

# Python programming : continued

ECE30007 Intro to AI Project

# Contents

- Python Function
- Python OOP
- Numpy
- Matplotlib
- Exercise

# Python Function

- Why we use functions?
  - Easy to modify
  - Flexible to maintain
  - Readability
- Built-in Function vs User-defined Function
- Local Variable vs Global Variable
- Recursion
- Lambda

# Python Function

- **Built-in Function**

- The Python interpreter has a number of functions and types built into it that are always available.
- Already defined in Python libraries and we can call them directly.

- `print()`

- `input()`

- `int()`

- `float()`

- `range()`

- `len()`

...

		Built-in Functions		
<code>abs()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>all()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>any()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>ascii()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bin()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>bool()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	
<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>	

<https://docs.python.org/3/library/functions.html>

# Python Function

- **User-Defined Function**

- User(programmer) define functions in a program and then call them wherever they are needed.
- In order to use user-defined functions
  - Define Function
  - Call Function

```
In [2]: 1 def user_define(x,y):  
        2     print(x,y,"is taking ECE30007")  
        3     return x+ " "+y  
        4  
        5 string = user_define("Hanse","21100394")  
        6 print(string)
```

```
Hanse 21100394 is taking ECE30007  
Hanse 21100394
```

# Python Function

- **Local Variable**

- Local variable can be accessed only inside the function in which they are declared.

```
1 def foo():
2     y = "local"
3     print("local : ", y)
4
5 foo()
6 print(y)
```

local : local

---

**NameError** Traceback (most recent call last)  
<ipython-input-3-39fd7c4ee928> in <module>  
4  
5 foo()  
----> 6 print(y)  
  
**NameError**: name 'y' is not defined

- **Global Variable**

- Global variables can be accessed throughout the program body by all functions.

```
1 x = "global"
2 def foo():
3     print("x inside:", x)
4
5 foo()
6 print("x outside:", x)
```

x inside: global  
x outside: global

# Python Function

- **Recursion**

- Function calls itself.

```
1 def recur_fac(x):  
2     if x<1:  
3         return 1  
4     return x*recur_fac(x-1)  
5  
6  
7 recur_fac(5)
```

120

- **Lambda**

- A lambda function is a small anonymous function. A lambda function can take any number of arguments but can only have one expression.

```
1 x = lambda a, b: print(a*b)  
2 x(5,6)
```

30

# Exercise(1) – Python Function

- Function that sums 0 to n using recursion
  - Get n as an integer parameter.
  - Return the added number

```
In [70]: def sum(n):
```

```
    sum(10)
```

```
Out[70]: 55
```

- Function that checks if the number is odd or even
  - Get n as an integer parameter.
  - Return string “Even” if the number is even.
  - Return string “Odd” if the number is odd

```
In [78]: def is_odd(num):
```

```
    is_odd(3)
```

```
Out[78]: 'Odd'
```

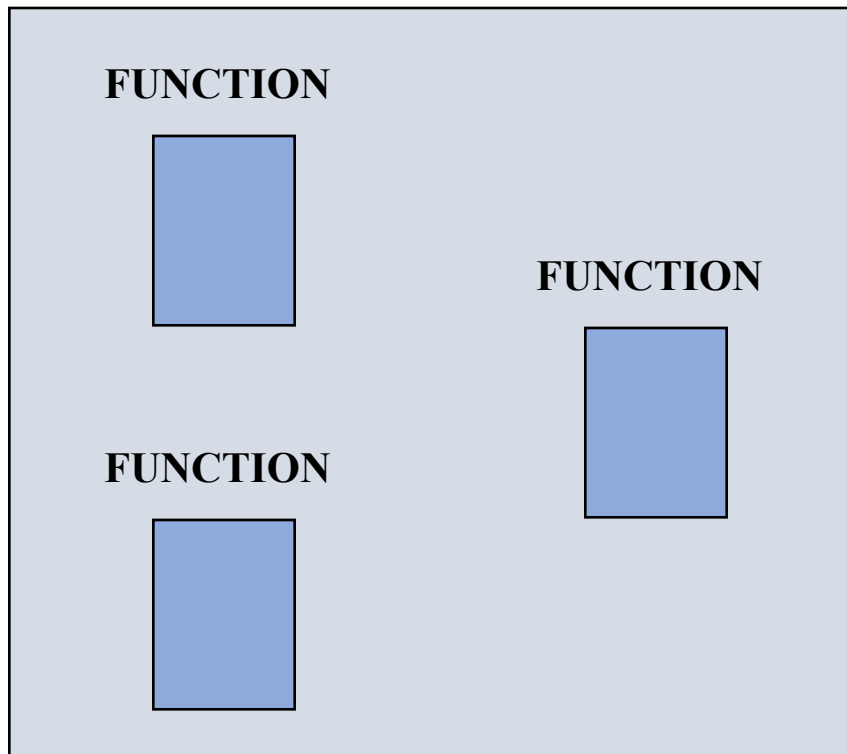


# object oriented programming

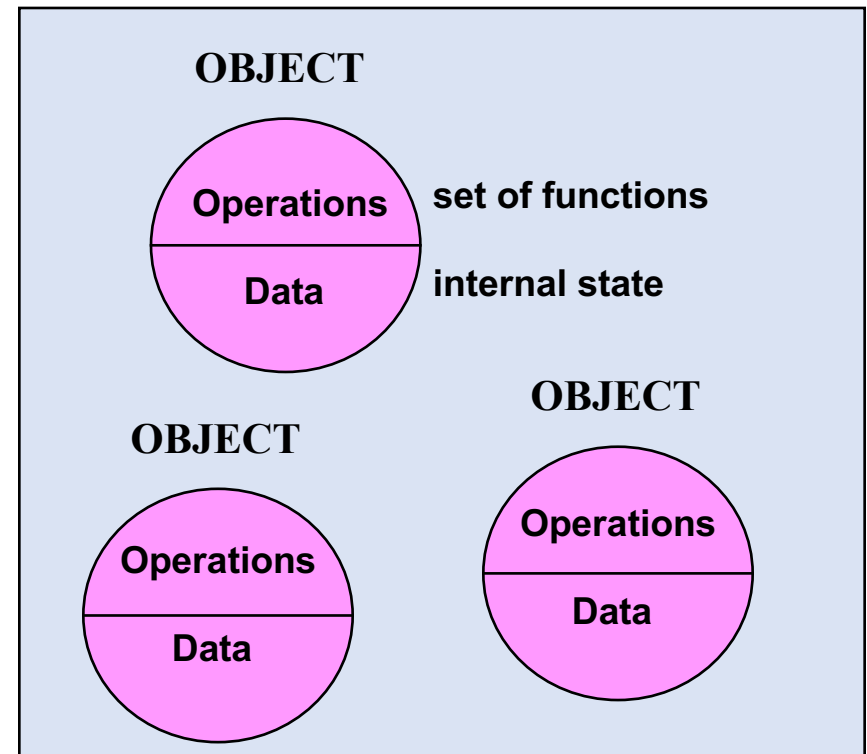
- object-oriented programming (OOP)
  - encapsulates data (attributes) and functions (behavior) into classes.
- so, classes are user-defined (programmer-defined) types.
  - data (data members)
  - functions (member functions or methods)
- they look like a structure with member functions
- programmer defines the attributes and behavior of objects.
- often the objects are modeled after real-world entities.
- very different approach than *function-based* programming (like C).

# two programming methodologies

## functional decomposition



## object-oriented design



# Python Object-Oriented Programming

- ex) A bird can be an object, as it has the following properties:
  - name, age, color as attributes
  - singing, flying as behavior

```
class Bird:
    def __init__(self, name, age, color):
        self.name = name
        self.age = age
        self.color = color
    def sing(self):
        print("Tweets! ")
    def flying(self):
        print(self.name + ' is flying...')
b0 = Bird('John', 3, 'yellow')
b0.sing()
b0.flying()
```

```
Tweets!
John is flying...
```

# inheritance and overriding

```
class Bird:
    def __init__(self, name, age, color):
        self.name = name
        self.age = age
        self.color = color
    def sing(self):
        print("Tweets! ")
    def flying(self):
        print(self.name + ' is flying...')
b0 = Bird('John', 3, 'yellow')
b0.sing()
b0.flying()
```

Tweets!  
John is flying...

```
class Parrot(Bird):
    def __init__(self, name, age, character):
        Bird.__init__(self, name, age, 'blue')
        self.character = character

class Crow(Bird):
    def __init__(self, name, age):
        Bird.__init__(self, name, age, 'black')
    def sing(self): # overriding
        print("krrrrrrrrrr")
```

```
c0 = Crow('John', 3)
c0.sing()
c0.flying()
p1 = Parrot("Hanse", 3, "mimic")
p1.sing()
p1.flying()
```

krrrrrrrrrr  
John is flying...  
Tweets!  
Hanse is flying...

# Python Numpy



- **Numpy** is the core library for scientific computing in Python.
  - It provides a high-performance multidimensional array object, tools for working with these arrays, and simple yet powerful data structure: the n-dimensional array.
  - The NumPy API is used extensively in Pandas, SciPy, Matplotlib, scikit-learn, scikit-image and most other data science and scientific Python packages.
  - **ndarray object**  
This encapsulates n-dimensional arrays of homogeneous data types, considering the mathematical operations

# Numpy

- How to install Numpy?

- conda install numpy
- pip install numpy

- How to use Numpy?

- `import numpy as np`



Q Search the docs ...

What is NumPy?  
Installation  
NumPy quickstart  
NumPy: the absolute basics for beginners  
NumPy basics  
Miscellaneous  
NumPy for MATLAB users  
Building from source  
Using NumPy C-API  
NumPy Tutorials  
NumPy How Tos  
Explanations  
F2PY Users Guide and Reference Manual  
Glossary  
Under-the-hood Documentation for developers  
NumPy's Documentation  
Reporting bugs  
Release Notes  
Documentation conventions  
NumPy license

## NumPy user guide

This guide is an overview and explains the important features; details are found in [NumPy Reference](#).

- [What is NumPy?](#)
- [Installation](#)
- [NumPy quickstart](#)
- [NumPy: the absolute basics for beginners](#)
- [NumPy basics](#)
- [Miscellaneous](#)
- [NumPy for MATLAB users](#)
- [Building from source](#)
- [Using NumPy C-API](#)
- [NumPy Tutorials](#)
- [NumPy How Tos](#)

<< NumPy Documentation

What is NumPy? >>

# Numpy Methods

## Adding/removing Elements

`np.append(arr, values)` | Appends values to end of `arr`  
`np.insert(arr, 2, values)` | Inserts values into `arr` before index 2  
`np.delete(arr, 3, axis=0)` | Deletes row on index 3 of `arr`  
`np.delete(arr, 4, axis=1)` | Deletes column on index 4 of `arr`

## Combining/splitting

`np.concatenate((arr1, arr2), axis=0)` | Adds `arr2` as rows to the end of `arr1`  
`np.concatenate((arr1, arr2), axis=1)` | Adds `arr2` as columns to end of `arr1`  
`np.split(arr, 3)` | Splits `arr` into 3 sub-arrays  
`np.hsplit(arr, 5)` | Splits `arr` horizontally on the 5th index

## Statistics

`np.mean(arr, axis=0)` | Returns mean along specific axis  
`arr.sum()` | Returns sum of `arr`  
`arr.min()` | Returns minimum value of `arr`  
`arr.max(axis=0)` | Returns maximum value of specific axis  
`np.var(arr)` | Returns the variance of array  
`np.std(arr, axis=1)` | Returns the standard deviation of specific axis  
`arr.corrcoef()` | Returns correlation coefficient of array

## Vector Math

`np.add(arr1, arr2)` | Elementwise add `arr2` to `arr1`  
`np.subtract(arr1, arr2)` | Elementwise subtract `arr2` from `arr1`  
`np.multiply(arr1, arr2)` | Elementwise multiply `arr1` by `arr2`  
`np.divide(arr1, arr2)` | Elementwise divide `arr1` by `arr2`  
`np.power(arr1, arr2)` | Elementwise raise `arr1` raised to the power of `arr2`  
`np.array_equal(arr1, arr2)` | Returns `True` if the arrays have the same elements and shape  
`np.sqrt(arr)` | Square root of each element in the array  
`np.sin(arr)` | Sine of each element in the array  
`np.log(arr)` | Natural log of each element in the array  
`np.abs(arr)` | Absolute value of each element in the array  
`np.ceil(arr)` | Rounds up to the nearest int  
`np.floor(arr)` | Rounds down to the nearest int  
`np.round(arr)` | Rounds to the nearest int

# Numpy Methods

- Create Array

**numpy.array**(object, dtype=None, \*, copy=True, order='K', subok=False, ndmin=0, like=None)

Create an array.

```
In [11]: np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15], dtype=int)
Out[11]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

**numpy.zeros**(shape, dtype=float, order='C', \*, like=None)

Return a new array of given shape and type, filled with zeros.

```
In [12]: np.zeros((3,5), dtype=int)
Out[12]: array([[0, 0, 0, 0, 0],
               [0, 0, 0, 0, 0],
               [0, 0, 0, 0, 0]])
```

**numpy.ones**(shape, dtype=None, order='C', \*, like=None)

Return a new array of given shape and type, filled with ones.

```
In [13]: np.ones((2,10), dtype=int)
Out[13]: array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
               [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
```



# Numpy Methods

- Create Array

**random.rand**(d0, d1, ..., dn)

Random values in a given shape.

```
In [14]: np.random.rand(3,4)
Out[14]: array([[0.77340866, 0.25094062, 0.94552963, 0.41094501],
                [0.91914791, 0.6209341 , 0.04587106, 0.16780006],
                [0.23261847, 0.82662203, 0.19127386, 0.60003303]])
```

**random.randn**(d0, d1, ..., dn)

Return a sample (or samples) from the “standard normal(mean 0, std 1)” distribution.

```
In [15]: np.random.randn(3,5)
Out[15]: array([[ 0.00955529, -0.25530476, -0.26400496, -0.42225605,  0.41110185],
                [-0.21870330,  1.21030862, -1.37108564, -1.52318496, -0.66407713],
                [ 0.09012102, -1.0090723 , -0.46048102, -0.03360769,  1.61832296]])
```

# Numpy Methods

- Methods

```
In [16]: ndarray=np.ones((2,4))
          ndarray
Out[16]: array([[1., 1., 1., 1.],
               [1., 1., 1., 1.]])
```

## **ndarray.shape**

Tuple of array dimensions.

```
In [17]: ndarray.shape
Out[17]: (2, 4)
```

## **ndarray.ndim**

Number of array dimensions.

```
In [18]: ndarray.ndim
Out[18]: 2
```

# Numpy Methods

- Methods

**ndarray.reshape(shape, order='C')**

Returns an array containing the same data with a new shape.

```
In [19]: ndarray.reshape(4,2)
Out [19]: array([[1., 1.],
                [1., 1.],
                [1., 1.],
                [1., 1.]])
```

cf) `ndarray.reshape(1,-1)` : One shape dimension can be -1. In this case, the value is inferred from the length of the array and remaining dimensions

```
In [21]: ndarray.reshape(1,-1)
Out [21]: array([[1., 1., 1., 1., 1., 1., 1., 1.]])
```

# Numpy Methods

- Methods

## `ndarray.dtype`

Data-type of the array's elements.

```
In [20]: ndarray.dtype
Out [20]: dtype('float64')
```

## `numpy.arange([start, ]stop, [step, ]dtype=None, *, like=None)`

Return evenly spaced values within a given interval.

```
In [23]: ndarray1=np.arange(2,22).reshape(4,5)
ndarray1
Out [23]: array([[ 2,  3,  4,  5,  6],
 [ 7,  8,  9, 10, 11],
 [12, 13, 14, 15, 16],
 [17, 18, 19, 20, 21]])
```

```
In [24]: ndarray2=np.arange(0,5).reshape(5,1)
ndarray2
Out [24]: array([[0],
 [1],
 [2],
 [3],
 [4]])
```

# Numpy Methods

- Methods

**ndarray.dot(a, b, out=None)**

Dot product of two arrays.

```
In [25]: ndarray3=nd.dot(ndarray1,ndarray2)
         ndarray3
Out[25]: array([[ 50],
               [100],
               [150],
               [200]])
```

**ndarray.T**

The transposed array.

```
In [26]: ndarray3.T
Out[26]: array([[ 50, 100, 150, 200]])
```

```
In [27]: ndarray4=ndarray3.reshape(2,2)
         ndarray4
Out[27]: array([[ 50, 100],
               [150, 200]])
```

```
In [28]: ndarray4.T
Out[28]: array([[ 50, 150],
               [100, 200]])
```

# Numpy Methods

- Methods

**numpy.add**(x1, x2, /, out=None, \*, where=True, casting='same\_kind', order='K', dtype=None, subok=True[, signature, extobj]) = <ufunc 'add'>  
Add arguments element-wise.

```
In [29]: np.add(ndarray4, ndarray4)
Out[29]: array([[100, 200],
               [300, 400]])
```

**numpy.subtract**(x1, x2, /, out=None, \*, where=True, casting='same\_kind', order='K', dtype=None, subok=True[, signature, extobj]) = <ufunc 'subtract'>  
Subtract arguments, element-wise.

```
In [30]: np.subtract(ndarray4, ndarray4.T)
Out[30]: array([[ 0, -50],
               [ 50,  0]])
```

# Numpy Methods

- Methods

**numpy.multiply**(x1, x2, /, out=None, \*, where=True, casting='same\_kind', order='K', dtype=None, subok=True[, signature, extobj]) = *<ufunc 'multiply'>*

Multiply arguments element-wise.

```
In [31]: np.multiply(ndarray4, ndarray4.T)
```

```
Out[31]: array([[ 2500, 15000],
               [15000, 40000]])
```

**numpy.divide**(x1, x2, /, out=None, \*, where=True, casting='same\_kind', order='K', dtype=None, subok=True[, signature, extobj]) = *<ufunc 'true\_divide'>*

Returns a true division of the inputs, element-wise.

```
In [32]: np.divide(ndarray4, ndarray4.T)
```

```
Out[32]: array([[1.         , 0.66666667],
               [1.5        , 1.         ]])
```

# Numpy Methods

- **Broadcasting** : How Numpy treats arrays with different shapes during arithmetic operations.
  - the smaller array is “broadcast” across the larger array so that they have compatible shapes

```
In [36]: 1 ndarray4=ndarray4+1
          2 ndarray4
```

```
Out[36]: array([[ 51, 101],
                [151, 201]])
```

```
In [37]: 1 ndarray4=ndarray4-5
          2 ndarray4
```

```
Out[37]: array([[ 46,  96],
                [146, 196]])
```

```
In [38]: 1 ndarray4=ndarray4*2
          2 ndarray4
```

```
Out[38]: array([[ 92, 192],
                [292, 392]])
```

```
In [54]: 1 ndarray4=ndarray4/3
          2 ndarray4
```

```
Out[54]: array([[ 30.66666667,  64.          ],
                [ 97.33333333, 130.66666667]])
```

```
In [55]: 1 ndarray4+[5,8]
```

```
Out[55]: array([[ 35.66666667,  72.          ],
                [102.33333333, 138.66666667]])
```

```
In [56]: 1 ndarray4-[[4],
          2             [2]]
          3
```

```
Out[56]: array([[ 26.66666667,  60.          ],
                [ 95.33333333, 128.66666667]])
```



# Numpy Methods

- **subarray**

```
ndarray1=np.arange(2,22).reshape(4,5)
ndarray1
```

```
array([[ 2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16],
       [17, 18, 19, 20, 21]])
```

```
index = ndarray1 < 10
index
```

```
array([[ True,  True,  True,  True,  True],
       [ True,  True,  True, False, False],
       [False, False, False, False, False],
       [False, False, False, False, False]])
```

```
ndarray1[index]
```

```
array([2, 3, 4, 5, 6, 7, 8, 9])
```

```
idx1 = ndarray1[:,0] > 10
ndarray1[idx1,:]
```

```
array([[12, 13, 14, 15, 16],
       [17, 18, 19, 20, 21]])
```

# Retrieving Data in Python

```
import numpy as np
import openpyxl # if you don't have this, then pip install openpyxl first
file= openpyxl.load_workbook('data_set_train.xlsx') # assuming that the file is in the current directory
ws = file.active

data=[] # list type
col_name=[] # list type
for row in ws.iter_rows(max_row=1): # read the first row
    for cell in row:
        col_name.append(cell.value)
for row in ws.iter_rows(min_row=2): # read from the 2nd row
    one_line=[]
    for cell in row:
        one_line.append(cell.value)
    data.append(one_line)
```

```
In [68]: data[:1]
Out[68]: [['강남역우정애썬트',
          '200901',
          9000.0,
          2004.0,
          '역삼동',
          17.23,
          7.0,
          37.4942041,
          127.0435446,
          225613.0,
          6.3,
          0.152880864,
          5.55949463170584,
          452.717843152131,
          794.544848855622,
          231.845431815105,
          2095.0282409097,
          291.140809061849,
          1694.41981012148,
          849.353652859484,
          0.0,
          52.0,
          1.0,
          0.75,
          '개별난방',
          536.0,
          58.0,
          13.0,
          12.0]]
```

# Numpy Methods

- Converting list to numpy ndarray

In [27]:

```
1 print(type(data))  
2 arr=np.array(data)  
3 print(type(arr))
```

<class 'list'>

<class 'numpy.ndarray'>

In [33]:

```
1 arr.shape
```

Out[33]: (17400, 29)

# Numpy Methods

```
In [39]: 1 data_set= arr[:,9]
          2 data_set
```

```
Out[39]: array([[ '강남역우정에쉐르', '2006Q1', '9000.0', ..., '7.0', '37.4942041',
                  '127.0435446'],
                [ '강남역우정에쉐르', '2006Q1', '9000.0', ..., '7.0', '37.4942041',
                  '127.0435446'],
                [ '개포주공1단지', '2006Q1', '73000.0', ..., '3.0', '37.4784072',
                  '127.061375'],
                ...,
                [ '현대하이츠', '2017Q3', '64000.0', ..., '1.0', '37.4913789',
                  '127.0348797'],
                [ '현대한강', '2017Q3', '170000.0', ..., '8.0', '37.5246752',
                  '127.056226'],
                [ '현대한강', '2017Q3', '170000.0', ..., '8.0', '37.5246752',
                  '127.056226']], dtype='<U32')
```

```
In [40]: 1 data_set.shape
```

```
Out[40]: (17400, 9)
```

## Exercise(2) – Slice only 2006

- Slice array based on a condition
  - Condition that only meets 2006 in column "yyyyqrt(거래년도 분기별)"
  - Slice rows that satisfy the condition.

```
In [11]:  
Out[11]: array([[ '강남역우정메퀘트', '200601', '9000.0', ..., '7.0', '37.4942041',  
                  '127.0435446'],  
               [ '강남역우정메퀘트', '200601', '9000.0', ..., '7.0', '37.4942041',  
                  '127.0435446'],  
               [ '개포주공1단지', '200601', '73000.0', ..., '3.0', '37.4784072',  
                  '127.061375'],  
               ...  
               [ '현대빌라트', '200604', '125000.0', ..., '3.0', '37.5175279',  
                  '127.0428004'],  
               [ '현대빌라트', '200604', '125000.0', ..., '3.0', '37.5175279',  
                  '127.0428004'],  
               [ '현대빌라트', '200604', '125000.0', ..., '3.0', '37.5175279',  
                  '127.0428004']], dtype='<U18')
```

- data\_set2006.shape

```
In [49]: data_set2006.shape  
Out[49]: (2617, 9)
```

# Numpy Methods

**`ndarray.astype(dtype, order='K', casting='unsafe', subok=True, copy=True)`**  
Copy of the array, cast to a specified type.

```
In [51]: 1 price = data_set2006[:,2]
          2 price.dtype
```

Out[51]: dtype('<U32') ← not a number type (32 character unicode), needs to be casted.

```
In [45]: 1 print(data_set2006[0])
          2
          3
          4 price = data_set2006[:,2].astype(float)
          5 lat = data_set2006[:,7].astype(float)
          6 lng = data_set2006[:,8].astype(float)
```

```
['강남역우정에쉐르' '2006Q1' '9000.0' '2004.0' '역삼동' '17.23' '7.0' '37.4942041'
 '127.0435446']
```

# Numpy Methods

**numpy.amax**(*a*, *axis=None*, *out=None*, *keepdims=<no value>*,  
*initial=<no value>*, *where=<no value>*)[\[source\]](#)

Return the maximum of an array or maximum along an axis.

**numpy.mean**(*a*, *axis=None*, *dtype=None*, *out=None*,  
*keepdims=<no value>*, \*, *where=<no value>*)[\[source\]](#)

Compute the arithmetic mean along the specified axis.

**numpy.amin**(*a*, *axis=None*, *out=None*, *keepdims=<no value>*,  
*initial=<no value>*, *where=<no value>*)[\[source\]](#)

Return the minimum of an array or minimum along an axis.

```
In [53]: 1 np.amax(price)
```

```
Out[53]: 310000.0
```

```
In [54]: 1 np.mean(price)
```

```
Out[54]: 67076.57928926252
```

```
In [55]: 1 np.amin(price)
```

```
Out[55]: 5500.0
```

## Exercise(3) – Convert type of ndarrays

- Find correlation between “price” and “dis\_subway”
  - Slice only dis\_subway column values `arr[:, 2]` and `arr[:,19]`
  - Slice only price column values
  - Convert datatype of two ndarray (string to float).
  - Use `corrcoef` method.

**`numpy.corrcoef`**(*x*, *y=None*, *rowvar=True*, *bias=<no value>*, *ddof=<no value>*, \*, *dtype=None*)[\[source\]](#)  
Return Pearson product-moment correlation coefficients.

```
In [74]: np.corrcoef(price, station)
Out[74]: array([[1., 0.03241268],
                [0.03241268, 1.]])
```



# Numpy Methods

**numpy.concatenate**((a1, a2, ...), axis=0, out=None, dtype=None, casting="same\_kind")  
Join a sequence of arrays along an existing axis.

- Combine data\_set and “dis\_subway”

```
station=arr[:,19]
print(data_set.shape)
print(station.shape)
station=station.reshape(-1,1)
print(station.shape)
```

```
(17400, 9)
(17400,)
(17400, 1)
```

```
concat=np.concatenate((data_set,station),axis=1) # axis indicates how to concatenate
print(concat.shape)
print(concat[0])
```

```
(17400, 10)
['강남역우정에셰르' '2006Q1' '9000.0' '2004.0' '역삼동' '17.23' '7.0' '37.4942041'
 '127.0435446' '849.353652859484']
```

# Python Matplotlib



- **Matplotlib** is one of the most popular and oldest plotting libraries in Python which is used in Machine Learning.
  - It helps to understand the huge amount of data through different visualizations.
  - It consists of several plots like the Line Plot, Bar Plot, Scatter Plot, Histogram etc. through which we can visualize various types of data.

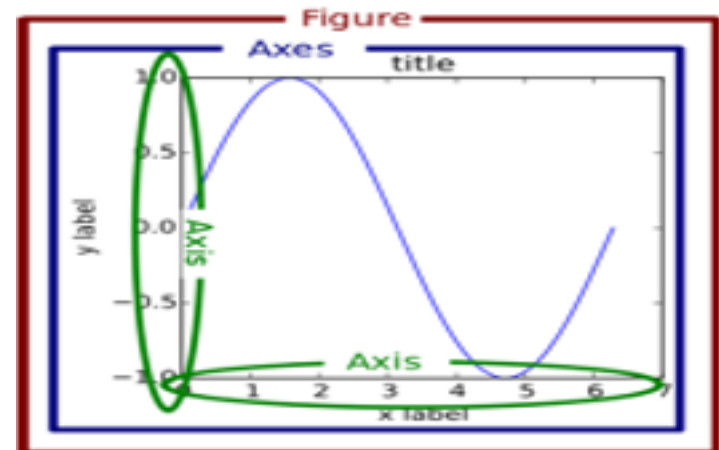
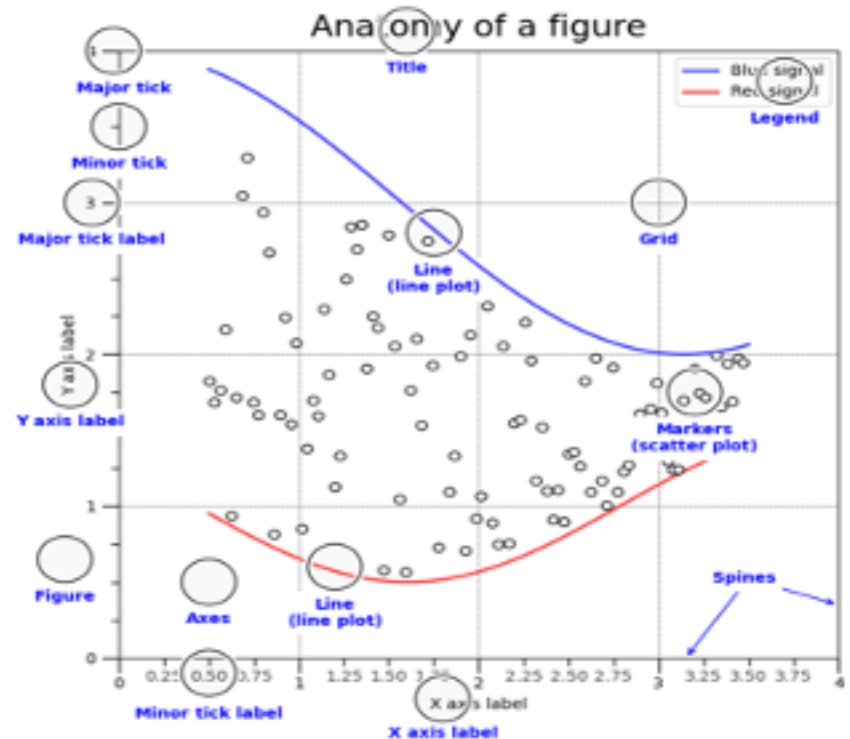
# Matplotlib

- How to install Matplotlib?
  - `conda install matplotlib`
  - `pip install matplotlib`
- How to use Matplotlib?
  - `import matplotlib.pyplot as plt`

# Matplotlib

<https://realpython.com/python-matplotlib-guide/>

- Concepts in Matplotlib
  - **Figure:** Top level container for all plot elements
  - **Axes:** An area where plot appears in
  - **Legend:** An area describing the elements of the graph
  - **Grid:** line that show axis divisions
  - **Plot:** draw lines or markers.
  - **Axis:** A reference line drawn on a graph



# Matplotlib Methods List

[tripcolor](#)

Create a pseudocolor plot of an unstructured triangular grid.

[triplot](#)

Draw a unstructured triangular grid as lines and/or markers.

[twinx](#)

Make and return a second axes that shares the x-axis.

[twiny](#)

Make and return a second axes that shares the y-axis.

[uninstall\\_repl\\_displayhook](#)

Uninstall the matplotlib display hook.

[violinplot](#)

Make a violin plot.

[vlines](#)

Plot vertical lines.

[xcorr](#)

Plot the cross correlation between x and y.

[xkcd](#)

Turn on [xkcd](#) sketch-style drawing mode.

[xlabel](#)

Set the label for the x-axis.

[xlim](#)

Get or set the x limits of the current axes.

[xscale](#)

Set the x-axis scale.

[xticks](#)

Get or set the current tick locations and labels of the x-axis.

[ylabel](#)

Set the label for the y-axis.

[ylim](#)

Get or set the y-limits of the current axes.

[yscale](#)

Set the y-axis scale.

[yticks](#)

Get or set the current tick locations and labels of the y-axis.

[subplot](#)

Add a subplot to the current figure.

[subplot2grid](#)

Create a subplot at a specific location inside a regular grid.

[subplot\\_mosaic](#)

Build a layout of Axes based on ASCII art or nested lists.

[subplot\\_tool](#)

Launch a subplot tool window for a figure.

[subplots](#)

Create a figure and a set of subplots.

[subplots\\_adjust](#)

Adjust the subplot layout parameters.

[suptitle](#)

Add a centered title to the figure.

[switch\\_backend](#)

Close all open figures and set the Matplotlib backend.

[table](#)

Add a table to an [Axes](#).

[text](#)

Add text to the axes.

[thetagrids](#)

Get or set the theta gridlines on the current polar plot.

[tick\\_params](#)

Change the appearance of ticks, tick labels, and gridlines.

[ticklabel\\_format](#)

Configure the [ScalarFormatter](#) used by default for linear axes.

[tight\\_layout](#)

Adjust the padding between and around subplots.

[title](#)

Set a title for the axes.

[tricontour](#)

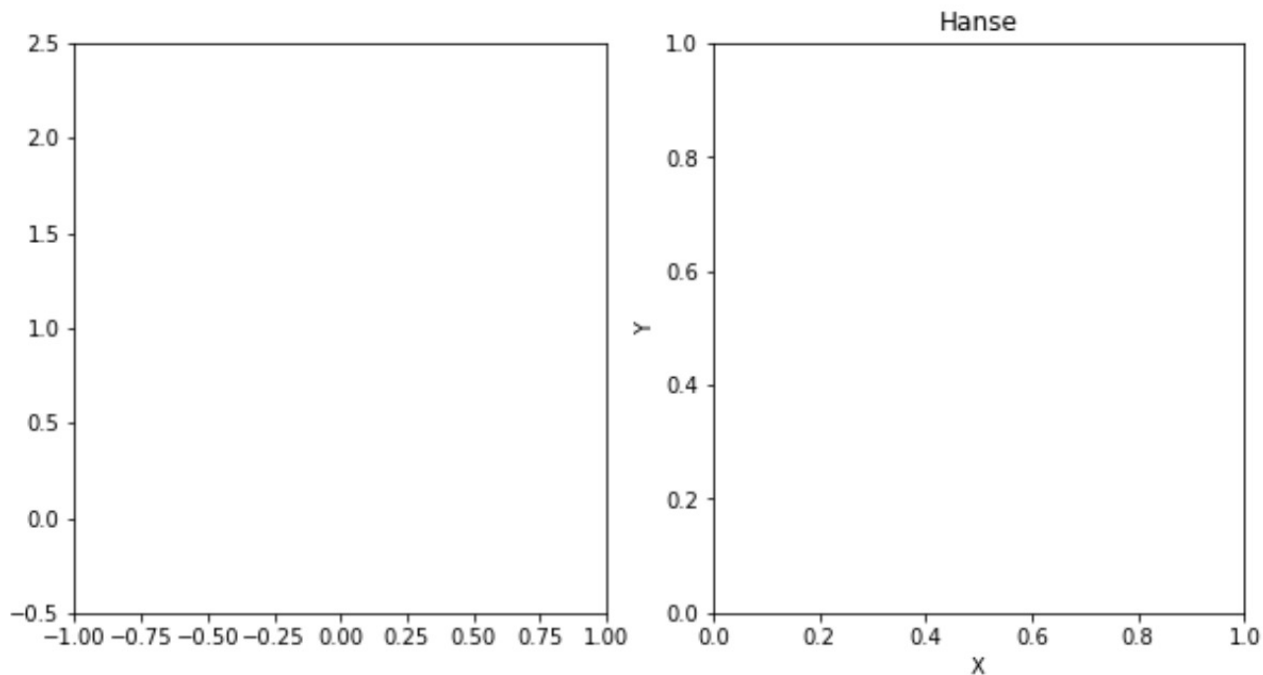
Draw contour lines on an unstructured triangular grid.

[tricontourf](#)

Draw contour regions on an unstructured triangular grid.

# Matplotlib Methods

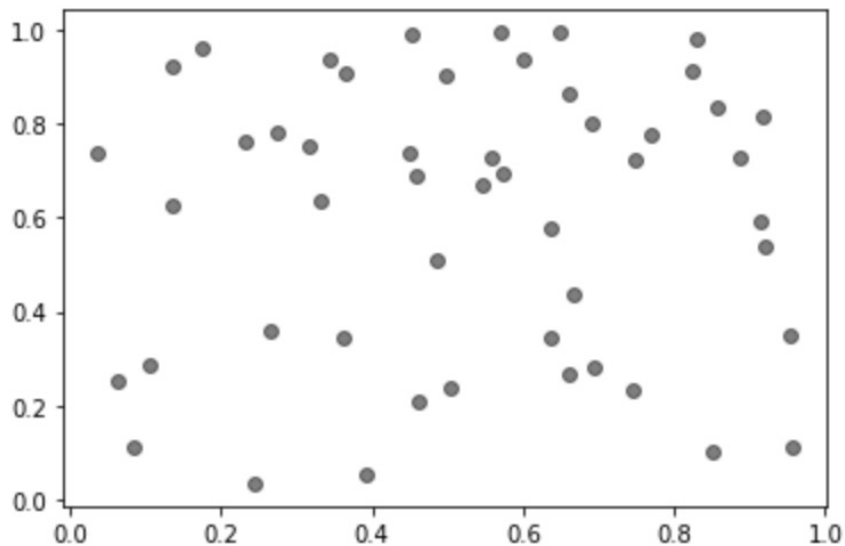
```
1 plt.figure(figsize=(10,5)) # Create figure
2 plt.subplot(1,2,1) # add a subplot to the current figure
3 plt.xlim([-1., 1.]) # set the x limits of the current axes
4 plt.ylim([-0.5, 2.5]) # set the y limits of the current axes.
5
6 plt.subplot(1,2,2) # 2nd subplot
7 plt.ylabel('Y') # set the y label
8 plt.xlabel('X') # set the x label
9 plt.title("Hanse") # Set the title
10 plt.show() # display this figure
```



# Matplotlib Methods

- Scatter plot

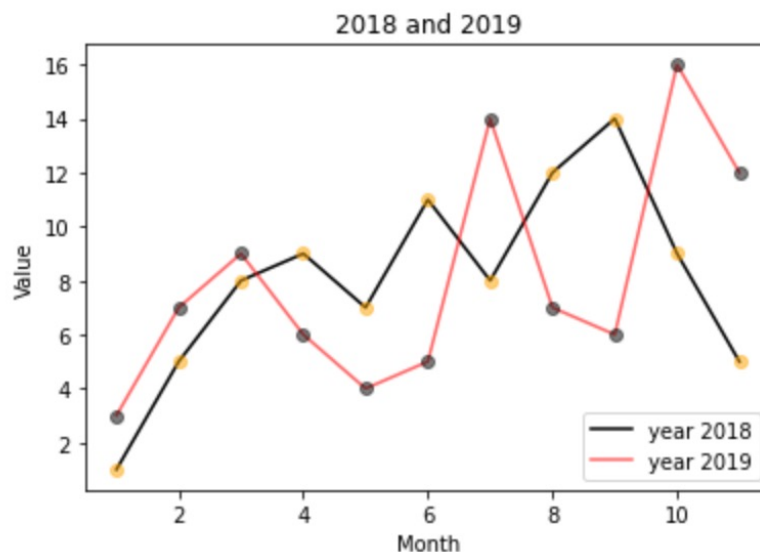
```
1 np.random.seed(500)
2 N = 50
3 x = np.random.rand(N)
4 y = np.random.rand(N)
5
6 #creating scatter plot
7 plt.scatter(x, y, c='black', alpha=0.5)
8 plt.show()
```



# Matplotlib Methods

- Line and point plot

```
1 year2018 = [1, 5, 8, 9, 7, 11, 8, 12, 14, 9, 5]
2 year2019 = [3, 7, 9, 6, 4, 5, 14, 7, 6, 16, 12]
3 #plotting line plots
4 plt.plot(range(1,12), year2018,color='black')
5 plt.plot(range(1,12), year2019,color='r',alpha=0.6)
6 #creating points
7 plt.plot(range(1,12), year2019, 'COo', alpha=0.5,color='black')
8 plt.plot(range(1,12), year2018, 'COo', alpha=0.5,color='orange')
9 #creating title,Y-label,X-label & legend
10 plt.title('2018 and 2019')
11 plt.ylabel('Value')
12 plt.xlabel('Month')
13 plt.legend(['year 2018', 'year 2019'], loc=4)
14 plt.show() # plot shows automatically in Jupyter
```





# Week 3 Exercise(4) : Yearly Average Price

- output

- Print yearly average price
- Note that “yyyyqrt(거래년도 분기별)” values consist of year and quarter.

```
2006 avg : 67076.58
2007 avg : 58405.72
2008 avg : 66136.67
2009 avg : 83060.50
2010 avg : 77282.03
2011 avg : 76748.97
2012 avg : 72714.52
2013 avg : 78839.02
2014 avg : 84002.70
2015 avg : 87213.44
2016 avg : 97149.95
2017 avg : 114656.79
```

# Week 3 Exercise(5) : Quarterly Average Price

- output

- Print quarterly average price

2006Q1 avg : 65661.17	2006Q2 avg : 59592.49	2006Q3 avg : 62410.72	2006Q4 avg : 79588.83
2007Q1 avg : 47336.40	2007Q2 avg : 65344.61	2007Q3 avg : 58634.69	2007Q4 avg : 59687.89
2008Q1 avg : 63515.58	2008Q2 avg : 62458.85	2008Q3 avg : 72233.96	2008Q4 avg : 72608.62
2009Q1 avg : 85574.77	2009Q2 avg : 84314.42	2009Q3 avg : 83246.47	2009Q4 avg : 78589.30
2010Q1 avg : 80386.73	2010Q2 avg : 71885.82	2010Q3 avg : 72697.58	2010Q4 avg : 82475.98
2011Q1 avg : 77359.50	2011Q2 avg : 68071.24	2011Q3 avg : 78499.77	2011Q4 avg : 82607.86
2012Q1 avg : 73633.61	2012Q2 avg : 73102.35	2012Q3 avg : 78050.49	2012Q4 avg : 69313.77
2013Q1 avg : 77980.51	2013Q2 avg : 76391.85	2013Q3 avg : 76638.47	2013Q4 avg : 84570.14
2014Q1 avg : 84038.77	2014Q2 avg : 81931.40	2014Q3 avg : 79218.36	2014Q4 avg : 90134.39
2015Q1 avg : 76180.14	2015Q2 avg : 81677.28	2015Q3 avg : 98509.43	2015Q4 avg : 93279.71
2016Q1 avg : 87456.07	2016Q2 avg : 94492.74	2016Q3 avg : 104774.82	2016Q4 avg : 102236.93
2017Q1 avg : 113074.15	2017Q2 avg : 110766.67	2017Q3 avg : 124098.44	

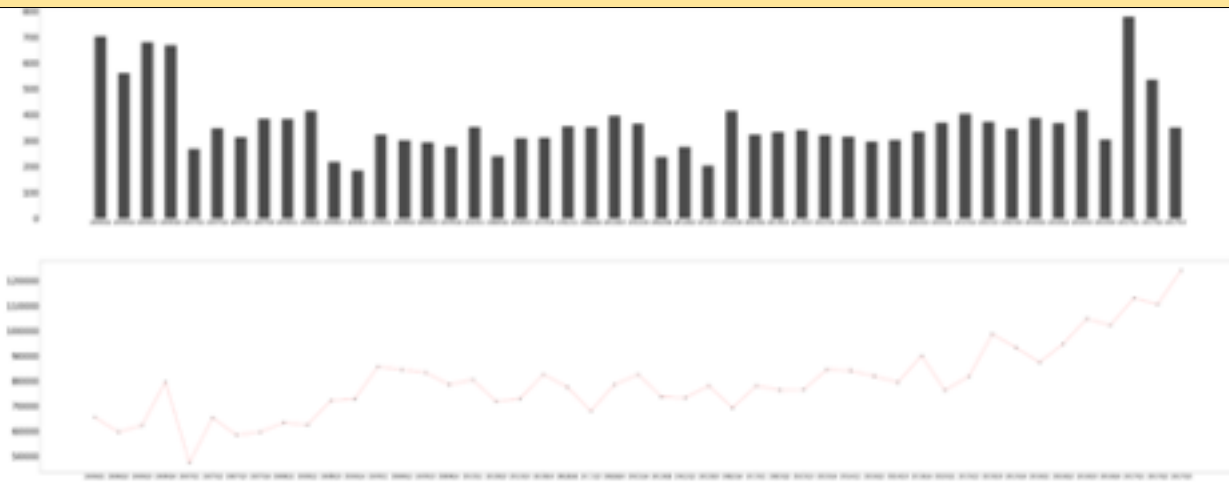
# Week 3 Exercise(6) : Quarterly Price Graph & Histogram

- Output (use subplot)

- Plot the count of trading in quarters, (quarterly on x-axis, counts on y-axis.)
- Draw bar graph using counted values in quarterly

use `matplotlib.pyplot.bar(x, height, width=0.5, bottom=None, *, align='center', data=None, **kwargs)`

- Draw quarterly average price.



# References

<https://www.programiz.com/python-programming/global-local-nonlocal-variables#:~:text=In%20Python%2C%20a%20variable%20declared,variable%20is%20created%20in%20Python.>

[https://www.w3schools.com/python/python\\_lambda.asp](https://www.w3schools.com/python/python_lambda.asp)

<https://docs.python.org/3/library/functions.html>

<https://www.programiz.com/python-programming/object-oriented-programming>

<https://numpy.org/doc/stable/reference/>

<https://www.dataquest.io/blog/numpy-cheat-sheet/>

[https://www.w3schools.com/python/numpy\\_data\\_types.asp](https://www.w3schools.com/python/numpy_data_types.asp)

<https://www.javatpoint.com/numpy-datatypes>

[https://matplotlib.org/3.1.1/api/pyplot\\_summary.html](https://matplotlib.org/3.1.1/api/pyplot_summary.html)