FACULTY OF ENGINEERING
AT SRI RACHA
DEPARMENT OF COMPUTER ENGINEERING

03603351 วิทยาศาสตร์ข้อมูลเบื้องต้น

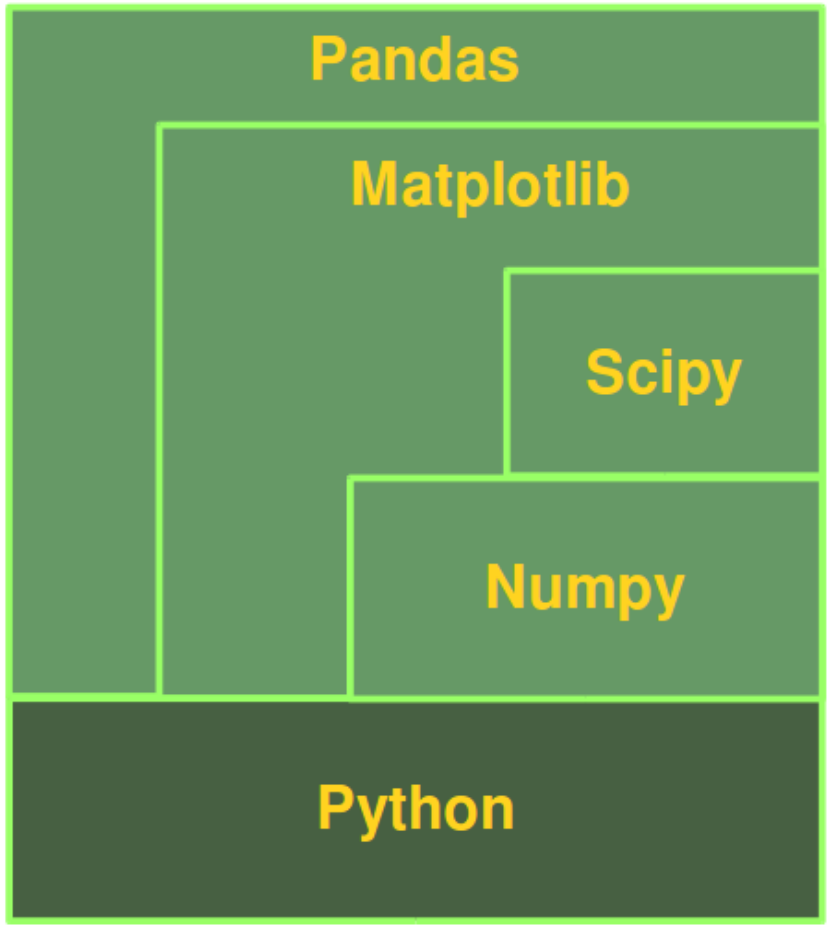
Introduction to Data Science

NumPy, SciPy, Matplotlib, Pandas

ผศ.ดร. กุลวดี สมบูรณ์วิวัฒน์

kulwadee@eng.src.ku.ac.th

ภาคการศึกษาที่ 1 ปี 2565



- **Pure Python** without any numerical modules couldn't be used for numerical tasks Matlab, R and other languages are designed for. If it comes to computational problem solving, it is of greatest importance to consider the performance of algorithms, both concerning speed and data usage

- **Numpy** is a module which provides the basic data structures, implementing/manipulating multi-dimensional arrays and matrices.

- **Scipy** extends the capabilities of NumPy with useful functions for minimization, regression, Fourier transformation, and many others.

- **Matplotlib** is a plotting library for the Python

- **Pandas** is based on Numpy, Scipy, Matplotlib
  - Provides capabilities for data manipulation and analysis, focusing on **numeric tables and time series.**
  - **Pandas** comes from "Panel Data"

NumPy

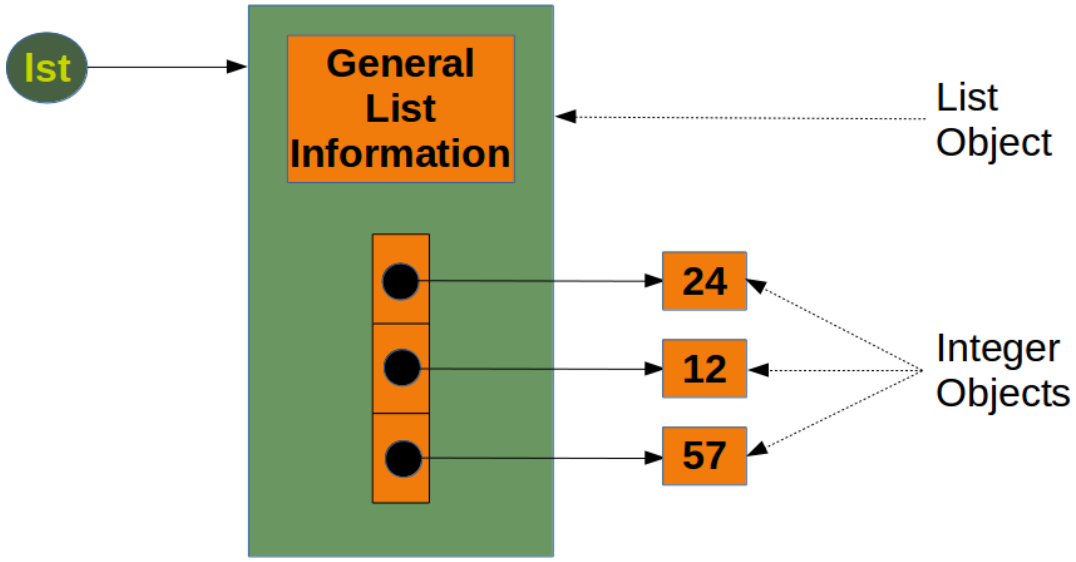
- Written mostly in C => precompiled mathematical and numerical functions guarantee great execution speed.

- Powerful data structures
  - Multi-dimensional arrays and matrices
  - Vectorized Computation

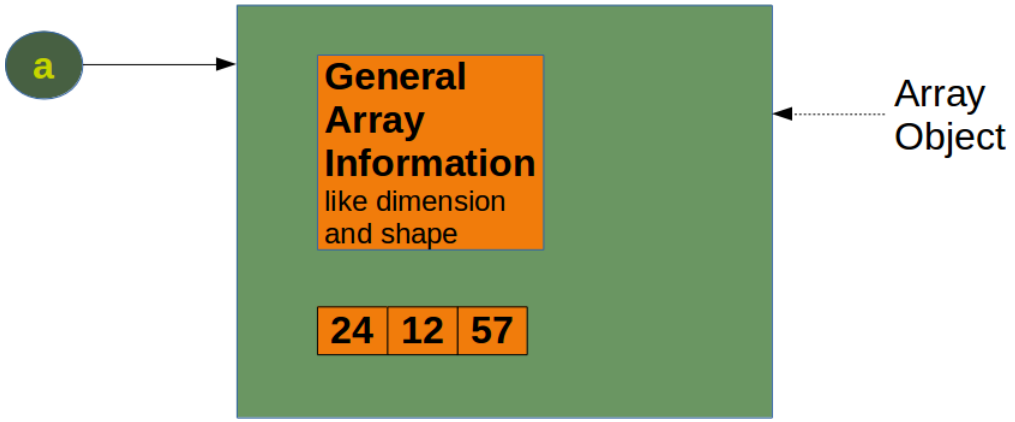
```
In [8]: import numpy as np

In [9]: celcius = [20.1, 20.8, 21.9, 22.5, 22.7, 22.3, 21.8, 21.2, 20.9, 20.1]

In [10]: # Python List Comprehension

In [11]: farenheit = [ c*9/5 + 32 for c in celcius ]

In [12]: print(farenheit)
[68.18, 69.44, 71.42, 72.5, 72.86, 72.14, 71.24000000000001, 70.16, 69.62, 68.18]

In [13]: # NumPy

In [14]: C = np.array(celcius)

In [15]: F = C *9/5 + 32

In [16]: print(F)
[68.18 69.44 71.42 72.5  72.86 72.14 71.24 70.16 69.62 68.18]
```

Why NumPy?

Python List



NumPy ndarray



Space Comparison

Python List

```
In [31]: def showsize_pythonlist(lst):
    ...:     size_of_list = size(lst)
    ...:     print([size(elem) for elem in lst])
    ...:     size_of_elements = sum([size(elem) for elem in lst])
    ...:     totsize_python_list = size_of_list + size_of_elements
    ...:     print("size without the size of elements: ", size_of_list)
    ...:     print("size of all elements: ", size_of_elements)
    ...:     print("total size of list, including elements: ", totsize_python_list)
    ...:

In [32]: showsize_pythonlist([])
[]
size without the size of elements:  64
size of all elements:  0
total size of list, including elements:  64

In [33]: showsize_pythonlist([24,57,3])
[28, 28, 28]
size without the size of elements:  88
size of all elements:  84
total size of list, including elements:  172

In [34]: showsize_pythonlist([24,57,3,-1])
[28, 28, 28, 28]
size without the size of elements:  96
size of all elements:  112
total size of list, including elements:  208
```

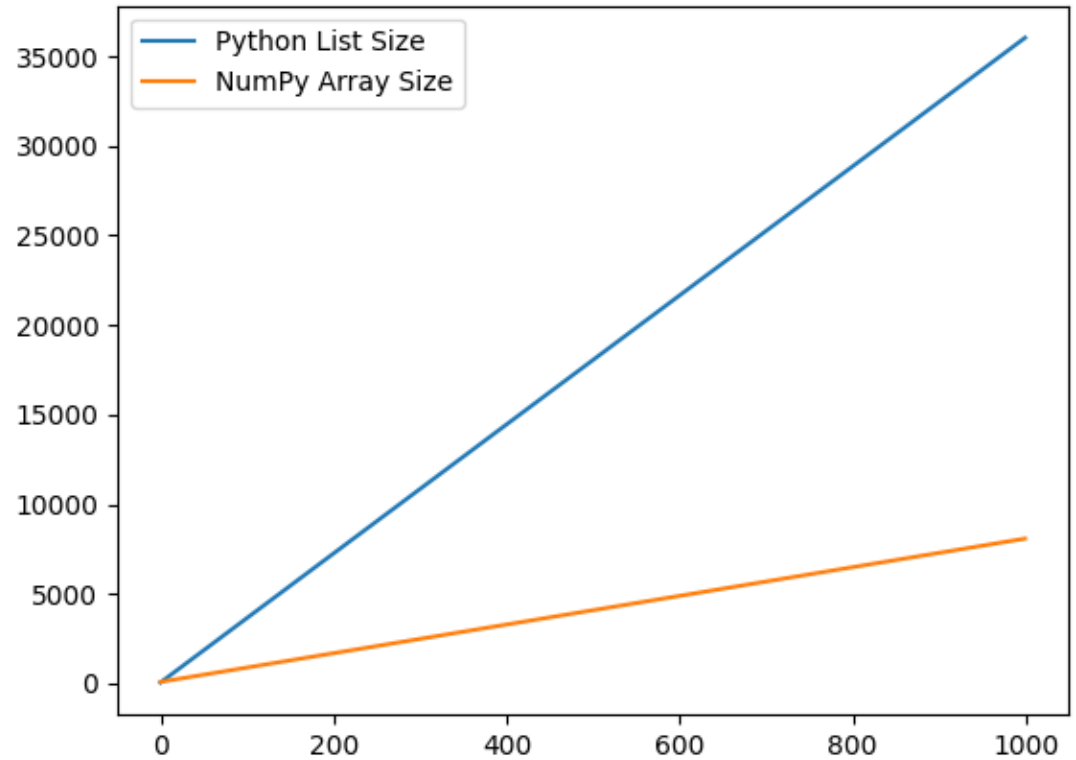
Totsize(num_elem) = 64+8*num_elem + 28*num_elem

NumPy ndarray

```
In [35]: lst = np.array([])

In [36]: print(size(lst))
96

In [37]: lst = np.array([24,57,3])

In [38]: print(size(lst))
120

In [39]: lst = np.array([24,57,3,-1])

In [40]: print(size(lst))
128
```

Totsize(num_elem) = 96+8*num_elem

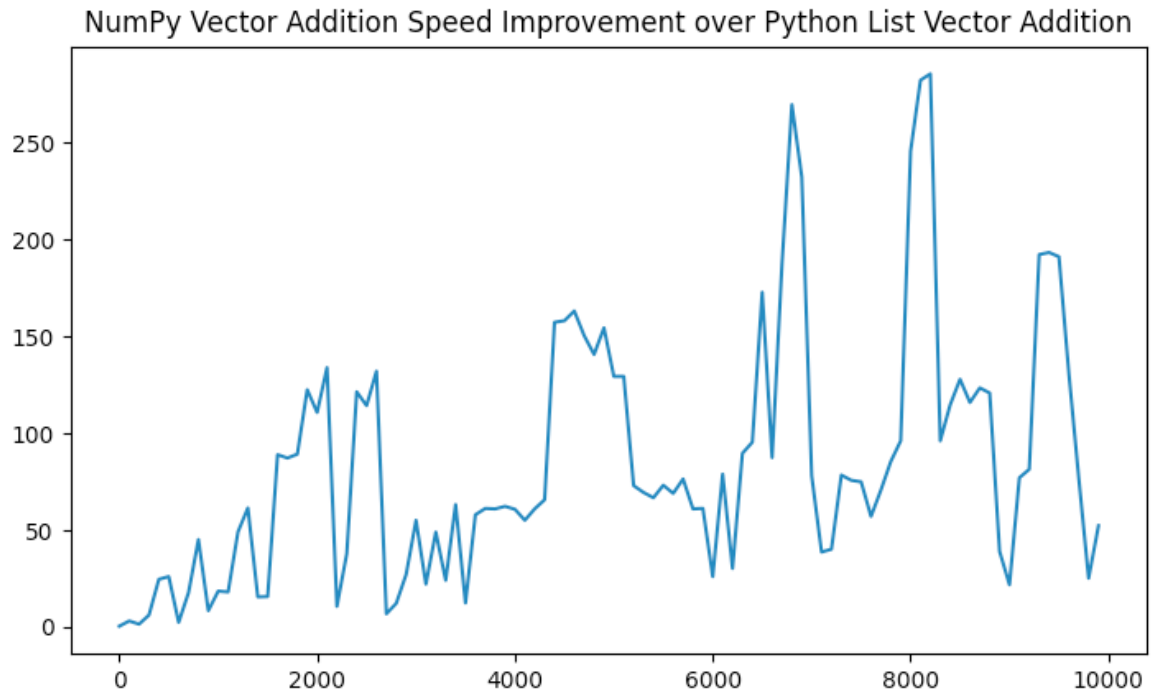
```
In [53]: def python_listsize(num_elem):
    ...:     return 64+8*num_elem+28*num_elem
    ...:

In [54]: def numpy_arraysize(num_elem):
    ...:     return 96+8*num_elem
    ...:

In [55]: Pvec = np.vectorize(python_listsize)

In [56]: Nvec = np.vectorize(numpy_arraysize)

In [57]: PG = Pvec(N); NG = Nvec(N)

In [58]: plt.plot(N, PG, label="Python List Size")
Out[58]: [<matplotlib.lines.Line2D at 0x7fb430a91908>]

In [59]: plt.plot(N, NG, label="NumPy Array Size")
Out[59]: [<matplotlib.lines.Line2D at 0x7fb430b12f98>]

In [60]: plt.legend()
Out[60]: <matplotlib.legend.Legend at 0x7fb4308c7ef0>

In [61]: plt.show()
```



Time Comparison

```
In [85]: def pure_python(size_of_vec):
    ...:     t1 = time.time()
    ...:     X = range(size_of_vec)
    ...:     Y = range(size_of_vec)
    ...:     Z = [X[i] + Y[i] for i in range(size_of_vec)]
    ...:     return time.time() - t1
    ...:

In [86]: def numpy_version(size_of_vec):
    ...:     t1 = time.time()
    ...:     X = np.arange(size_of_vec)
    ...:     Y = np.arange(size_of_vec)
    ...:     Z = X + Y
    ...:     return time.time() - t1
    ...:

In [87]: Pvec = np.vectorize(pure_python)

In [88]: Nvec = np.vectorize(numpy_version)

In [89]: Ptime = Pvec(S)

In [90]: Ntime = Nvec(S)
```

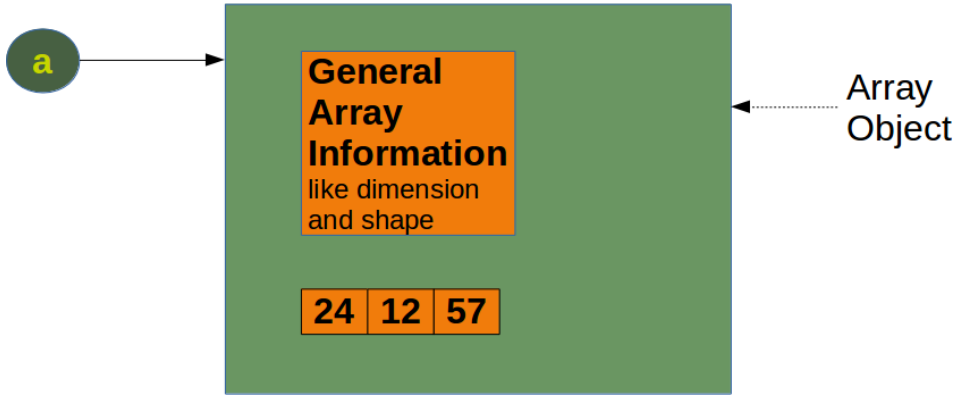
```
In [103]: plt.plot(S, Ptime/Ntime)
Out[103]: [<matplotlib.lines.Line2D at 0x7fb43023acf8>]

In [104]: plt.title('NumPy Vector Addition Speed Improvement over Python List Vector Addition')
Out[104]: Text(0.5, 1.0, 'NumPy Vector Addition Speed Improvement over Python List Vector Addition')
```

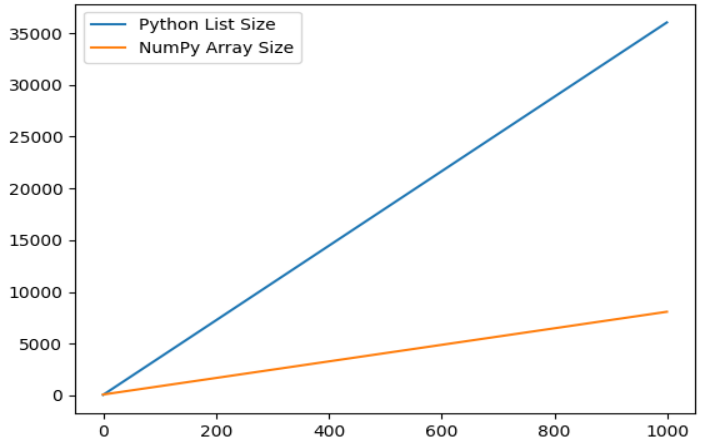


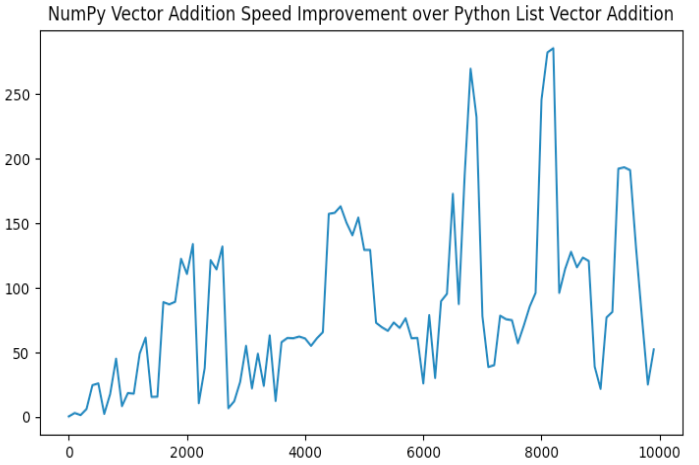

```
In [97]: print("On average, Numpy is faster than Python vector addion about {}".format(sum(Ptime/Ntime)/len(S)))
On average, Numpy is faster than Python vector addion about 84.18309227241475
```

สรุป: NumPy

- NumPy เป็น Python library สำหรับการประมวลผล multi-dimentional array และ matrices ขนาดใหญ่ที่มีประสิทธิภาพสูง
  - โครงสร้างข้อมูลหลักของ NumPy คือ ndarray
  - ใช้เนื้อที่หน่วยความจำน้อยกว่าและมีความเร็วในการประมวลผลสูงกว่า Python List







Creating NumPy Arrays

- **numpy.array**(*PythonList*)

```
import numpy as np
np.array([0,1,2,3,4])
```

```
array([0., 1., 2., 3., 4.])
```

- **numpy.arange**(*start, stop, step*)

```
import numpy as np
np.arange(0,5,1)
```

- **numpy.linspace**(*start, end, number_of_points*)

```
import numpy as np
np.linspace(0, 4, 5)
```

Creating NumPy Arrays

- **numpy.diag**(*1D_array*)

```
import numpy as np
np.diag([1,2,3])
```

- **numpy.zeros***(shape)*

```
import numpy as np
np.zeros((3,3))
```

- **numpy.ones***(shape)*

```
import numpy as np
np.ones((3,3))
```

- **numpy.eye(N)**

```
import numpy as np
np.eye(3)
```

- **numpy.random.rand***(d0, d1,…,dn)*

```
import numpy as np
np.random.rand(3,3)
```

Accesing NumPy Arrays

```
In [310]: M = np.random.rand(3,4)

In [311]: M
Out[311]:
array([[0.93801864, 0.17225443, 0.98679902, 0.48372231],
       [0.52705304, 0.57015024, 0.08606747, 0.0436057 ],
       [0.75643244, 0.75361269, 0.44621982, 0.48491875]])

In [312]: # access row 1

In [313]: M[1,:]
Out[313]: array([0.52705304, 0.57015024, 0.08606747, 0.0436057 ])

In [314]: M[1]
Out[314]: array([0.52705304, 0.57015024, 0.08606747, 0.0436057 ])

In [315]: # access col 1

In [316]: M[:,1]
Out[316]: array([0.17225443, 0.57015024, 0.75361269])

In [317]: # access cell (1,1)

In [318]: M[1,1]
Out[318]: 0.570150236486981

In [319]: M[1][1]
Out[319]: 0.570150236486981

In [320]: M[1,1]=0

In [321]: M[0] = 0

In [322]: M
Out[322]:
array([[0.        , 0.        , 0.        , 0.        ],
       [0.52705304, 0.        , 0.08606747, 0.0436057 ],
       [0.75643244, 0.75361269, 0.44621982, 0.48491875]])
```

Slicing NumPy Arrays

```
In [332]: M
Out[332]:
array([[0.        , 0.        , 0.        , 0.        ],
       [0.52705304, 0.        , 0.08606747, 0.0436057 ],
       [0.75643244, 0.75361269, 0.44621982, 0.48491875]])


In [323]: # slices of M

In [324]: M[1:3]
Out[324]:
array([[0.52705304, 0.        , 0.08606747, 0.0436057 ],
       [0.75643244, 0.75361269, 0.44621982, 0.48491875]])

In [325]: M[1:3, 1:2]
Out[325]:
array([[0.        ],
       [0.75361269]])

In [326]: M[-2]
Out[326]: array([0.52705304, 0.        , 0.08606747, 0.0436057 ])
```

NumPy Array Operations - scalar

```
In [332]: M
Out[332]:
array([[0.        , 0.        , 0.        , 0.        ],
       [0.52705304, 0.        , 0.08606747, 0.0436057 ],
       [0.75643244, 0.75361269, 0.44621982, 0.48491875]])
```

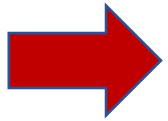
```
In [328]: M*2
Out[328]:
array([[0.        , 0.        , 0.        , 0.        ],
       [1.05410608, 0.        , 0.17213494, 0.08721141],
       [1.51286488, 1.50722537, 0.89243964, 0.9698375 ]])

In [329]: M -2
Out[329]:
array([[-2.        , -2.        , -2.        , -2.        ],
       [-1.47294696, -2.        , -1.91393253, -1.9563943 ],
       [-1.24356756, -1.24638731, -1.55378018, -1.51508125]])
```

NumPy Array Operations – Matrix Multiplication

```
In [340]: M = np.ones((3,3))

In [341]: M
Out[341]:
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])

In [342]: M*M                    # Element-wise multiplication
Out[342]:
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])

In [343]: np.dot(M,M)            # Matrix multiplication
Out[343]:
array([[3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.]])
```

NumPy Array Operations – vectorize

```
In [344]: def theta(x):
     ...:     if x >= 0: return 1
     ...:     else:      return 0
     ...:

In [345]: theta(1.0)
Out[345]: 1

In [346]: theta(-1.0)
Out[346]: 0

In [347]: theta(M)
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-347-f8ea58c9a110> in <module>
----> 1 theta(M)

<ipython-input-344-9625831ad55a> in theta(x)
      1 def theta(x):
----> 2     if x >= 0: return 1
      3     else:      return 0
      4
```



```
In [365]: M = np.random.randn(3,3)

In [366]: M
Out[366]:
array([[ 1.55102353, -0.69610317,  0.13832054],
       [-2.00539404, -0.99261748,  0.33799685],
       [-0.77093719, -0.39705166, -1.05252702]])

In [367]: thetaM = [theta(m) for r in M for m in r]

In [368]: thetaM
Out[368]: [1, 0, 1, 0, 0, 1, 0, 0, 0]

In [369]: tvec = np.vectorize(theta)

In [370]: thetaM2 = tvec(M)

In [371]: thetaM2
Out[371]:
array([[1, 0, 1],
       [0, 0, 1],
       [0, 0, 0]])
```

NumPy Array Operations – Conditions

```
In [372]: M
Out[372]:
array([[ 1.55102353, -0.69610317,  0.13832054],
       [-2.00539404, -0.99261748,  0.33799685],
       [-0.77093719, -0.39705166, -1.05252702]])

In [373]: (M>3)
Out[373]:
array([[False, False, False],
       [False, False, False],
       [False, False, False]])

In [374]: (M>3).any()
Out[374]: False

In [375]: (M>1).any()
Out[375]: True

In [376]: (M>1).all()
Out[376]: False
```

- Built on top of the NumPy framework
- Scientific algorithms
  - Integration
  - Optimization
  - Linear Algebra
  - Statistics
  - I/O
  - Fourier Transform
  - …

Linear Algebra

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 5 \\ 2 & 5 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 6 \\ -4 \\ 27 \end{bmatrix}$$

$$AX = B$$

where

- A is the 3x3 matrix of x, y and z **coefficients**
- X is **x, y and z**, and
- B is **6, −4 and 27**

Then (as shown on the Inverse of a Matrix page) the solution is this:

$$X = A^{-1}B$$

What does that mean?

It means that we can find the values of x, y and z (the X matrix) by multiplying the **inverse of the A matrix** by the **B matrix**.

So let's go ahead and do that.

First, we need to find the **inverse of the A matrix** (assuming it exists!)

Using the Matrix Calculator we get this:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 5 \\ 2 & 5 & -1 \end{bmatrix}^{-1} = \frac{1}{-21} \begin{bmatrix} -27 & 6 & 3 \\ 10 & -3 & -5 \\ -4 & -3 & 2 \end{bmatrix}$$

(I left the 1/determinant outside the matrix to make the numbers simpler)

Then multiply A^{-1} by B (we can use the Matrix Calculator again):

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \frac{1}{-21} \begin{bmatrix} -27 & 6 & 3 \\ 10 & -3 & -5 \\ -4 & -3 & 2 \end{bmatrix} \begin{bmatrix} 6 \\ -4 \\ 27 \end{bmatrix} = \frac{1}{-21} \begin{bmatrix} -105 \\ -63 \\ 42 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \\ -2 \end{bmatrix}$$

And we are done! The solution is:

$$x = 5,$$
$$y = 3,$$
$$z = -2$$

Source: https://www.mathsisfun.com/algebra/systems-linear-equations-matrices.html

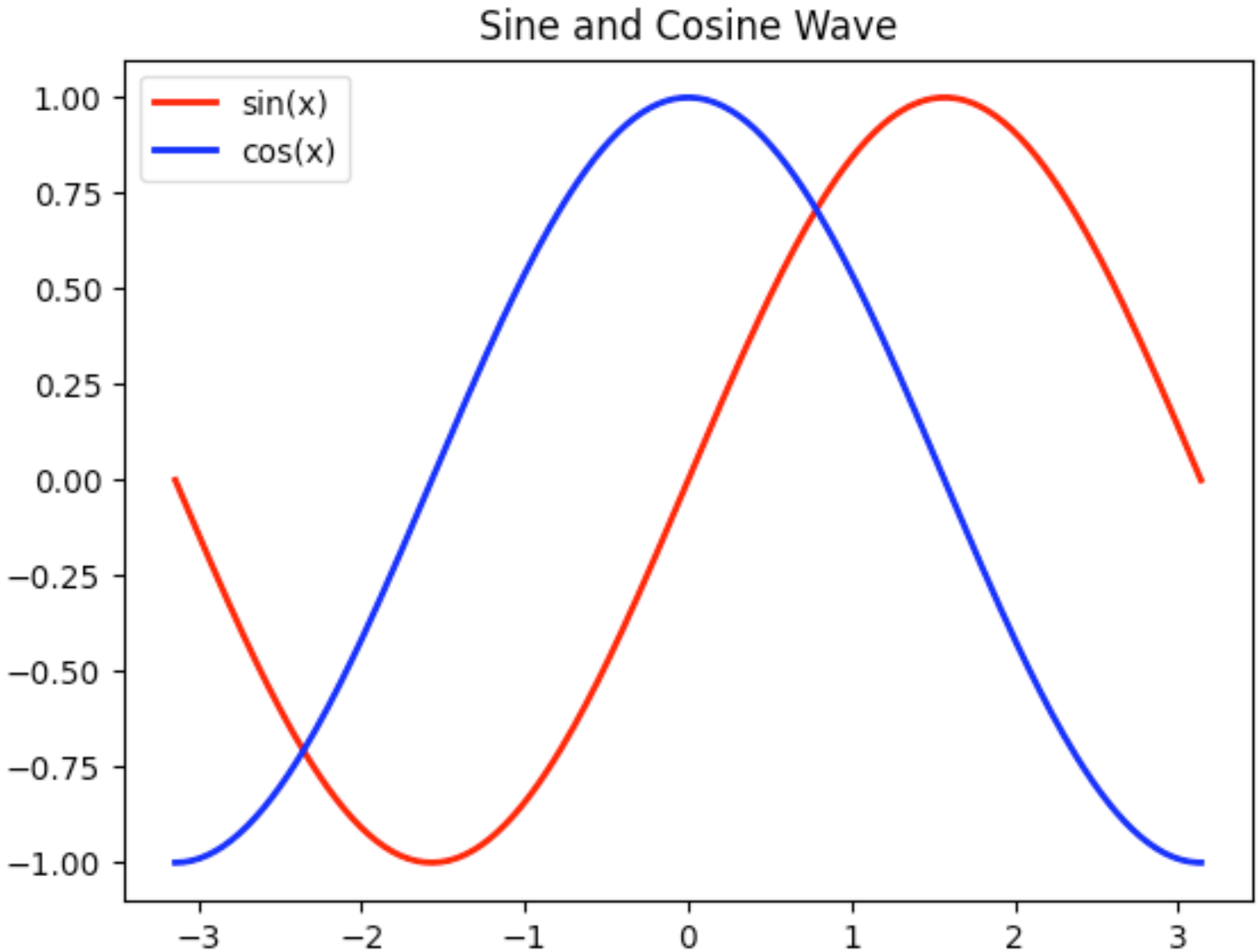
Solving Ax = b with SciPy

```
In [125]: A = np.array([[1,1,1],[0,2,5],[2,5,-1]])

In [126]: b = np.array([6,-4,27])

In [127]: x = linalg.solve(A, b)

In [128]: print(x)
[ 5.  3. -2.]

In [129]: linalg.norm(np.dot(A, x) - b)
Out[129]: 0.0
```

- Used for generating 2D and 3D scientific plots
- Support for LaTeX
- Various output formats: png, pdf, svg, eps
- Fine-grained control of plotting options

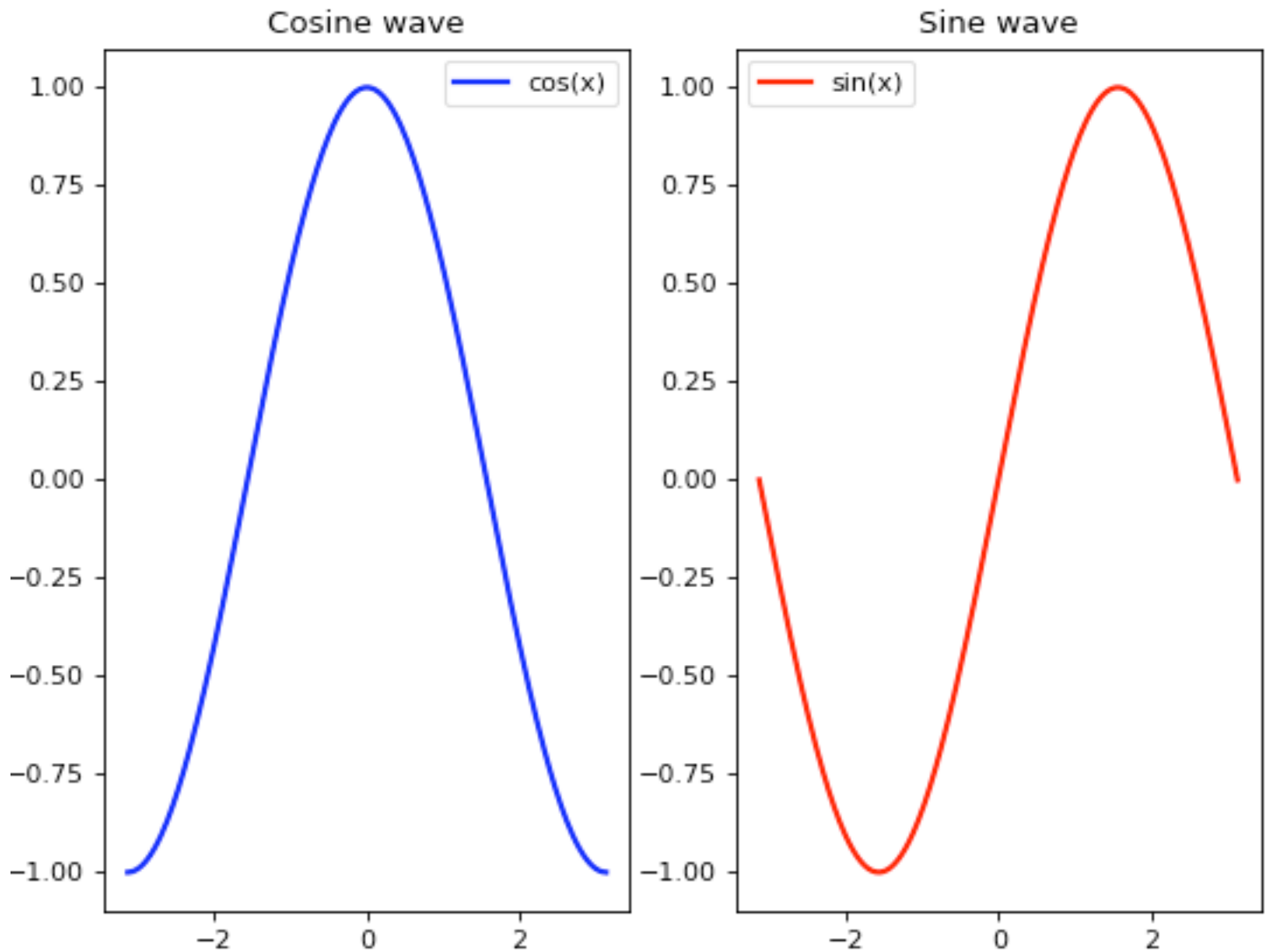
Ploting Sine and Cosine

```
In [218]: import matplotlib.pyplot as plt

In [219]: X = np.linspace(-np.pi, np.pi, 256, endpoint=True)

In [220]: C, S = np.cos(X), np.sin(X)

In [221]: plt.plot(X, C, color='blue', linewidth=2, linestyle='-', label='cos(x)')
Out[221]: [<matplotlib.lines.Line2D at 0x7fb43483e8d0>]

In [222]: plt.plot(X, S, color='red', linewidth=2, linestyle='-', label='sin(x)')
Out[222]: [<matplotlib.lines.Line2D at 0x7fb4352da198>]

In [223]: plt.title("Sine and Cosine Wave")
Out[223]: Text(0.5, 1.0, 'Sine and Cosine Wave')

In [224]: plt.legend()
Out[224]: <matplotlib.legend.Legend at 0x7fb434820c50>

In [225]: plt.show()
```


Sine and Cosine Wave
sin(x)
cos(x)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−3
−2
−1
0
1
2
3

Ploting Sine and Cosine (Subplots)

```
In [247]: X = np.linspace(-np.pi, np.pi, 256, endpoint=True)

In [248]: C, S = np.cos(X), np.sin(X)

In [249]: plt.figure(figsize=(8,6), dpi=80)
Out[249]: <Figure size 640x480 with 0 Axes>

In [250]: plt.subplot(1,2,1)
Out[250]: <AxesSubplot:>

In [251]: plt.plot(X, C, color='blue', linewidth=2, linestyle='-', label='cos(x)')
Out[251]: [<matplotlib.lines.Line2D at 0x7fb436b33ba8>]

In [252]: plt.legend()
Out[252]: <matplotlib.legend.Legend at 0x7fb436ce5208>

In [253]: plt.title('Cosine wave')
Out[253]: Text(0.5, 1.0, 'Cosine wave')

In [254]: plt.subplot(1,2,2)
Out[254]: <AxesSubplot:>

In [255]: plt.plot(X, S, color='red', linewidth=2, linestyle='-', label='sin(x)')
Out[255]: [<matplotlib.lines.Line2D at 0x7fb4370e6048>]

In [256]: plt.legend()
Out[256]: <matplotlib.legend.Legend at 0x7fb4370f0f28>

In [257]: plt.title('Sine Wave')
Out[257]: Text(0.5, 1.0, 'Sine Wave')
```


Cosine wave
cos(x)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−2
0
2
Sine wave
sin(x)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−2
0
2

- High-performance, user-friendly data structures and data analysis tools
- Built on top of NumPy, (SciPy, Matplotlib)
- Data structures for manipulate numeric tables and data series
  - Series
  - DataFrame (similar to database)

Pandas Series

Pandas Series

คือ โครงสร้างข้อมูลที่ประกอบด้วย อาร์เรย์ 2 อาร์เรย์ คือ index array และ values array

```
import pandas as pd
S = pd.Series([11, 28, 72, 3, 5, 8])
S
```

Output::

```
0    11
1    28
2    72
3     3
4     5
5     8
dtype: int64
```

```
print(S.index)
print(S.values)
```

```
RangeIndex(start=0, stop=6, step=1)
[11 28 72  3  5  8]
```

Pandas Series vs Numpy Array

คือ pandas series มีข้อดีคือ สามารถใช้ index ที่ไม่ใช่ตัวเลขจำนวนเต็มตามลำดับ 0...จำนวน element ได้

```
import numpy as np
X = np.array([11, 28, 72, 3, 5, 8])
print(X)
print(S.values)
# both are the same type:
print(type(S.values), type(X))
```

```
[11 28 72  3  5  8]
[11 28 72  3  5  8]
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

```
[In [102]: fruits = ['ส้ม','แอปเปิ้ล','กล้วย','เชอรี่', 'ฝรั่ง','แตงโม']

[In [103]: quantities = [11,28,72,3,5,8]

[In [104]: S = pd.Series(quantities, index=fruits)

[In [105]: S
Out[105]:
ส้ม          11
แอปเปิ้ล     28
กล้วย        72
เชอรี่        3
ฝรั่ง         5
แตงโม         8
dtype: int64
```

Pandas Series – Scalar Operation

คือ สามารถใช้ได้เหมือนกันกับ nump arrray

```
import numpy as np
print((S + 3) * 4)
print("========================")
print(np.sin(S))
```

```
apples       92
oranges     144
cherries    220
pears        52
dtype: int64
========================
apples      0.912945
oranges     0.999912
cherries    0.986628
pears      -0.544021
dtype: float64
```

Pandas Series – Applying Function Element-wise

Apply numpy functions

```
S.apply(np.log)
```

Output::

```
apples      2.995732
oranges     3.496508
cherries    3.951244
pears       2.302585
dtype: float64
```

Apply user-defined functions: e.g. เพิ่มจำนวนผลไม้ที่มีจำนวนน้อยกว่า 50 อีก 10 ลูก

```
S.apply(lambda x: x if x > 50 else x+10 )
```

Output::

```
apples      30
oranges     43
cherries    52
pears       20
dtype: int64
```

Pandas Series – Accessing with loc, iloc

```
In [117]: S
Out[117]:
ส้ม          11
แอปเปิ้ล     28
กล้วย        72
เชอรี่        3
ฝรั่ง         5
แตงโม         8
dtype: int64
```

```
In [109]: S['ส้ม']
Out[109]: 11

In [110]: S[['ส้ม','กล้วย']]
Out[110]:
ส้ม      11
กล้วย    72
dtype: int64
```

```
In [114]: S.loc[['ส้ม','กล้วย']]
Out[114]:
ส้ม      11
กล้วย    72
dtype: int64

In [115]: S.iloc[1]
Out[115]: 28

In [116]: S.iloc[0]
Out[116]: 11
```

Pandas Series – Filtering by a Boolean array

เลือกผลไม้ที่มีจำนวนมากกว่า 30

```
S[S>30]
```

```
Output:: oranges     33
         cherries    52
         dtype: int64
```

มี “ส้ม” ใน S ?

```
In [108]: "ส้ม" in S
Out[108]: True
```

Pandas Series – Creating from Dictionary

```
cities = {"London":    8615246,
          "Berlin":    3562166,
          "Madrid":    3165235,
          "Rome":      2874038,
          "Paris":     2273305,
          "Vienna":    1805681,
          "Bucharest": 1803425,
          "Hamburg":   1760433,
          "Budapest":  1754000,
          "Warsaw":    1740119,
          "Barcelona": 1602386,
          "Munich":    1493900,
          "Milan":     1350680}

city_series = pd.Series(cities)
print(city_series)
```

Pandas Series – Missing Values

```
my_cities = ["London", "Paris", "Zurich", "Berlin",
             "Stuttgart", "Hamburg"]

my_city_series = pd.Series(cities,
                           index=my_cities)
my_city_series
```

Output::

```
London       8615246.0
Paris        2273305.0
Zurich             NaN
Berlin       3562166.0
Stuttgart          NaN
Hamburg      1760433.0
dtype: float64
```

dropna()

```
print(my_city_series.dropna())
```

```
London      8615246.0
Paris       2273305.0
Berlin      3562166.0
Hamburg     1760433.0
dtype: float64
```

isnull(), notnull()

```
my_cities = ["London", "Paris", "Zurich", "Berlin",
             "Stuttgart", "Hamburg"]

my_city_series = pd.Series(cities,
                           index=my_cities)
print(my_city_series.isnull())
```

```
London       False
Paris        False
Zurich        True
Berlin       False
Stuttgart     True
Hamburg      False
dtype: bool
```

```
print(my_city_series.notnull())
```

```
London        True
Paris         True
Zurich       False
Berlin        True
Stuttgart    False
Hamburg       True
dtype: bool
```

Pandas Series – Missing Values

fillna()

```
print(my_city_series.fillna(0))
```

```
London       8615246.0
Paris        2273305.0
Zurich             0.0
Berlin       3562166.0
Stuttgart          0.0
Hamburg      1760433.0
dtype: float64
```

```
cities = {"London":    8615246,
          "Berlin":    3562166,
          "Madrid":    3165235,
          "Rome":      2874038,
          "Paris":     2273305,
          "Vienna":    1805681,
          "Bucharest":1803425,
          "Hamburg":   1760433,
          "Budapest":  1754000,
          "Warsaw":    1740119,
          "Barcelona":1602386,
          "Munich":    1493900,
          "Milan":     1350680}

my_cities = ["London", "Paris", "Zurich", "Berlin",
             "Stuttgart", "Hamburg"]

my_city_series = pd.Series(cities,
                           index=my_cities)
my_city_series = my_city_series.fillna(0).astype(int)
print(my_city_series)
```

```
London       8615246
Paris        2273305
Zurich             0
Berlin       3562166
Stuttgart          0
Hamburg      1760433
dtype: int64
```

สรุป Pandas Series

- คือโครงสร้างข้อมูล มีลักษณะคล้าย numpy 1-d array แต่มีความสามารถมากกว่า
  - ประกอบด้วย อาร์เรย์สองอาร์เรย์คือ values, index
  - อินเด็กซ์ ไม่จำเป็นต้องเป็นจำนวนเต็มเรียงลำดับจาก 0...ขนาดอาร์เรย์
  - สามารถใช้ scalar operation เช่น +, * ได้เหมือนกันกับ numpy array
  - สามารถ apply user-defined functions กับสมาชิกแต่ละตัวได้โดยใช้ method *apply*
  - access ข้อมูล
    - ใช้ค่าตำแหน่ง (iloc) (สามารถกำหนดได้หลายตัว)
    - ใช้ค่าของ index (loc) (สามารถกำหนดได้หลายตัว)
    - Boolean Array
  - Missing Values
    - NaN
    - Testing: isnull(), notnull()
    - Filtering: dropna()
    - Filling: fillna()

Pandas Data Frame

Pandas DataFrame

คือ โครงสร้างข้อมูลที่ประกอบด้วยกลุ่มคอลัมน์ที่มีลำดับ (ordered collection of column)

- สร้างจาก Series หลายๆ ซีรียส์

```
import pandas as pd

years = range(2014, 2018)

shop1 = pd.Series([2409.14, 2941.01, 3496.83, 3119.55], ind
ex=years)
shop2 = pd.Series([1203.45, 3441.62, 3007.83, 3619.53], ind
ex=years)
shop3 = pd.Series([3412.12, 3491.16, 3457.19, 1963.10], ind
ex=years)
```

```
shops_df = pd.concat([shop1, shop2, shop3], axis=1)
shops_df
```

Output::

	0	1	2
2014	2409.14	1203.45	3412.12
2015	2941.01	3441.62	3491.16
2016	3496.83	3007.83	3457.19
2017	3119.55	3619.53	1963.10

```
cities = ["Zürich", "Winterthur", "Freiburg"]
shops_df.columns = cities
print(shops_df)

# alternative way: give names to series:
shop1.name = "Zürich"
shop2.name = "Winterthur"
shop3.name = "Freiburg"

print("------")
shops_df2 = pd.concat([shop1, shop2, shop3], axis=1)
print(shops_df2)
```

Pandas DataFrame

คือ โครงสร้างข้อมูลที่ประกอบด้วยกลุ่มคอลัมน์ที่มีลำดับ (ordered collection of column)

- สร้างจาก Dictionaries

```
cities = {"name": ["London", "Berlin", "Madrid", "Rome",
                   "Paris", "Vienna", "Bucharest", "Hamburg",
                   "Budapest", "Warsaw", "Barcelona",
                   "Munich", "Milan"],
          "population": [8615246, 3562166, 3165235, 2874038,
                         2273305, 1805681, 1803425, 1760433,
                         1754000, 1740119, 1602386, 1493900,
                         1350680],
          "country": ["England", "Germany", "Spain", "Italy",
                      "France", "Austria", "Romania",
                      "Germany", "Hungary", "Poland", "Spain",
                      "Germany", "Italy"]}

city_frame = pd.DataFrame(cities)
city_frame
```

	name	population	country
0	London	8615246	England
1	Berlin	3562166	Germany
2	Madrid	3165235	Spain
3	Rome	2874038	Italy
4	Paris	2273305	France
5	Vienna	1805681	Austria
6	Bucharest	1803425	Romania
7	Hamburg	1760433	Germany
8	Budapest	1754000	Hungary
9	Warsaw	1740119	Poland
10	Barcelona	1602386	Spain
11	Munich	1493900	Germany
12	Milan	1350680	Italy

Pandas DataFrame

คือ โครงสร้างข้อมูลที่ประกอบด้วยกลุ่มคอลัมน์ที่มีลำดับ (ordered collection of column)

- สร้างจาก random values

```
import numpy as np

names = ['Frank', 'Eve', 'Stella', 'Guido', 'Lara']
index = ["January", "February", "March",
         "April", "May", "June",
         "July", "August", "September",
         "October", "November", "December"]
df = pd.DataFrame((np.random.randn(12, 5)*1000).round(2),
                  columns=names,
                  index=index)
df
```

Pandas DataFrame – Reading from CSV ด้วย read_csv

```
In [182]: exchanges =pd.read_csv('dollar_euro.txt',sep=' ')

In [183]: exchanges
Out[183]:
```

	Year	Average	Min_USD/EUR	Max_USD/EUR	Working_days
0	2016	0.901696	0.864379	0.959785	247
1	2015	0.901896	0.830358	0.947688	256
2	2014	0.753941	0.716692	0.823655	255
3	2013	0.753234	0.723903	0.783208	255
4	2012	0.778848	0.743273	0.827198	256
5	2011	0.719219	0.671953	0.775855	257
6	2010	0.755883	0.686672	0.837381	258
7	2009	0.718968	0.661376	0.796495	256
8	2008	0.683499	0.625391	0.802568	256
9	2007	0.730754	0.672314	0.775615	255
10	2006	0.797153	0.750131	0.845594	255
11	2005	0.805097	0.740357	0.857118	257
12	2004	0.804828	0.733514	0.847314	259
13	2003	0.885766	0.791766	0.963670	255
14	2002	1.060945	0.953562	1.165773	255
15	2001	1.117587	1.047669	1.192748	255
16	2000	1.085899	0.962649	1.211827	255
17	1999	0.939475	0.848176	0.998502	261

Pandas DataFrame – Reading from CSV ด้วย read_csv

```
In [190]: exchanges =pd.read_csv('dollar_euro.txt',sep=' ', header=0, names=['year','average','min','max','days'])
     ...:

In [191]: exchanges.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18 entries, 0 to 17
Data columns (total 5 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   year     18 non-null     int64
 1   average  18 non-null     float64
 2   min      18 non-null     float64
 3   max      18 non-null     float64
 4   days     18 non-null     int64
dtypes: float64(3), int64(2)
memory usage: 848.0 bytes
```

```
In [193]: exchanges
Out[193]:
    year   average       min       max  days
0   2016  0.901696  0.864379  0.959785   247
1   2015  0.901896  0.830358  0.947688   256
2   2014  0.753941  0.716692  0.823655   255
3   2013  0.753234  0.723903  0.783208   255
4   2012  0.778848  0.743273  0.827198   256
5   2011  0.719219  0.671953  0.775855   257
6   2010  0.755883  0.686672  0.837381   258
7   2009  0.718968  0.661376  0.796495   256
8   2008  0.683499  0.625391  0.802568   256
9   2007  0.730754  0.672314  0.775615   255
10  2006  0.797153  0.750131  0.845594   255
11  2005  0.805097  0.740357  0.857118   257
12  2004  0.804828  0.733514  0.847314   259
13  2003  0.885766  0.791766  0.963670   255
14  2002  1.060945  0.953562  1.165773   255
15  2001  1.117587  1.047669  1.192748   255
16  2000  1.085899  0.962649  1.211827   255
17  1999  0.939475  0.848176  0.998502   261
```

Pandas DataFrame – ใช้คอลัมน์เป็น index ของ DataFrame

```
city_frame = pd.DataFrame(cities,
                          columns=["name", "population"],
                          index=cities["country"])

city_frame
```

Output::

	name	population
England	London	8615246
Germany	Berlin	3562166
Spain	Madrid	3165235
Italy	Rome	2874038
France	Paris	2273305
Austria	Vienna	1805681
Romania	Bucharest	1803425
Germany	Hamburg	1760433
Hungary	Budapest	1754000
Poland	Warsaw	1740119
Spain	Barcelona	1602386
Germany	Munich	1493900
Italy	Milan	1350680

Pandas DataFrame – เพิ่มคอลัมน์ใหม่ ด้วย Python list

```
# area in square km:
area = [1572, 891.85, 605.77, 1285,
        105.4, 414.6, 228, 755,
        525.2, 517, 101.9, 310.4,
        181.8]
# area could have been designed as a list, a Series, an array or a scalar

city_frame["area"] = area
print(city_frame)
```

```
           country     area  population
London     England  1572.00     8615246
Berlin     Germany   891.85     3562166
Madrid       Spain   605.77     3165235
Rome         Italy  1285.00     2874038
Paris       France   105.40     2273305
Vienna     Austria   414.60     1805681
Bucharest  Romania   228.00     1803425
Hamburg    Germany   755.00     1760433
Budapest   Hungary   525.20     1754000
Warsaw      Poland   517.00     1740119
Barcelona    Spain   101.90     1602386
Munich     Germany   310.40     1493900
Milan        Italy   181.80     1350680
```

Pandas DataFrame – เพิ่มคอลัมน์ใหม่ ด้วย insert

```
In [145]: city_frame.insert(column='cum_population', loc=3, value=city_frame['population'].cumsum())

In [146]: city_frame
Out[146]:
```

	name	population	country	cum_population
0	London	8615246	England	8615246
1	Berlin	3562166	Germany	12177412
2	Madrid	3165235	Spain	15342647
3	Rome	2874038	Italy	18216685
4	Paris	2273305	France	20489990
5	Vienna	1805681	Austria	22295671
6	Bucharest	1803425	Romania	24099096
7	Hamburg	1760433	Germany	25859529
8	Budapest	1754000	Hungary	27613529
9	Warsaw	1740119	Poland	29353648
10	Barcelona	1602386	Spain	30956034
11	Munich	1493900	Germany	32449934
12	Milan	1350680	Italy	33800614

Pandas DataFrame – ลบคอลัมน์ ด้วย drop

```
In [148]: city_frame.drop('cum_population',axis=1, inplace=True)

In [149]: city_frame
Out[149]:
          name  population  country
0       London     8615246  England
1       Berlin     3562166  Germany
2       Madrid     3165235    Spain
3         Rome     2874038    Italy
4        Paris     2273305   France
5       Vienna     1805681  Austria
6    Bucharest     1803425  Romania
7      Hamburg     1760433  Germany
8     Budapest     1754000  Hungary
9       Warsaw     1740119   Poland
10   Barcelona     1602386    Spain
11      Munich     1493900  Germany
12       Milan     1350680    Italy
```

Pandas DataFrame – ใช้ loc ในการ access Rows

```
city_frame = pd.DataFrame(cities,
                          columns=("name", "population"),
                          index=cities["country"])
print(city_frame.loc["Germany"])
```

```
            name  population
Germany   Berlin     3562166
Germany  Hamburg     1760433
Germany   Munich     1493900
```

```
print(city_frame.loc[["Germany", "France"]])
```

```
            name  population
Germany   Berlin     3562166
Germany  Hamburg     1760433
Germany   Munich     1493900
France     Paris     2273305
```

```
print(city_frame.loc[city_frame.population>2000000])
```

```
            name  population
England   London     8615246
Germany   Berlin     3562166
Spain     Madrid     3165235
Italy       Rome     2874038
France     Paris     2273305
```

Pandas DataFrame – ใช้ index ในการ access Rows

```
In [152]: city_frame[['population']]
Out[152]:
    population
0      8615246
1      3562166
2      3165235
3      2874038
4      2273305
5      1805681
6      1803425
7      1760433
8      1754000
9      1740119
10     1602386
11     1493900
12     1350680
```

```
In [153]: city_frame[['name','population']]
Out[153]:
         name  population
0      London     8615246
1      Berlin     3562166
2      Madrid     3165235
3        Rome     2874038
4       Paris     2273305
5      Vienna     1805681
6   Bucharest     1803425
7     Hamburg     1760433
8    Budapest     1754000
9      Warsaw     1740119
10  Barcelona     1602386
11     Munich     1493900
12      Milan     1350680
```

```
In [154]: city_frame.population
Out[154]:
0     8615246
1     3562166
2     3165235
3     2874038
4     2273305
5     1805681
6     1803425
7     1760433
8     1754000
9     1740119
10    1602386
11    1493900
12    1350680
Name: population, dtype: int64
```

Pandas DataFrame – เรียงลำดับด้วย sort_values

```
city_frame = city_frame.sort_values(by="area", ascending=False)
print(city_frame)
```

```
           country     area  population
London     England  1572.00     8615246
Rome         Italy  1285.00     2874038
Berlin     Germany   891.85     3562166
Hamburg    Germany   755.00     1760433
Madrid       Spain   605.77     3165235
Budapest   Hungary   525.20     1754000
Warsaw      Poland   517.00     1740119
Vienna     Austria   414.60     1805681
Munich     Germany   310.40     1493900
Bucharest  Romania   228.00     1803425
Milan        Italy   181.80     1350680
Paris       France   105.40     2273305
Barcelona    Spain   101.90     1602386
```

Pandas DataFrame – Changing one value in Data Frame

ใช้ loc (ใช้เปลี่ยนทีละหลายๆ ค่าได้), at (เร็วกว่า loc)

```
# accessing the job of Bill:
print(df.loc['Bill', 'job'])
# alternative way to access it with at:
print(df.at['Bill', 'job'])

# setting the job of Bill to 'data analyst' with 'loc'
df.loc['Bill', 'job'] = 'data analyst'
# let us check it:
print(df.loc['Bill', 'job'])

# setting the job of Bill to 'computer scientist' with 'at'
df.at['Pete', 'language'] = 'Python'
```

```
data scientist
data scientist
data analyst
```

```
%timeit df.loc['Bill', 'language'] = 'Python'

129 µs ± 2.1 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

%timeit df.at['Bill', 'language'] = 'Python'

4.69 µs ± 209 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Pandas DataFrame – Replacing Values in Data Frame

ใช้ replace

```
import pandas as pd

first = ('Mike', 'Dorothee', 'Tom', 'Bill', 'Pete', 'Kate')
last = ('Meyer', 'Maier', 'Meyer', 'Mayer', 'Meyr', 'Mair')
job = ('data analyst', 'programmer', 'computer scientist',
       'data scientist', 'programmer', 'psychiatrist')
language = ('Python', 'Perl', 'Java', 'Pithon', 'Pythen', 'Brainfuck')

df = pd.DataFrame(list(zip(last, job, language)),
                  columns =['last', 'job', 'language'],
                  index=first)

df
```

Output::

	last	job	language
Mike	Meyer	data analyst	Python
Dorothee	Maier	programmer	Perl
Tom	Meyer	computer scientist	Java
Bill	Mayer	data scientist	Pithon
Pete	Meyr	programmer	Pythen
Kate	Mair	psychiatrist	Brainfuck

```
df.replace("programmer",
           "computer scientist",
           inplace=True)
df
```

```
df.replace(to_replace=['Mike', 'Tom', 'Perl'],
           value= ['Michael', 'Thomas', 'Python'],
           inplace=True)
df
```

Example

```
df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],
                  index=['cobra', 'viper', 'sidewinder'],
                  columns=['max_speed', 'shield'])

df
```

Output::

	max_speed	shield
cobra	1	2
viper	4	5
sidewinder	7	8

1.แสดงค่า max_speed, shield ของ viper

```
df.loc['viper']   df.loc['viper'][['max_speed','shield']]
```

2.แสดงค่า max_speed, shield ของ viper และ sidewinder

```
df.loc[['viper','sidewinder']]        df.loc['viper':'sidewinder']
```

Example

```
df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],
                  index=['cobra', 'viper', 'sidewinder'],
                  columns=['max_speed', 'shield'])

df
```

Output::

	max_speed	shield
cobra	1	2
viper	4	5
sidewinder	7	8

3.แสดงค่า max_speed ของ cobra

```
df.loc['cobra']['max_speed']
```

4.แสดงค่า max_speed ของ cobra และ viper โดยใช้ slice index

```
df.loc['cobra':'viper']['max_speed']
```

Example

```
df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],
                  index=['cobra', 'viper', 'sidewinder'],
                  columns=['max_speed', 'shield'])

df
```

Output::

	max_speed	shield
cobra	1	2
viper	4	5
sidewinder	7	8

5.แสดงค่า max_speed และ shield ของ cobra โดยใช้ Boolean Array

```
df.loc[[True,False,False],['max_speed']]
```

6.แสดงค่า max_speed ของแถวที่มีค่า shield มากกว่า 6

```
df.loc[df['shield']>6,['max_speed']]
```

Example

```
df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],
                  index=['cobra', 'viper', 'sidewinder'],
                  columns=['max_speed', 'shield'])

df
```

Output::

	max_speed	shield
cobra	1	2
viper	4	5
sidewinder	7	8

7.แสดงค่า max_speed ของแถวที่มีค่า shield เท่ากับ 8 โดยใช้ lambda function

```
df.loc[lambda df: df['shield'] == 8]['max_speed']
```

Example

```
df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],
                  index=['cobra', 'viper', 'sidewinder'],
                  columns=['max_speed', 'shield'])

df
```

Output::

	max_speed	shield
cobra	1	2
viper	4	5
sidewinder	7	8

8. Set ค่าทุกค่าของแถวที่มีอินเด็กซ์เท่ากับ cobra เป็น 10
9. Set ค่าคอลัมน์ shield เป็น 20
10. Set ค่าของแถวที่มีค่า shield น้อยกว่า 20 เป็น 0

```
df.loc['cobra']=10
df.loc[:,'shield'] = 20
df.loc[df['shield']<20] = 0
```

สรุป Pandas Dataframe

- คือโครงสร้างข้อมูลแบบ heterogenous data type มีลักษณะคล้าย Excel Spreadsheet
  - การสร้าง dataframe
    - Concatenation of multiple pandas series
    - Python Dictionaries
    - Random values
    - Read from CSV file
  - Access
    - subscript
    - loc
    - at
  - Query
  - Aggregate
  - Plotting Values

Reference

- https://www.python-course.eu/
- https://srijithr.gitlab.io/post/scipy/