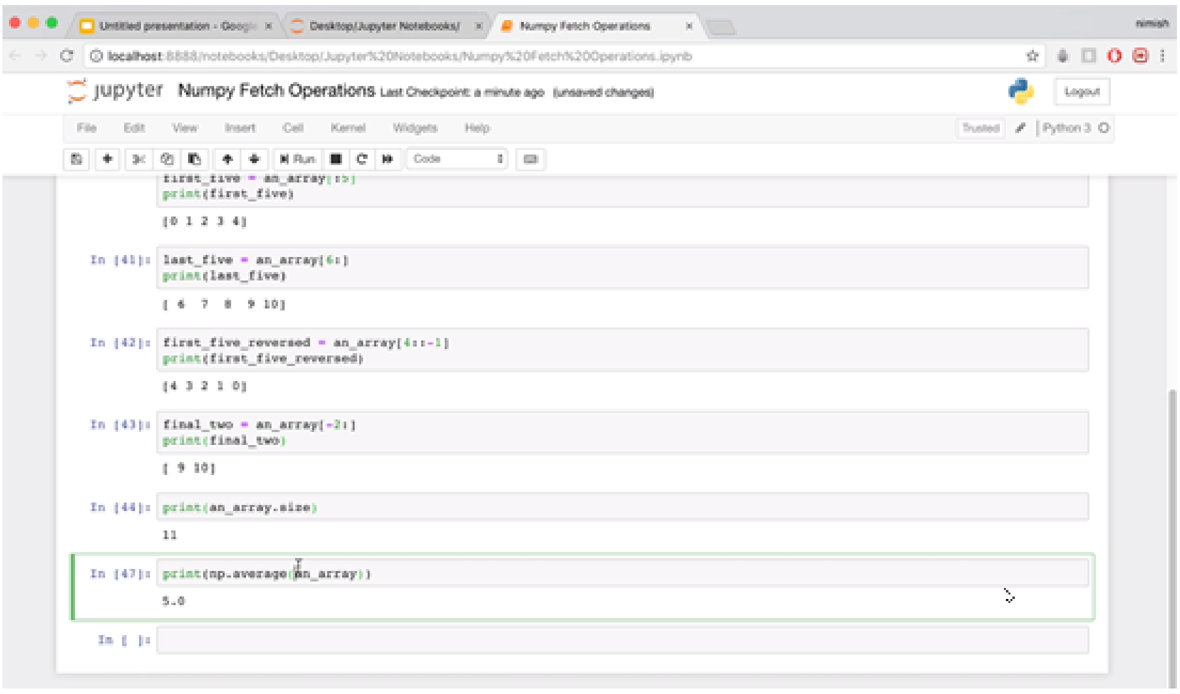


```
first_five = an_array[:5]
print(first_five)

[0 1 2 3 4]

In [41]: last_five = an_array[6:]
         print(last_five)

[ 6  7  8  9 10]

In [42]: first_five_reversed = an_array[4::-1]
         print(first_five_reversed)

[4 3 2 1 0]

In [43]: final_two = an_array[-2:]
         print(final_two)

[ 9 10]

In [44]: print(an_array.size)

11

In [46]: print(np.amin(an_array))

0
```

See the code below:

```
print(np.average(an_array))
```

Run the cell.



```
In [47]: print(np.average(an_array))

5.0
```

Modify Elements & Properties of Arrays

- Append, insert, delete operations
- Scalar and vector operations
- Sorting operations

Create a new Jupyter Notebook and call it “Numpy Modification Operations.”

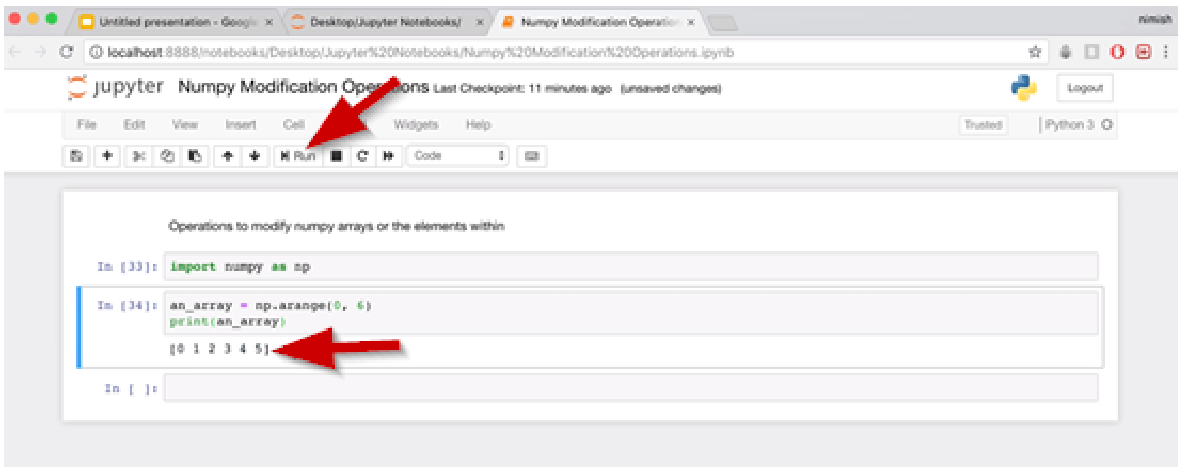
Create a header and **call it “Operations to modify numpy arrays or the elements within.”**

Import the numpy and **run the cell.**

The first thing we want to do is **create an array of elements.** See the code below and follow along:

```
an_array = np.arange(0, 6)
print(an_array)
```

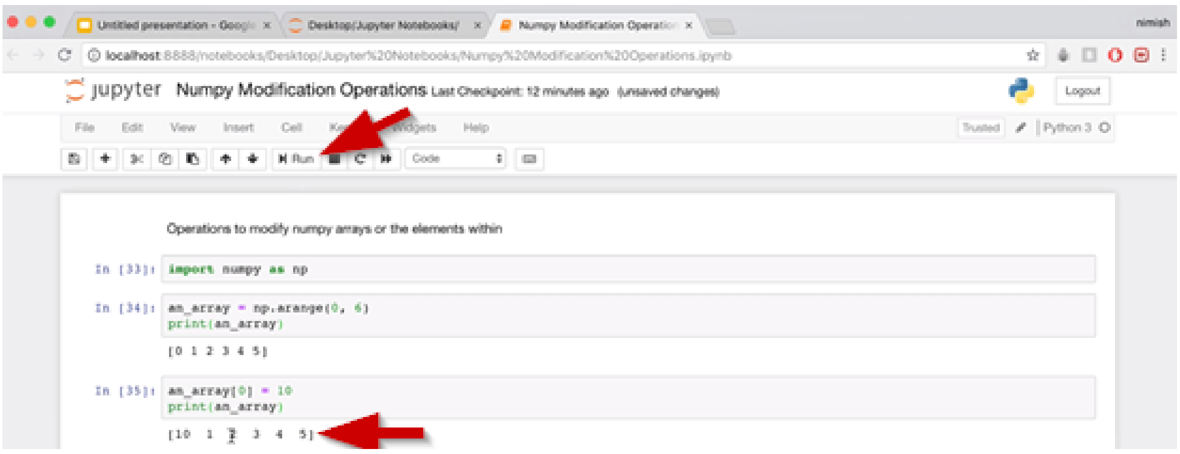
Run the cell.


```
Operations to modify numpy arrays or the elements within

In [33]: import numpy as np

In [34]: an_array = np.arange(0, 6)
         print(an_array)

         [0 1 2 3 4 5]

In [ ]:
```

Let’s say we want to **modify a single element within an array.** See the code below and follow along:

```
an_array[0] = 10
print(an_array)
```

Run the cell.


jupyter Numpy Modification Operations Last Checkpoint: 12 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

Operations to modify numpy arrays or the elements within

```
In [33]: import numpy as np

In [34]: an_array = np.arange(0, 6)
         print(an_array)

         [0 1 2 3 4 5]

In [35]: an_array[0] = 10
         print(an_array)

         [10  1  2  3  4  5]
```

We are going to **reset the "an_array"** see the code below:

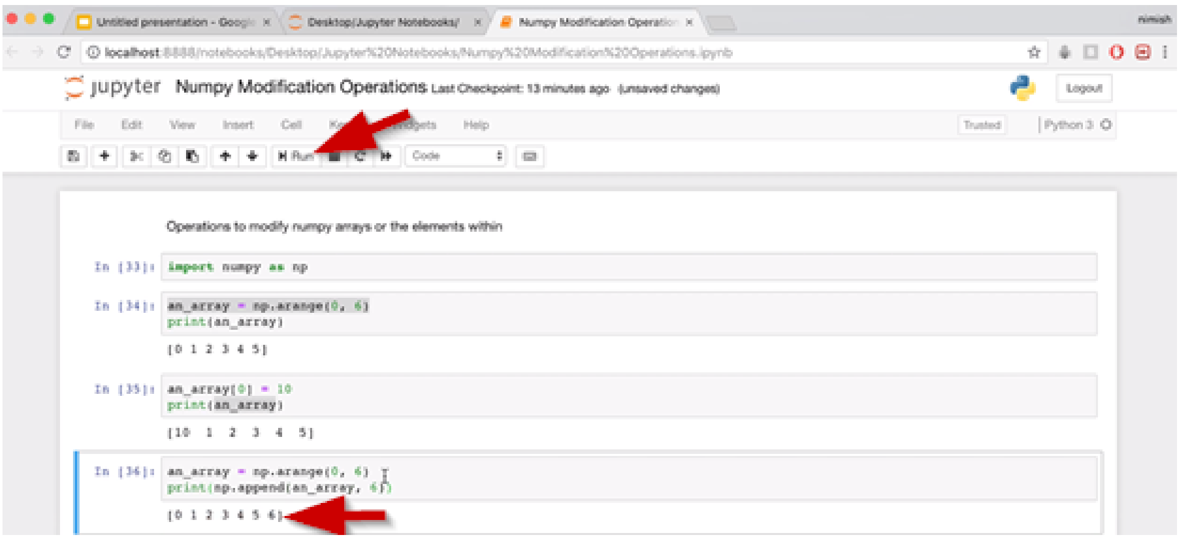
```
an_array = np.arange(0, 6)
```

Let's take a look at the **insert, append, and delete operations.**

See the code below and follow along:

```
an_array = np.arange(0, 6)
print(np.append(an_array, 6))
```

Run the cell.

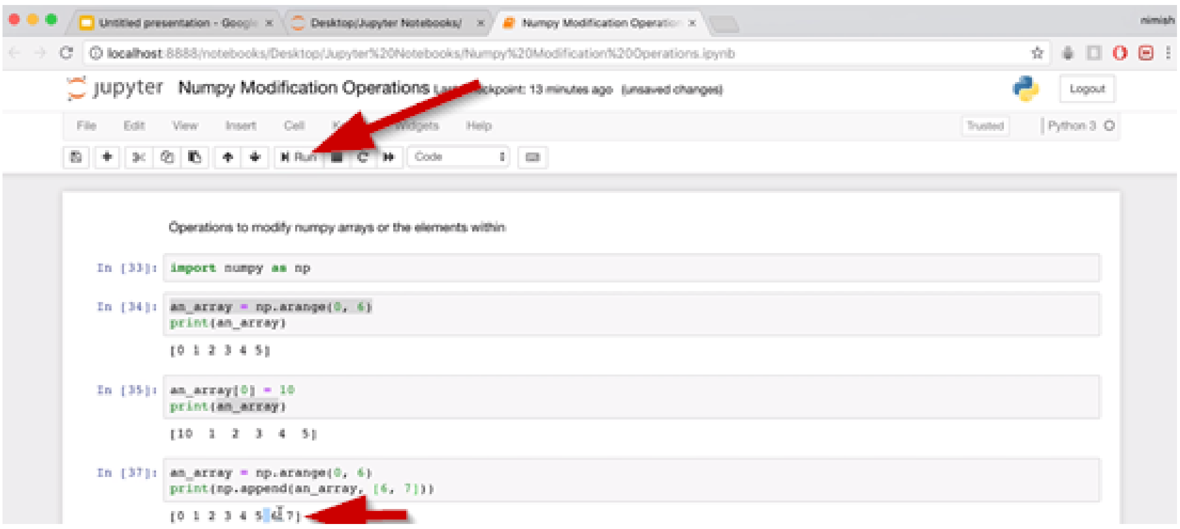

jupyter Numpy Modification Operations Last Checkpoint: 13 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

Operations to modify numpy arrays or the elements within

```
In [33]: import numpy as np

In [34]: an_array = np.arange(0, 6)
         print(an_array)

         [0 1 2 3 4 5]

In [35]: an_array[0] = 10
         print(an_array)

         [10  1  2  3  4  5]

In [36]: an_array = np.arange(0, 6)
         print(np.append(an_array, 6))

         [0 1 2 3 4 5 6]
```

```
an_array = np.arange(0, 6)
print(np.append(an_array, [6, 7]))
```

Run the cell.



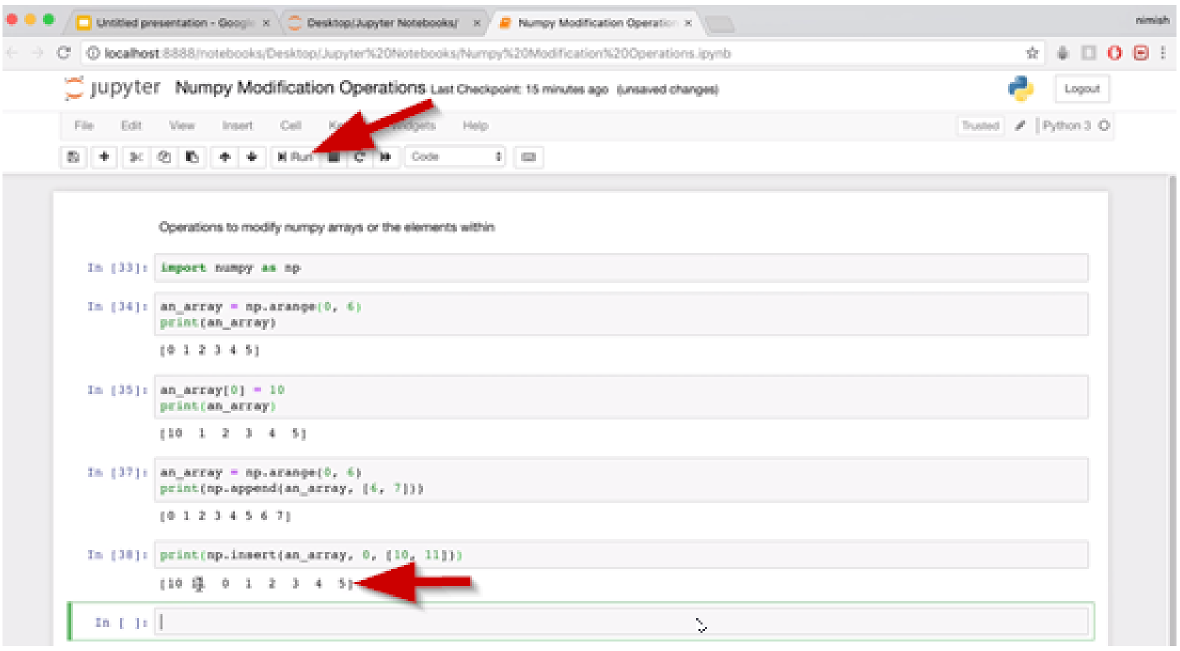
Operations to modify numpy arrays or the elements within

```
In [33]: import numpy as np

In [34]: an_array = np.arange(0, 6)
         print(an_array)

         [0 1 2 3 4 5]

In [35]: an_array[0] = 10
         print(an_array)

         [10  1  2  3  4  5]

In [37]: an_array = np.arange(0, 6)
         print(np.append(an_array, [6, 7]))

         [0 1 2 3 4 5 6 7]
```

Next is an **insert operation**. Insert operation allows us to specify where we want to stick the elements. See the code below and follow along:

```
print(np.insert(an_array, 0, [10, 11]))
```

Run the cell.



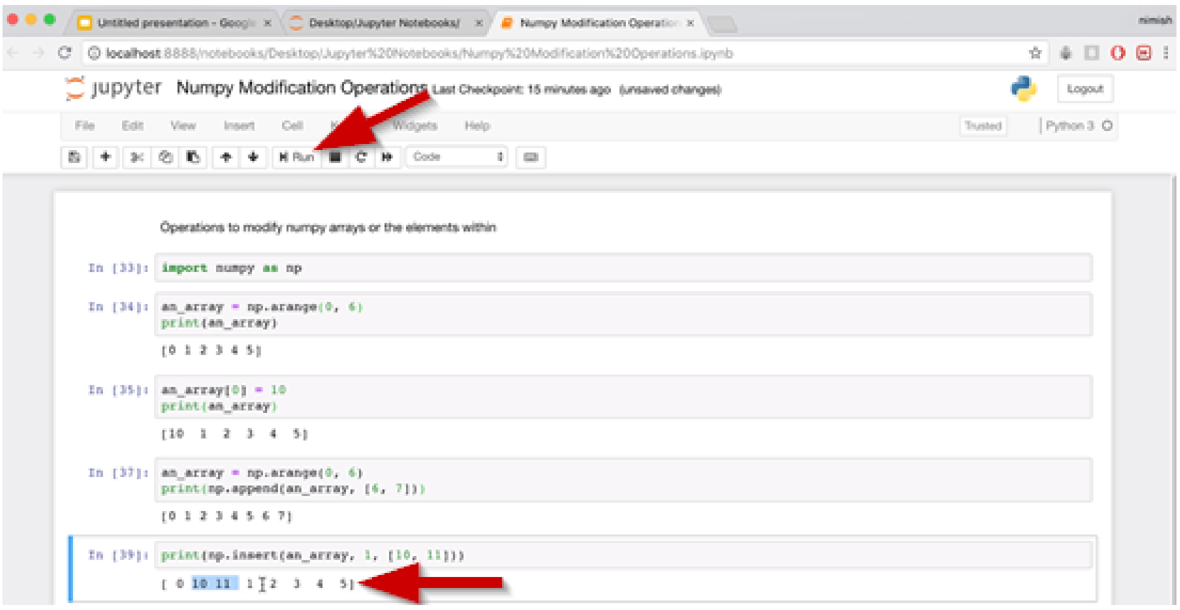
Operations to modify numpy arrays or the elements within

```
In [33]: import numpy as np

In [34]: an_array = np.arange(0, 6)
         print(an_array)

         [0 1 2 3 4 5]

In [35]: an_array[0] = 10
         print(an_array)

         [10  1  2  3  4  5]

In [37]: an_array = np.arange(0, 6)
         print(np.append(an_array, [6, 7]))

         [0 1 2 3 4 5 6 7]

In [38]: print(np.insert(an_array, 0, [10, 11]))

         [10 11  0  1  2  3  4  5]

In [ ]:
```

See the code below:

```
print(np.insert(an_array, 1, [10, 11]))
```

Run the cell.

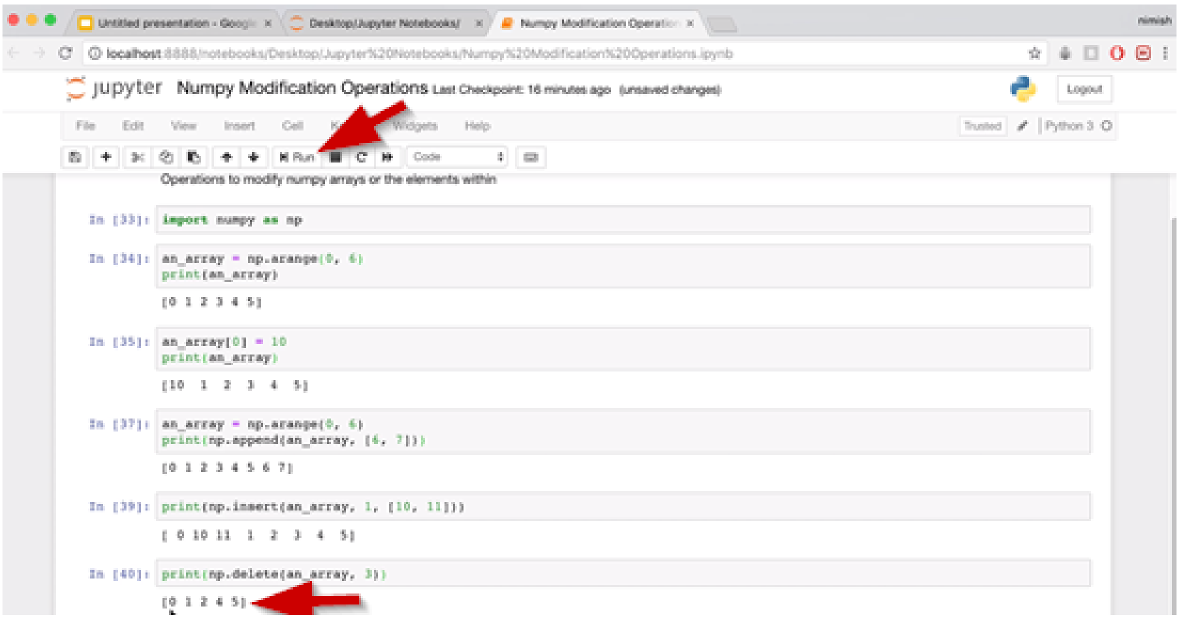

Operations to modify numpy arrays or the elements within

```
In [33]: import numpy as np

In [34]: an_array = np.arange(0, 6)
         print(an_array)

         [0 1 2 3 4 5]

In [35]: an_array[0] = 10
         print(an_array)

         [10  1  2  3  4  5]

In [37]: an_array = np.arange(0, 6)
         print(np.append(an_array, [6, 7]))

         [0 1 2 3 4 5 6 7]

In [39]: print(np.insert(an_array, 1, [10, 11]))

         [ 0 10 11  1  2  3  4  5]
```

Let’s now take a look at **delete**. See the code below and follow along:

```
print(np.delete(an_array, 3))
```

Run the cell.


Operations to modify numpy arrays or the elements within

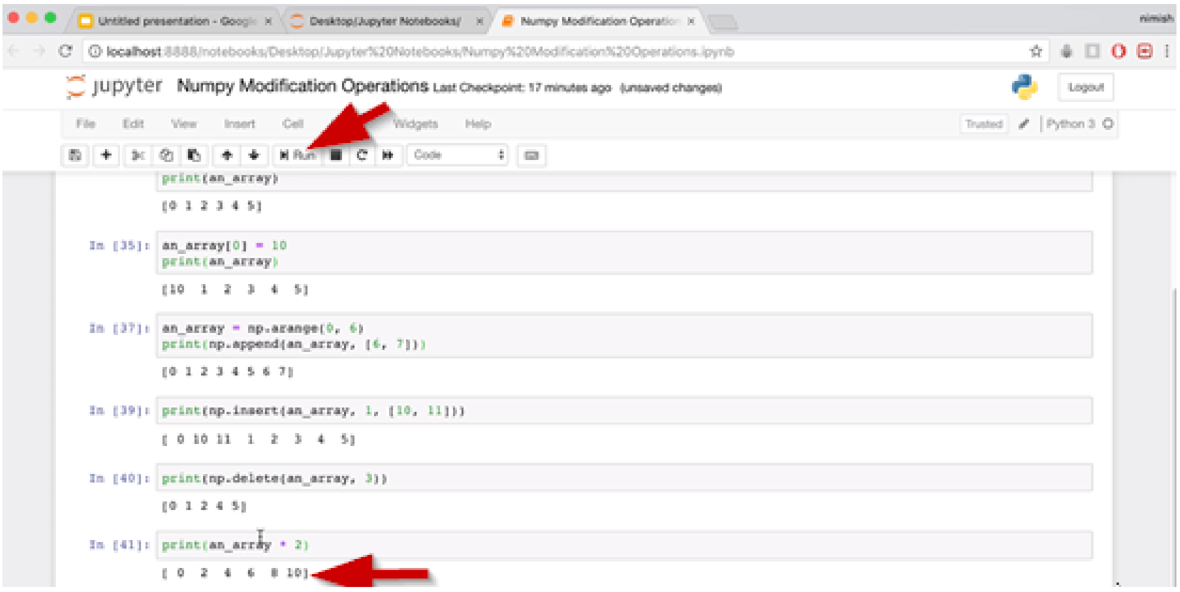
```
In [33]: import numpy as np

In [34]: an_array = np.arange(0, 6)
         print(an_array)

         [0 1 2 3 4 5]

In [35]: an_array[0] = 10
         print(an_array)

         [10  1  2  3  4  5]

In [37]: an_array = np.arange(0, 6)
         print(np.append(an_array, [6, 7]))

         [0 1 2 3 4 5 6 7]

In [39]: print(np.insert(an_array, 1, [10, 11]))

         [ 0 10 11  1  2  3  4  5]

In [40]: print(np.delete(an_array, 3))

         [0 1 2 4 5]
```

Next on the list is our simple scalar or vector operations. This is just going to provide a way to modify multiple elements within an array.

We can take an array and let’s say **we want to multiply everything in that array by two**, so we are essentially doubling all of the elements. See the code below and follow along:

```
print(an_array * 2)
```

Run the cell.



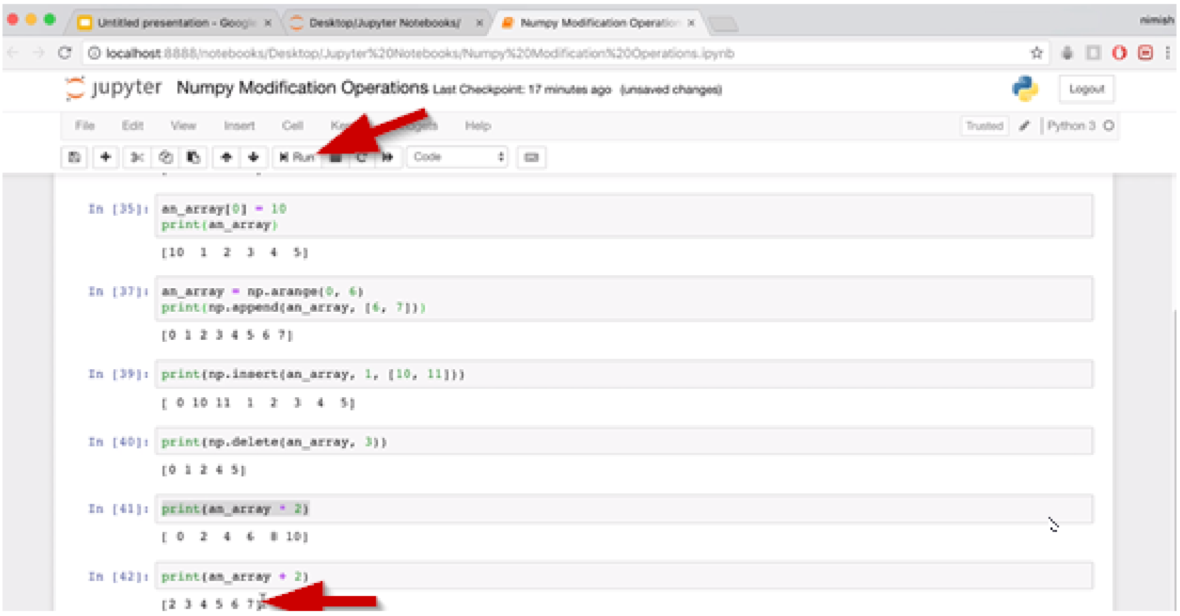
jupyter Numpy Modification Operations Last Checkpoint: 17 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

```
print(an_array)

[0 1 2 3 4 5]

In [35]: an_array[0] = 10
         print(an_array)

[10  1  2  3  4  5]

In [37]: an_array = np.arange(0, 6)
         print(np.append(an_array, [6, 7]))

[0 1 2 3 4 5 6 7]

In [39]: print(np.insert(an_array, 1, [10, 11]))

[ 0 10 11  1  2  3  4  5]

In [40]: print(np.delete(an_array, 3))

[0 1 2 4 5]

In [41]: print(an_array * 2)

[ 0  2  4  6  8 10]
```

So this has taken every element in the array and it simply multiplied it by two. We can do the same thing with addition, subtraction, and division. see the code below and follow along:

```
print(an_array + 2)
```

Run the cell.



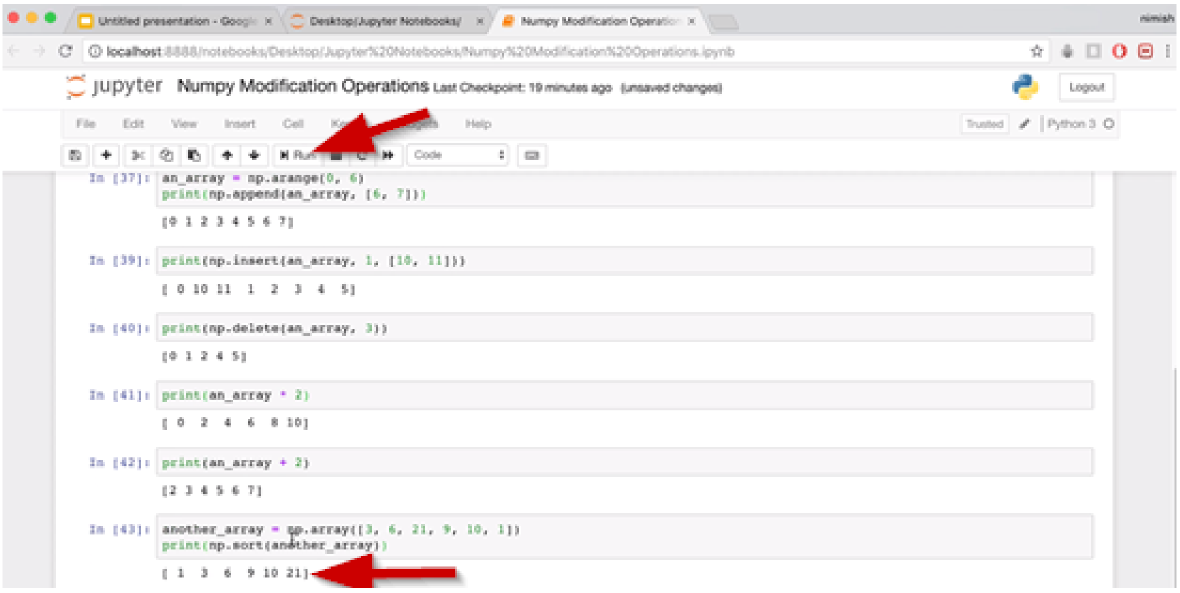
jupyter Numpy Modification Operations Last Checkpoint: 17 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

```
In [35]: an_array[0] = 10
         print(an_array)

[10  1  2  3  4  5]

In [37]: an_array = np.arange(0, 6)
         print(np.append(an_array, [6, 7]))

[0 1 2 3 4 5 6 7]

In [39]: print(np.insert(an_array, 1, [10, 11]))

[ 0 10 11  1  2  3  4  5]

In [40]: print(np.delete(an_array, 3))

[0 1 2 4 5]

In [41]: print(an_array * 2)

[ 0  2  4  6  8 10]

In [42]: print(an_array + 2)

[2 3 4 5 6 7]
```

The default setting for sort is to **sort them in ascending order**. So this will take some jumbled array, or maybe its already in order and it would just order it from lowest to highest value. See the code below and follow along:

```
another_array = np.array([3, 6, 21, 9, 10, 1])
print(np.sort(another_array))
```

Run the cell.

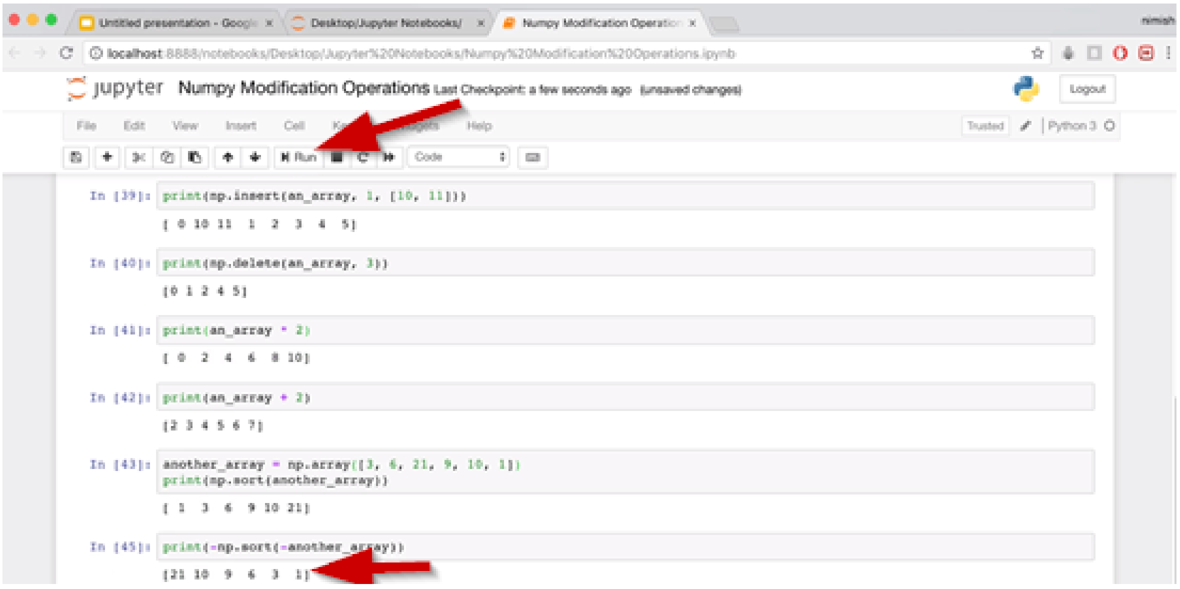

localhost:8888/notebooks/Desktop/Jupyter%20Notebooks/Numpy%20Modification%20Operations.ipynb
jupyter Numpy Modification Operations Last Checkpoint: 19 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [37]: an_array = np.arange(0, 6)
         print(np.append(an_array, [6, 7]))

         [0 1 2 3 4 5 6 7]

In [39]: print(np.insert(an_array, 1, [10, 11]))

         [ 0 10 11  1  2  3  4  5]

In [40]: print(np.delete(an_array, 3))

         [0 1 2 4 5]

In [41]: print(an_array * 2)

         [ 0  2  4  6  8 10]

In [42]: print(an_array + 2)

         [2 3 4 5 6 7]

In [43]: another_array = np.array([3, 6, 21, 9, 10, 1])
         print(np.sort(another_array))

         [ 1  3  6  9 10 21]
```

If we wanted it to be **sorted in descending order.** See the code below and follow along:

```
print(-np.sort(-another_array))
```

Run the cell.


localhost:8888/notebooks/Desktop/Jupyter%20Notebooks/Numpy%20Modification%20Operations.ipynb
jupyter Numpy Modification Operations Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [39]: print(np.insert(an_array, 1, [10, 11]))

         [ 0 10 11  1  2  3  4  5]

In [40]: print(np.delete(an_array, 3))

         [0 1 2 4 5]

In [41]: print(an_array * 2)

         [ 0  2  4  6  8 10]

In [42]: print(an_array + 2)

         [2 3 4 5 6 7]

In [43]: another_array = np.array([3, 6, 21, 9, 10, 1])
         print(np.sort(another_array))

         [ 1  3  6  9 10 21]

In [45]: print(-np.sort(-another_array))

         [21 10  9  6  3  1]
```

Example of Student Grades

- Start with a list of grades
- Obtain the max, min, and average
- Sort in ascending and descending order
- Increase all grades by a % and by adding ones

Create a new Jupyter notebook and call it **"Student Grades Example."**

Create a new header and call it **"Explore what we have learned with the practical example of a list of student grades."**

Import the numpy statement and **run** the cell. This gives us access to the numpy library.

```
import numpy as np
```

Follow along with code from the cells in this below:

```
student_grades = np.array([56, 78, 98, 90, 58, 64, 67, 72, 93, 51])
print(student_grades)
```

Run the cell.

The result printed out:

```
[56 78 98 90 58 64 67 72 93 51]
```

```
class_average = np.average(student_grades)
print(class_average)
```

Run the cell.

The result printed out:

```
72.7
```

```
highest_grade = np.amax(student_grades)
print(highest_grade)

lowest_grade = np.amin(student_grades)
print(lowest_grade)
```

Run the cell.

The result printed out:

```
98
51
```

```
print(np.argmax(student_grades))
print(np.argmin(student_grades))
```

Run the cell.

The result printed out:

```
2
9
```

```
print(np.sort(student_grades))
print(-np.sort(-student_grades))
```

Run the cell.

The result printed out:

```
[51 56 58 64 67 72 78 90 93 98]
[98 93 90 78 72 67 64 58 56 51]
```

```
increased_grades = student_grades * 1.1
print(increased_grades)
```

Run the cell.

The result printed out:

```
[ 61.6  85.8 107.8  99.   63.8  70.4  73.7  79.2 102.3  56.1]
```

```
increased_grades = student_grades + 10
print(increased_grades)
```

Run the cell.

The result printed out:

```
[ 66  88 108 100  68  74  77  82 103  61]
```

```
print(np.add(student_grades, np.ones(10)))
```

Run the cell.

The result printed out:

```
[57. 79. 99. 91. 59. 65. 68. 73. 94. 52.]
```

```
print(np.add(student_grades, np.array([1,2,3,4,5,6,7,8,9,10])))
```

Run the cell.

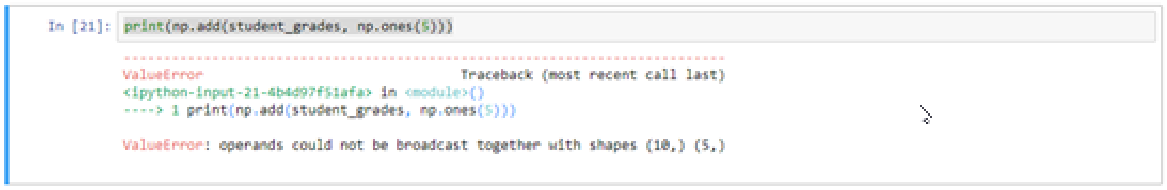
The result printed out:

```
[ 57  80 101  94  63  70  74  80 102  61]
```

```
print(np.add(student_grades, np.ones(5)))
```

Run the cell.

The result printed out. Results in an error:



```
In [21]: print(np.add(student_grades, np.ones(5)))

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-21-4b4d97f51afa> in <module>()
----> 1 print(np.add(student_grades, np.ones(5)))

ValueError: operands could not be broadcast together with shapes (10,) (5,)
```

The error is telling us that it cannot deal with arrays of two different shapes. This function just doesn’t work like that, and this is something to keep in mind when moving forward.

Intro to Matrices in Numpy

- Talk about what matrices are and why they're important.
- Create a custom matrix
- Create a matrix of ones or zeroes

What are matrices?

- A matrix is a multidimensional array
- Essentially an array of arrays
- Used in images, geological data, machine learning, and anything graph related

Image Example

- An example of a 1 might look like this:

[[0,0,1,00]],

[0,0,1,00],

[0,0,1,0,0],

[0,0,1,0,0],

[0,0,1,0,0],]

Here we have five lists of five elements. This is a matrix of five by five.

Each of the elements within the matrix is considered an array here, and each array has five elements.

You can see that there are a bunch of zeroes, that would maybe just represent white pixels, and then the ones down the middle would represent black pixels. This is how we might represent just a straight up one, as an image.

You will often see images represented like this, especially when it comes to machine learning, and image processing, and while software is designed to break an image that we see down into just basically a bunch of ones and zeroes. You should get used to seeing images like this.

Create a new Jupyter Notebook and call it **"Creating Numpy Matrices."**

Create a new header and call it **"Create Matrices in Numpy."**

Import the numpy statement. Make sure you **run** the cell, this gives us access to the library.

```
import numpy as np
```

We will start off by **learning how to create matrices.** They start with **np.matrix.**

See the code below for the cells from the Jupyter Notebook:

```
two_by_two = np.matrix([[1, 2],[3, 4]])
```

```
print(two_by_two)
```

Run the cell.

The results printed:

```
[[1 2]
 [3 4]]
```

```
two_by_three = np.matrix('1 2; 3 4; 5 6')
print(two_by_three)
```

Run the cell.

The results printed out:

```
[[1 2]
 [3 4]
 [5 6]]
```

```
print(np.zeros([3, 2]))
```

Run the cell.

The results printed out:

```
[[0. 0.]
 [0. 0.]
 [0. 0.]]
```

```
print(np.ones([2, 3]))
```

Run the cell.

The results printed out:

```
[[1. 1. 1.]
 [1. 1. 1.]]
```

This is the basics of setting up matrices. We can create custom matrices in a couple if different ways, although they both have the same result.

This is very good when representing image data, if we have an image, but we don’t know what pixel values everything should take on.

Operations to Fetch Elements of Properties

- Get elements based on index
- Get slices
- Get size and shape
- Get max, min, and average

Create a new Jupyter Notebook and call it **“Matrix Fetch Operations.”**

Create a new header and call it **“Functions to get elements or properties of matrices.”**

Import the numpy statement and run the cell. This is so we get access to the numpy library.

```
import numpy as np
```

See the code below and follow along for the cells form the notebook:

```
a_matrix = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(a_matrix)
```

Run the cell.

The results printed:

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
print(a_matrix[0])
```

Run the cell.

The results printed:

```
[[1 2 3]]
```

```
print(a_matrix[1, 0])
```

Run the cell.

The results printed:

```
4
```

```
print(a_matrix[2, 1:3])
```

Run the cell.

The results printed:

```
[[8 9]]
```

```
print(a_matrix.size)
```

Run the cell.

The results printed:

```
9
```

```
print(a_matrix.shape)
```

Run the cell.

The results printed:

```
(3, 3)
```

```
print(np.amax(a_matrix))
print(np.amin(a_matrix))
print(np.average(a_matrix))
```

Run the cell.

The results printed:

```
9
1
5.0
```

```
print(a_matrix.max())
print(a_matrix.min())
print(a_matrix.mean())
```

Run the cell.

The results printed:

```
9
1
5.0
```

Operations to Change Elements or Properties

- Change individual elements or rows
- Sort matrices
- Transpose matrices
- Multiplication and addition
- Matrix multiplication

Create a new Jupter Notebook and call it **"Matrix Modification Operations."**

Create a new header and call it **"Functions to modify elements or properties of a matrix."**

Import the Numpy statement and run the cell. We need this to import the numpy library.

```
import numpy as np
```

See the code below and follow along for the code in the cells from the notebook:

```
a_matrix = np.matrix([[1, 2], [3, 4]])
print(a_matrix)
```

Run the cell.

The results printed:

```
[[1 2]
 [3 4]]
```

```
a_matrix[1, 0] = 10
print(a_matrix)
```

Run the cell.

The results printed:

```
[[ 1  2]
 [10  4]]
```

```
a_matrix[0] = [6, 3]
print(a_matrix)
```

Run the cell.

The results printed:

```
[[ 6  3]
 [10  4]]
```

```
print(-np.sort(-a_matrix))
```

Run the cell.

The results printed:

```
[[ 6  3]
 [10  4]]
```

```
a_matrix = np.matrix([[1, 2], [3, 4]])
print(a_matrix.T)
```

Run the cell.

The results printed:

```
[[1 3]
 [2 4]]
```

```
print(a_matrix.flatten())
```

Run the cell.

The results printed:

```
[[1 2 3 4]]
```

```
print(a_matrix * 2)
```

Run the cell.

The results printed:

```
[[2 4]
 [6 8]]
```

```
print(np.add(a_matrix, a_matrix))
```

Run the cell.

The results printed:

```
[[2 4]
 [6 8]]
```

```
print(np.matmul(a_matrix, a_matrix))
```

Run the cell.

The results printed:

```
[[ 7 10]
 [15 22]]
```

Look at a Quick Example of Matrices in Action

- Create a matrix to represent a number
- Change the number from black to white
- Rotate the number

Images in terms of a computer, like a Hubble computer recognizes an image are often just a series of pixel values. So this could be one representing the darkest pixel, zero representing the lightest pixel, and some number in between representing some kind of a grey, or this could be the numbers from zero to 255, regardless of how we want to represent it, basically, we'll have the highest number being the darkest, and the lowest number being the lightest kind of pixel.

There are also different kinds of RGB codes, if you would like to add color.

We will keep things really simple, and just stick to ones and zeroes. When we are using images and performing a series of transform operations on them, especially when it comes to stuff like machine learning with regards to image recognition or image classification, we generally want to make sure our images are squares. That means we have the same number of rows and columns. This is especially useful if we're performing any kind of transpose operation, that way we retain the integrity of the image somewhat.

Create a new Jupyter Notebook and call it **"Matrix Operations by Example."**

Create a new header and call it **"Go over a quick example of matrices in action."**

Import the numpy statement, and run the cell. We need this to get access to the numpy library.

```
import numpy as np
```

See the code below and follow along for the cells from the notebook:

```
image_matrix = np.matrix([[0, 0, 1, 0, 0], [0, 0, 1, 0, 0], [0, 0, 1, 0, 0], [0, 0, 1
, 0, 0], [0, 0, 1, 0, 0]])
print(image_matrix)
```

Run the cell.

The printed results:

```
[[0 0 1 0 0]
 [0 0 1 0 0]
 [0 0 1 0 0]
 [0 0 1 0 0]
 [0 0 1 0 0]]
```

```
rows = image_matrix.shape[0]
columns = image_matrix.shape[1]
print(rows)
print(columns)
```

Run the cell.

The printed results:

```
5
5
```

```
for row in range(rows):
    for column in range(columns):
        if image_matrix[row, column] == 0:
            image_matrix[row, column] = 1
        else:
            image_matrix[row, column] = 0
print(image_matrix)
```

Run the cell.

The printed results:

```
[[1 1 0 1 1]
 [1 1 0 1 1]
 [1 1 0 1 1]
 [1 1 0 1 1]
 [1 1 0 1 1]]
```

```
print(image_matrix.T)
```

Run the cell.

The printed results:

```
[[1 1 1 1 1]
 [1 1 1 1 1]
 [0 0 0 0 0]
 [1 1 1 1 1]
 [1 1 1 1 1]]
```

```
print(image_matrix.flatten())
```

Run the cell.

The printed results:

```
[[1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1]]
```