

DATA SCIENCE WITH PYTHON



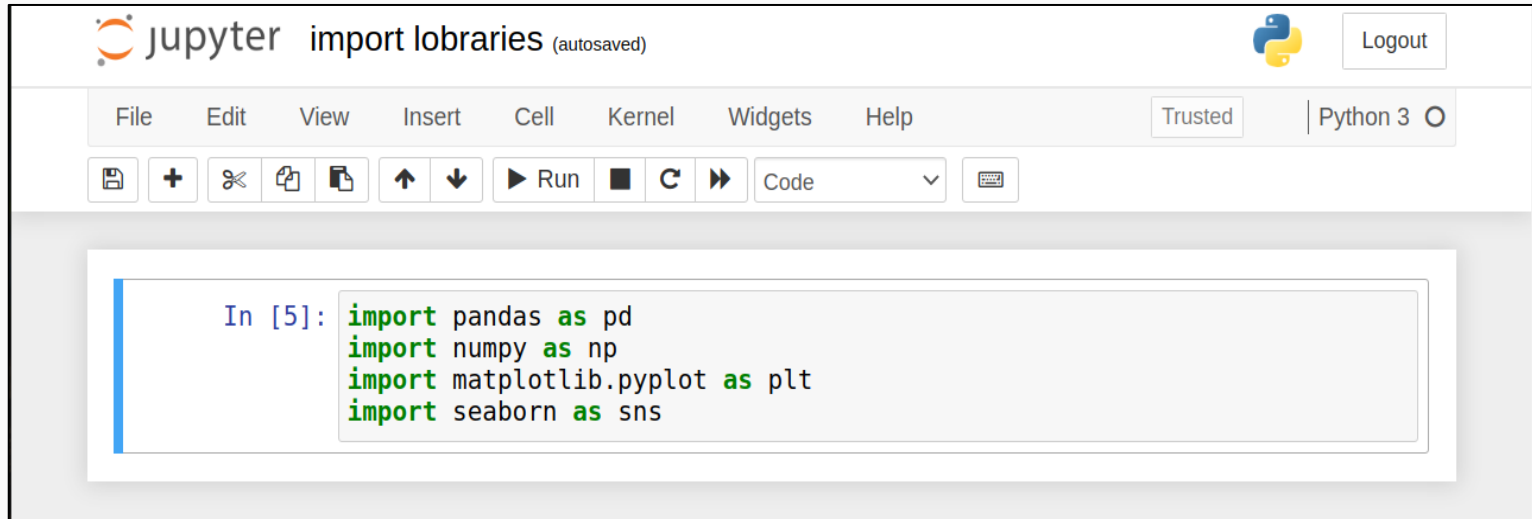
Installing Libraries

If you installed Anaconda, you do not need to download any libraries as it automatically installs all the popular data science libraries such as Pandas, Numpy, Matplotlib, Seaborn, etc.



Importing Libraries

- Open your Jupyter Notebook.
- To import a library we use the keyword **import** followed by library name.
- We can use the **as** keyword to use abbreviations for our library names.
- The common abbreviations used are
 - pd for pandas
 - np for numpy
 - plt for matplotlib.pyplot
 - sns for seaborn



The screenshot shows a Jupyter Notebook window titled "import libraries (autosaved)". The interface includes a top bar with the Jupyter logo, a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a "Trusted" status indicator, and a "Python 3" version selector. Below the menu bar is a toolbar with icons for saving, adding, deleting, and running code. The main area contains a code cell with the following Python code:

```
In [5]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```



Pandas Library for Data Science

- Pandas is a Python library for data manipulation and analysis.
- It allows exploring, cleaning, and processing tabular data.
- It provides two ways for storing data;
 - Series, which is one dimensional data structure
 - Data Frame, which is two dimensional data structure

DataFrame

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
2	All-Bran	70	4	25	59.425505
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843
5	Apple Cinnamon Cheerios	110	2	25	29.509541
6	Apple Jacks	110	2	25	33.174094
7	Basic 4	130	3	25	37.038562
8	Bran Chex	90	2	25	49.120253
9	Bran Flakes	90	3	25	53.313813

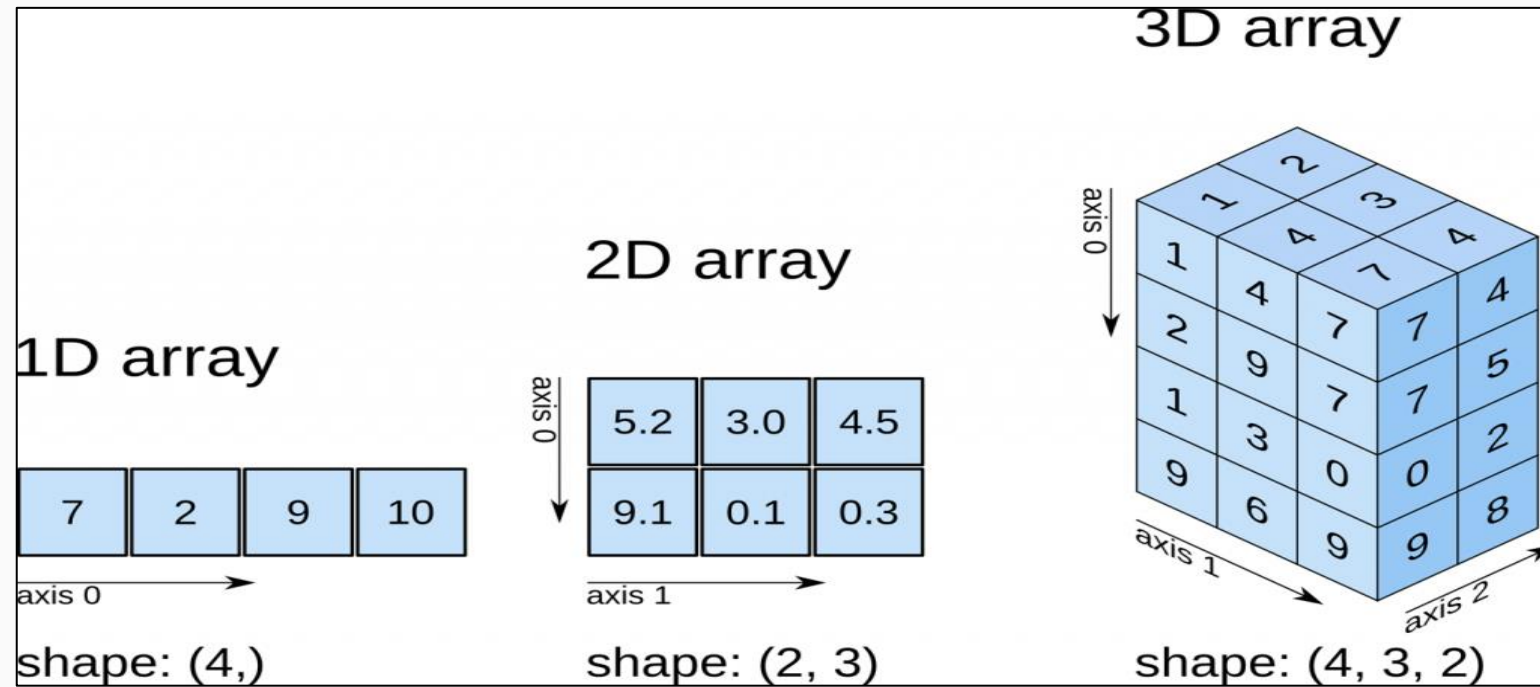
```
0    70
1   120
2    70
3    50
4   110
5   110
6   110
7   130
8    90
9    90
Name: calories, dtype: int64
```

Series



NumPy Library for Data Science

- NumPy stands for Numerical Python.
- It provides a data structure called NumPy array, which is a grid of values.
- It also provides a collection of high-level mathematical functions which can be performed on multi-dimensional NumPy arrays.





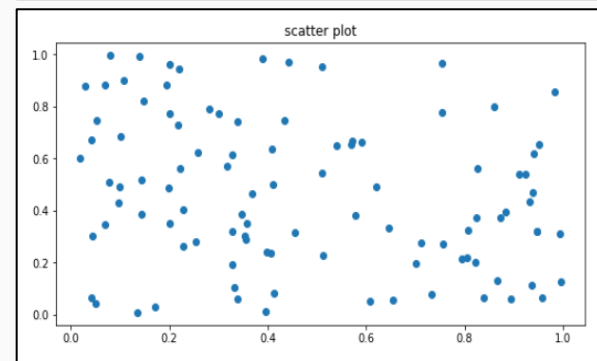
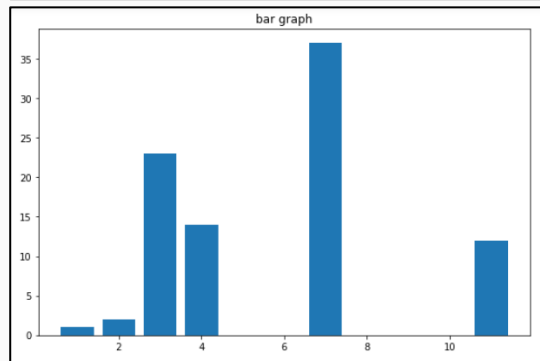
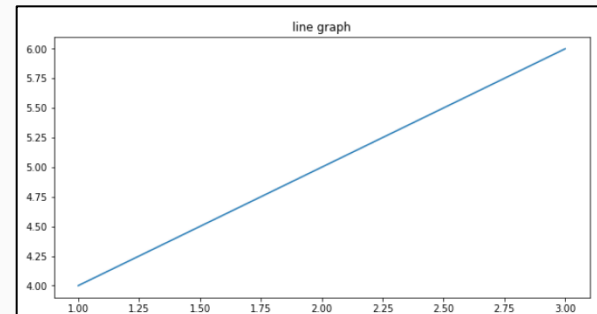
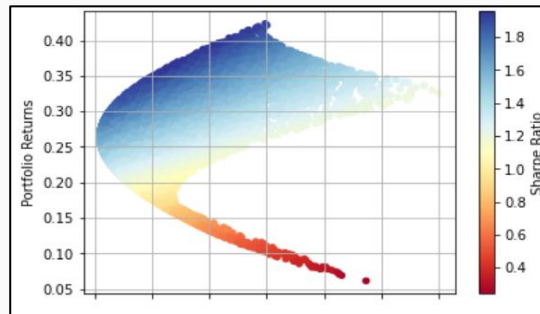
Pandas vs NumPy

NumPy	Pandas
NumPy and Pandas are both Python libraries for Data Science	
It is used for scientific computing	It is used for data manipulation such as storing, exploring, cleaning, and processing the data
It provides NumPy arrays which can be multidimensional	It provides two data structures; <ul style="list-style-type: none">• Series (one dimensional)• Data frames (two dimensional)
We use Pandas for data manipulation and NumPy for Mathematical Computations	
Since Pandas Series and Data Frames can be thought of as one and two dimensional NumPy arrays respectively, we can apply NumPy mathematical functions on them as well	



Matplotlib Library for Data Science

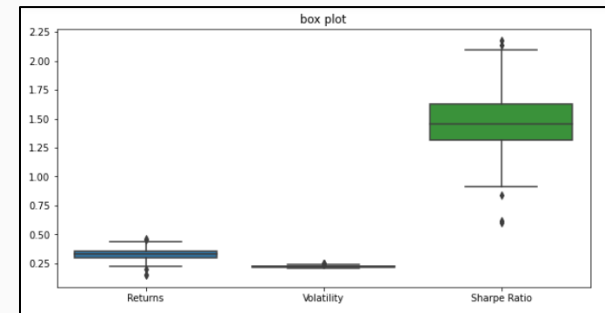
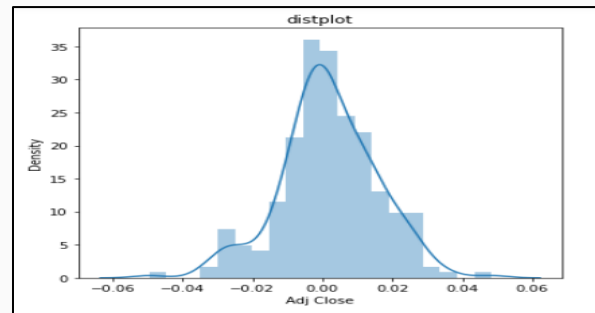
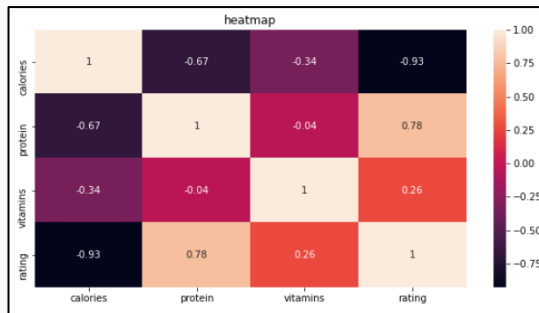
- Matplotlib is a visualization Python library, i.e., it is used for plotting graphs.
- The pyplot module inside of the Matplotlib provides the interface to underlying plotting functionality of the Matplotlib.
- We can create a number of different types of graphs using Matplotlib such as line graphs, bar graphs, histograms, scatter plots, area plots, pie plots, and so on.





Seaborn Library for Data Science

- Seaborn is another visualization Python library built on top of Matplotlib.
- It extends the functionality of Matplotlib and allows creating a variety of different graphs with fewer syntax.





NumPy Arrays

What are NumPy Arrays

- NumPy array is a multidimensional data structure designed to handle large data sets easily.
- A NumPy array is called **ndarray**.
- We can find the number of dimensions of a NumPy array using **.ndim**.

NumPy Arrays vs Python Lists

- NumPy arrays provide more built-in functionality as compared to Python lists.
- NumPy arrays make working with huge multi-dimensional data sets much easier with fewer syntax.
- NumPy arrays are also more efficient than Python lists in terms of memory consumption and speed.



Creating NumPy Arrays (1/3)

1-D NumPy Arrays

- A 1-D NumPy array is where each element of the outermost array is a 0-D array (scalar).
- We can create a NumPy array using the `array()` function in the NumPy library.
- We can create a NumPy array using either Python lists or tuples.
- To create a 1-D NumPy Array, we provide a 1-D Python list or tuple to the `array()` function.

The image shows a Jupyter Notebook interface with the title "1-D Numpy Array". The notebook contains three code cells. The first cell imports NumPy as np. The second cell creates a 1-D array named oneDArray from a list of integers and prints it. The third cell prints the number of dimensions (ndim) of the array, which is 1.

```
In [66]: import numpy as np

In [72]: oneDArray = np.array([1, 2, 3, 4, 5])
         print(oneDArray)
         [1 2 3 4 5]

In [73]: print(oneDArray.ndim)
         1

In [ ]:
```



Creating NumPy Arrays (2/3)

2-D NumPy Arrays

- A 2-D NumPy array is where each element in the outermost array is a 1-D array.
- To create a 2-D NumPy Array, we provide a 2-D Python list or tuple to the array() function.

The image shows a Jupyter Notebook interface with the title "2-D Numpy Array" and a status bar indicating "Last Checkpoint: a few seconds ago (autosaved)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for saving, adding cells, undo, redo, and running code. The code is written in three cells:

```
In [66]: import numpy as np
```

```
In [74]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
          print(twoDArray)
          [[1 2 3]
           [4 5 6]]
```

```
In [75]: print(twoDArray.ndim)
          2
```



Creating NumPy Arrays (3/3)

3-D NumPy Arrays

- A 3-D NumPy array is where each element of the outermost array is a 2-D array.
- To create a 3-D NumPy Array, we provide a 3-D Python list or tuple to the `array()` function.

```
jupyter 3-D Numpy Array Last Checkpoint: 3 minutes ago (unsaved changes) Python 3
```

```
In [66]: import numpy as np
```

```
In [84]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])
         print(threeDArray)
```

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

  [[ 4  5  6]
  [-4 -5 -6]]]
```

```
In [85]: print(threeDArray.ndim)
```

```
3
```



Quiz Time

1. How many dimensions are in the array `[[[1, 2, 3, 4]]]`
- a) 1
 - b) 2
 - c) 3



Quiz Time

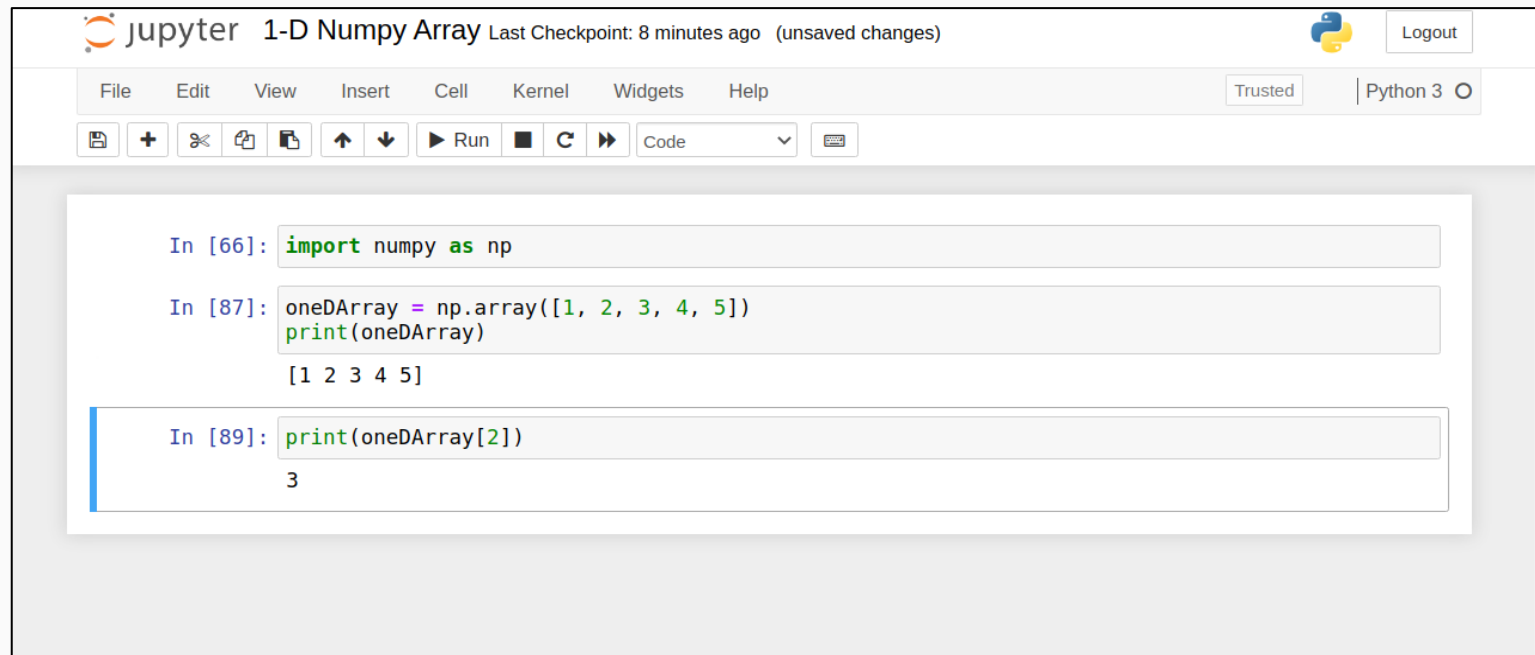
1. How many dimensions are in the array `[[[1, 2, 3, 4]]]`
- a) 1
 - b) 2
 - c) 3



Indexing NumPy Arrays (1/8)

1-D NumPy Arrays

- Indexing a 1-D NumPy array is the same as indexing a 1-D Python list.
- Provide the index of the element inside the square brackets to get that element.



The image shows a Jupyter Notebook interface with the title "1-D Numpy Array". The notebook contains three code cells. The first cell imports NumPy as np. The second cell creates a 1-D NumPy array named oneDArray with values [1, 2, 3, 4, 5] and prints it, showing the output [1 2 3 4 5]. The third cell prints the element at index 2 of oneDArray, showing the output 3.

```
In [66]: import numpy as np

In [87]: oneDArray = np.array([1, 2, 3, 4, 5])
         print(oneDArray)
         [1 2 3 4 5]

In [89]: print(oneDArray[2])
         3
```



Indexing NumPy Arrays (2/8)

2-D NumPy Arrays

- To index a 2-D NumPy array, we provide 2 values inside the square brackets ([]).
 - First value is the index of the inner array
 - Second value is the index of the element inside the inner array
- In the following example, we get the first element of the second array.

The image shows a Jupyter Notebook interface with the title "2-D Numpy Array (unsaved changes)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for saving, adding cells, undo, redo, and running code, and a code editor. The code editor contains the following code:

```
In [66]: import numpy as np

In [90]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
         print(twoDArray)
```

The output of the code is a 2x3 NumPy array:

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

The element '4' in the second row, first column is highlighted with a red box. Below the output, there is an empty input field for the next code cell.



Indexing NumPy Arrays (3/8)

2-D NumPy Arrays

- The first dimension contains 2 arrays.
- If we say `twoDArray[1]`, we get the second array.

```
jupyter 2-D Numpy Array (unsaved changes) Python 3
```

```
In [66]: import numpy as np
```

```
In [90]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
          print(twoDArray)
          [[1 2 3]
           [4 5 6]]
```

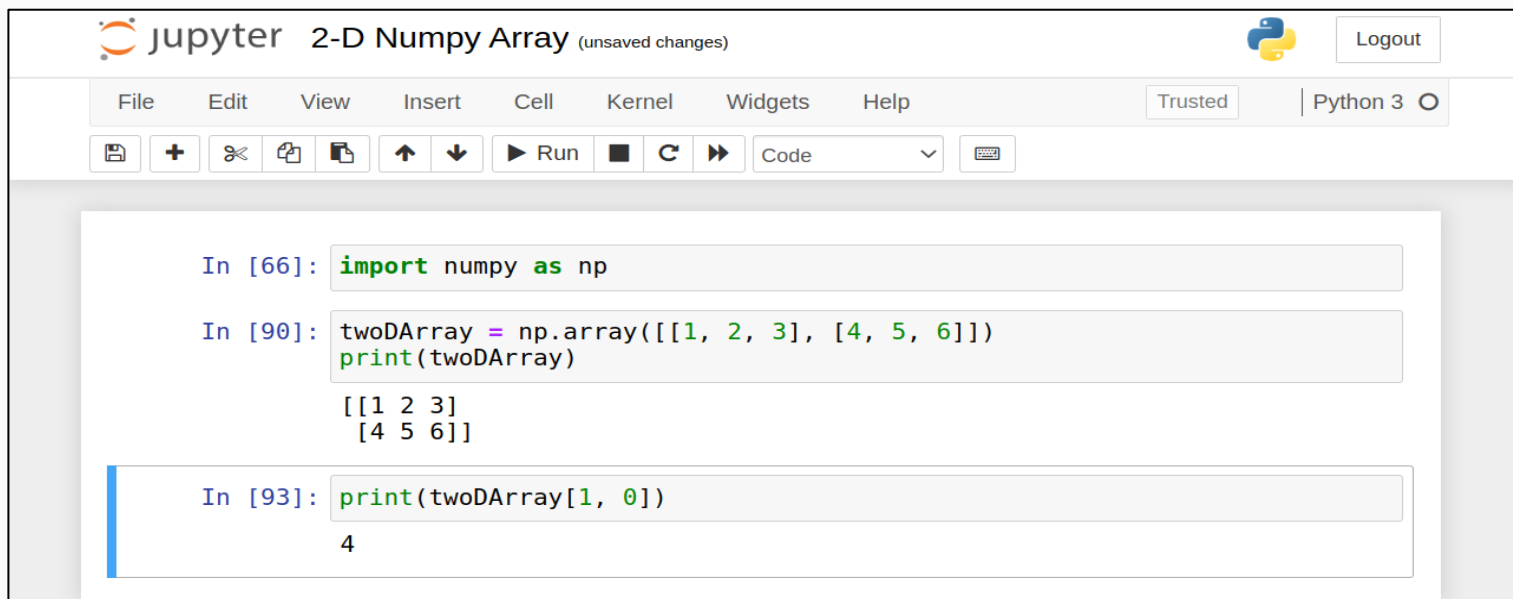
```
In [92]: print(twoDArray[1])
          [4 5 6]
```



Indexing NumPy Arrays (4/8)

2-D NumPy Arrays

- The second dimension contains 3 elements.
- If we say `twoDArray[1, 0]`, we get the first element of the second array.



The image shows a Jupyter Notebook interface with the title "2-D Numpy Array (unsaved changes)". The interface includes a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, Help. Below the menu bar is a toolbar with icons for saving, adding, deleting, copying, pasting, undo, redo, and running code. The notebook contains three code cells. The first cell imports numpy as np. The second cell creates a 2D array named twoDArray with values [[1, 2, 3], [4, 5, 6]] and prints it. The output of the second cell is a 2x3 array. The third cell prints the element at index [1, 0] of twoDArray, which is 4.

```
In [66]: import numpy as np

In [90]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
         print(twoDArray)

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

In [93]: print(twoDArray[1, 0])

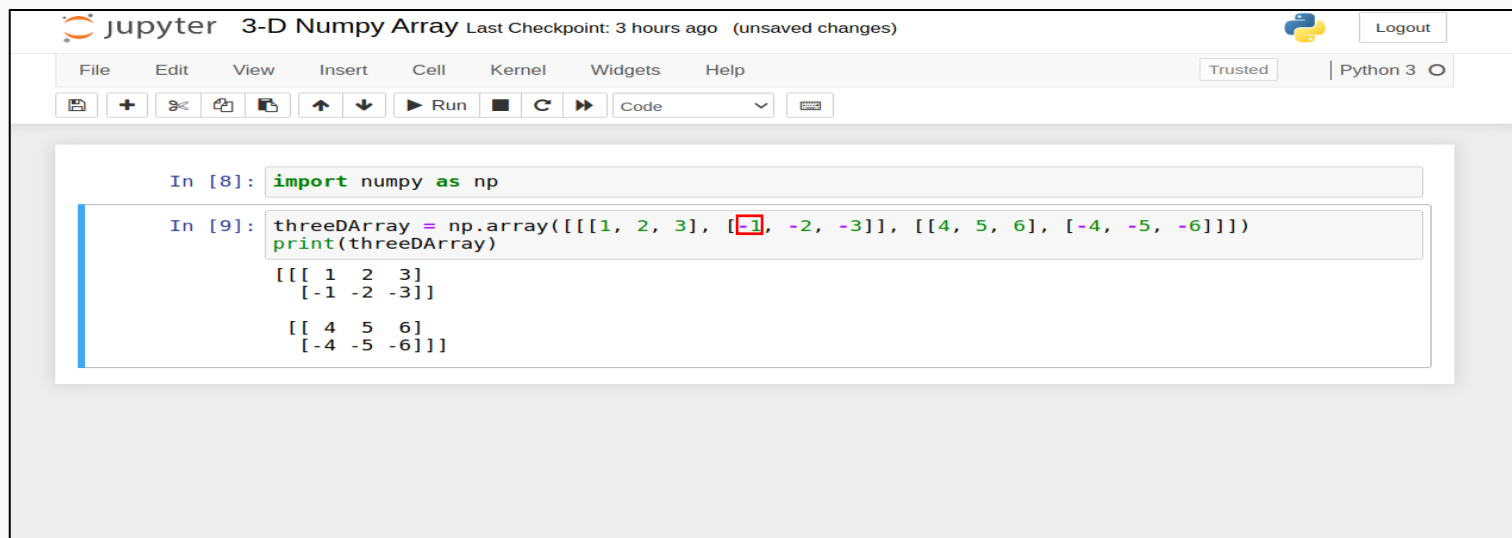
         4
```



Indexing NumPy Arrays (5/8)

3-D NumPy Arrays

- To index a 3-D NumPy array, we provide 3 values inside the square brackets ([]).
 - First value is the index of the inner 2-D array in the first dimension.
 - Second value is the index of the inner 1-D array in the second dimension.
 - Third value is the index of the element in the third dimension.
- In the following example, we get the first element of the second array of the first array.



The image shows a Jupyter Notebook interface with the title "3-D Numpy Array". The notebook contains two code cells. The first cell (In [8]) imports numpy as np. The second cell (In [9]) creates a 3-D array named 'threeDArray' with the following structure: a list of three 2-D arrays. The first 2-D array is [[1, 2, 3], [-1, -2, -3]], the second is [[4, 5, 6]], and the third is [[-4, -5, -6]]. The value -1 in the first 2-D array is highlighted with a red box. The output of the print statement shows the array structure.

```
In [8]: import numpy as np

In [9]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6]], [[-4, -5, -6]]])
print(threeDArray)

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



Indexing NumPy Arrays (6/8)

3-D NumPy Arrays

- The first dimension contains 2 arrays.
- If we say `threeDArray[0]`, we get the first array.

```
jupyter 3-D Numpy Array Last Checkpoint: 3 hours ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [8]: import numpy as np
In [10]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])
          print(threeDArray[0])
          [[ 1  2  3]
           [-1 -2 -3]]
```



Indexing NumPy Arrays (7/8)

3-D NumPy Arrays

- The second dimension again contains 2 arrays.
- If we say `threeDArray[0, 1]`, we get the second array of the first array.

```
jupyter 3-D Numpy Array Last Checkpoint: 3 hours ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [8]: import numpy as np
In [11]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])
          print(threeDArray[0, 1])
          [-1 -2 -3]
```



Indexing NumPy Arrays (8/8)

3-D NumPy Arrays

- The third dimension contains 3 values.
- If we say `threeDArray[0, 1, 0]`, we get the first element of the second array of the first array.

```
jupyter 3-D Numpy Array Last Checkpoint: 3 hours ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [8]: import numpy as np
In [12]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])
         print(threeDArray[0, 1, 0])
         -1
```



Quiz Time

1. Consider a numpy array `x = [[[1, 2, 3, 4]]]`. What will `x[0, 0, 0]` return?
 - a) 1
 - b) 3
 - c) 4



Quiz Time

1. Consider a numpy array `x = [[[1, 2, 3, 4]]]`. What will `x[0, 0, 0]` return?

- a) 1
- b) 3
- c) 4



Array Shape

- NumPy arrays have a shape attribute which returns a tuple.
 - First value of the tuple gives the number of dimensions in the array
 - Second value of the tuple gives the number of elements in each dimension

A screenshot of a Jupyter Notebook interface. The title bar shows 'jupyter .shape' and 'Last Checkpoint: 3 hours ago (autosaved)'. The top menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. Below the menu is a toolbar with icons for saving, adding cells, undo, redo, and running code. The notebook contains two code cells. The first cell, labeled 'In [8]:', contains the code 'import numpy as np'. The second cell, labeled 'In [16]:', contains the code 'twoDArray = np.array([[1, 2, 3], [4, 5, 6]])' followed by 'print(twoDArray.shape)'. The output of the second cell is '(2, 3)'.

```
In [8]: import numpy as np

In [16]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
          print(twoDArray.shape)
          (2, 3)
```



Iterating Over NumPy Arrays (1/8)

1-D NumPy Arrays

- We can use a for loop to iterate over a 1-D array just as we do in a 1-D Python list.

The image shows a Jupyter Notebook interface with the title "iterating over arrays" and a status bar indicating "Last Checkpoint: 3 hours ago (unsaved changes)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and other functions. The code is written in two cells:

```
In [8]: import numpy as np
```

```
In [19]: oneDArray = [1, 2, 3, 4, 5]
         for i in oneDArray:
             print(i)
```

The output of the second cell shows the numbers 1, 2, 3, 4, and 5, each on a new line, indicating that the loop successfully iterated over each element of the 1-D array.



Iterating Over NumPy Arrays (2/8)

2-D NumPy Arrays

- We can use a nested for loop to iterate over a 2-D array.
 - The outer for loop iterates over the outer array.
 - The inner for loop iterates over the inner array.



Iterating Over NumPy Arrays (3/8)

2-D NumPy Arrays

- We use a for loop to iterate over the outer array.
- We print all the inner arrays.

The image shows a Jupyter Notebook window titled "iterating over arrays". The interface includes a top bar with the Jupyter logo, the title, and a "Logout" button. Below the top bar is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A toolbar with various icons for file operations and execution is located below the menu bar. The main area contains two code cells. The first cell, labeled "In [8]:", contains the code `import numpy as np`. The second cell, labeled "In [17]:", contains the code `twoDArray = np.array([[1, 2, 3], [4, 5, 6]])`, `for i in twoDArray:`, and `print(i)`. The output of the second cell is displayed below the code: `[1 2 3]` and `[4 5 6]`.

```
jupyter iterating over arrays Last Checkpoint: 3 hours ago (unsaved changes) Logout
```

```
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
```

```
In [8]: import numpy as np
```

```
In [17]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
         for i in twoDArray:
             print(i)
```

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



Iterating Over NumPy Arrays (4/8)

2-D NumPy Arrays

- We use another for loop nested inside the outer for loop to iterate over the inner array.
- We print all the elements in each of the inner arrays.

The image shows a Jupyter Notebook window titled "iterating over arrays". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for saving, adding cells, undo, redo, and running code, and a status bar indicating "Trusted" and "Python 3". The notebook contains two code cells. The first cell, labeled "In [8]:", imports NumPy as np. The second cell, labeled "In [18]:", creates a 2D array named twoDArray with the values [[1, 2, 3], [4, 5, 6]] and uses nested for loops to iterate over each element, printing the value of j. The output of the second cell shows the numbers 1 through 6, each on a new line.

```
In [8]: import numpy as np

In [18]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
        for i in twoDArray:
            for j in i:
                print(j)
```

1
2
3
4
5
6



Iterating Over NumPy Arrays (5/8)

3-D NumPy Arrays

- We can use 3 nested for loops to iterate over a 3-D array.
 - The outermost for loop iterates over the arrays in the first dimension.
 - The middle for loop iterates over the arrays in the second dimension.
 - The innermost for loop iterates over all the elements in the third dimension.



Iterating Over NumPy Arrays (6/8)

3-D NumPy Arrays

- Outermost array contains 2 arrays both of which are 2-D.
- We use a for loop to print these 2-D arrays.

```
jupyter iterating over arrays Last Checkpoint: 3 hours ago (autosaved) Logout
File Edit View Insert Cell Kernel Widgets Help Notebook saved Trusted Python 3
In [8]: import numpy as np
In [20]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])
         for i in threeDArray:
             print(i)
[[ 1  2  3]
 [-1 -2 -3]]
[[ 4  5  6]
 [-4 -5 -6]]
```



Iterating Over NumPy Arrays (7/8)

3-D NumPy Arrays

- Each of the 2-D array contains 2 arrays in the second dimension, each of which is 1-D.
- We use another for loop nested within the first for loop to print these 1-D arrays.

```
jupyter iterating over arrays Last Checkpoint: 3 hours ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [8]: import numpy as np
In [21]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])
        for i in threeDArray:
            for j in i:
                print(j)
[1 2 3]
[-1 -2 -3]
[4 5 6]
[-4 -5 -6]
```




Iterating Over NumPy Arrays (8/8)

3-D NumPy Arrays

- Each of the 1-D array contains 3 elements in the third dimension, each of which is 1-D.
- We use another for loop nested within the first 2 for loops to print these elements.

```
jupyter iterating over arrays Last Checkpoint: 3 hours ago (unsaved changes) Python 3
```

```
In [8]: import numpy as np
```

```
In [22]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])
         for i in threeDArray:
             for j in i:
                 for k in j:
                     print(k)
```

```
1
2
3
-1
-2
-3
4
5
6
-4
-5
-6
```



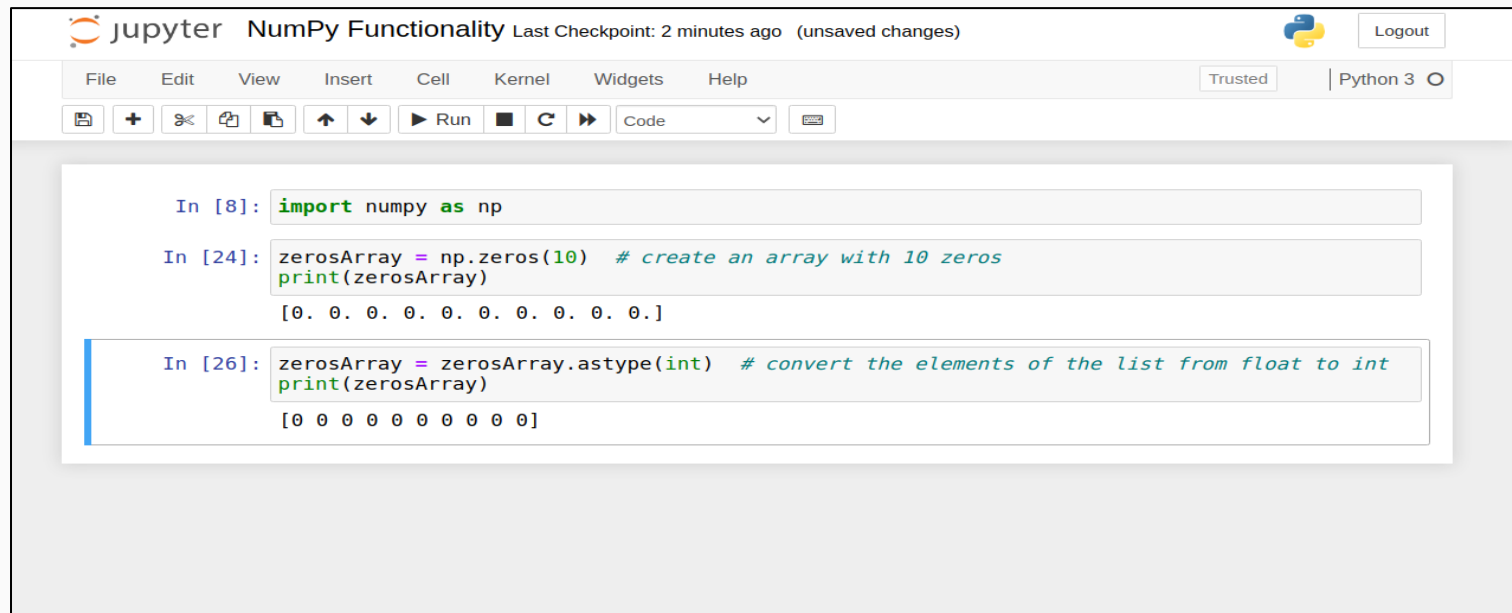
Mathematics for Data Science

- NumPy provides us with a huge collection of high-level functions for multi-dimensional arrays.
- Let's have a look at some of the functionality provided by NumPy.



.zeros()

- To create a NumPy array prefilled with zeros, we can use the `.zeros()` built-in NumPy function.
- `.zeros()` gives us a list prefilled with float zeros. To convert this list into integer list, we use the `.astype()` function.



```
In [8]: import numpy as np

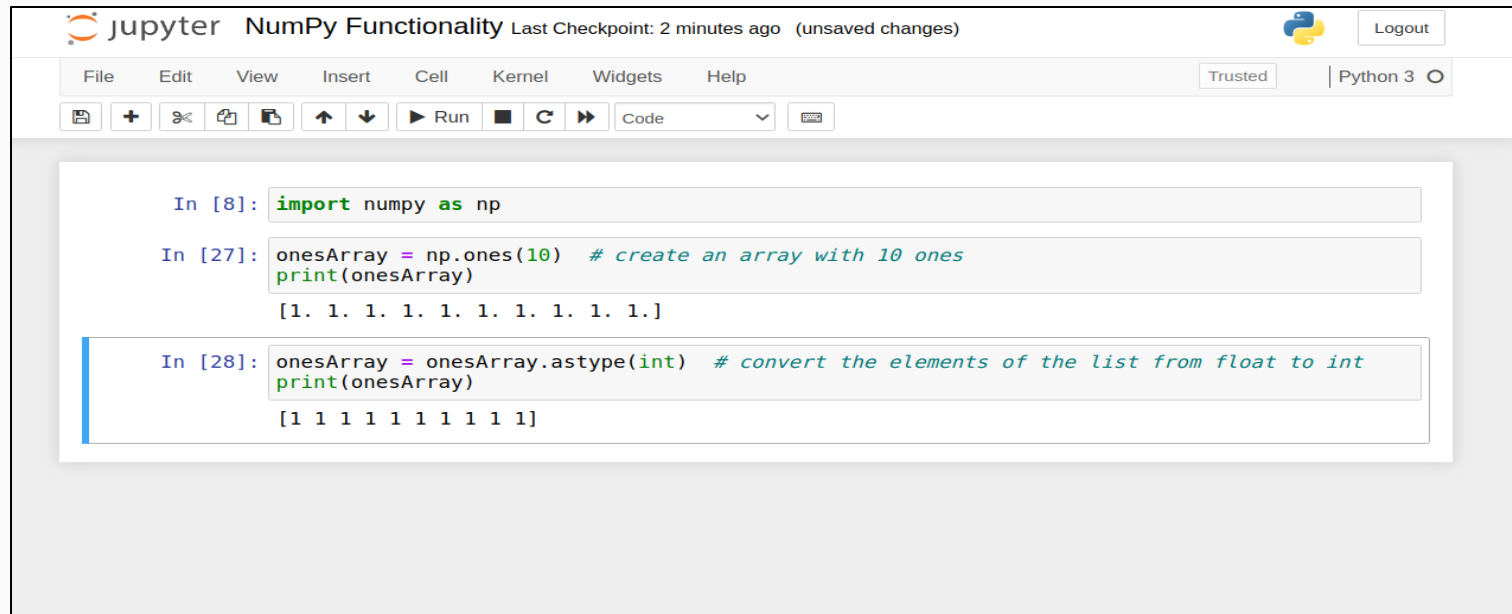
In [24]: zerosArray = np.zeros(10) # create an array with 10 zeros
print(zerosArray)
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

In [26]: zerosArray = zerosArray.astype(int) # convert the elements of the list from float to int
print(zerosArray)
[0 0 0 0 0 0 0 0 0 0]
```



.ones()

- To create a NumPy array prefilled with ones, we can use the `.ones()` built-in NumPy function.
- `.ones()` gives us a list prefilled with float ones. To convert this list into integer list, we use the `.astype()` function.



```
In [8]: import numpy as np

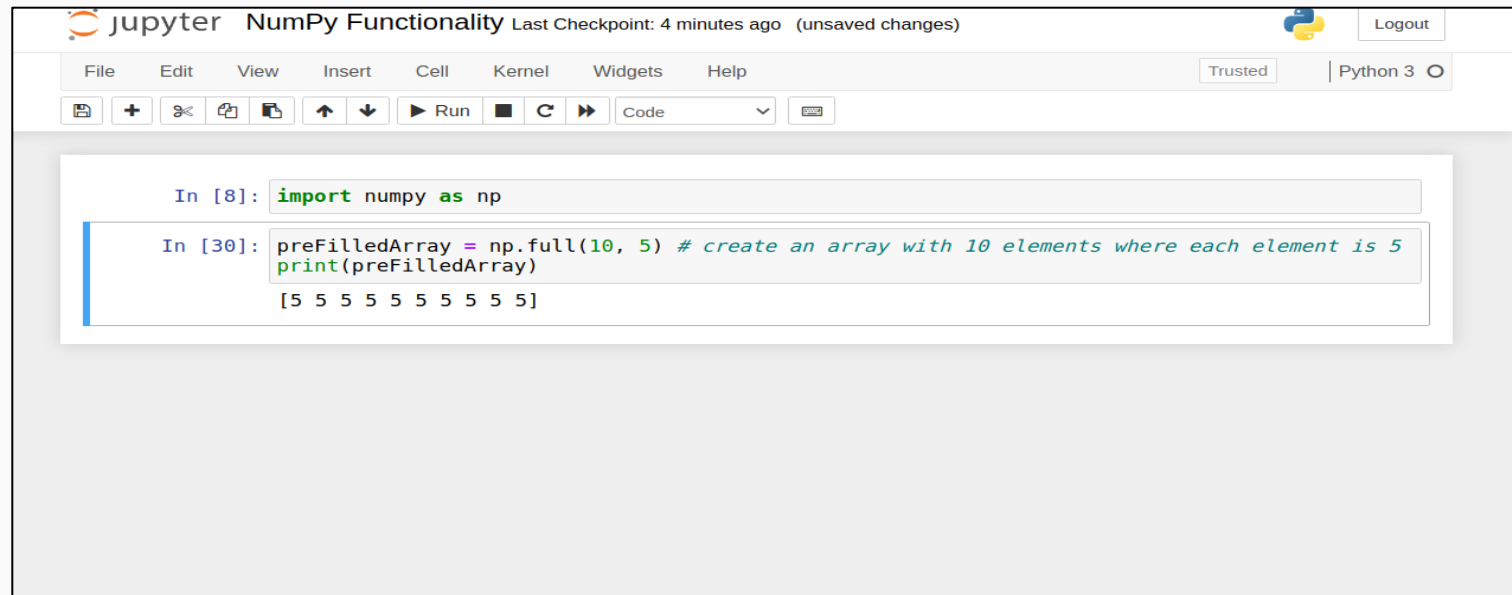
In [27]: onesArray = np.ones(10) # create an array with 10 ones
print(onesArray)
[1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]

In [28]: onesArray = onesArray.astype(int) # convert the elements of the list from float to int
print(onesArray)
[1 1 1 1 1 1 1 1 1 1]
```



.full()

- To create a NumPy array prefilled with some specific number, we can use the `.full()` built-in NumPy function.
 - First argument in the `.full()` function is the size of the array
 - Second argument in the `.full()` function is the value that we want our list to be pre-filled with



The image shows a Jupyter Notebook interface titled "NumPy Functionality". The top bar includes the Jupyter logo, the title, and a "Logout" button. Below the title bar is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A toolbar with various icons for file operations and execution is located below the menu bar. The main area displays two code cells. The first cell, labeled "In [8]:", contains the code `import numpy as np`. The second cell, labeled "In [30]:", contains the code `preFilledArray = np.full(10, 5) # create an array with 10 elements where each element is 5` followed by `print(preFilledArray)`. The output of the second cell is displayed as `[5 5 5 5 5 5 5 5 5 5]`.

```
In [8]: import numpy as np

In [30]: preFilledArray = np.full(10, 5) # create an array with 10 elements where each element is 5
         print(preFilledArray)
         [5 5 5 5 5 5 5 5 5 5]
```



Quiz Time

1. What is the correct syntax to create a numpy array of 9 elements filled with all zeros (float)?

- a) `np.zeros()`
- b) `np.zeros(0)`
- c) `np.zeros(9)`



Quiz Time

1. What is the correct syntax to create a numpy array of 9 elements filled with all zeros (float)?

- a) `np.zeros()`
- b) `np.zeros(0)`
- c) `np.zeros(9)`



Scalar Operations (1/5)

Addition

- We can add a scalar to a NumPy array simply by using the (+) operator.
- The scalar quantity is added to each of the elements of the array.
- Note that adding a scalar to a Python list will result in an error.

The image shows a Jupyter Notebook interface with the title "NumPy Functionality". The top bar includes a "Logout" button and a "Python 3" selector. The menu bar contains "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Below the menu is a toolbar with icons for saving, adding cells, undo, redo, and running code. The notebook contains three code cells:

```
In [38]: matrix = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix)

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

In [39]: print(matrix + 2)

[[3 4 5]
 [6 7 8]]

In [40]: pythonList = [[1, 2, 3], [4, 5, 6]]
print(pythonList + 2)

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-40-788a2ae497a2> in <module>
      1 pythonList = [[1, 2, 3], [4, 5, 6]]
----> 2 print(pythonList + 2)

TypeError: can only concatenate list (not "int") to list
```




Scalar Operations (2/5)

Subtraction

- We can subtract a scalar from a NumPy array simply by using the (-) operator.
- The scalar quantity is subtracted from each of the elements of the array.
- Note that subtracting a scalar from a Python list will result in an error.

```
jupyter NumPy Functionality Last Checkpoint: 8 minutes ago (unsaved changes) Logout
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [41]: matrix = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix)
[[1 2 3]
 [4 5 6]]
In [42]: print(matrix - 2)
[[-1  0  1]
 [ 2  3  4]]
In [43]: pythonList = [[1, 2, 3], [4, 5, 6]]
print(pythonList - 2)
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-43-0cdc8cdac650> in <module>
      1 pythonList = [[1, 2, 3], [4, 5, 6]]
----> 2 print(pythonList - 2)
TypeError: unsupported operand type(s) for -: 'list' and 'int'
```



Scalar Operations (3/5)

Multiplication

- We can multiply a scalar with a NumPy array simply by using the (*) operator.
- The scalar quantity is multiplied with each of the elements of the array.
- Note that multiplying a scalar with a Python list will result in list concatenation.

The image shows a Jupyter Notebook interface with the title "NumPy Functionality". The top bar includes a "Logout" button and a "Python 3" selector. The menu bar contains "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Below the menu is a toolbar with icons for saving, adding cells, undo, redo, and running code. The notebook contains four code cells:

```
In [8]: import numpy as np

In [41]: matrix = np.array([[1, 2, 3], [4, 5, 6]])
         print(matrix)
         [[1 2 3]
          [4 5 6]]

In [44]: print(matrix * 2)
         [[ 2  4  6]
          [ 8 10 12]]

In [45]: pythonList = [[1, 2, 3], [4, 5, 6]]
         print(pythonList * 2)
         [[1, 2, 3], [4, 5, 6], [1, 2, 3], [4, 5, 6]]
```



Scalar Operations (4/5)

Division

- We can divide a NumPy array by a scalar simply by using the (/) operator for float division or (//) operator for integer division.
- Each of the elements of the array is divided by the scalar.
- Note that dividing a Python list by a scalar will result in an error.

The screenshot shows a Jupyter Notebook interface with the title 'NumPy Functionality'. The notebook contains four code cells. The first three cells demonstrate dividing a NumPy array by a scalar. The first cell creates a 2x3 array. The second cell divides the array by 2 using the / operator, resulting in a float array. The third cell divides the array by 2 using the // operator, resulting in an integer array. The fourth cell attempts to divide a Python list by 2 using the / operator, which results in a TypeError. The error message states: 'TypeError: unsupported operand type(s) for /: 'list' and 'int''. The traceback shows the error occurred in the fourth cell.

```
jupyter NumPy Functionality Last Checkpoint: 10 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [41]: matrix = np.array([[1, 2, 3], [4, 5, 6]])
         print(matrix)
         [[1 2 3]
          [4 5 6]]

In [47]: print(matrix / 2)
         [[0.5 1.  1.5]
          [2.  2.5 3.  ]]

In [48]: print(matrix // 2)
         [[0 1 1]
          [2 2 3]]

In [49]: pythonList = [[1, 2, 3], [4, 5, 6]]
         print(pythonList / 2)

         -----
         TypeError                                 Traceback (most recent call last)
         <ipython-input-49-76559e2dbe98> in <module>
             1 pythonList = [[1, 2, 3], [4, 5, 6]]
         ----> 2 print(pythonList / 2)

         TypeError: unsupported operand type(s) for /: 'list' and 'int'
```



Scalar Operations (5/5)

Power

- We can raise each element of a NumPy array to a power simply by using the `(**)` operator.
- Note that raising the elements of a Python list using `(**)` operator will result in an error.

The screenshot shows a Jupyter Notebook interface with the title "NumPy Functionality". The top bar includes a "Logout" button and a "Python 3" selector. The menu bar contains "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Below the menu is a toolbar with icons for file operations, running, and other notebook functions. The notebook contains three code cells:

```
In [41]: matrix = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix)

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

In [50]: print(matrix ** 2)

[[ 1  4  9]
 [16 25 36]]

In [51]: pythonList = [[1, 2, 3], [4, 5, 6]]
print(pythonList ** 2)
```

The output of the third cell is a `TypeError` traceback:

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-51-301391818ac0> in <module>
      1 pythonList = [[1, 2, 3], [4, 5, 6]]
----> 2 print(pythonList ** 2)

TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```



Transpose

- We can take the transpose of a NumPy array by putting .T at the end of the array.
- Note that taking transpose of a Python list will result in an error.

The image shows a Jupyter Notebook interface with the title "NumPy Functionality". The top bar includes a "Logout" button and a "Python 3" selector. The notebook contains three code cells. The first cell, labeled "In [41]:", defines a NumPy array named "matrix" and prints it, showing the output as a 2x3 array. The second cell, labeled "In [52]:", prints the transpose of "matrix" using ".T", showing the output as a 3x2 array. The third cell, labeled "In [53]:", defines a Python list named "pythonList" and attempts to print its transpose using ".T". This results in an "AttributeError" because lists do not have a ".T" attribute. The error message is displayed in red text, including a traceback that shows the error occurred in the second line of the code cell.

```
jupyter NumPy Functionality Last Checkpoint: 12 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [41]: matrix = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix)
[[1 2 3]
 [4 5 6]]

In [52]: print(matrix.T)
[[1 4]
 [2 5]
 [3 6]]

In [53]: pythonList = [[1, 2, 3], [4, 5, 6]]
print(pythonList.T)

-----
AttributeError                                Traceback (most recent call last)
<ipython-input-53-f8b6fc22b132> in <module>
      1 pythonList = [[1, 2, 3], [4, 5, 6]]
----> 2 print(pythonList.T)

AttributeError: 'list' object has no attribute 'T'
```



Element-wise Operations (1/4)

Addition

- We can add the elements of two NumPy arrays simply by using (+) operator.
- Each element of the first array is added to the corresponding element of the second array.
- Note that adding two Python lists using plus (+) operator is not possible. Instead the lists are concatenated if we use (+) operator.

```
jupyter NumPy Functionality Last Checkpoint: 13 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [8]: import numpy as np

In [54]: matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix1)
[[1 2 3]
 [4 5 6]]

In [56]: matrix2 = np.array([[-1, -2, -3], [-4, -5, -6]])
print(matrix2)
[[-1 -2 -3]
 [-4 -5 -6]]

In [57]: print(matrix1 + matrix2)
[[0 0 0]
 [0 0 0]]

In [58]: pythonList1 = [[1, 2, 3], [4, 5, 6]]
pythonList2 = [[-1, -2, -3], [-4, -5, -6]]
print(pythonList1 + pythonList2)
[[1, 2, 3], [4, 5, 6], [-1, -2, -3], [-4, -5, -6]]
```



Element-wise Operations (2/4)

Subtraction

- We can subtract the elements of two NumPy arrays simply by using (-) operator.
- Each element of the second array is subtracted from the corresponding element of the first array.
- Note that subtracting the elements of two Python lists using (-) operator will result in an error.

```
jupyter NumPy Functionality Last Checkpoint: 14 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [54]: matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
         print(matrix1)
[[1 2 3]
 [4 5 6]]

In [56]: matrix2 = np.array([[-1, -2, -3], [-4, -5, -6]])
         print(matrix2)
[[-1 -2 -3]
 [-4 -5 -6]]

In [59]: print(matrix1 - matrix2)
[[ 2  4  6]
 [ 8 10 12]]

In [60]: pythonList1 = [[1, 2, 3], [4, 5, 6]]
         pythonList2 = [[-1, -2, -3], [-4, -5, -6]]
         print(pythonList1 - pythonList2)

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-60-b0182aedaa2a> in <module>
      1 pythonList1 = [[1, 2, 3], [4, 5, 6]]
      2 pythonList2 = [[-1, -2, -3], [-4, -5, -6]]
----> 3 print(pythonList1 - pythonList2)

TypeError: unsupported operand type(s) for -: 'list' and 'list'
```



Element-wise Operations (3/4)

Multiplication

- We can multiply the elements of two NumPy arrays simply by using (*) operator.
- Each element of the first array is multiplied by the corresponding element of the second array.
- Note that multiplying the elements of two Python lists using (*) operator will result in an error.

```
jupyter NumPy Functionality Last Checkpoint: 15 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [54]: matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
         print(matrix1)
         [[1 2 3]
          [4 5 6]]

In [56]: matrix2 = np.array([[1, 2, 3], [-4, -5, -6]])
         print(matrix2)
         [[1 2 3]
          [-4 -5 -6]]

In [62]: print(matrix1 * matrix2)
         [[ 1 -4 -9]
          [-16 -25 -36]]

In [61]: pythonList1 = [[1, 2, 3], [4, 5, 6]]
         pythonList2 = [[1, 2, 3], [-4, -5, -6]]
         print(pythonList1 * pythonList2)

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-61-e9ba5f4c2fff> in <module>
      1 pythonList1 = [[1, 2, 3], [4, 5, 6]]
      2 pythonList2 = [[1, 2, 3], [-4, -5, -6]]
----> 3 print(pythonList1 * pythonList2)

TypeError: can't multiply sequence by non-int of type 'list'
```




Element-wise Operations (4/4)

Division

- We can divide the elements of two NumPy arrays simply by using (/) operator.
- Each element of the first array is divided by the corresponding element of the second array.
- Note that dividing the elements of two Python lists using (/) operator will result in an error.

```
jupyter NumPy Functionality Last Checkpoint: 16 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [54]: matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
         print(matrix1)
         [[1 2 3]
          [4 5 6]]

In [56]: matrix2 = np.array([[-1, -2, -3], [-4, -5, -6]])
         print(matrix2)
         [[-1 -2 -3]
          [-4 -5 -6]]

In [67]: print(matrix1 / matrix2)
         [[-1. -1. -1.]
          [-1. -1. -1.]]

In [68]: print(matrix1 // matrix2)
         [[-1 -1 -1]
          [-1 -1 -1]]

In [69]: pythonList1 = [[1, 2, 3], [4, 5, 6]]
         pythonList2 = [[-1, -2, -3], [-4, -5, -6]]
         print(pythonList1 / pythonList2)
         -----
         Traceback (most recent call last)
         <ipython-input-69-7faf5c77848e> in <module>
             1 pythonList1 = [[1, 2, 3], [4, 5, 6]]
             2 pythonList2 = [[-1, -2, -3], [-4, -5, -6]]
         ----> 3 print(pythonList1 / pythonList2)
         TypeError: unsupported operand type(s) for /: 'list' and 'list'
```



Matrix Multiplication

- Apart from elementwise multiplication, NumPy also provides us with a built-in function to compute the matrix multiplication of two arrays.
- We use the `.matmul()` function inside the NumPy library for matrix multiplication of two arrays.

```
jupyter NumPy Functionality Last Checkpoint: 19 minutes ago (unsaved changes) Python 3
```

```
File Edit View Insert Cell Kernel Widgets Help Trusted
```

```
In [75]: matrix1 = np.array([[1, 2, 3], [4, 5, 6], [0, 0, 0]])
         print(matrix1)
         [[1 2 3]
          [4 5 6]
          [0 0 0]]

In [76]: matrix2 = np.array([[-1, -2, -3], [-4, -5, -6], [0, 0, 0]])
         print(matrix2)
         [[-1 -2 -3]
          [-4 -5 -6]
          [ 0  0  0]]

In [77]: np.matmul(matrix1, matrix2)
Out[77]: array([[ -9, -12, -15],
                [-24, -33, -42],
                [  0,  0,  0]])
```



Quiz Time

1. Which of the following statements is correct?
 - a) * is used for array multiplication
 - b) * is used for element-wise multiplication
 - c) * is used for power



Quiz Time

1. Which of the following statements is correct?

- a) * is used for array multiplication
- b) * is used for element-wise multiplication
- c) * is used for power



Statistics (1/7)

.min()

- .min() function gives us the minimum value in a NumPy array.
- This function can also be applied on Python lists.

```
In [8]: import numpy as np

In [80]: matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
          print(matrix1)
          [[1 2 3]
           [4 5 6]]

In [81]: np.min(matrix1)
Out[81]: 1
```



Statistics (2/7)

.max()

- .max() function gives us the maximum value in a NumPy array.
- This function can also be applied on Python lists.

```
jupyter NumPy Functionality Last Checkpoint: 20 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Notebook saved Trusted Python 3

In [8]: import numpy as np

In [80]: matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
         print(matrix1)
         [[1 2 3]
          [4 5 6]]

In [82]: np.max(matrix1)
Out[82]: 6
```



Statistics (3/7)

.sum()

- .sum() function gives us the sum of all the values in a NumPy array.
- This function can also be applied on Python lists.

The image shows a Jupyter Notebook interface with the title "NumPy Functionality". The top bar includes a "Logout" button and a "Python 3" selector. The menu bar contains "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Below the menu is a toolbar with icons for saving, adding cells, undo, redo, and running code. The notebook contains three code cells:

```
In [8]: import numpy as np
```

```
In [80]: matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
          print(matrix1)
          [[1 2 3]
           [4 5 6]]
```

```
In [85]: np.sum(matrix1)
Out[85]: 21
```



Statistics (4/7)

.mean()

- .mean() function gives us the mean of all the values in a NumPy array.
- This function can also be applied on Python lists.

```
jupyter NumPy Functionality Last Checkpoint: 20 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [8]: import numpy as np

In [80]: matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
         print(matrix1)
         [[1 2 3]
          [4 5 6]]

In [83]: np.mean(matrix1)
Out[83]: 3.5
```




Statistics (5/7)

.std()

- .std() function gives us the standard deviation of a NumPy array.
- This function can also be applied on Python lists.

```
In [8]: import numpy as np

In [80]: matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
          print(matrix1)
          [[1 2 3]
           [4 5 6]]

In [84]: np.std(matrix1)
Out[84]: 1.707825127659933
```



Statistics (6/7)

.median()

- .median() function gives us the median of a NumPy array.
- This function can also be applied on Python lists.

```
In [8]: import numpy as np

In [80]: matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
         print(matrix1)
         [[1 2 3]
          [4 5 6]]

In [86]: np.median(matrix1)
Out[86]: 3.5
```



Statistics (7/7)

- A detailed list of NumPy statistical functions can be found at the link below;
<https://numpy.org/doc/stable/reference/routines.statistics.html>



Resources

- https://www.w3schools.com/python/numpy/numpy_intro.asp
- <https://www.tutorialspoint.com/numpy/index.htm>