Python Modules

NEE2106 Computer Programming for Electrical Engineers

Prepared by: Dr. Rui Li (rui.li@vu.edu.au)

VICTORIA UNIVERSITY
MELBOURNE AUSTRALIA

< Contents >

- Python Modules
- Commonly used modules
  - Matplotlib
  - Numpy

< Python Module >

A Python module is a file containing Python definitions and statements.

- **Organise** code by grouping related functions and variables.
- Modules are **reused** in different programs.
- Separate **namespace** to prevent conflict with other parts of code.
- **Built-in modules**: come with Python installation such as “math”, “random”, “os”, “datetime”, etc.
- **Third-party modules**: available through Python Package Index (PyPI), installed using tools like “pip”

< Built-in Modules >

Import a module vs. Import certain functions in a module

```
import math       # import the whole module
x = math.sqrt(9) # call a function under this module
print(x)

from math import sqrt,factorial # import certain functions under a module
x = sqrt(9)  # use the function name directly
y = factorial(3)
print("x=",x)
print("y=",y)
```

Be careful – the name of objects imported should not conflict with other objects in the main program

It is possible to rename your module or package to shorten long names

```
1  from math import sqrt as sq, factorial as fa
2  x = sq(9)  # use a shorter function name
3  y = fa(3)
4  print("x=",x)
5  print("y=",y)
```

```
# "matplotlib" is a Package
# "pyplot" is a Module in this Package
# rename the Package.Module structure using "plt"
import matplotlib.pyplot as plt
x = [1,2,3]
y = [10,15,20]
plt.plot(x,y) # create the plot
plt.show()    # display the plot
```

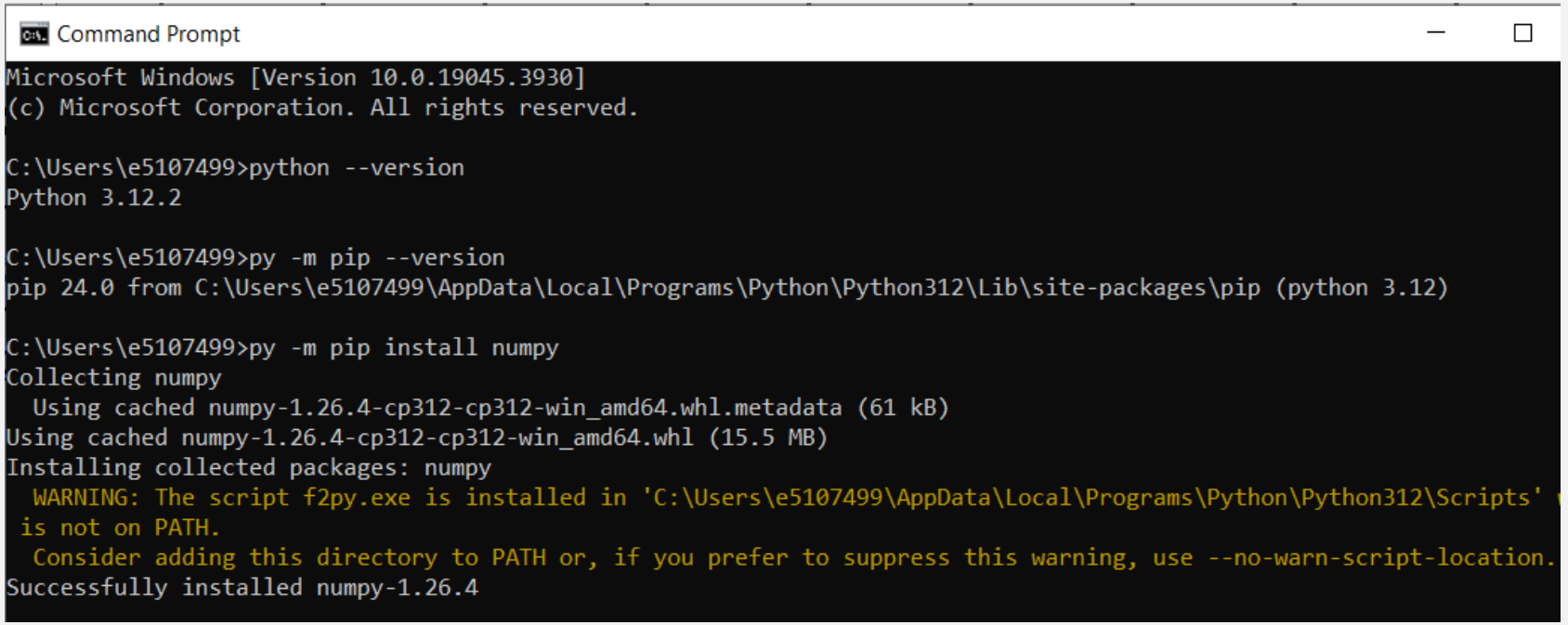
```
# You may directly define a function name
# "matplotlib.pyplot" is a Module
# "plot" is a function
from matplotlib.pyplot import plot as plt
x = [1,2,3]
y = [10,15,20]
plt(x,y)
```

< Third-party Modules >

There’re thousands of third-party modules developed by Python enthusiasts. To install:

- If you use default Python 3.12 app –

1. Install the “pip” Python Package Installer
2. Download the .py module or package from https://pypi.org/
3. Type “**py –m pip install module_name**” in system command window (type “cmd” at Start Menu to open)



< Third-party Modules >

- If you use Anaconda –

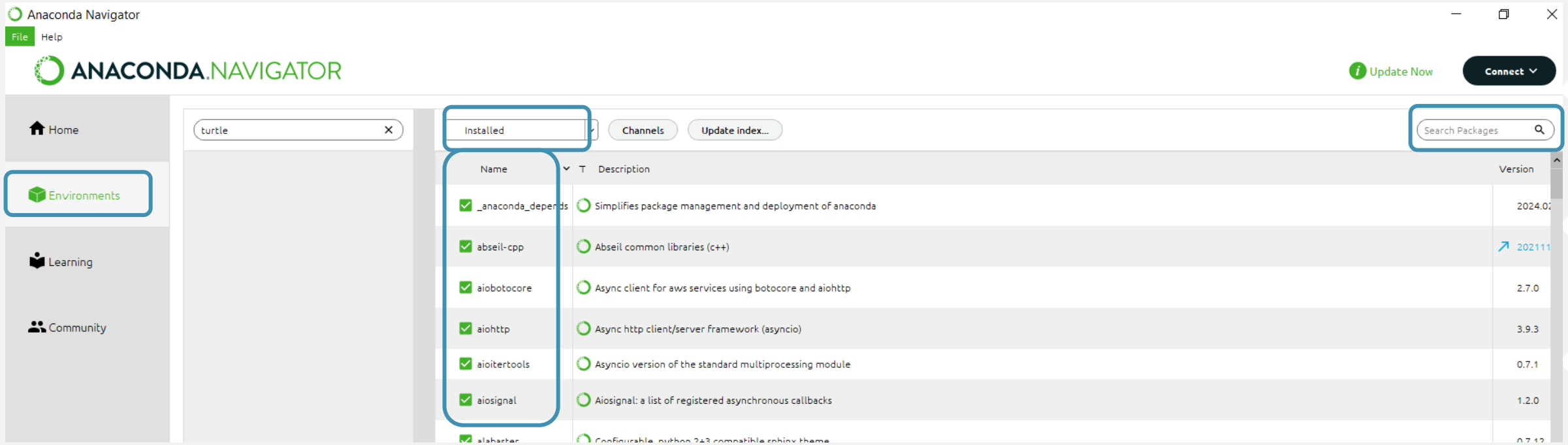
1. Open Anaconda Prompt
2. Type “**conda install module_name**” in Anaconda Prompt

```
(base) C:\Users\e5107499>python --version
Python 3.11.7

(base) C:\Users\e5107499>conda install scikit-learn
```

```
Downloading and Extracting Packages:

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```



< Exercise>

Install (or check if they're installed) the following packages or modules to your Anaconda Environment:

- Numpy
- Scipy
- Matplotlib
- Pandas

Confirm the successful installation in Anaconda Navigator Environments.

< Contents >

- Python Modules
- Commonly used modules
  - Matplotlib
  - Numpy

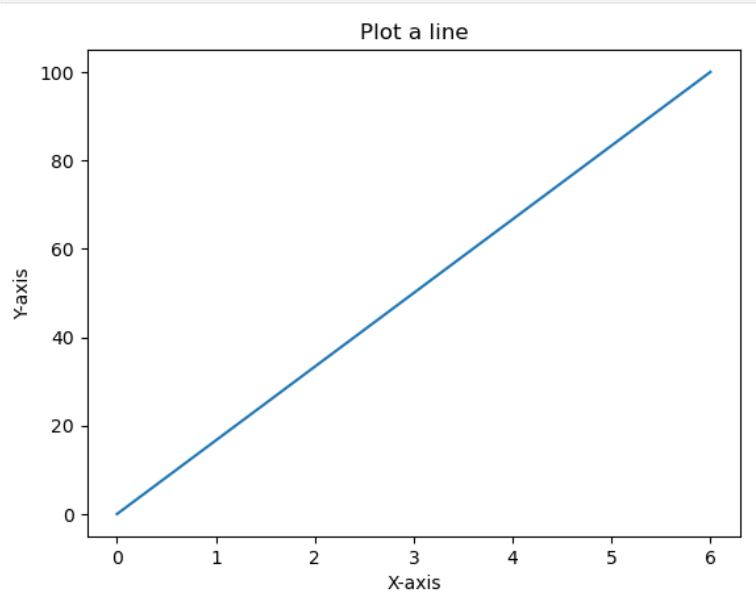
< Matplotlib >

- Matlab Plot Library (simulate Matlab data visualisation environment)
- Primary visualization tool for scientific graphics in Python
- Most of **matplotlib** utilities lie under the **pyplot** module, which can be imported using:

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

```
import matplotlib.pyplot as plt

x = [0,6]     # define x values
y = [0,100]   # define y values
plt.plot(x,y) # plot a line from (0,0) to (6,100)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Plot a line')
plt.show()    # show the graph
```



< Format the plot >

- Format the marker type, marker size, line type, and color using:

```
plt.plot(x, y, ‘r--o’)  # red, dashed line, circular marker
# alternatively
plt.plot(x, y, color=‘red', linestyle=‘--', marker='o', markersize=8, linewidth=10)
```

- Format axes and labels:

```
plt.xlabel('X-axis', fontsize=12, color='blue', fontstyle='italic',fontweight='bold’)
```

- Gridline and Legend:

```
plt.grid(True, linestyle='--', linewidth=0.5, color='gray')
plt.legend(['Data'], loc='upper right', fontsize=10, frameon=True)
```

Marker	Description
'o'	Circle
'*'	Star
'.'	Point
'x'	X
'X'	X (filled)
'+'	Plus
'P'	Plus (filled)
's'	Square
'D'	Diamond

Line Syntax	Description
‘-'	Solid Line
‘:'	Dotted Line
‘--'	Dashed Line
‘-.'	Dashed/ Dotted Line

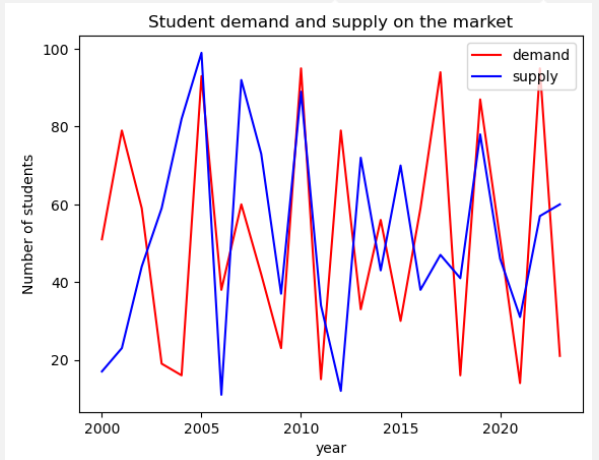
Color Syntax	Description
‘r'	Red
‘g'	Green
‘b'	Blue
‘c'	Cyan
‘m’	Magenta
‘y’	Yellow
‘k’	Black
‘w’	White

< Multiple plots>

- **Multiple plots on the same figure**

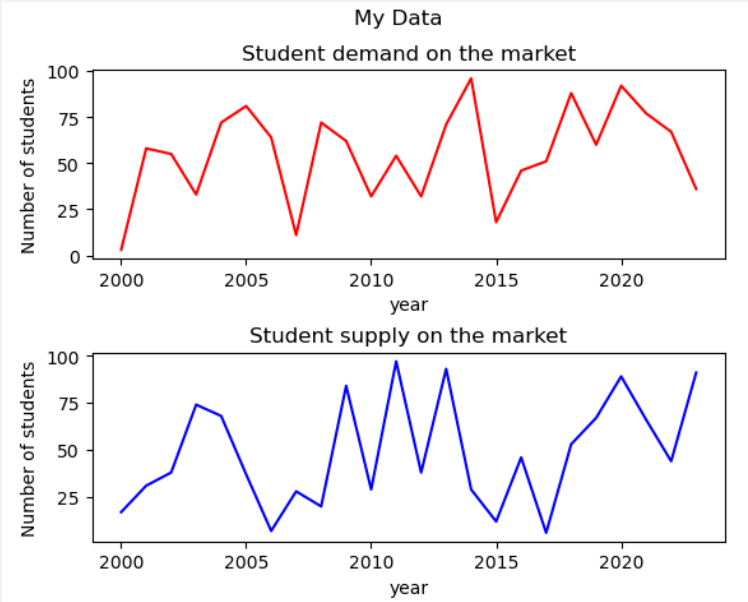
```
import random
import numpy as np

yr = list(range(2000,2024))
demand = np.random.randint(1, 101, size=len(yr))
supply = np.random.randint(1, 101, size=len(yr))
plt.plot(yr,demand,'r',yr,supply,'b')
plt.xlabel('year')
plt.ylabel('Number of students')
plt.title('Student demand and supply on the market')
plt.legend(['demand','supply'],loc='upper right')
plt.show()
```



- **Subplot**

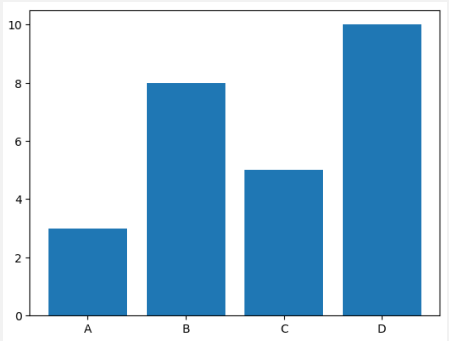
```
yr = list(range(2000,2024)) # x values
demand = np.random.randint(1, 101, size=len(yr)) # y of the 1st graph
supply = np.random.randint(1, 101, size=len(yr)) # y of the 2nd graph
plt.subplot(2,1,1) # 2 rows, 1 column, index 1
plt.plot(yr,demand,'r')
plt.xlabel('year')
plt.ylabel('Number of students')
plt.title('Student demand on the market')
plt.subplot(2,1,2) # 2 rows, 1 column, index 2
plt.subplots_adjust(hspace=0.5) # adjust the height space between two subplots
plt.plot(yr,supply,'b')
plt.xlabel('year')
plt.ylabel('Number of students')
plt.title('Student supply on the market')
plt.suptitle("My Data") # the overall super title
plt.show()
```



< Other plots>

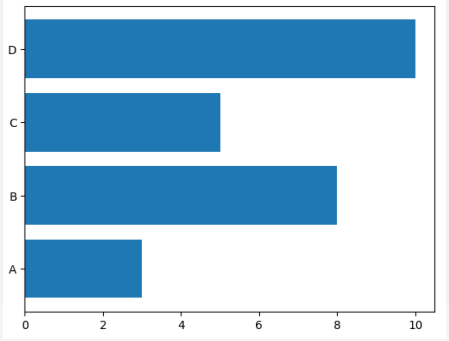
- Bar

```
# bar
x = ["A","B","C","D"]
y = [3,8,5,10]
plt.bar(x,y)
plt.show()
```



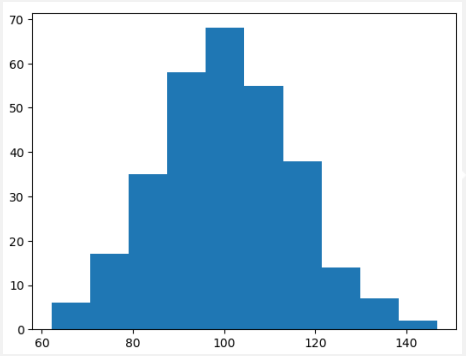
- Horizontal bar

```
# horizontal bar
x = ["A","B","C","D"]
y = [3,8,5,10]
plt.barh(x,y)
plt.show()
```



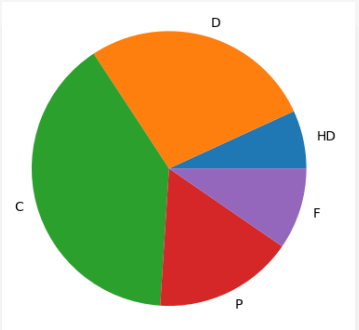
- Histogram

```
# histogram
mean_value = 100
std_value = 15
size=300 # number of elements in output array
data = np.random.normal(mean_value,std_value,size) # create a normal dist.
plt.hist(data) # plot histogram
plt.show()
```



- Pie

```
# pie
portion = [5,20,29,12,7]
portion_label = ["HD","D","C","P","F"]
plt.pie(portion,labels=portion_label)
plt.show()
```



< Exercise>

1. Create a bar chart that shows the number of books read by a group of 5 students in a month.

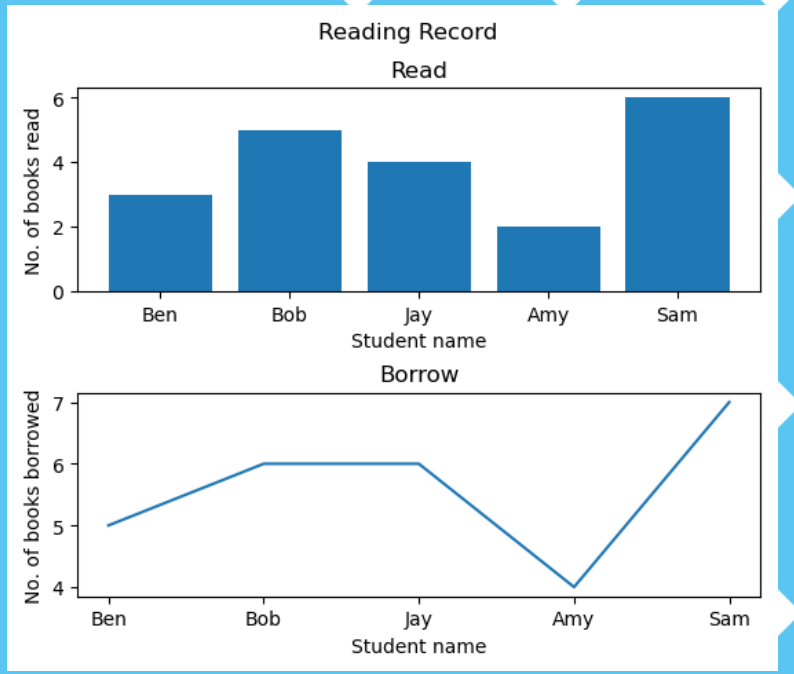
2. Under the bar char, create another line graph to show the number of books borrowed in the month. Use circular markers to highlight each data point.

3. Properly label each graph and axes. Create a super title named “Reading Record”.

Student	Read	Borrowed
Ben	3	5
Bob	5	6
Jay	4	6
Amy	2	4
Sam	6	7

```
import matplotlib.pyplot as plt

student = ["Ben","Bob","Jay","Amy","Sam"]
read = [3,5,4,2,6]
borrow = [5,6,6,4,7]
plt.subplot(2,1,1) # bar chart
plt.bar(student,read)
plt.xlabel("Student name")
plt.ylabel("No. of books read")
plt.title("Read")
plt.subplot(2,1,2) # line graph
plt.subplots_adjust(hspace=0.5) # adjust the space
plt.plot(student,borrow)
plt.xlabel("Student name")
plt.ylabel("No. of books borrowed")
plt.title("Borrow")
plt.suptitle("Reading Record")
plt.show()
```



< Contents >

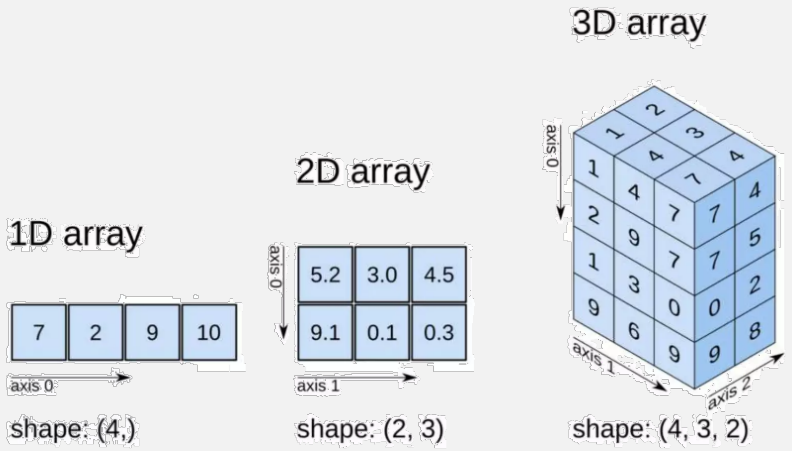
- Python Modules
- Commonly used modules - Numpy
  - Matplotlib
  - Numpy

< Numpy >

- Numpy – Numerical Python
- For high-performance computing and data analysis, i.e. better efficiency for large arrays

- One of the key features of NumPy is its N-Dimensional Array object (**ndarray**), which is a fast (10-100 times faster as it uses less memory space), flexible container for large datasets in Python

- Array: a data structure that contains elements of the same type
  - Vectors (1-dimensional tensor)
  - Matries (2-dimensional tensor)
  - Tensors (an array of any dimension, normally refer to high-order tensors)
  - Images (3-dimensional tensors)



< Numpy - define an array >

```
#%% array
import numpy as np
a = np.array([1, 2, 3]) # 1D vector
b = np.array([[1,2,3],[4,5,6]]) # 2D matrix
c = np.zeros(2) # 2 zeros in floating-point format
d = np.ones(3) # 3 ones in floating-point format
e = np.empty(4) # an empty array to contain 4 elements (with random initial values)
f = np.arange(2,9,3) # from 2 to 9, with a step of 3
g = np.linspace(2,9,3) # from 2 to 9, create 3 elements that are spaced evenly
h = np.ones(2,dtype=np.int8) # specify data type = 8-bit integer
```

- **np.array()** – use [ ] to list all the elements
- **np.zeros()** – all zeros
- **np.ones()** – all ones
- **np.empty()** – empty array
- **np.arange()** – arrange between (start, end, step)
- **np.linspace()** – evenly spaced between (start, end, # elements)
- **dtype** – specify data type

Nam	Type	Size	
a	Array of int32	(3,)	[1 2 3]
b	Array of int32	(2, 3)	[[1 2 3] [4 5 6]]
c	Array of float64	(2,)	[0. 0.]
d	Array of float64	(3,)	[1. 1. 1.]
e	Array of float64	(4,)	[0. 0. 6. 4.]
f	Array of int32	(3,)	[2 5 8]
g	Array of float64	(3,)	[2. 5.5 9.]
h	Array of int8	(2,)	[1 1]

< Numpy – array IO >

- **Save to Text file:**

```
#%% array IO
import numpy as np
# Create a 2D array
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# Save the array to a text file
np.savetxt('ArrayFile.txt', a) # by default, delimiter = space , format = f.p.
np.savetxt('ArrayFile.txt', a, delimiter=",",fmt="%d")
```

```
1.000000000000000000e+00 2.000000000000000000e+00 3.000000000000000000e+00
4.000000000000000000e+00 5.000000000000000000e+00 6.000000000000000000e+00
7.000000000000000000e+00 8.000000000000000000e+00 9.000000000000000000e+00
```

```
1,2,3
4,5,6
7,8,9
```

- **Save to Excel file:**

```
#%% arry to Excel
import pandas as pd
import numpy as np
# Create a 2D array
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# Create a dataframe from the array
df = pd.DataFrame(a)

# Save dataframe (df) to Excel
df.to_excel("ArrayFile.xlsx")  # data and row/ column numbers
df.to_excel("ArrayFile.xlsx",index=False,header=False)
# Set index=False to exclude row numbers, header=False to exclude headers
```

	A	B	C	D
1		**0**	**1**	**2**
2	**0**	1	2	3
3	**1**	4	5	6
4	**2**	7	8	9

	A	B	C
1	1	2	3
2	4	5	6
3	7	8	9

< Numpy – array IO >

- **Load from Text file:**

```
#%% load from txt
import numpy as np
data = np.loadtxt("ArrayFile.txt",delimiter=',', skiprows=1, dtype=int)
```

Nam	Type	Size	Value
data	Array of int32	(2, 3)	[[4 5 6] [7 8 9]]

- **Load from Excel file:**

```
#%% load from .xlsx
import pandas as pd
df = pd.read_excel("ArrayFile.xlsx",header=None) # Load Excel file into a DataFrame, data has no header
data = df.values # Convert DataFrame to Array
```

df - DataFrame

Index	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

data - NumPy object array

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

< Numpy – array inspection and mathematics >

➢ Inspection

```python
#%% array inspection
x = np.array([[1,3,5],[2,4,6]])
print("array dimension: ", x.shape) # rows, columns
print("array size: ", x.size) # No. of elements
print("array length: ", len(x)) # length (No. of rows)
print("array No. of axes: ", x.ndim) # No. of array dimension, 1D -> vector, 2D -> matrix
print("array data type: ", x.dtype) # data type
print("array in f.p. format: ",x.astype(float)) # convert to other data type
```

```
array dimension:  (2, 3)
array size:  6
array length:  2
array No. of axes:  2
array data type:  int32
array in f.p. format:  [[1. 3. 5.]
 [2. 4. 6.]]
```

➢ Mathematics

```python
#%% array manipulation
import numpy as np
x = np.array([[1,3,5],[2,4,6]])
y = np.array([[10,30,50],[20,40,60]])
a = x+y # or np.add(x,y)
b = x-y # or np.subtract(x,y)
c = x*y # or np.multiplt(x,y), element-wise
d = x/y # or np.divide(x,y), element-wise
e = np.dot(x[0,:],y[0,:]) # matrix multiplication or dot product -> 1x10+3*30+5*50

f = np.exp(x)
g = np.sqrt(x)
h = np.sin(x)
i = np.cos(x)
j = np.log(x)

k = x.sum() # sum all elements
l = x.sum(axis=0) # sum over rows
m = x.sum(axis=1) # sum over columns, result a 1D vector
n = x.sum(axis=1, keepdims=True) # sum over columns, result the true positions
```

➢ Statistics

```python
#%% statistics
import numpy as np
data = np.array([12, 15, 17, 20, 22, 25, 30, 32, 35])

# Calculate minimum and maximum
minimum = np.min(data)
maximum = np.max(data)
# Calculate quartiles
q1 = np.percentile(data, 25)
median = np.percentile(data, 50) # or np.median(data)
q3 = np.percentile(data, 75)
# mean and std
mean = np.mean(data)
std = np.std(data)
```

np. argmin()
to find the index

< Numpy – array manipulation>

- **Shape your array:**

Function	Overview
new_array=array.**reshape**(r,c)	changes the shape of an array without changing its data
array.**resize**(r,c)	changes the shape and size of the array itself
new_array=array.**ravel**()	returns a flattened view of the array
new_array=array.**flatten**()	returns a copy of one-dimentional array (separate from origin)

```
#%% array manipulation
import numpy as np
# ARRAY SHAPE
a = np.array([1,2,3,4,5,6]) # 1D vector with 6 elements
b = a.reshape(3,2) # reshape to 3 rows x 2 columns
c = a.reshape(2,-1) # reshape to 2 rows, auto calc No. of columns
d = c.ravel() # flatten multi-dim array to 1D vector (no copy)
e = c.flatten() # flatten multi-dim array to 1D vector (copy version)

f = np.array([1,2,3,4,5,6])
f.resize(3,3) # resize to 3r x 3c, add 0s to fill up positions (modify the origin)

g = np.arange(10).reshape(5,2)
h = g.T  # transpose of g, or g.transpose()
```

What's in "g"?

0 1
2 3
4 5
6 7
8 9

a	Array of int32	(6,)	[1 2 3 4 5 6]
b	Array of int32	(3, 2)	[[1 2] [3 4] [5 6]]
c	Array of int32	(2, 3)	[[1 2 3] [4 5 6]]
d	Array of int32	(6,)	[1 2 3 4 5 6]
e	Array of int32	(6,)	[1 2 3 4 5 6]
f	Array of int32	(3, 3)	[[1 2 3] [4 5 6] [0 0 0]]
g	Array of int32	(5, 2)	[[0 1] [2 3] [4 5]]
g_delete_column	Array of int32	(5, 1)	[[1] [3] [5]]
g_delete_row	Array of int32	(3, 2)	[[0 1] [4 5] [8 9]]
g_delete_vector	Array of int32	(8,)	[0 2 4 5 6 7 8 9]
h	Array of int32	(2, 5)	[[0 2 4 6 8] [1 3 5 7 9]]
i	Array of int32	(3,)	[1 2 3]
i_append	Array of int32	(6,)	[1 2 3 4 5 6]
i_hstack	Array of int32	(6,)	[1 2 3 4 5 6]
i_vstack	Array of int32	(2, 3)	[[1 2 3] [4 5 6]]
j	Array of int32	(5,)	[1 0 0 2 3]
k	Array of int32	(3,)	[1 0 3]
lc	Array of int32	(2, 4)	[[1 2 5 6] [3 4 7 8]]
lr	Array of int32	(4, 2)	[[1 2] [3 4] [5 6]]
m	Array of int32	(2, 2)	[[1 2] [3 4]]
n	Array of int32	(2, 2)	[[5 6] [7 8]]

< Numpy – array manipulation>

➢ **Add or remove:**

Function	Overview
np.**append**(array, new_array)	appends values to the end of an array
np.**vstack**((array, new_array)), np.**hstack**((array, new_array))	stacks arrays vertically (row-wise) or horizontally (column-wise) ** 2 sets of () needed
np.**insert**(array, index, new_array)	inserts values into an array at specified index
np.**delete**(array, [indices])	removes elements at specified indices from an array
np.**concatenate**((array1,array2))	joins two or more arrays along an existing axis Default axis=0, row-wise. Change to axis = 1 for column-wise ** 2 sets of () needed
np.**vsplit**(), np.**hsplit**()	splits an array into multiple sub-arrays vertically (row-wise), or horizontally (column-wise)

Note:
append, insert, delete functions flatten the input array if the axis parameter is not specified.
To apply these function at a certain axis, add “axis=0” (row-wise) or “axis=1” (column-wise) in the function.

< Numpy – array manipulation>

- **Add or remove:**

```
# ADD / REMOVE
i = np.array([1, 2, 3])
i_append = np.append(i, [4, 5, 6]) # append to the end of array
i_vstack = np.vstack((i,np.array([4,5,6]))) # stack array vertically, the added rows are arrays
i_hstack = np.hstack((i,np.array([4,5,6]))) # stack array horizontally

j = np.insert(i,1,[0,0]) # insert [0,0] in front of index =1 of variable i
k = np.delete(j,[1,3]) # delete the values at index = [1,3] of j

g = np.arange(10).reshape(5,2) # 5r x 2c
g_delete_vector = np.delete(g,[1,3]) # 5x2 matrix is flattened, values at index = [1,3] are deleted
g_delete_row = np.delete(g,[1,3],axis=0) # delete rows (axis=0) at index = [1,3]
g_delete_column = np.delete(g,[0],axis=1) # delete columns (axis=1) at index = [0]

m = np.array([[1,2],[3,4]])
n = np.array([[5,6],[7,8]])
lr = np.concatenate((m,n)) # (default) axis=0, concatenate along rows
lc = np.concatenate((m,n),axis=1) # axis=1, concatenate along columns

o = np.arange(18).reshape(3,6) # 3r x 6c
ov = np.vsplit(o,3) # split the array vertically into 3 equal-height arrays
oh = np.hsplit(o,2) # split the array horizontally into 2 equal-width arrays
```

a	Array of int32	(6,)	[1 2 3 4 5 6]
b	Array of int32	(3, 2)	[[1 2] [3 4] [5 6]]
c	Array of int32	(2, 3)	[[1 2 3] [4 5 6]]
d	Array of int32	(6,)	[1 2 3 4 5 6]
e	Array of int32	(6,)	[1 2 3 4 5 6]
f	Array of int32	(3, 3)	[[1 2 3] [4 5 6] [7 8 9]]
g	Array of int32	(5, 2)	[[0 1] [2 3] [4 5]]
g_delete_column	Array of int32	(5, 1)	[[1] [3] [5]]
g_delete_row	Array of int32	(3, 2)	[[0 1] [4 5] [8 9]]
g_delete_vector	Array of int32	(8,)	[0 2 4 5 6 7 8 9]
h	Array of int32	(2, 5)	[[0 2 4 6 8] [1 3 5 7 9]]
i	Array of int32	(3,)	[1 2 3]
i_append	Array of int32	(6,)	[1 2 3 4 5 6]
i_hstack	Array of int32	(6,)	[1 2 3 4 5 6]
i_vstack	Array of int32	(2, 3)	[[1 2 3] [4 5 6]]
j	Array of int32	(5,)	[1 0 0 2 3]
k	Array of int32	(3,)	[1 0 3]
lc	Array of int32	(2, 4)	[[1 2 5 6] [3 4 7 8]]
lr	Array of int32	(4, 2)	[[1 2] [3 4] [5 6] [7 8]]
m	Array of int32	(2, 2)	[[1 2] [3 4]]
n	Array of int32	(2, 2)	[[5 6] [7 8]]

< Exercise>

1. Generate 50 random numbers ranged from 30 to 100 using **np.random.randint()** function to simulate students' scores of Class 1.
2. Reshape the 50 scores to represent 10 teams with 5 scores each (i.e. 10 rows x 5 columns).
3. Find the maximum score of each team and print results on screen.
4. Class 2 has 30 students. Generate these 30 scores using the same manner.
5. Properly reshape the Class 2 scores so it can be vertically stacked under Class 1 scores. Then, stack it.
6. Find the average score of each team and print results on screen.
7. Find the teams that received highest and lowest average scores. Make sure you find these two scores (**np.max()** or **np.min()** functions) and their indices (**np.argmax()** or **np.argmin()** functions).
8. Remove the highest and lowest, then print average scores of the rest teams.

```
The highest score of each team is:  76 72 63 95 97 87 87 87 89 95
The average score of each team is:  59.4 50.4 51.8 69.6 69.4 76.4 65.  79.4 66.8 64.2 66.2 68.2 65.8 72.4
 69.6 80.2
Max average score is 80.20 of team 15. Min average score is 50.40 of team 1.
After removing the highest and lowest, the rest average scores are:  [59.4 51.8 69.6 69.4 76.4 65.  79.4 66.8
64.2 66.2 68.2 65.8 72.4 69.6]
```

< Exercise>

```python
import numpy as np
# class 1 scores
class1 = np.random.randint(30,100,50).reshape(10,5)
# max score of each team
class1_team_max = np.max(class1,axis=1)
print("The highest score of each team is: ",str(class1_team_max)[1:-1])
# class 2 scores
class2 = np.random.randint(30,100,30).reshape(6,5)
# class 1 and 2 scores
class_tot = np.vstack((class1,class2))
# average score of each team of two classes
class_ave = np.average(class_tot,axis=1)
print("The average score of each team is: ",str(class_ave)[1:-1])
# max and min score team
class_ave_max = np.max((class_ave))
class_ave_max_idx = np.argmax((class_ave))
class_ave_min = np.min((class_ave))
class_ave_min_idx = np.argmin((class_ave))
print("Max average score is %.2f of team %d. Min average score is %.2f of team %d."
      %(class_ave_max,class_ave_max_idx,class_ave_min,class_ave_min_idx))
# remove these two teams
class_new = np.delete(class_ave,[class_ave_max_idx,class_ave_min_idx])
print("After removing the highest and lowest, the rest average scores are: ", class_new)
```

Convert to string and remove [ ] when display