Basic Programming in Python

7. Session: Matplotlib

Nohayr Muhammad Abdelmoneim

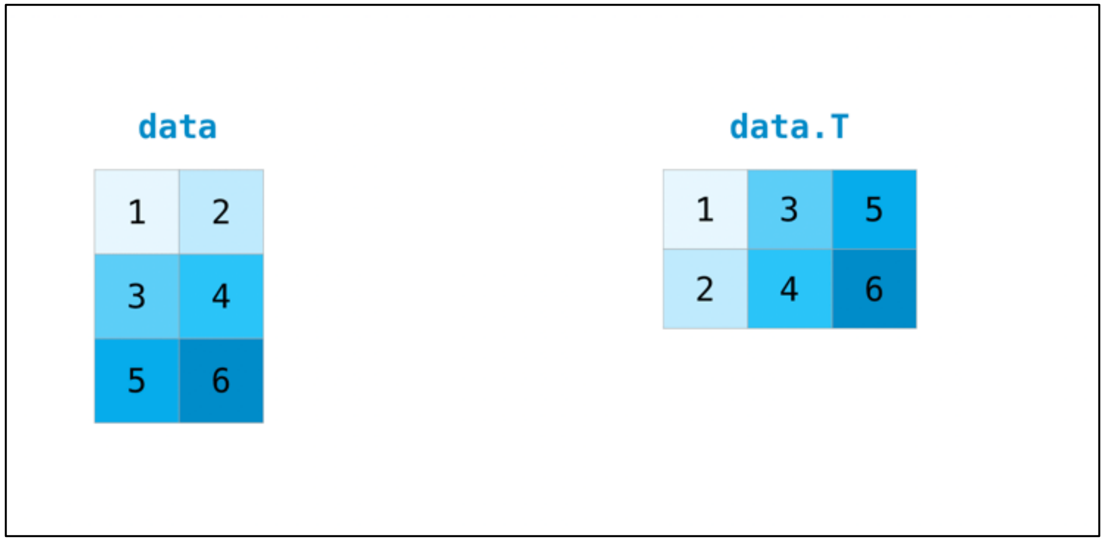
Summer Term 2023

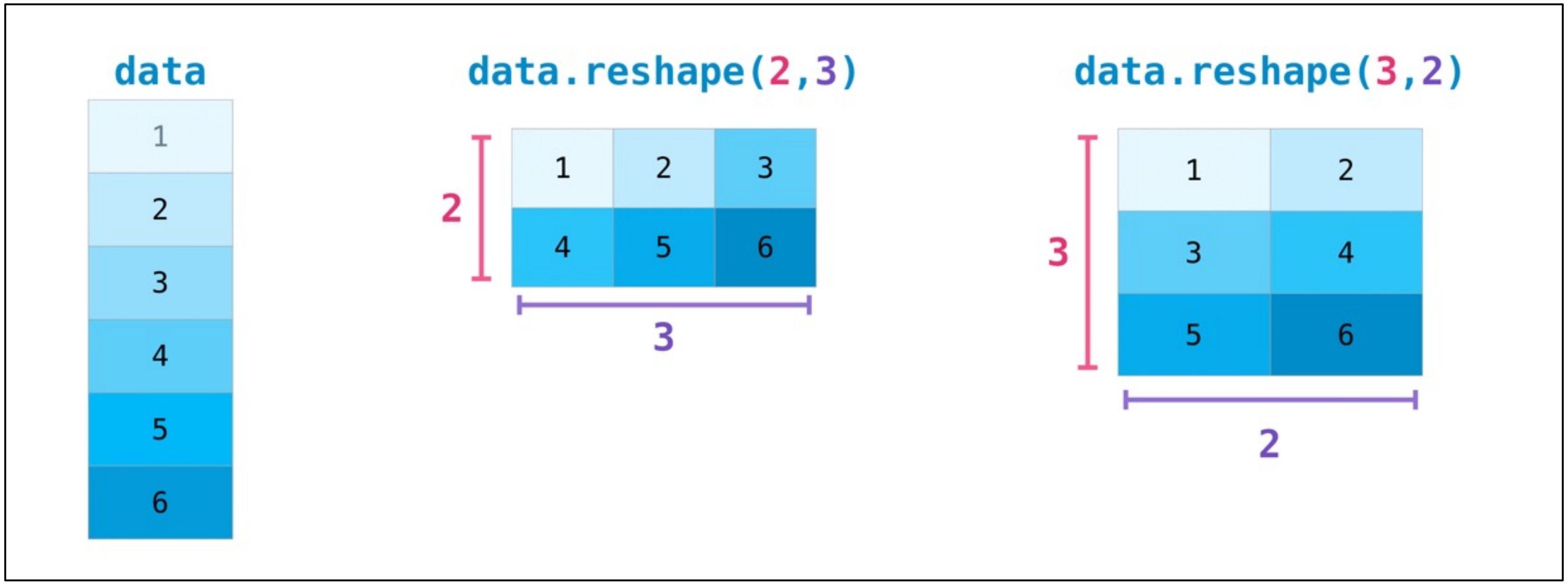
June 12th, 2023

Overview

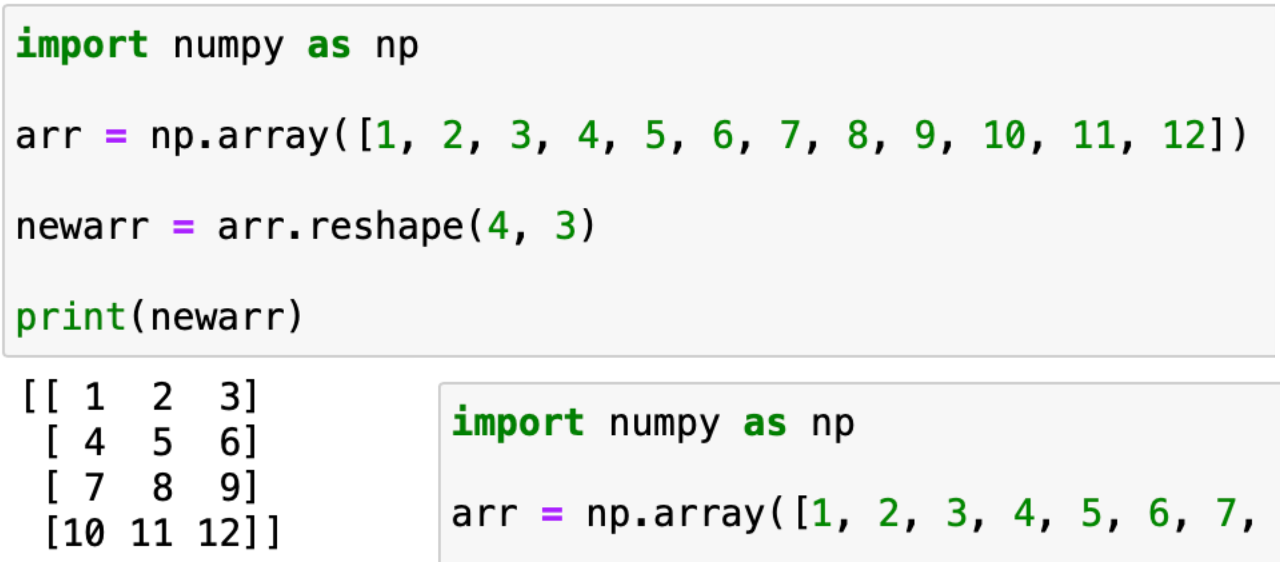
- Recap on NumPy
- Some more NumPy
- Matplotlib
- Properties of plt.plot()
- Different plotting variations

Shape manipulation



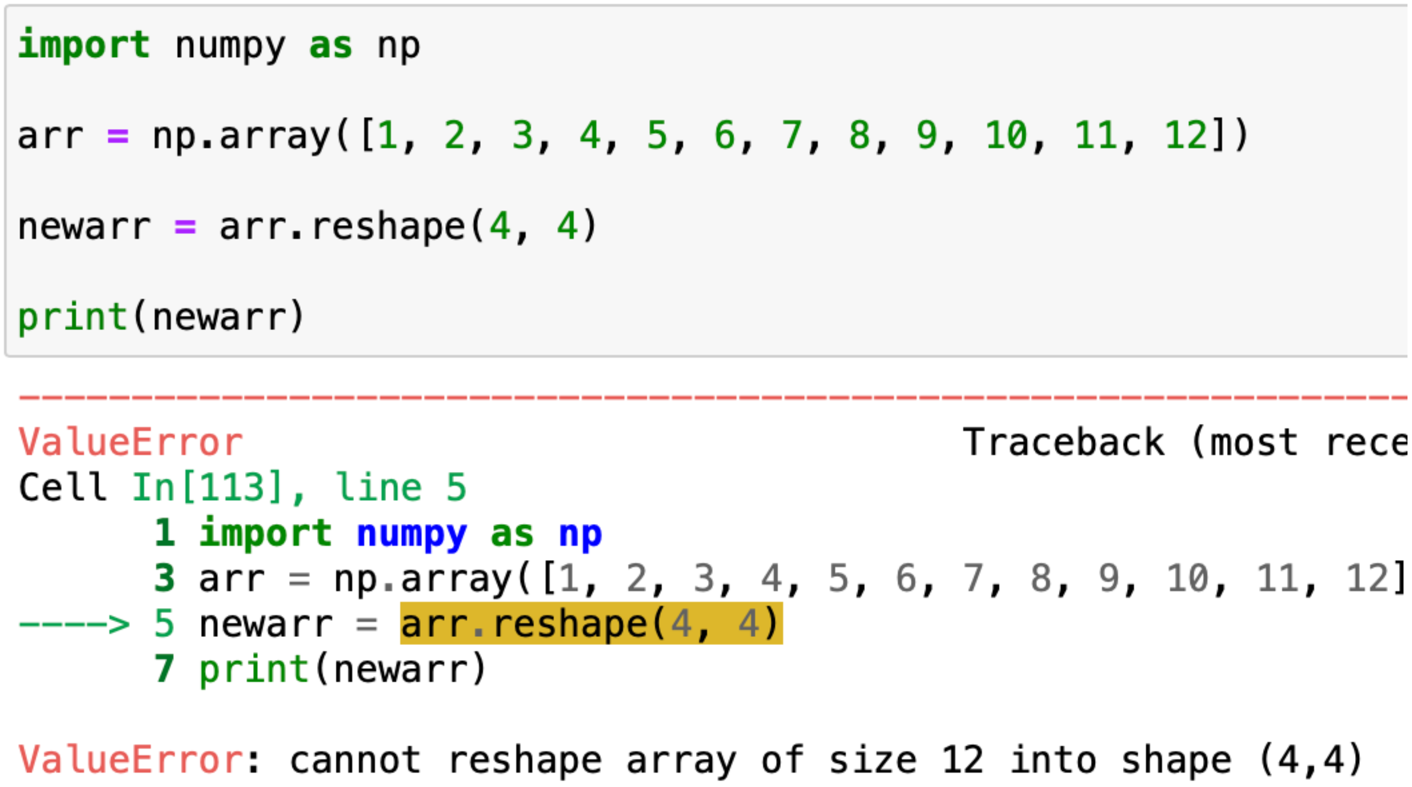


Shape manipulation



```python
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3)

print(newarr)
```

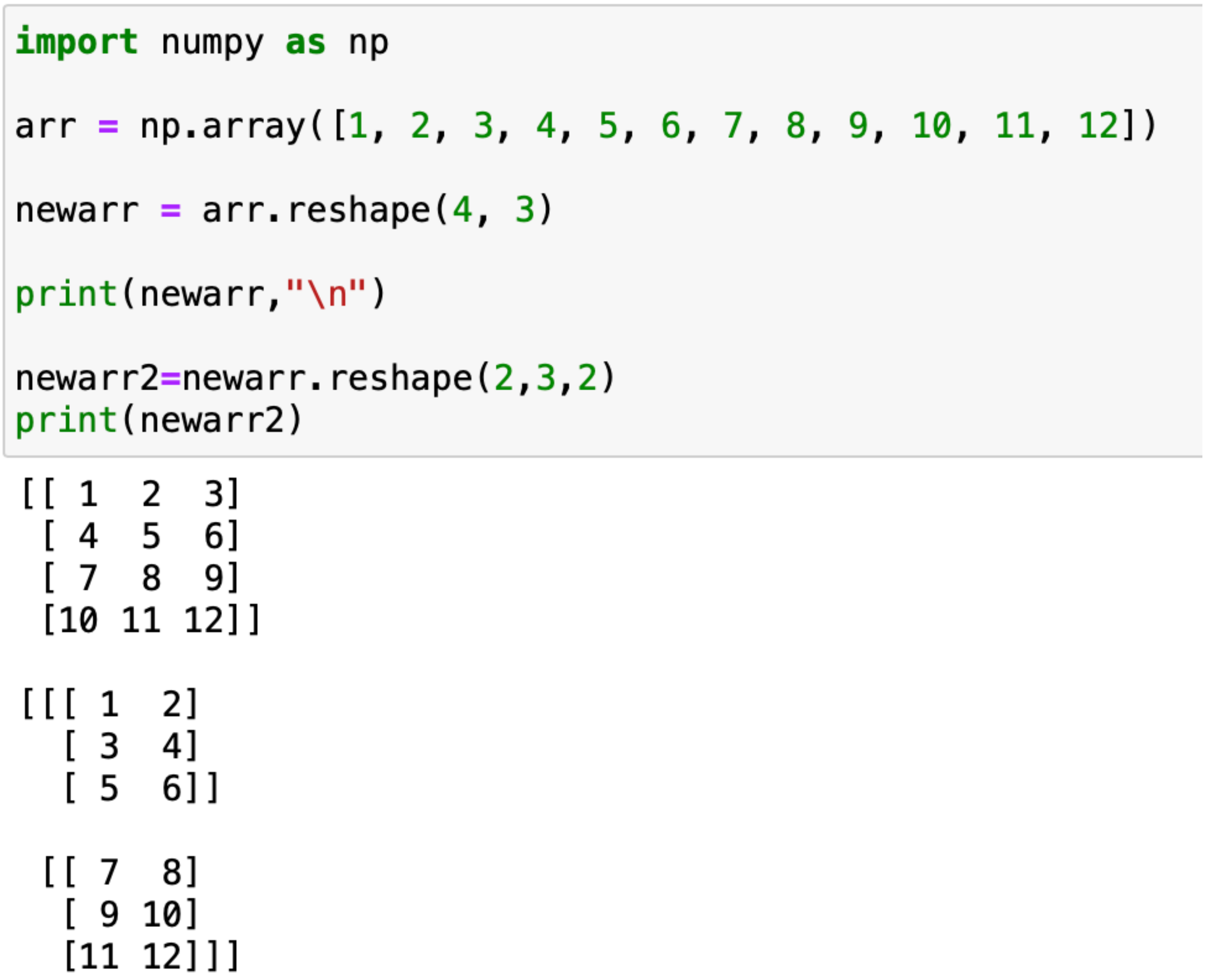
```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```



```python
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 4)

print(newarr)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most rece
Cell In[113], line 5
      1 import numpy as np
      3 arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
----> 5 newarr = arr.reshape(4, 4)
      7 print(newarr)

ValueError: cannot reshape array of size 12 into shape (4,4)
```

Shape manipulation



```python
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3)

print(newarr,"\n")

newarr2=newarr.reshape(2,3,2)
print(newarr2)
```

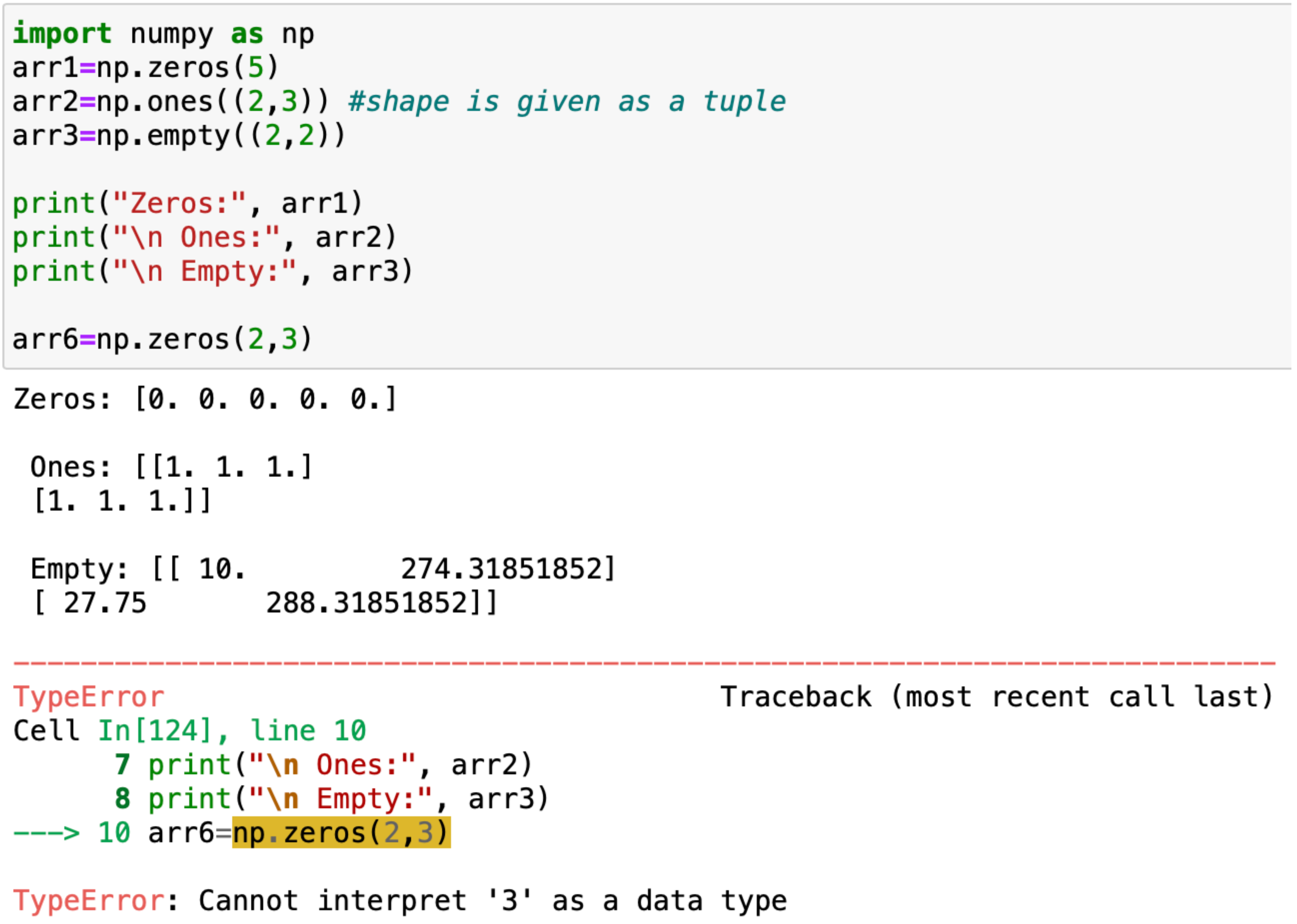
```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]

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

 [[ 7  8]
  [ 9 10]
  [11 12]]]
```

Automatic array creation

- np.zeros(shape): Creates an array full of zeros of the given shape.
- np.ones(shape): Creates an array full of ones of the given shape.
- np.empty(shape): Creates an array of initially random values of the given shape.

P.S. For more than 1-D, shape should be given as a tuple.

Automatic array creation



```python
import numpy as np
arr1=np.zeros(5)
arr2=np.ones((2,3)) #shape is given as a tuple
arr3=np.empty((2,2))

print("Zeros:", arr1)
print("\n Ones:", arr2)
print("\n Empty:", arr3)

arr6=np.zeros(2,3)
```

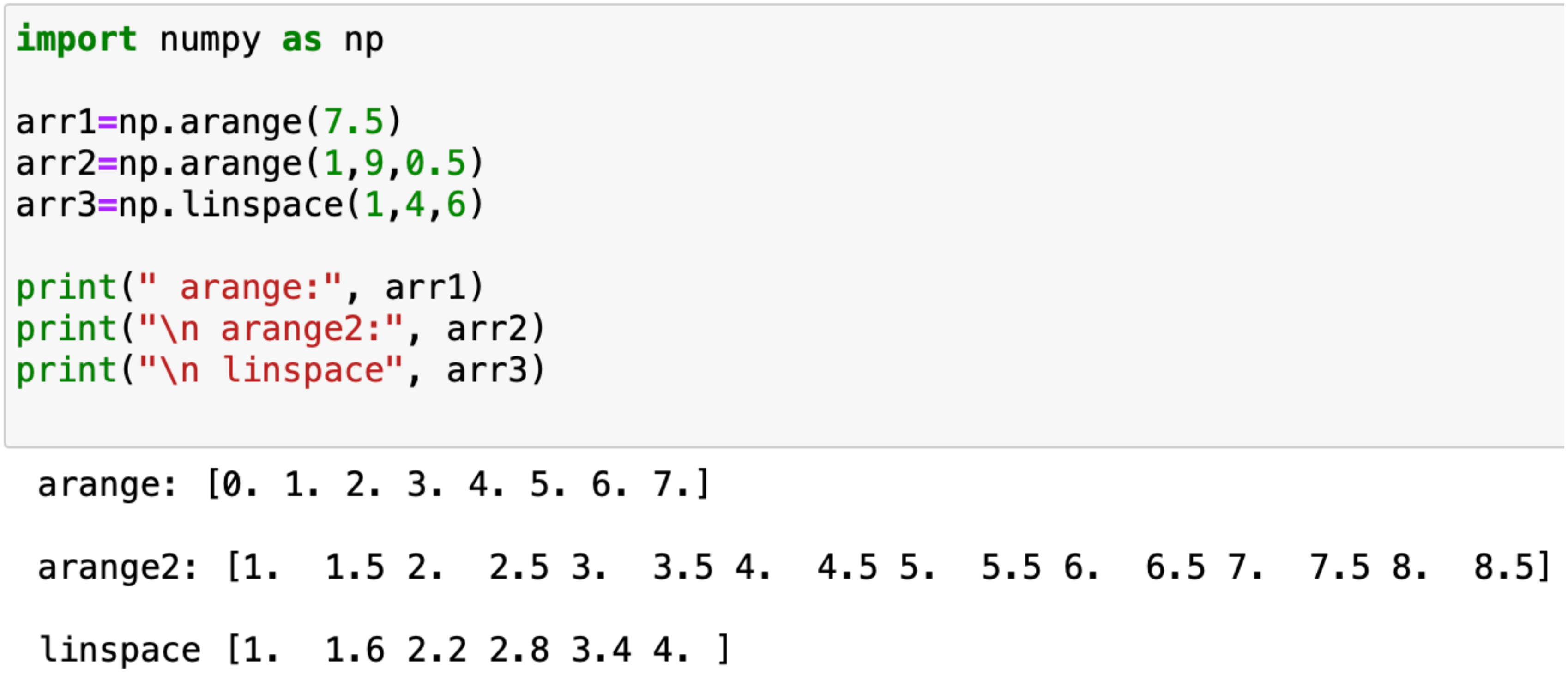
```
Zeros: [0. 0. 0. 0. 0.]

 Ones: [[1. 1. 1.]
 [1. 1. 1.]]

 Empty: [[ 10.         274.31851852]
 [ 27.75       288.31851852]]

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[124], line 10
      7 print("\n Ones:", arr2)
      8 print("\n Empty:", arr3)
---> 10 arr6=np.zeros(2,3)

TypeError: Cannot interpret '3' as a data type
```

Automatic array creation cont.

- np.arange(end)
  np.arange(start,end)
  np.arange(start,end,step).
- np.linspace(start, end, number of elements).
- np.eye(shape) Creates identity matrix, i.e., ones in the diagonal and zeros elsewehere.

P.S. In np.eye, shape is not given as a tuple. It’s given as number of rows and number of columns. If number of columns is not given then it’s considered equal to rows be default.

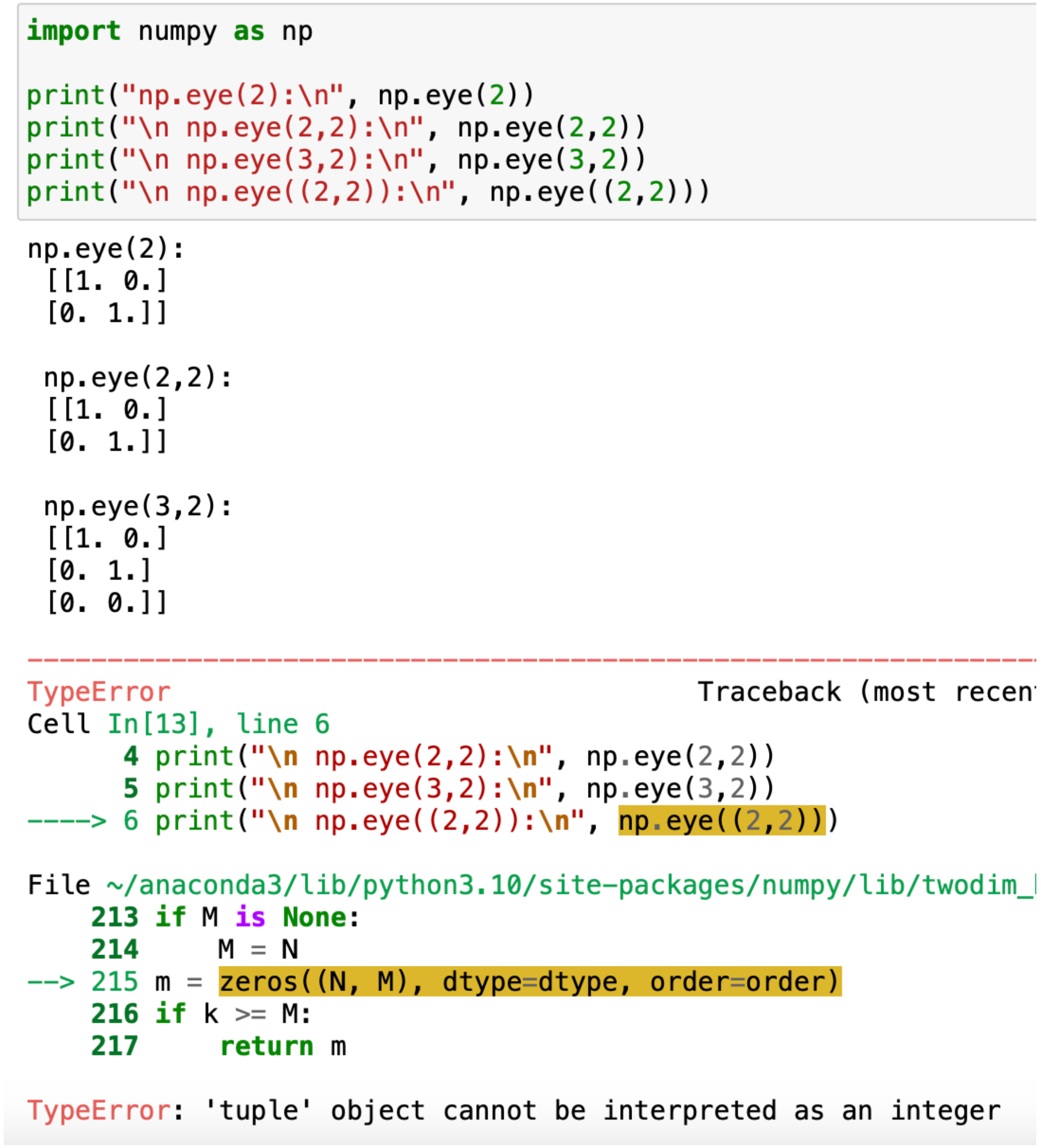
Automatic array creation



```python
import numpy as np

arr1=np.arange(7.5)
arr2=np.arange(1,9,0.5)
arr3=np.linspace(1,4,6)

print(" arange:", arr1)
print("\n arange2:", arr2)
print("\n linspace", arr3)
```

```
 arange: [0. 1. 2. 3. 4. 5. 6. 7.]

 arange2: [1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5 6.  6.5 7.  7.5 8.  8.5]

 linspace [1.  1.6 2.2 2.8 3.4 4. ]
```

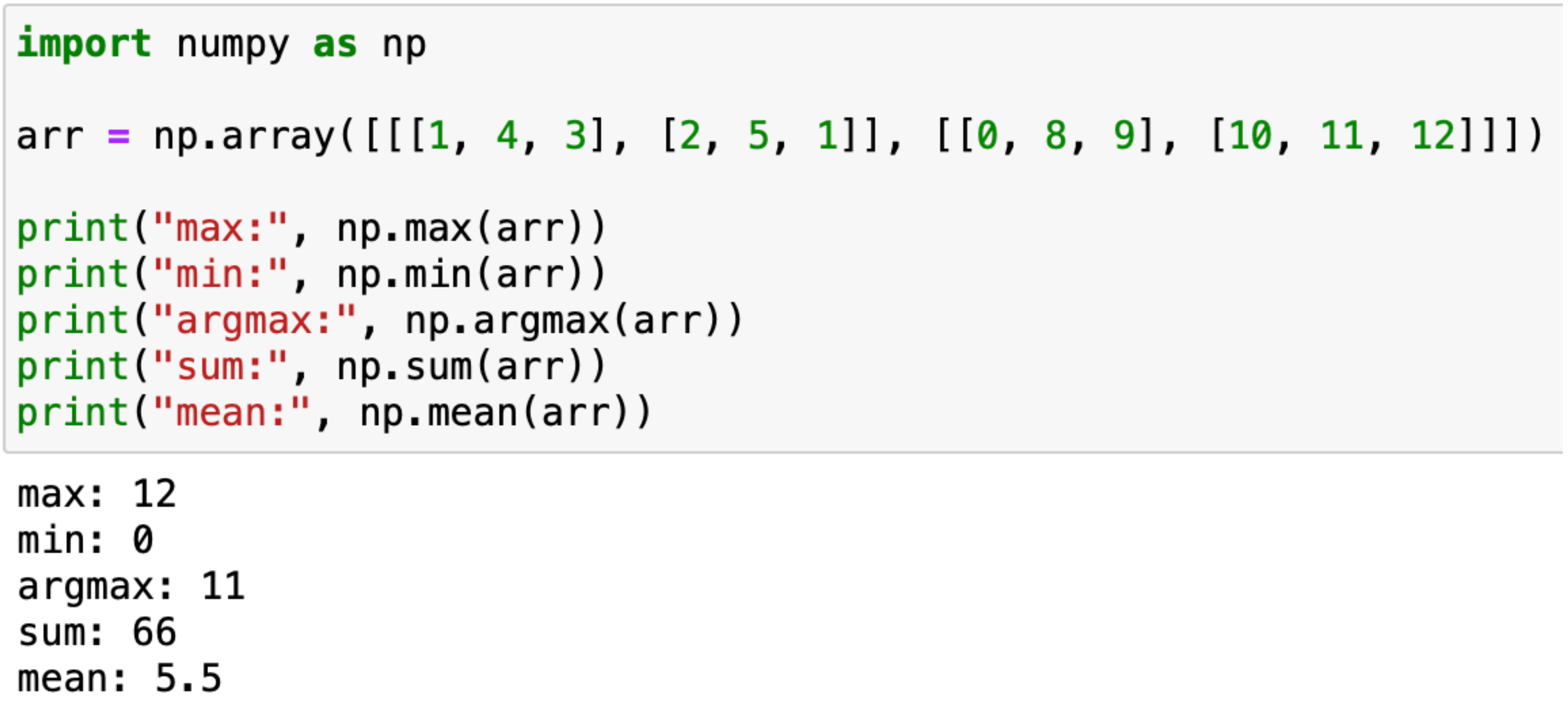
Automatic array creation



```
import numpy as np

print("np.eye(2):\n", np.eye(2))
print("\n np.eye(2,2):\n", np.eye(2,2))
print("\n np.eye(3,2):\n", np.eye(3,2))
print("\n np.eye((2,2)):\n", np.eye((2,2)))
```

```
np.eye(2):
 [[1. 0.]
 [0. 1.]]

 np.eye(2,2):
 [[1. 0.]
 [0. 1.]]

 np.eye(3,2):
 [[1. 0.]
 [0. 1.]
 [0. 0.]]

---------------------------------------------------------------
TypeError                                 Traceback (most recen
Cell In[13], line 6
      4 print("\n np.eye(2,2):\n", np.eye(2,2))
      5 print("\n np.eye(3,2):\n", np.eye(3,2))
----> 6 print("\n np.eye((2,2)):\n", np.eye((2,2)))

File ~/anaconda3/lib/python3.10/site-packages/numpy/lib/twodim_
    213 if M is None:
    214     M = N
--> 215 m = zeros((N, M), dtype=dtype, order=order)
    216 if k >= M:
    217     return m

TypeError: 'tuple' object cannot be interpreted as an integer
```

Some arithmetic methods



```python
import numpy as np

arr = np.array([[[1, 4, 3], [2, 5, 1]], [[0, 8, 9], [10, 11, 12]]])

print("max:", np.max(arr))
print("min:", np.min(arr))
print("argmax:", np.argmax(arr))
print("sum:", np.sum(arr))
print("mean:", np.mean(arr))
```

```
max: 12
min: 0
argmax: 11
sum: 66
mean: 5.5
```

Simple search

```python
import numpy as np

arr = np.array([[[1, 4, 3], [2, 5, 1]], [[0, 8, 9], [10, 11, 12]]])

print(arr>3)
print("\n Elements greater than 3: ", arr[arr>3])
```

```
[[[False  True False]
  [False  True False]]

 [[False  True  True]
  [ True  True  True]]]

 Elements greater than 3:  [ 4  5  8  9 10 11 12]
```

- Practice: Can this be used to replace zero elements from an identity matrix by elements from another matrix? (Assignment 7 task 3)
  Use the above property to replace the given code:
- identity_matrix[~np.eye(5, dtype=bool)] = random_integers[~np.eye(5, dtype=bool)]

NumPy Random

- random is a NumPy module that deals with random numbers, random distributions, etc.

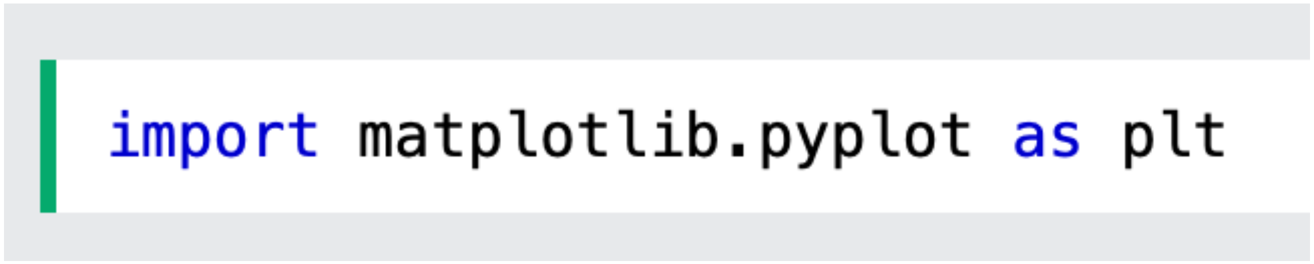
```python
from numpy import random

x=random.randint(100)#Generates a random int between 0 and 100
print("randint:" ,x)
x2=random.randint(100,size=(2,3))#Generates an array of random int between 0 and 100 of given size
print("randint array:\n" ,x2)
y=random.rand()#Generates a random float between 0 and 1
print("rand:\n" ,y)
z2=random.rand(2,3)#Generates an array of random float between 0 and 1 of given size
print("rand array:\n" ,z2)
```

```
randint: 70
randint array:
 [[45 32  8]
 [92 13 41]]
rand:
 0.4597258740429102
rand array:
 [[0.31473771 0.41489667 0.07804528]
 [0.6918078  0.28935831 0.07779068]]
```

Matplotlib

Matplotlib

- Matplotlib is arguably the most popular graphing and data visualization library for Python.
- Install matplotlib (e.g. using pip)
- Import matplotlib
- Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias.



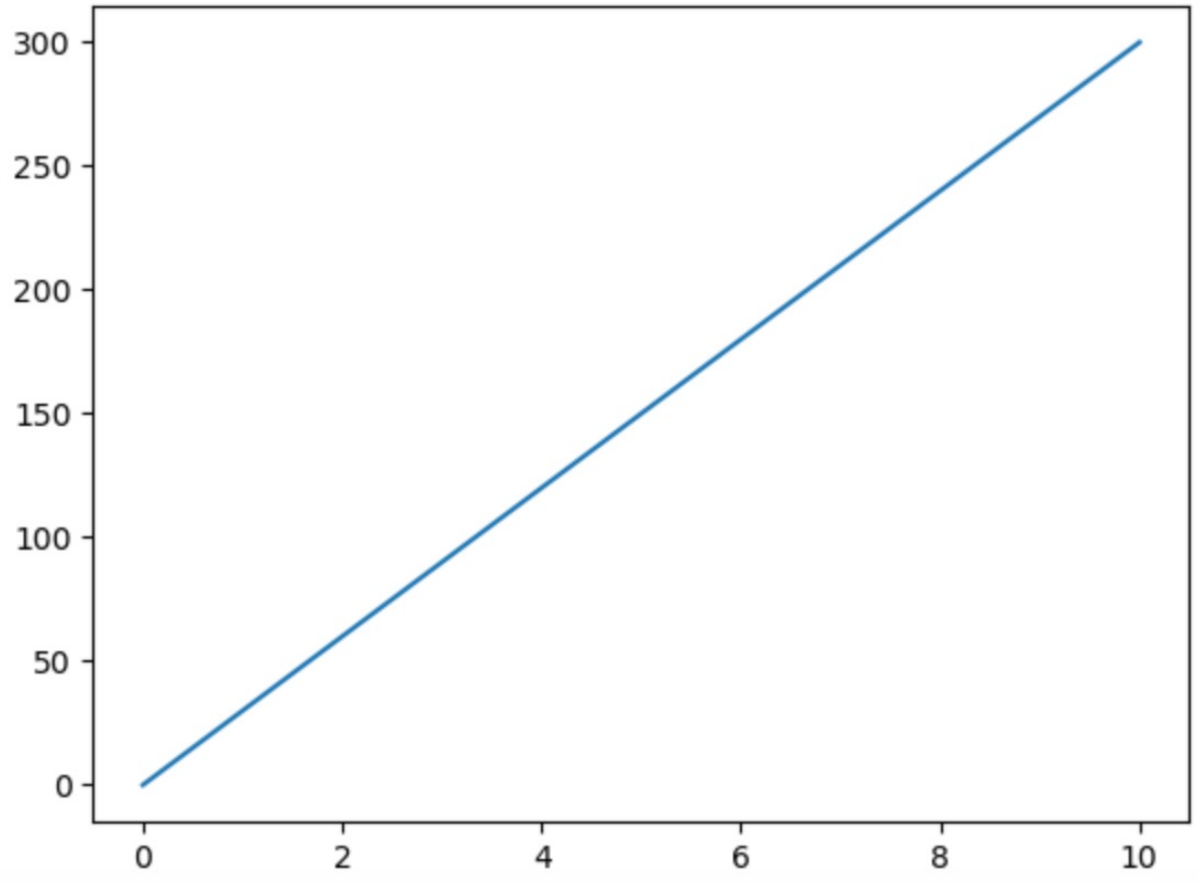
```python
import matplotlib.pyplot as plt
```

Matplotlib

- The plot() method in pyplot is responsible for plotting.
- In the simplest form, plot() will draw a line between two points p_1 and p_2
- To do so, plot will take two parameters plot(x,y)
- Where x contains the points on x-axis, and y contains the points on y-axis.
- The properties of the graph like: line shape, points shape, colors, labels, etc., can be modified using plot().

Plot Example

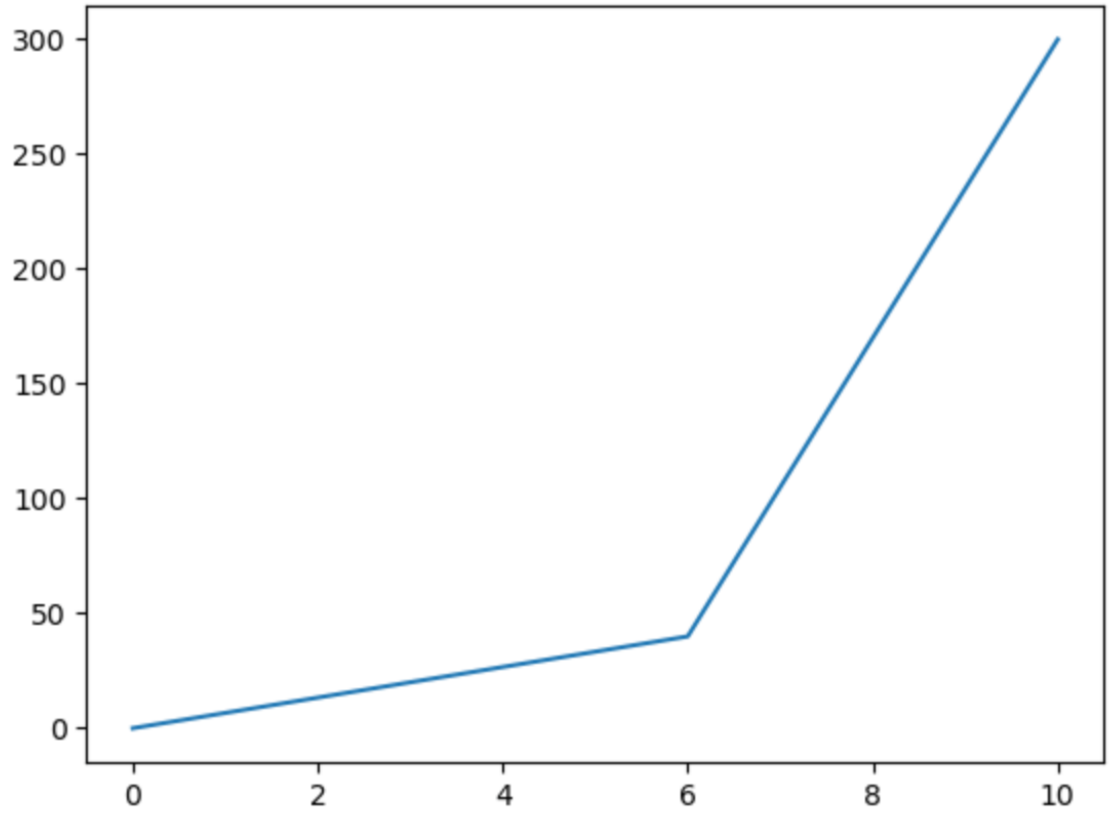
```
import matplotlib.pyplot as plt

xpoints = (0, 10)
ypoints = (0, 300)
#This means that I want to draw a line between point (0,0) and (10,300)
plt.plot(xpoints, ypoints)
plt.show()
```



Example

- How many points do we have in this case?
- How would the graph look like?

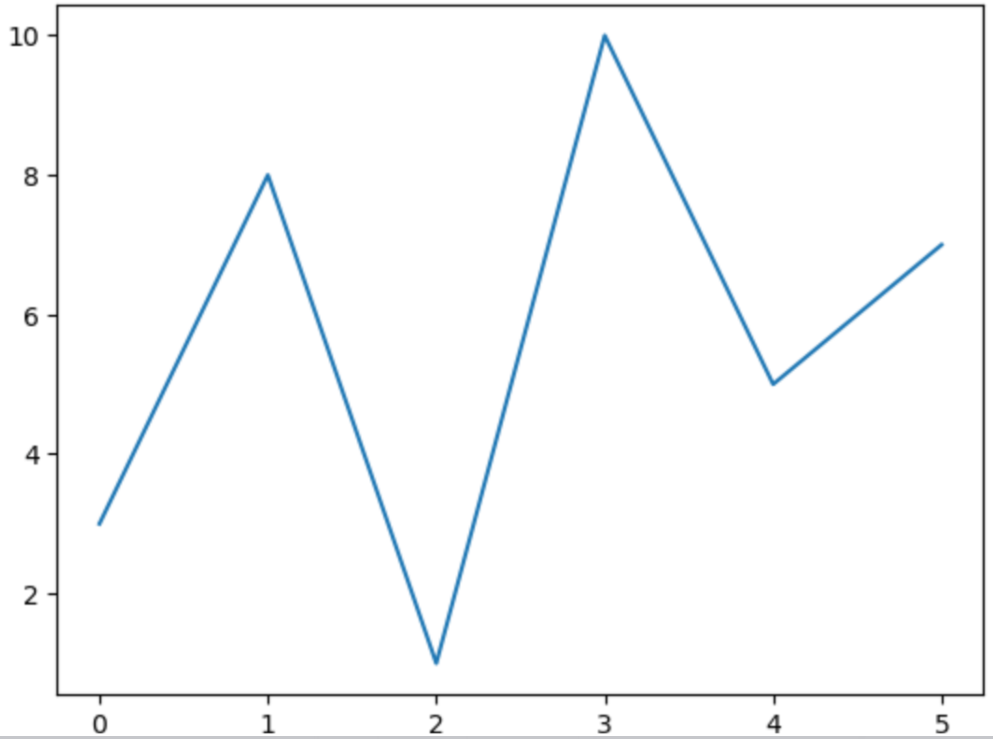
```
import matplotlib.pyplot as plt

xpoints = (0, 6, 10)
ypoints = (0, 40, 300)

plt.plot(xpoints, ypoints)
plt.show()
```



Default values

- If one parameter is passed to plot(), it's considered y-values.
- In this case, x-axis take values 0,1,2,3,…,n-1
  where n is the number of the given y values

```
import matplotlib.pyplot as plt
import numpy as np

points = np.array([3, 8, 1, 10, 5, 7])

plt.plot(points)
plt.show()
```

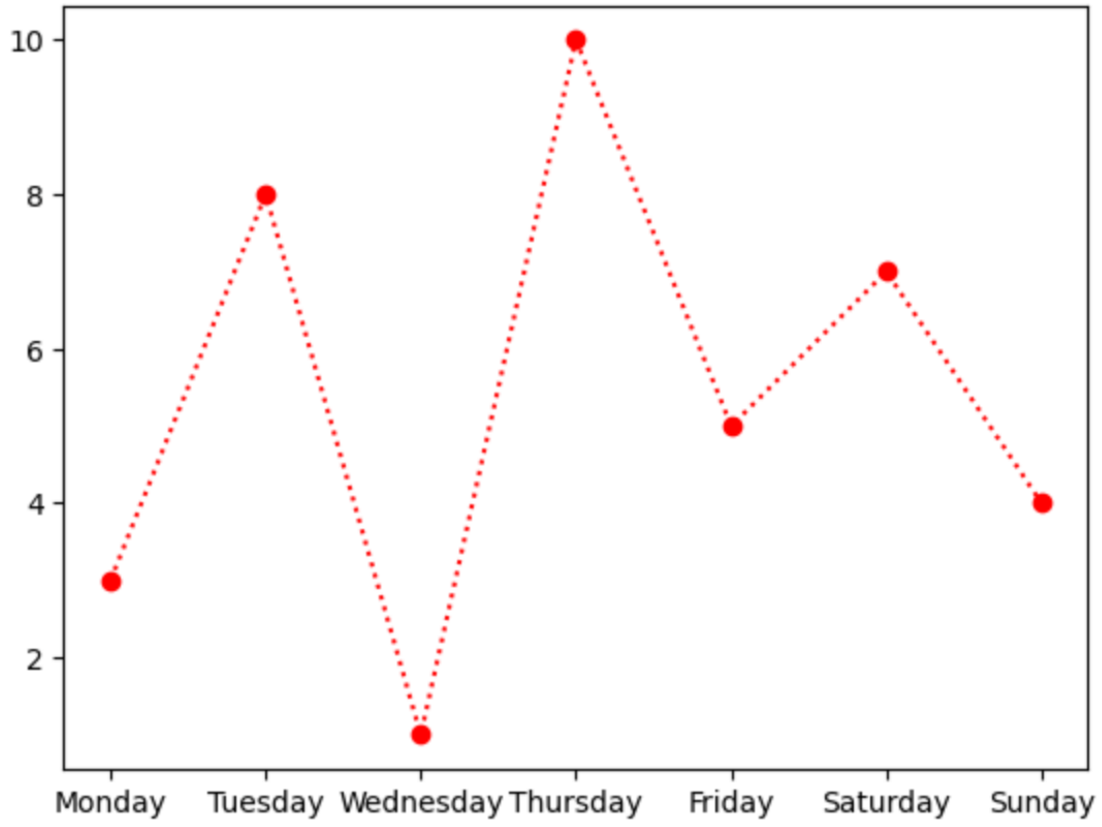


Graph properties

- The graph properties can be specified either by setting the values for keywords defining style, color, etc.
- Keywords:
  - marker: specifies the shape of the point (‘o’, ‘*’, ‘.’, etc.)
  - markersize or ms: specifies the size of the point (takes a numerical value)
  - markeredgecolor or mec: specifies the color of the point outline
  - markerfacecolor or mfc: specifies the color of the point filling
  - linestyle or ls: specifies the style of the line (dotted ‘,’, dashed ‘:’, etc.)
  - color or c: specifies the color of the line
  - linewidth or lw: specifies the width of the line (takes a numerical value)
- Or by using string format that includes all properties.
  - By giving the values directly ’marker linestyle color’ (Example: ‘*:r’)
  - P.S. when using string format, unmentioned values will not be drawn in the graph.

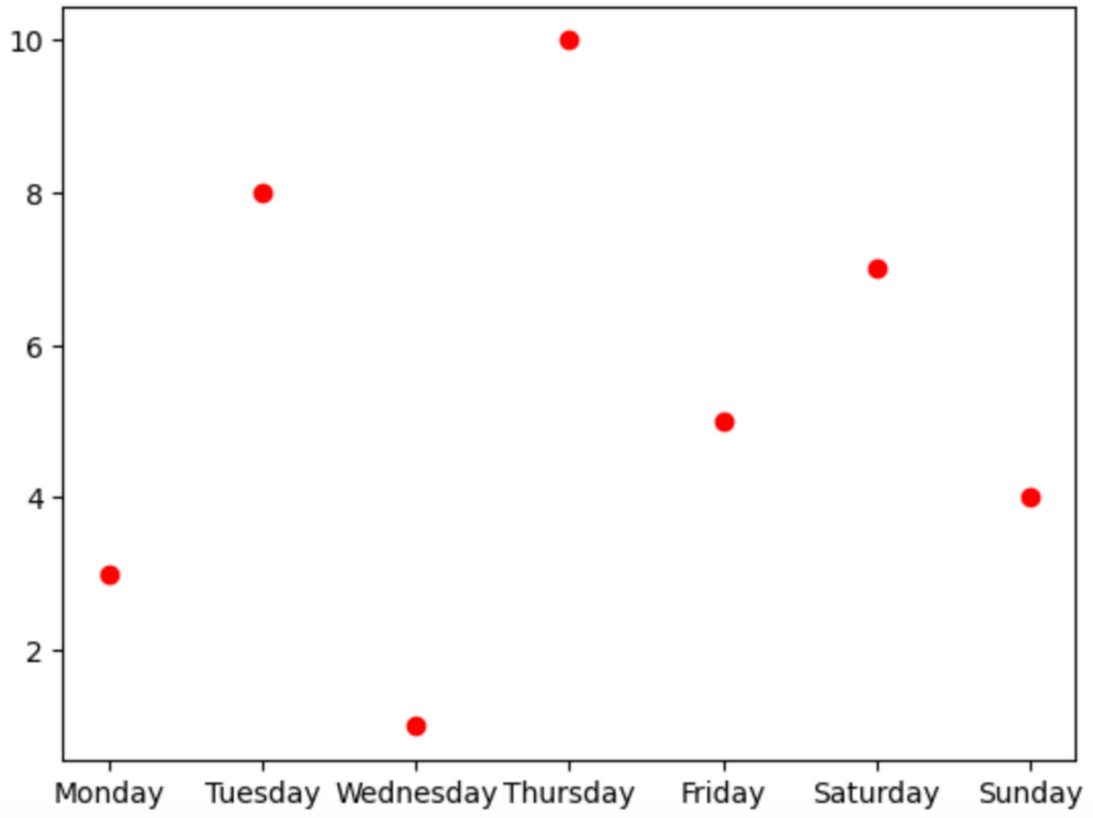
Examples: string format

```python
import matplotlib.pyplot as plt
import numpy as np

xpoints=np.array(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"])
ypoints = np.array([3, 8, 1, 10, 5, 7, 4])

plt.plot(xpoints,ypoints, 'o:r')
plt.show()
```



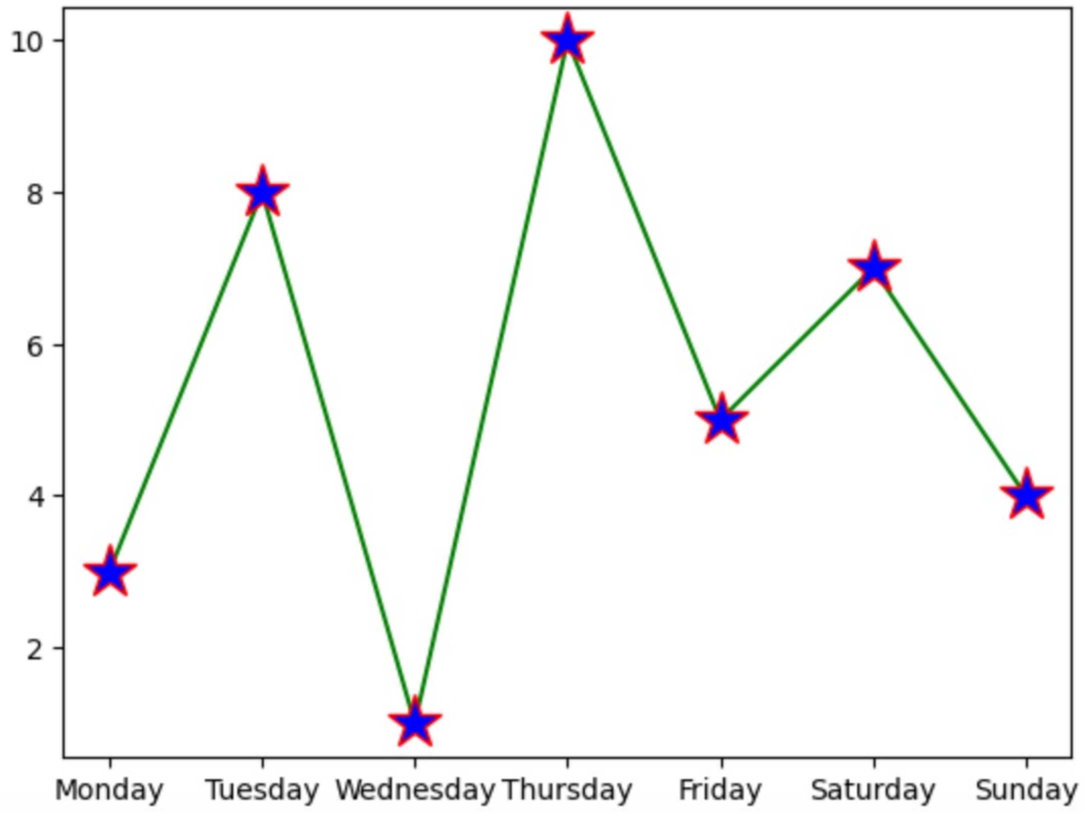
Examples: string format

```python
import matplotlib.pyplot as plt
import numpy as np

xpoints=np.array(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"])
ypoints = np.array([3, 8, 1, 10, 5, 7, 4])

plt.plot(xpoints,ypoints, 'or')
plt.show()
```

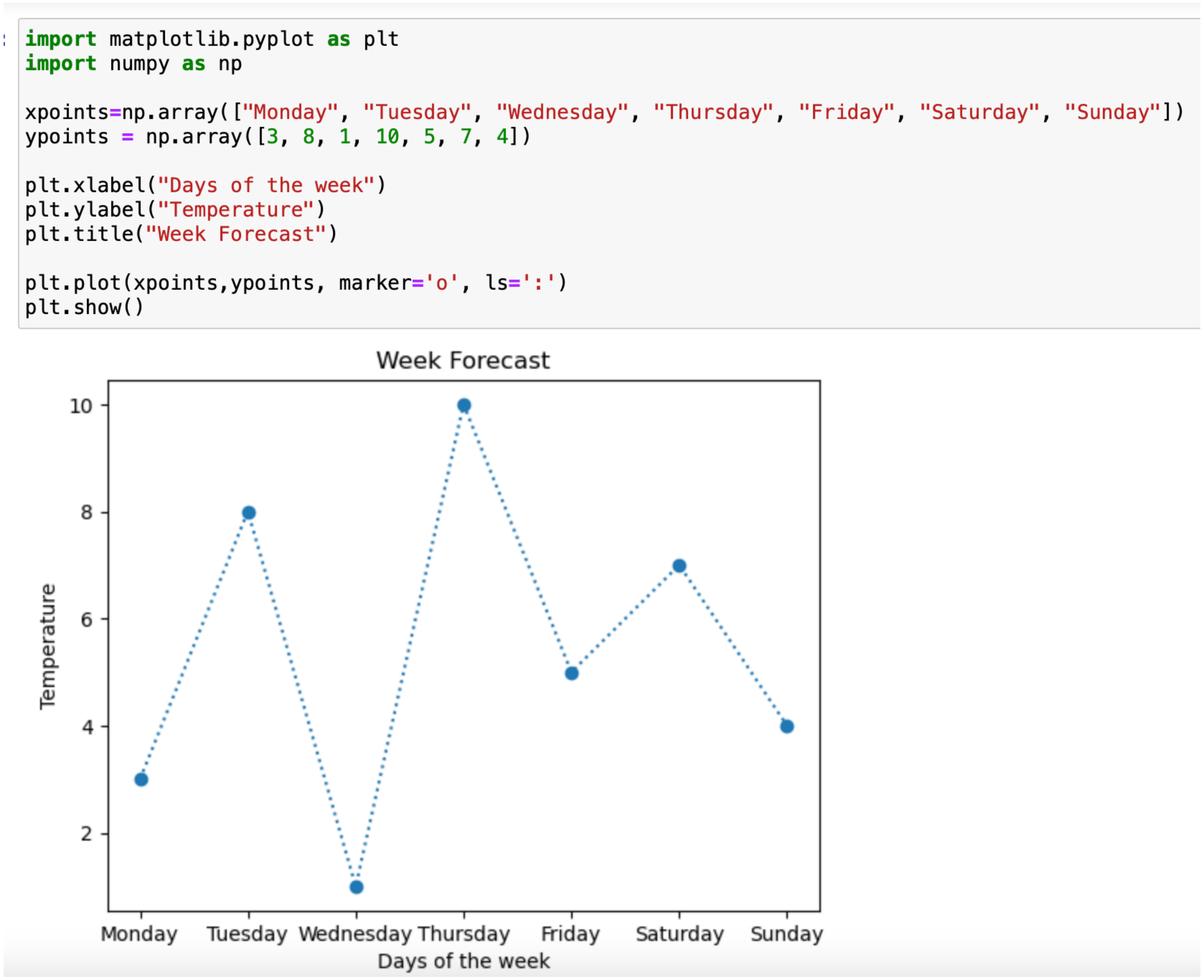


Examples: Using keywords

```python
import matplotlib.pyplot as plt
import numpy as np

xpoints=np.array(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"])
ypoints = np.array([3, 8, 1, 10, 5, 7, 4])

plt.plot(xpoints,ypoints, marker = '*', ms = 20, mec = 'r', mfc = 'b', color='g')
plt.show()
```



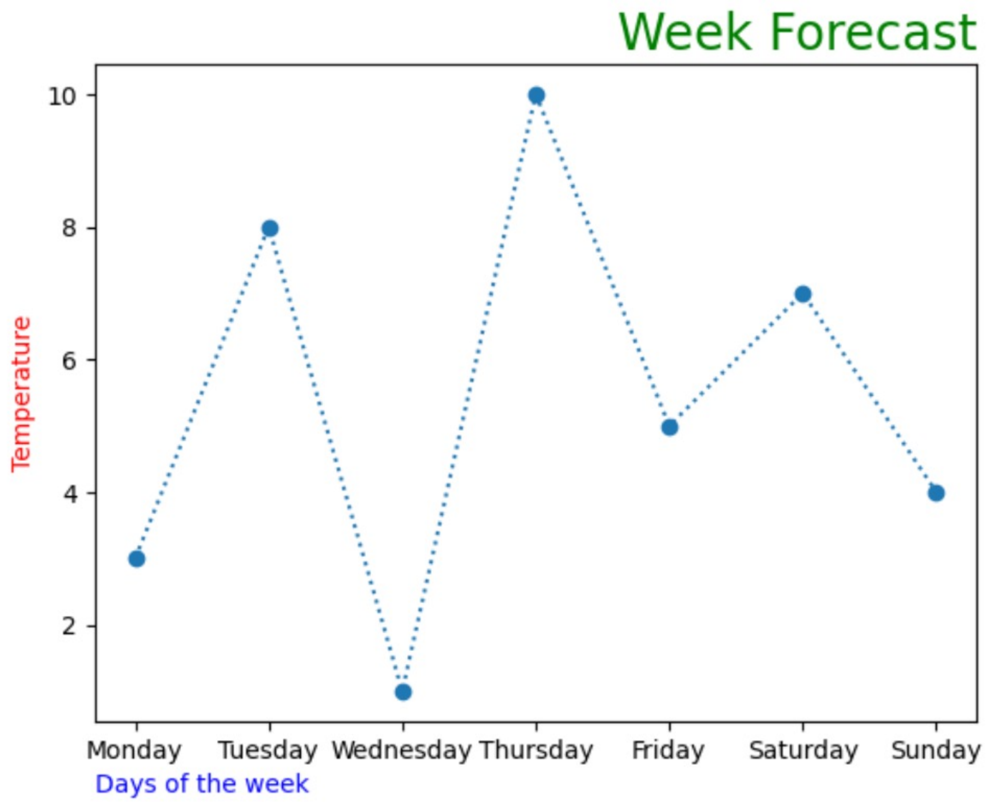
Labels and title



```
import matplotlib.pyplot as plt
import numpy as np

xpoints=np.array(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"])
ypoints = np.array([3, 8, 1, 10, 5, 7, 4])

plt.xlabel("Days of the week")
plt.ylabel("Temperature")
plt.title("Week Forecast")

plt.plot(xpoints,ypoints, marker='o', ls=':')
plt.show()
```

Labels and title properties

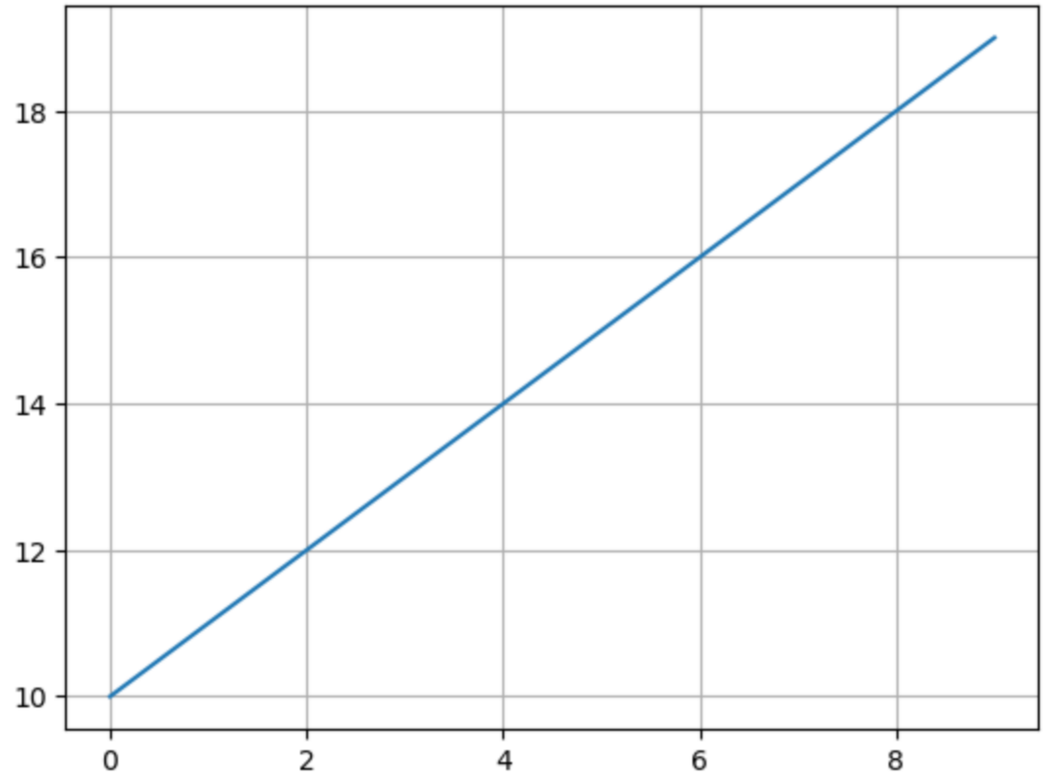
```
import matplotlib.pyplot as plt
import numpy as np

xpoints=np.array(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"])
ypoints = np.array([3, 8, 1, 10, 5, 7, 4])

f1={'color':'blue','size':10}
plt.xlabel("Days of the week", loc='left', fontdict=f1)
plt.ylabel("Temperature", fontdict={'color':'red','size':10})
plt.title("Week Forecast", {'color':'green','size':20}, loc='right')

plt.plot(xpoints,ypoints, marker='o', ls=':')
plt.show()
```



grid

- The grid property simply adds grid lines to the plot

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(10)
y = np.arange(10,20)

plt.plot(x, y)
plt.grid()
plt.show()
```

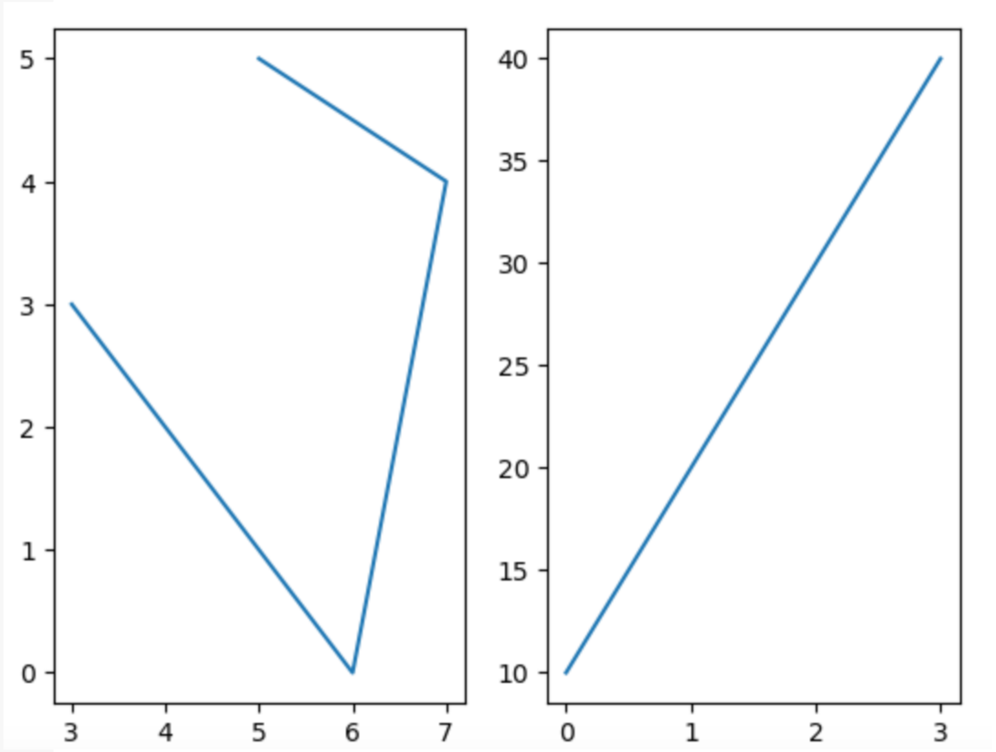


Subplot

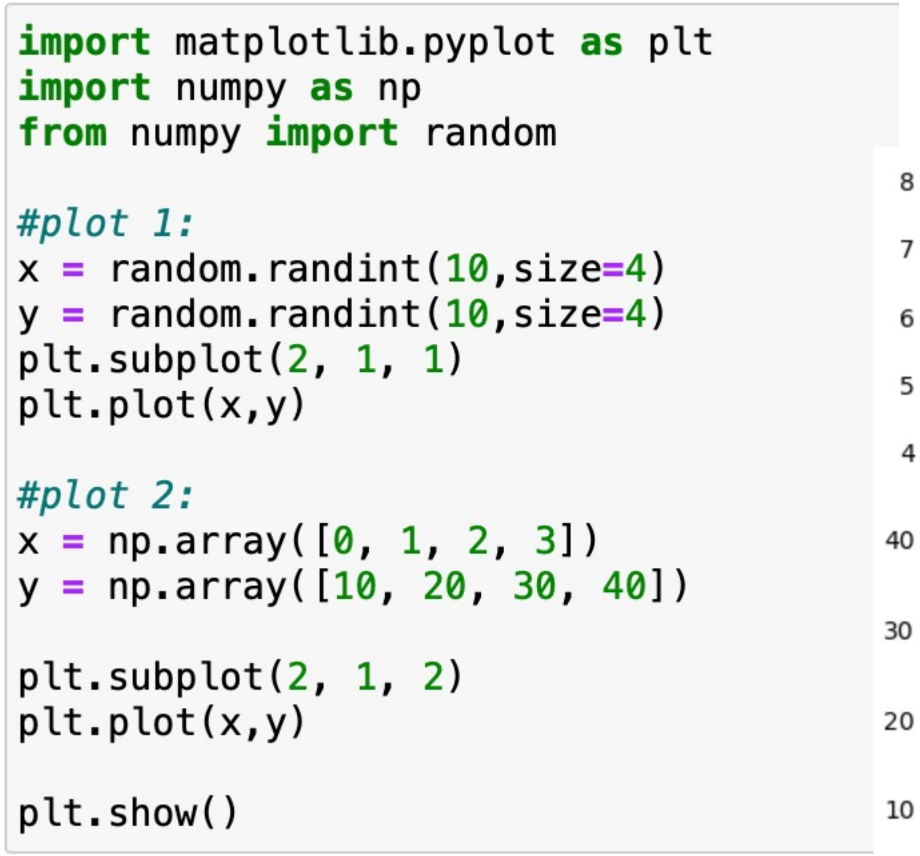
- Subplot enables plotting multiple plots in the same figure
- plt.subplot(r,c,i) where:
  - r: is the number of rows in the figure
  - c: is the number of columns in the figure
  - i: is the index or the position of the current plot with respect to the entire figure
- plt.sublot() should be called before plt.plot() to specifiy the position of the plot.

Example

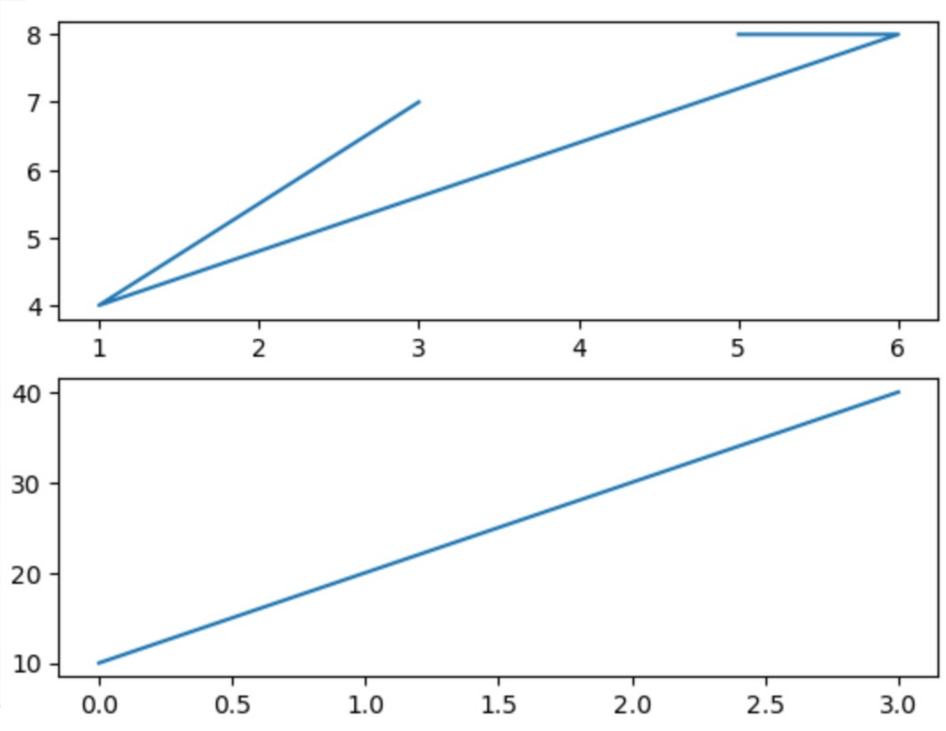
```
import matplotlib.pyplot as plt
import numpy as np
from numpy import random

#plot 1:
x = random.randint(10,size=4)
y = random.randint(10,size=4)
plt.subplot(1, 2, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)

plt.show()
```



Example



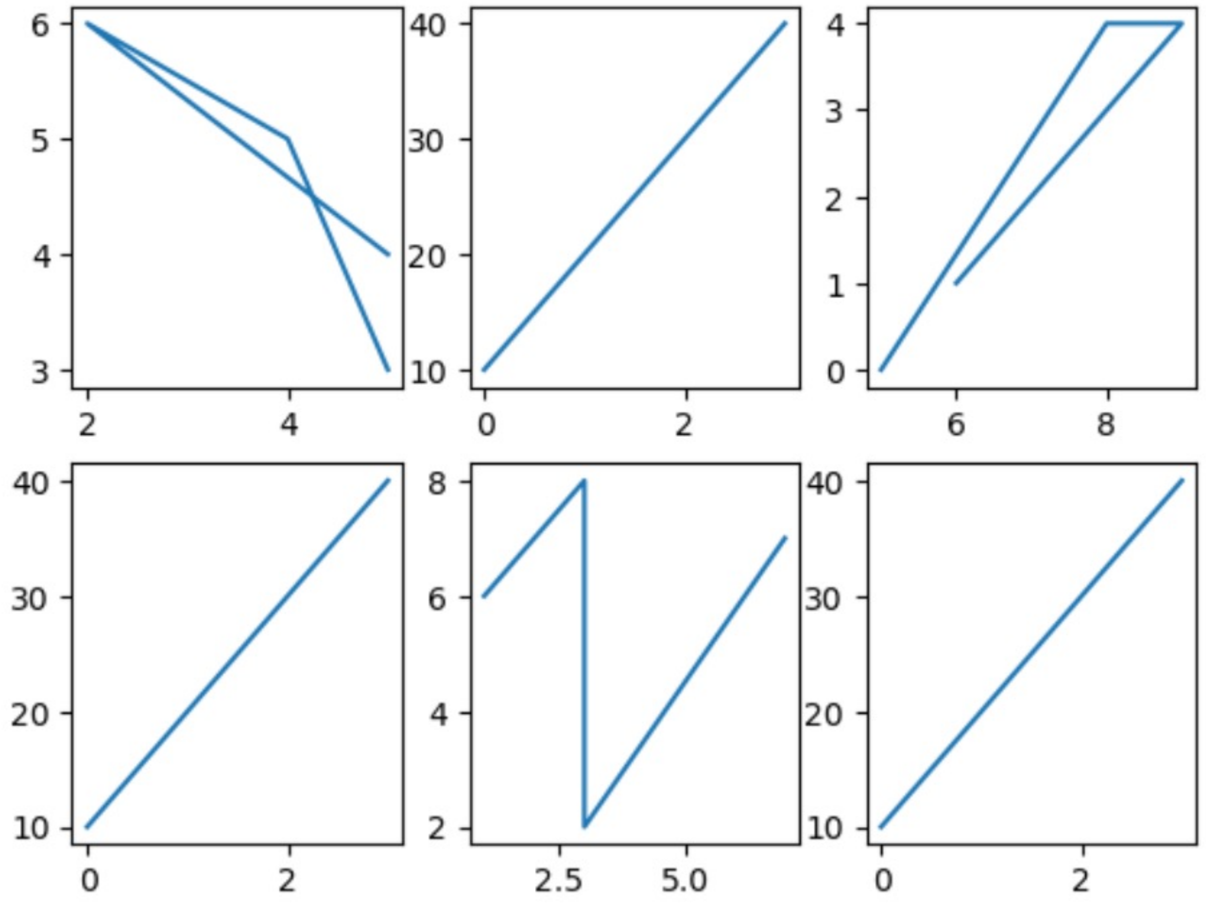
```
import matplotlib.pyplot as plt
import numpy as np
from numpy import random

#plot 1:
x = random.randint(10,size=4)
y = random.randint(10,size=4)
plt.subplot(2, 1, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 1, 2)
plt.plot(x,y)

plt.show()
```



- How many plots there will be with plt.sublot(2,3,i)?

Example

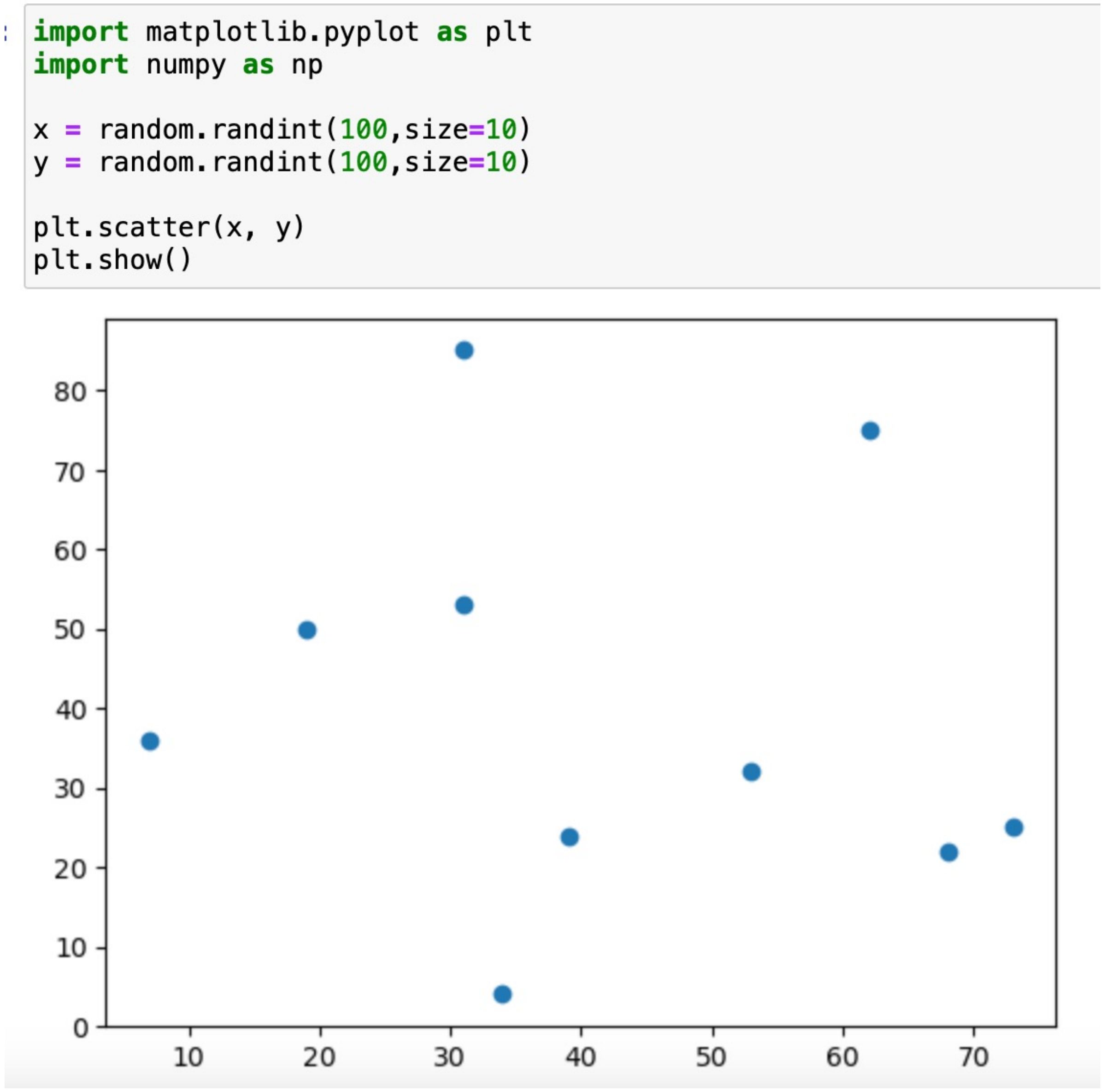
```python
import matplotlib.pyplot as plt
import numpy as np
from numpy import random

#plot 1:
x = random.randint(10,size=4)
y = random.randint(10,size=4)
plt.subplot(2, 3, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 2)
plt.plot(x,y)

#plot 3:
x = random.randint(10,size=4)
y = random.randint(10,size=4)
plt.subplot(2, 3, 3)
plt.plot(x,y)

#plot 4:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x,y)

#plot 5:
x = random.randint(10,size=4)
y = random.randint(10,size=4)
plt.subplot(2, 3, 5)
plt.plot(x,y)

#plot 6:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 6)
plt.plot(x,y)

plt.show()
```



Plt.plot() variations

- Pyplot offers many variations to the plot() method that offers different ways of data visualization.
  - plt.scatter(): Scatter is like plot without lines. Each data point is represented as a dot.
  - plt.bar(): bar creates a bar graph for the data points.
  - plt.barh(): hbar creates a horizontal bar graph for the data points
  - plt.hist(): hist creates a histogram for the data points
  - plt.pie(): pie creates a pie chart for the data points
- Each of them has certain properties that can be modified for various styles.

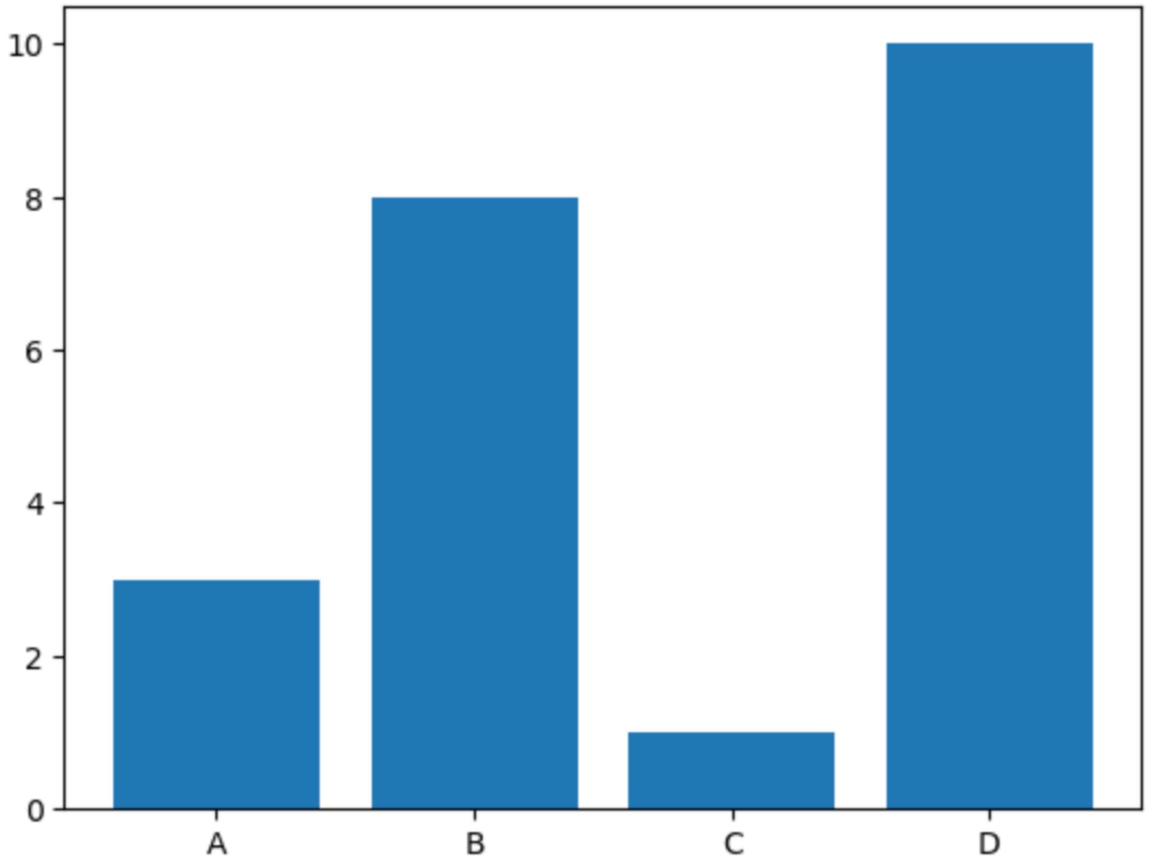
Scatter



```
import matplotlib.pyplot as plt
import numpy as np

x = random.randint(100,size=10)
y = random.randint(100,size=10)

plt.scatter(x, y)
plt.show()
```

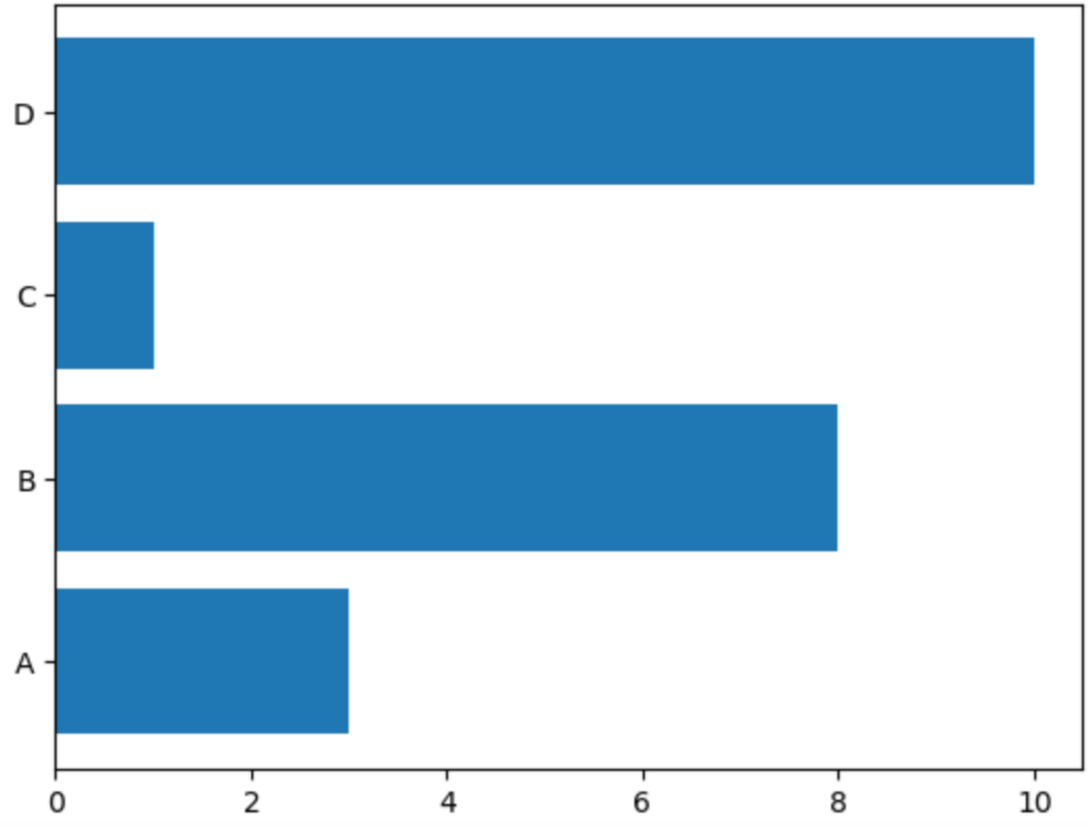
Bar chart

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x,y)
plt.show()
```



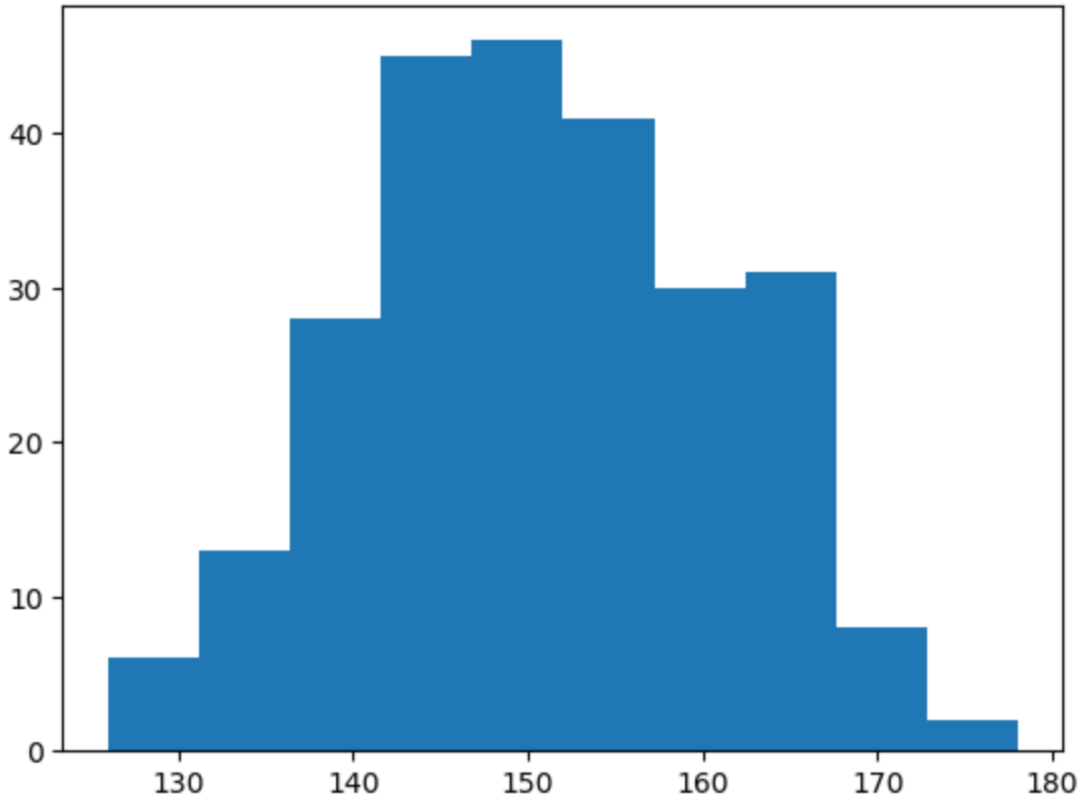
Horizontal Bar chart

```python
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x,y)
plt.show()
```

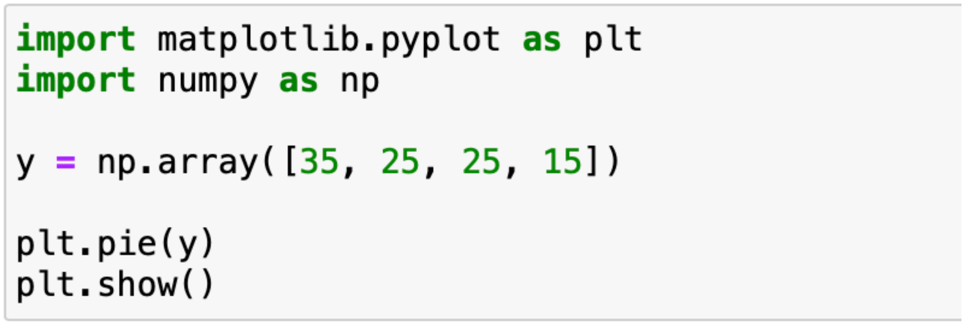


Histogram

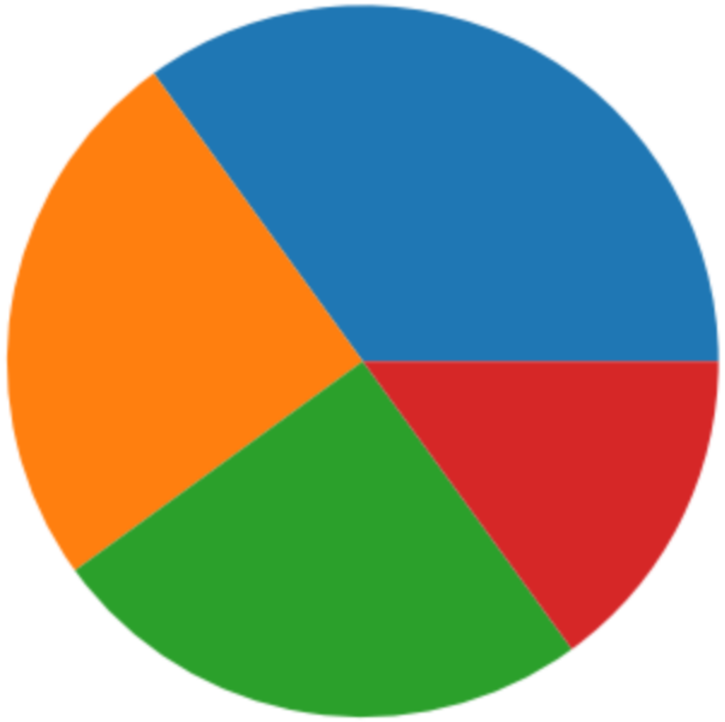
```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.normal(150, 10, 250)
#Creates normal distribution with mean=150, standard deviation=10 and 250 data points

plt.hist(x)
plt.show()
```



Pie chart



```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])

plt.pie(y)
plt.show()
```



References

- https://www.w3schools.com/python/matplotlib_intro.asp

QUESTIONS?