파이썬을 이용한 데이터과학과 머신러닝 입문

Introduction to Python for Data Science & Machine Learning

August, 2019

Ein Intelligence

* *Microsoft DAT208x Introduction to Python for Data Science 교육 자료에 기반하여 작성*

CONTENTS

파이썬 및 구글 코랩 소개

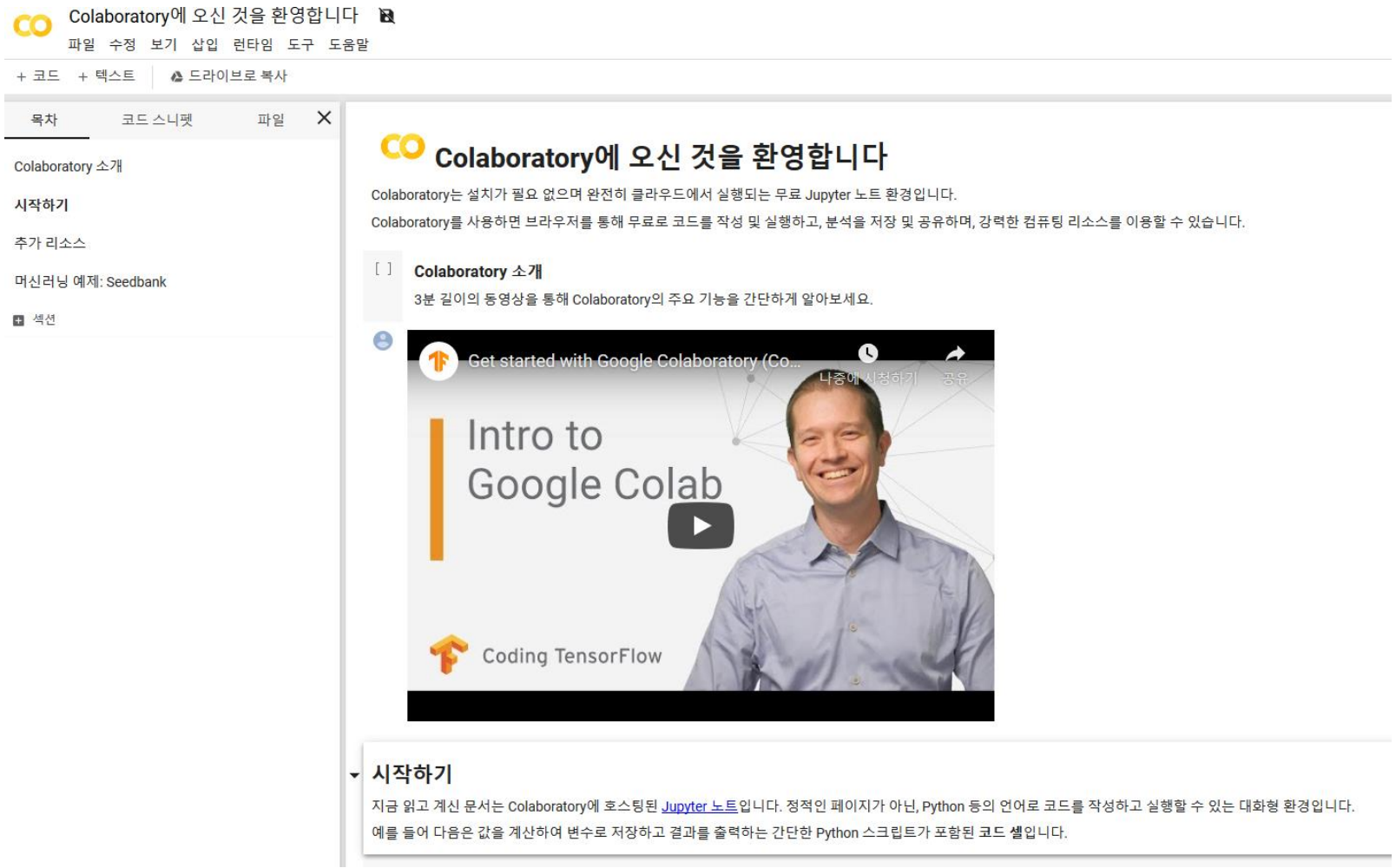
Introduction : Python & Google CoLab

파이썬

- 1991년 네덜란드의 귀도 반 로썸(Guido van Rossum) 이 창시
- 일반용도의 언어 (General Purpose)
- 오픈 소스 & 무료 (비영리 파이썬 소프트웨어 재단이 관리)
- 인터프리터 (vs. 컴파일러)
- 데이터 처리에 적합
- 과학 응용분야 라이브러리 : Numpy, Matplotlib, Pandas, Scikit Learn
- Download at https://www.python.org/
- Version 2 vs. version 3 (Lastest : Python 3.7.4)
- Python Script : ~~~~.py (vs. ~~~~.ipynb of Jupyter Notebook)

Colaboratory

https://colab.research.google.com/



코랩 소개

- Jupyter Notebook 개발 환경의 웹 서비스 버전
- 정적인 페이지가 아닌, Python 등의 언어로 코드를 작성하고 실행할 수 있는 대화형 환경

  → 실시간으로 데이터 처리 결과를 눈으로 보면서 작업 가능
- iPython 기반 (ipynb vs. py)
- 깃허브 (GitHub) 및 구글드라이브 (Google Drive) 연동 가능
- 일반 데이터 파일 업로드/다운로드 가능
- 파이썬 모듈 import 하여 처리 가능

파이썬 기초 (1) : 변수와 자료형

Python Basics (1) : Variables and Types

파이썬 변수

- Specific, case-sensitive name
- Call up value through variable name
- 1.79 m - 68.7 kg

```
In [1]: height = 1.79

In [2]: weight = 68.7

In [3]: height
Out[3]: 1.79
```

체질량 지수 (Body Mass Index) 계산 예

Calculate BMI

```
In [1]: height = 1.79

In [2]: weight = 68.7

In [3]: height
Out[3]: 1.79

In [4]: 68.7 / 1.79 ** 2
Out[4]: 21.4413

In [5]: weight / height ** 2
Out[5]: 21.4413

In [6]: bmi = weight / height ** 2

In [7]: bmi
Out[7]: 21.4413
```

$$\text{BMI} = \frac{\text{weight}}{\text{height}^2}$$

Reproducibility

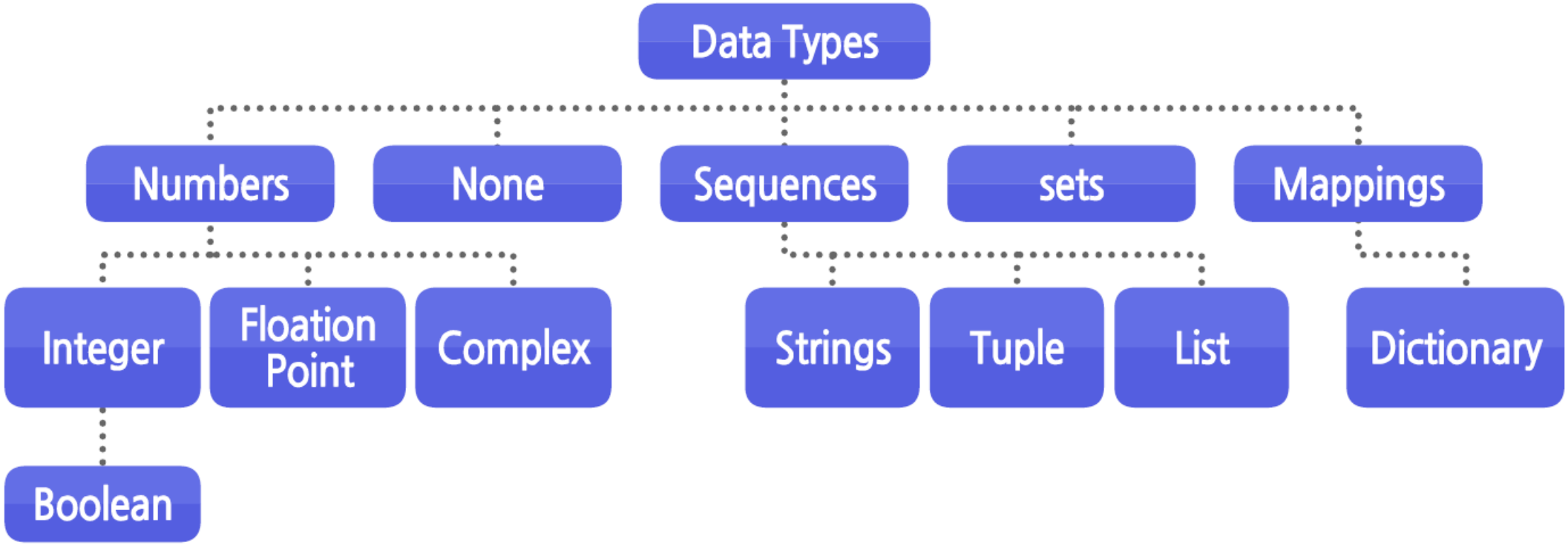
my_script.py

```
height = 1.79
weight = 68.7
bmi = weight / height ** 2
print(bmi)
```

```
Output:
21.4413
```

파이썬 기본 자료형

- `float` - real numbers
- `int` - integer numbers
- `str` - string, text
- `bool` - `True`, `False`



파이썬 자료형 예

Python Types

```
In [8]: type(bmi)
Out[8]: float

In [9]: day_of_week = 5

In [10]: type(day_of_week)
Out[10]: int
```

Python Types (2)

```
In [11]: x = "body mass index"

In [12]: y = 'this works too'

In [13]: type(y)
Out[13]: str

In [14]: z = True

In [15]: type(z)
Out[15]: bool
```

Python Types (3)

```
In [16]: 2 + 3
Out[16]: 5

In [17]: 'ab' + 'cd'
Out[17]: 'abcd'
```

Different type = different behavior!

파이썬 기초 (2) : 리스트와 딕셔너리

Python Basics (2) : Lists & Dictionaries

파이썬 리스트

Problem

- Data Science: many data points
- Height of entire family

```
In [3]: height1 = 1.73

In [4]: height2 = 1.68

In [5]: height3 = 1.71

In [6]: height4 = 1.89
```

- Inconvenient

Python List [a, b, c]

```
In [7]: [1.73, 1.68, 1.71, 1.89]
Out[7]: [1.73, 1.68, 1.71, 1.89]

In [8]: fam = [1.73, 1.68, 1.71, 1.89]

In [9]: fam
Out[9]: [1.73, 1.68, 1.71, 1.89]
```

- Name a collection of values
- Contain any type
- Contain different types

파이썬 리스트

- 이중/다중리스트 : 리스트 안에 리스트 (LoL, List of Lists)

Python List **[a, b, c]** **List type**

```
In [10]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]

In [11]: fam
Out[11]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]


In [11]: fam2 = [["liz", 1.73],
                 ["emma", 1.68],
                 ["mom", 1.71],
                 ["dad", 1.89]]

In [12]: fam2
Out[12]: [['liz', 1.73], ['emma', 1.68],
              ['mom', 1.71], ['dad', 1.89]]
```

```
In [13]: type(fam)
Out[13]: list

In [14]: type(fam2)
Out[14]: list
```

파이썬 리스트 인덱스

- 리스트의 인덱스 : 왼쪽 0부터 시작
- 오른쪽부터 셀 경우 -1부터 작아지는 순서로

```
In [1]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]

In [2]: fam
Out[2]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
    index:   0      1      2      3      4      5      6      7

                          "zero-based indexing"
```

```
In [3]: fam[3]
Out[3]: 1.68

In [4]: fam[6]
Out[4]: 'dad'

In [5]: fam[-1]
Out[5]: 1.89

In [6]: fam[-2]
Out[6]: 'dad'
```

파이썬 리스트 범위와 슬라이싱

- 슬라이싱 : 콜론 ':' 을 이용하여 리스트 내의 중간 범위만을 잘라낼 수 있음
- 오른쪽부터 셀 경우 -1부터 작아지는 순서로

```
In [1]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]

In [2]: fam
Out[2]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
    index:   0      1      2      3      4      5      6      7
```

[start : end]

inclusive exclusive

```
In [8]: fam[3:5]
Out[8]: [1.68, 'mom']

In [9]: fam[1:4]
Out[9]: [1.73, 'emma', 1.68]

In [10]: fam[:4]
Out[10]: ['liz', 1.73, 'emma', 1.68]

In [11]: fam[5:]
Out[11]: [1.71, 'dad', 1.89]
```

파이썬 리스트 조작 (Manipulation)

- 리스트 요소(element)에 대한 조작 예

```
In [2]: fam
Out[2]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]

In [3]: fam[7] = 1.86

In [4]: fam
Out[4]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]

In [5]: fam[0:2] = ["lisa", 1.74]

In [6]: fam
Out[6]: ['lisa', 1.74, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]
```

- 리스트 요소(element)의 추가와 삭제

```
In [7]: fam + ["me", 1.79]
Out[7]: ['lisa', 1.74,'emma', 1.68,
                          'mom', 1.71, 'dad', 1.86, 'me', 1.79]

In [8]: fam_ext = fam + ["me", 1.79]
```

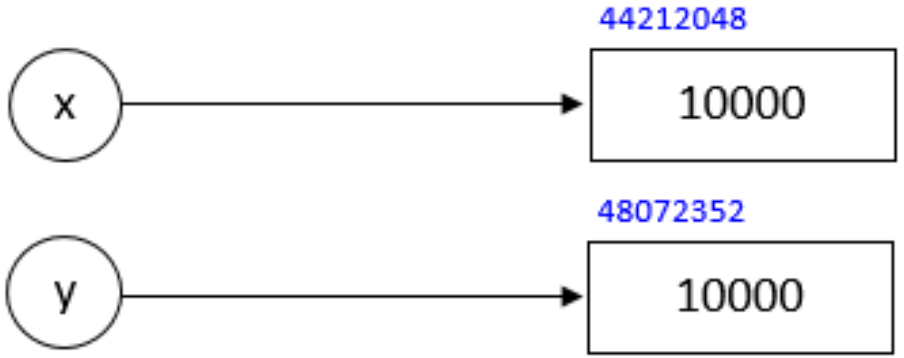
```
In [9]: del(fam[2])

In [10]: fam
Out[10]: ['lisa', 1.74, 1.68, 'mom', 1.71, 'dad', 1.86]

In [11]: del(fam[2])

In [12]: fam
Out[12]: ['lisa', 1.74, 'mom', 1.71, 'dad', 1.86]
```

파이썬 메모리(Memory) & 객체(Object)

- 파이썬에서는 지정된 메모리 주소(Address)에 변수를 넣고, 해당되는 변수와 같은 객체와 연결(Binding)하여 참조/호출

```
>>> x = 100
>>> id(x)
1773787088
```

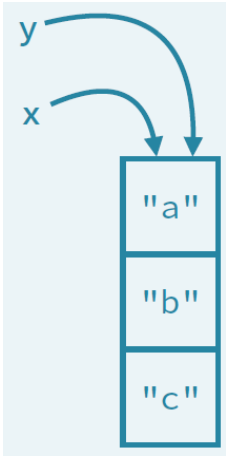

```
>>> x = 10000
>>> y = 10000
>>> id(x), id(y)
(44212048, 48072352)
```



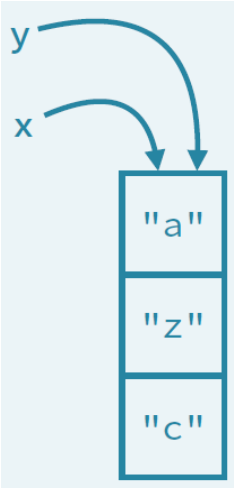
파이썬 객체(Object) 개념 (1)

- 동일한 주소의 메모리를 참조하는 객체의 경우, 원본 내용이 바뀌면 다른 객체의 내용도 따라서 바뀜

```
In [13]: x = ["a", "b", "c"]

In [14]: y = x
```




```
In [15]: y[1] = "z"

In [16]: y
Out[16]: ['a', 'z', 'c']

In [17]: x
Out[17]: ['a', 'z', 'c']
```



파이썬 객체(Object) 개념 (2)

- 따라서, 원본 객체의 내용을 바꾸지 않고 사용하려면 동일 내용만 복사하여 사용

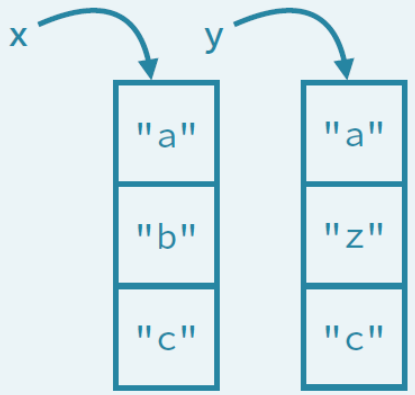
```
new_list = old_list.copy()
```

```
new_list = old_list[:]
```

```
new_list = list(old_list)
```

```
import copy
new_list = copy.copy(old_list)
```

```
In [18]: x = ["a", "b", "c"]

In [19]: y = list(x)

In [20]: y = x[:]

In [21]: y[1] = "z"

In [22]: x
Out[22]: ['a', 'b', 'c']
```



파이썬 기초 (3) : 함수, 메소드 그리고 패키지

Python Basics (3) : Functions, Methods and Packages

Functions

- Nothing new!
- `type()`
- Piece of reusable code
- Solves particular task
- Call function instead of writing code yourself

```
In [1]: fam = [1.73, 1.68, 1.71, 1.89]

In [2]: fam
Out[2]: [1.73, 1.68, 1.71, 1.89]

In [3]: max(fam)
Out[3]: 1.89

In [4]: tallest = max(fam)

In [5]: tallest
Out[5]: 1.89
```

함수의 종류

- 파이썬 함수의 종류

함수의 종류	의미
라이브러리 함수	표준함수 또는 내장함수라고 부르며 시스템에서 미리 작성해 놓은 함수를 의미한다. 삼각함수, 지수함수, 날짜 정보함수, 파일함수, 데이터베이스 함수 등을 말한다.
사용자 정의 함수	프로그램안에서 필요한 기능들을 사용자가 직접 만들어서 사용하는 함수를 말한다.
Built-in 함수, 기본 함수	python설치 시 기본적으로 제공하는 함수를 말하며 print(), type()등의 함수를 말한다.

import *library* (as *alias*)
Import *function* from *library*

def *function*(*parameter*)
.....
return(*result*)

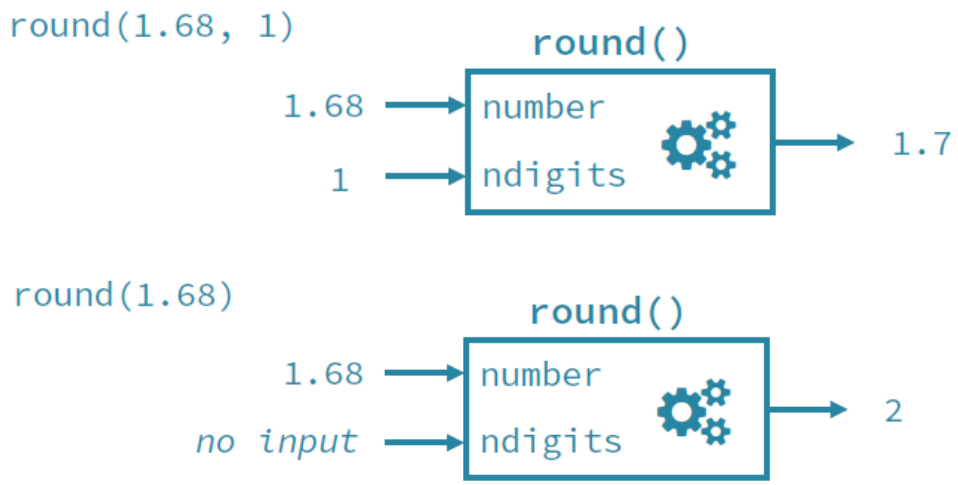
내장 함수

Built-in Functions

- Maximum of list: `max()`
- Length of list or string: `len()`
- Get index in list: ?
- Reversing a list: ?

- **내장함수의 예 : 반올림 함수 round()**

```
In [6]: round(1.68, 1)
Out[6]: 1.7

In [7]: round(1.68)
Out[7]: 2
```



사용자 함수 정의

- 함수의 정의

  - 함수 선언은 def 로 시작
  - 함수의 시작과 끝은 들여쓰기(indentation)로 구분
  - 시작과 끝을 명시하지 않음
  - 함수 이름 뒤에 오는 ( ) 안에 함수로 전달하는 인자(파라미터)를 적음

- 사용자 함수의 예

```
def times(a, b):
    print(a * b)

times(3,5)
```

메소드(Methods)

- Everything = object
- Object have methods associated, depending on type

		type	examples of methods
`In [1]: sister = "liz"`	Object	`str`	`capitalize()` `replace()`
`In [2]: height = 1.73`	Object	`float`	`bit_length()` `conjugate()`
`In [3]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]`	Object	`list`	`index()` `count()`

Methods: Functions that belong to objects

메소드 이용 예 (1)

- List methods

```
In [4]: fam
Out[4]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]

In [5]: fam.index("mom")
Out[5]: 4

In [6]: fam.count(1.73)
Out[6]: 1
```

"Call method index() on fam"

- Str methods

```
In [7]: sister
Out[7]: 'liz'

In [8]: sister.capitalize()
Out[8]: 'Liz'

In [9]: sister.replace("z", "sa")
Out[9]: 'lisa'
```

메소드 이용 예 (2)

```
In [10]: sister.replace("z", "sa")
Out[10]: 'lisa'

In [11]: fam.replace("mom", "mommy")
AttributeError: 'list' object has no attribute 'replace'

In [12]: sister.index("z")
Out[12]: 2

In [13]: fam.index("mom")
Out[13]: 4
```

```
In [14]: fam
Out[14]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]

In [15]: fam.append("me")

In [16]: fam
Out[16]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89, 'me']

In [17]: fam.append(1.79)

In [18]: fam
Out[18]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89, 'me', 1.79]
```

모듈(Modules)

- 파이썬에서는 몇 개의 함수를 정의하여 모듈 파일로 저장하고 필요하여 import하여 사용할 수 있음

```
def cal_upper(price):
    increment = price * 0.3
    upper_price = price + increment
    return upper_price

def cal_lower(price):
    decrement = price * 0.3
    lower_price = price - decrement
    return lower_price

author = "pystock"
```

stock.py 로 저장

```
>>> stock.cal_upper(10000)
13000.0
>>> stock.cal_lower(10000)
7000.0
```

import stock

패키지(Packages)

- Functions and methods are powerful
- All code in Python distribution?
  - Huge code base: messy
  - Lots of code you won't use
  - Maintenance problem
- Directory of Python Scripts
- Each script = module
- Specify functions, methods, types
- Thousands of packages available

```
pkg/
  mod1.py
  mod2.py
  ...
```

- Numpy
- Matplotlib
- Scikit-learn

패키지 설치(Installation)와 임포트(Import)

- http://pip.readthedocs.org/en/stable/installing/
- Download `get-pip.py`
- Terminal:
  - `python3 get-pip.py`
  - `pip3 install numpy`

```
In [1]: import numpy

In [2]: array([1, 2, 3])
NameError: name 'array' is not defined

In [3]: numpy.array([1, 2, 3])
Out[3]: array([1, 2, 3])

In [4]: import numpy as np

In [5]: np.array([1, 2, 3])
Out[5]: array([1, 2, 3])

In [6]: from numpy import array

In [7]: array([1, 2, 3])
Out[7]: array([1, 2, 3])
```

패키지 임포트 예

- from numpy import array

```
from numpy import array

fam = ["liz", 1.73, "emma", 1.68,
       "mom", 1.71, "dad", 1.89]

...

fam_ext = fam + ["me", 1.79]

...

print(str(len(fam_ext)) + " elements in fam_ext")

...

np_fam = array(fam_ext)
```

Using Numpy, but not very clear

- import numpy

```
import numpy

fam = ["liz", 1.73, "emma", 1.68,
       "mom", 1.71, "dad", 1.89]

...

fam_ext = fam + ["me", 1.79]

...

print(str(len(fam_ext)) + " elements in fam_ext")

...

np_fam = numpy.array(fam_ext)
```

Clearly using Numpy

파이썬 기초 (4) : 부울 연산

Python Basics (4) : Boolean Operations

부울 자료형와 관계 연산자

- **Booleans**

```
In [4]: 2 < 3
Out[4]: True

In [5]: 2 == 3
Out[5]: False

In [6]: x = 2

In [7]: y = 3

In [8]: x < y
Out[8]: True

In [9]: x == y
Out[9]: False
```

- **Rational Operatiors**

<	strictly less than
<=	less than or equal
>	strictly greater than
>=	greater than or equal
==	equal
!=	not equal

논리연산자 및 대입연산자

논리 연산자	연산자의 의미
and	두 값이 모두 참일 때 만 결과값이 'True'
or	두 값이 모두 참일 때 만 결과값이 'True'
not	결과값이 참이면 'False', 거짓이면 'True'로 반대로 리턴

연산자	대입연산자 예	의미
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

조건문(Conditional Statements) (1)

```
z = 4    True
if z % 2 == 0 :
    print("z is even")
```

```
Output:
z is even
```

```
z = 4
if z % 2 == 0 :
    print("checking "  + str(z))
    print("z is even")
```

```
Output:
checking 4
z is even
```

조건문(Conditional Statements) (2)

```
z = 5
if z % 2 == 0 :
    print("checking "  + str(z))
    print("z is even")
```

```
Output:
```

```
z = 5
if z % 2 == 0 :
    print("z is even")
else :
    print("z is odd")
```

False

```
Output:
z is odd
```

조건문(Conditional Statements) (3)

```
z = 3
if z % 2 == 0 :
    print("z is divisible by 2")
elif z % 3 == 0 :
    print("z is divisible by 3")
else :
    print("z is neither divisible by 2 nor by 3")
```

False

True

```
Output:
z is divisible by 3
```

```
z = 6
if z % 2 == 0 :
    print("z is divisible by 2")
elif z % 3 == 0 :
    print("z is divisible by 3")
else :
    print("z is neither divisible by 2 nor by 3")
```

True

Never reached

```
Output:
z is divisible by 2
```

파이썬 라이브러리 : 넘파이

Python Libraries : Numpy

넘파이(Numpy) 개요

- Numeric Python
- Alternative to Python List: Numpy Array
- Calculations over entire arrays
- Easy and Fast
- Installation
  - In the terminal: `pip3 install numpy`

넘파이(Numpy) 장점

- **Without Numpy**

```
In [1]: height = [1.73, 1.68, 1.71, 1.89, 1.79]

In [2]: height
Out[2]: [1.73, 1.68, 1.71, 1.89, 1.79]

In [3]: weight = [65.4, 59.2, 63.6, 88.4, 68.7]

In [4]: weight
Out[4]: [65.4, 59.2, 63.6, 88.4, 68.7]

In [5]: weight / height ** 2
TypeError: unsupported operand type(s) for **: 'list' and 'int'
```

- **With Numpy**

Element-wise calculations

```
In [6]: import numpy as np

In [7]: np_height = np.array(height)

In [8]: np_height
Out[8]: array([ 1.73,  1.68,  1.71,  1.89,  1.79])

In [9]: np_weight = np.array(weight)

In [10]: np_weight
Out[10]: array([ 65.4,  59.2,  63.6,  88.4,  68.7])

In [11]: bmi = np_weight / np_height ** 2

In [12]: bmi
Out[12]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])
```

```
= 65.5/1.73 ** 2
```

넘파이(Numpy) 장점

```
In [19]: np.array([1.0, "is", True])
Out[19]:
array(['1.0', 'is', 'True'],
      dtype='<U32')

In [20]: python_list = [1, 2, 3]

In [21]: numpy_array = np.array([1, 2, 3])

In [22]: python_list + python_list
Out[22]: [1, 2, 3, 1, 2, 3]

In [23]: numpy_array + numpy_array
Out[23]: array([2, 4, 6])
```

Numpy arrays: contain only one type

Different types: different behavior!

```
In [24]: bmi
Out[24]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])

In [25]: bmi[1]
Out[25]: 20.975

In [26]: bmi > 23
Out[26]: array([False, False, False,  True, False], dtype=bool)

In [27]: bmi[bmi > 23]
Out[27]: array([ 24.747])
```

넘파이 1차원/2차원 배열(Array)

```
In [1]: import numpy as np

In [2]: np_height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])

In [4]: type(np_height)
Out[4]: numpy.ndarray
```

ndarray = N-dimensional array

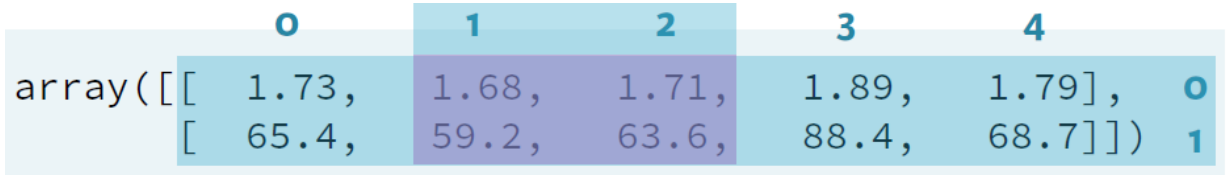
```
In [6]: np_2d = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
                          [65.4, 59.2, 63.6, 88.4, 68.7]])

In [7]: np_2d
Out[7]:
array([[  1.73,   1.68,   1.71,   1.89,   1.79],
       [ 65.4 ,  59.2 ,  63.6 ,  88.4 ,  68.7 ]])

In [8]: np_2d.shape
Out[8]: (2, 5)

In [9]: np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
                  [65.4, 59.2, 63.6, 88.4, "68.7"]])
Out[9]:
array([['1.73', '1.68', '1.71', '1.89', '1.79'],
       ['65.4', '59.2', '63.6', '88.4', '68.7']],
      dtype='<U32')
```

2 rows, 5 columns

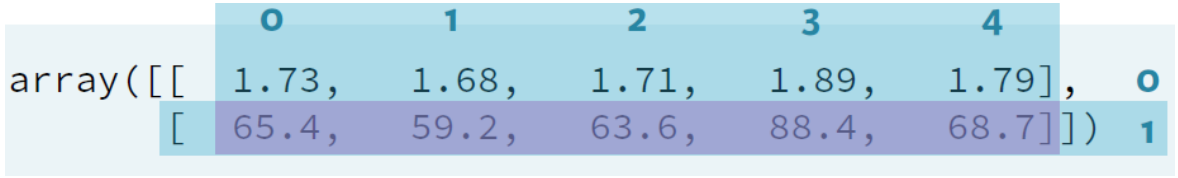
Single type!

배열 Subsetting

```
In [10]: np_2d[0]
Out[10]: array([ 1.73,  1.68,  1.71,  1.89,  1.79])

In [11]: np_2d[0][2]
Out[11]: 1.71

In [12]: np_2d[0,2]
Out[12]: 1.71

In [13]: np_2d[:,1:3]
Out[13]:
array([[  1.68,   1.71],
       [ 59.2 ,  63.6 ]])
```




```
In [14]: np_2d[1,:]
Out[14]: array([ 65.4,  59.2,  63.6,  88.4,  68.7])
```



numpy.ndarray.tolist

- **Return the array as an a.ndim -levels deep nested list of Python scalars.**
- **Return a copy of the array data as a (nested) Python list. Data items are converted to the nearest compatible builtin Python type, via the item function.**

1D Array

```
>>> a = np.array([1, 2])
>>> list(a)
[1, 2]
>>> a.tolist()
[1, 2]
```

2D Array

```
>>> a = np.array([[1, 2], [3, 4]])
>>> list(a)
[array([1, 2]), array([3, 4])]
>>> a.tolist()
[[1, 2], [3, 4]]
```

0D Array

```
>>> a = np.array(1)
>>> list(a)
Traceback (most recent call last):
  ...
TypeError: iteration over a 0-d array
>>> a.tolist()
1
```

- *Ref) https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.tolist.html*

넘파이 1D 배열 실습

```
[ ]  baseball = [180, 215, 210, 210, 188, 176, 209, 200]

     # Create a Numpy array from baseball: np_baseball
     np_baseball = np.array(baseball)

     # Print out type of np_baseball
     print(type(np_baseball))
```

```
[ ]  # Create data by random method
     height = np.random.normal(1.8/0.0254, 3.2, 500).tolist()
     weight = np.random.normal(140, 30, 500).tolist()
     print(height)
     print(weight)
```

```
[ ]  # Create array from height and weight and Convert unit
     np_height = np.array(height)
     np_height_m = np_height * 0.0254
     np_weight_kg = np.array(weight) * 0.453592
     print(np_height_m)
     print(np_weight_kg)
```

```
[ ]  # Calculate the BMI: bmi
     bmi = np_weight_kg / (np_height_m ** 2)
     print (bmi)
```

```
[ ]  light = bmi < 21
     print (light)  # Print out light in Boolean
     print (bmi[light])   # Print out BMIs of all baseball players whose BMI is below 21
```

- *Ref)*
  https://github.com/tomhostyn/DAT208x

넘파이 2D 행렬 실습 (1)

```
[1]   import numpy as np
      height = np.random.normal(1.75/0.0254, 0.20, 500).tolist()
      weight = np.random.normal(63, 20, 500).tolist()
```

```
[2]   baseball = [height, weight]
      np_baseball = np.array (baseball)
```

```
[3]   print (baseball)
      print (np_baseball)
      baseball = np_baseball.T
```

```
[4]   print (np_baseball.shape)
      print (baseball.shape)
      print (type(np_baseball))
      print (type(baseball))
```

```
[5]   # Print out the 50th row of baseball
      print (baseball[49,:])
      np_height_top10 = baseball[:10,0]
      np_weight_top10 = baseball[:10,1]
      print (np_height_top10)
      print (np_weight_top10)
```

- *Ref) https://github.com/datacamp/courses-intro-to-python/blob/master/chapter4.md*

넘파이 2D 행렬 실습 (2)

```
[1]  import pandas as pd
     baseball = pd.read_csv("http://s3.amazonaws.com/assets.datacamp.com/course/intro_to_python/baseball.csv")[['Height', 'Weight']].values.tolist()
```

```
[2]  print(baseball)
     print (type(baseball))
```

```
[3]  import numpy as np
     np_baseball = np.array(baseball)
     print (np_baseball.shape)
     print (np_baseball[49,:])  # Print out the 50th row of np_baseball
```

```
[4]  np_height_in = np_baseball[:,0]
     np_weight_lb = np_baseball[:,1]
     print (np_height_in)
     print (np_weight_lb)
```

- *Ref) https://github.com/datacamp/courses-intro-to-python/blob/master/chapter4.md*

넘파이 기본 통계분석 실습

```
[1]   import numpy as np
      heights_cm = np.random.normal(1.75, 0.30, 100).tolist()
      positions = ['GK', 'M', 'A', 'D'] * (100//4)
```

```
[2]   np_positions = np.array(positions)
      np_heights = np.array(heights_cm)
```

```
[3]   print (np_positions)
      print (np_heights)
```

```
[4]   # Print out the mean of np_height
      print(np.mean(np_heights))


      # Print out the median of np_height
      print(np.median(np_heights))
```

```
[5]   # Heights of the goalkeepers: gk_heights
      gk_heights = np_heights[np_positions == 'GK']

      # Heights of the other players: other_heights
      other_heights = np_heights[np_positions != 'GK']

      # Print out the median height of goalkeepers. Replace 'None'
      print("Median height of goalkeepers: " + str(np.median(gk_heights)))

      # Print out the median height of other players. Replace 'None'
      print("Median height of other players: " + str(np.median(other_heights)))
```

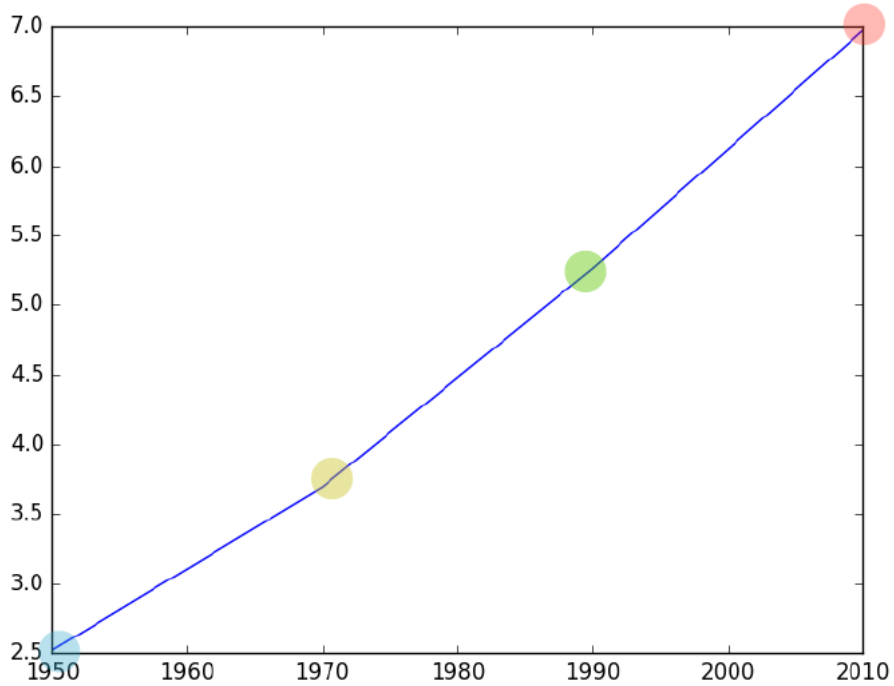
- *Ref) https://github.com/tomhostyn/DAT208x/blob/master/Week4.ipynb*

파이썬 라이브러리 : 맷플롯립

Python Libraries : Matplotlib

Matplotlib

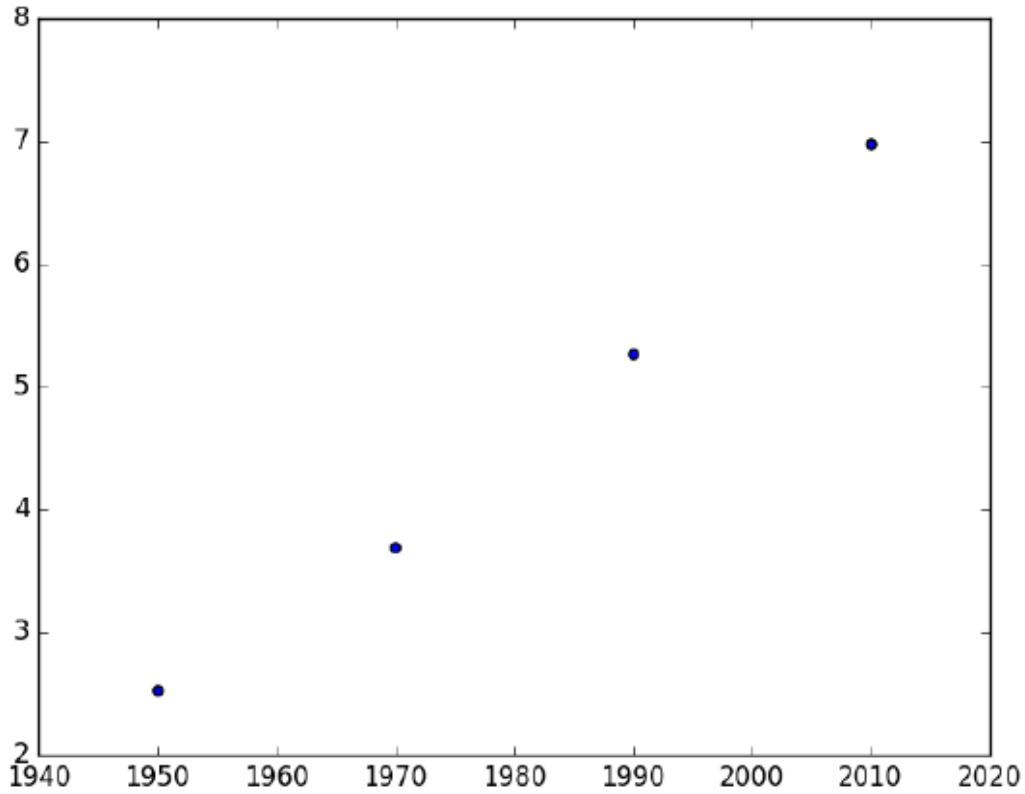
```
In [1]: import matplotlib.pyplot as plt

In [2]: year = [1950, 1970, 1990, 2010]

In [3]: pop = [2.519, 3.692, 5.263, 6.972]

In [4]: plt.plot(year, pop)

In [5]: plt.show()
```




```
year = [1950, 1970, 1990, 2010]
pop  = [2.519, 3.692, 5.263, 6.972]
```

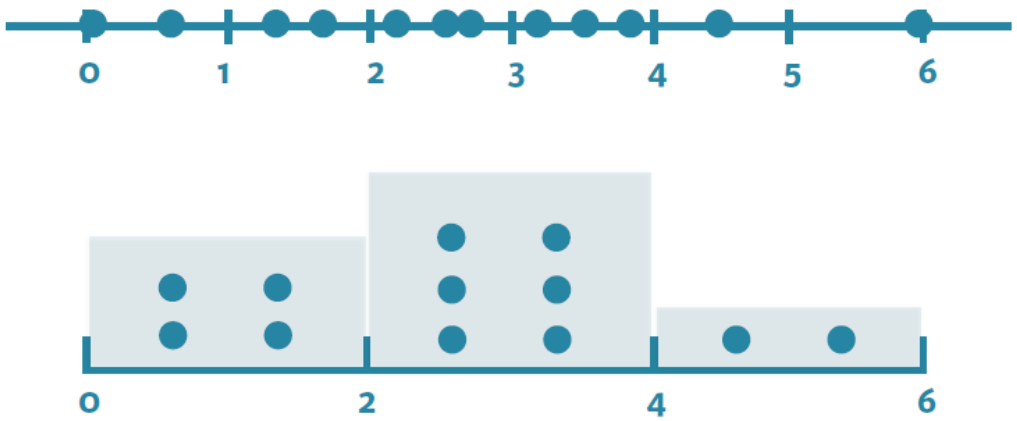
Scatter Plot

```
In [6]: plt.scatter(year, pop)

In [7]: plt.show()
```



Histogram

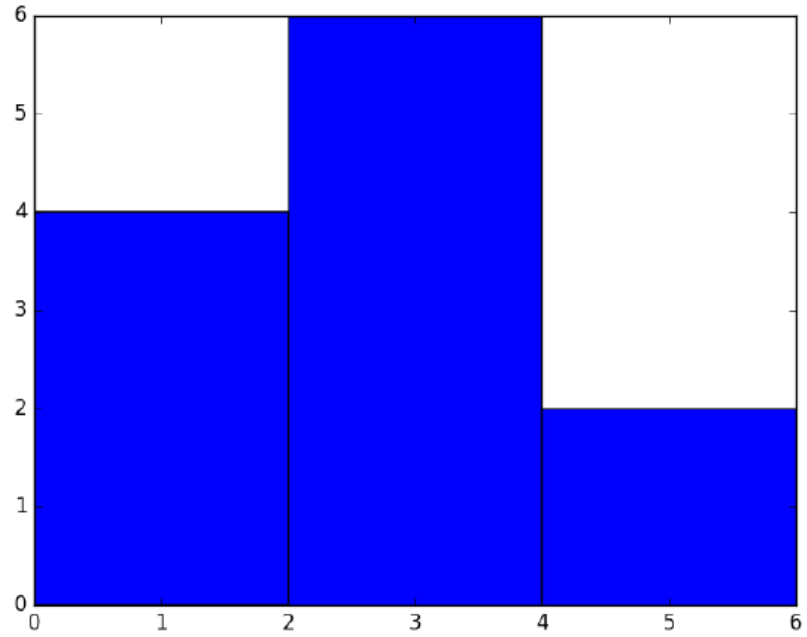
- Explore dataset
- Get idea about distribution



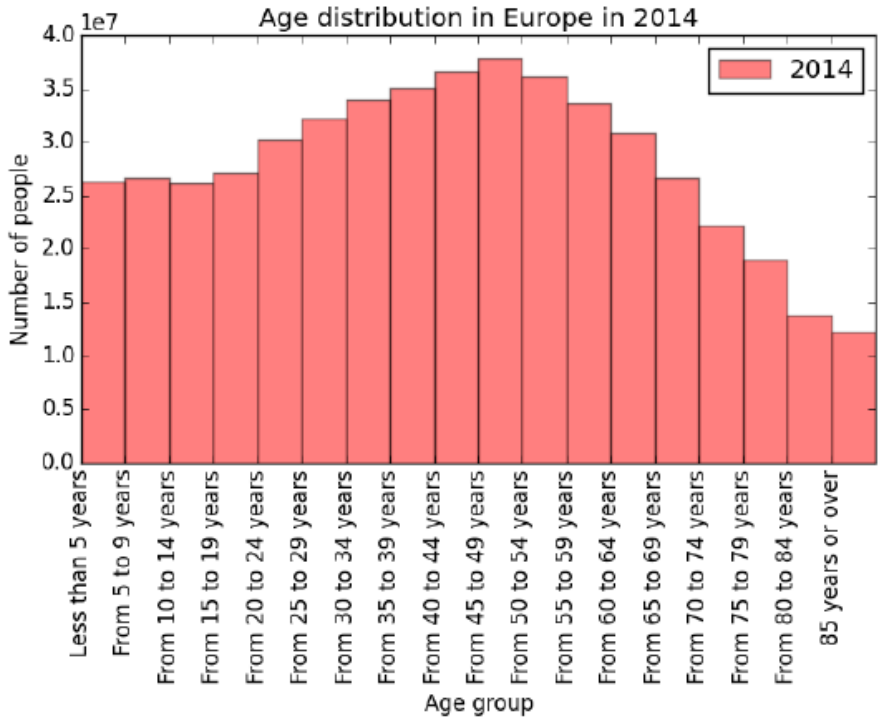

```
In [1]: import matplotlib.pyplot as plt

In [2]: help(plt.hist)

  Help on function hist in module matplotlib.pyplot:

  hist(x, bins=10, range=None, normed=False, weights=None,
  cumulative=False, bottom=None, histtype='bar', align='mid',
  orientation='vertical', rwidth=None, log=False, color=None,
  label=None, stacked=False, hold=None, data=None, **kwargs)
      Plot a histogram.

      Compute and draw the histogram of *x*. The return value is a
      tuple (*n*, *bins*, *patches*) or ([*n0*, *n1*, ...],
      *bins*, [*patches0*, *patches1*,...]) if the input contains
      multiple data.

  ...
```

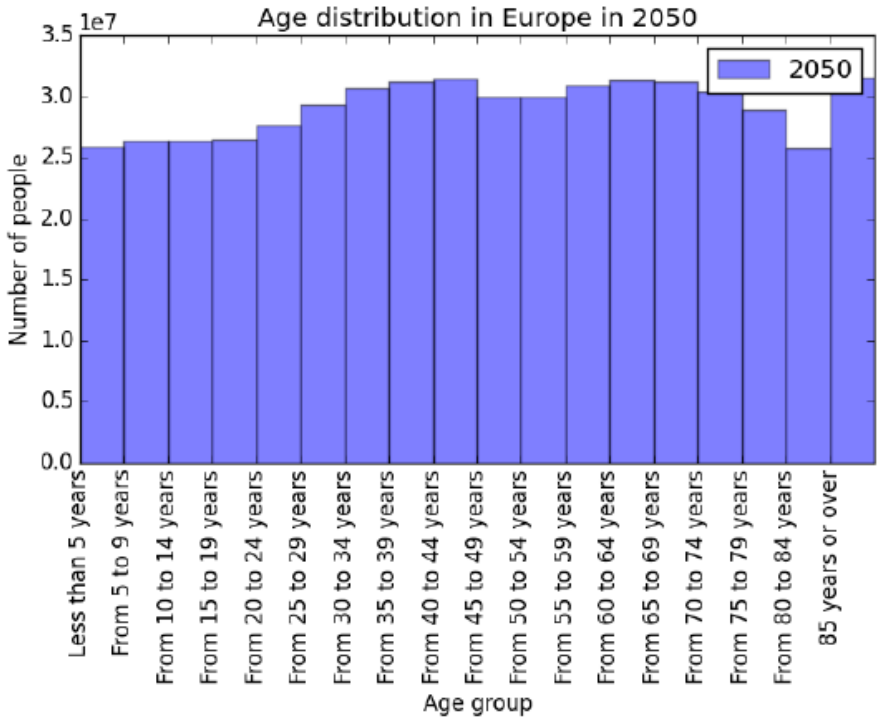
Histogram Example

```
In [3]: values = [0,0.6,1.4,1.6,2.2,2.5,2.6,3.2,3.5,3.9,4.2,6]
In [4]: plt.hist(values, bins = 3)
In [5]: plt.show()
```



Guess Condition for the Graphs



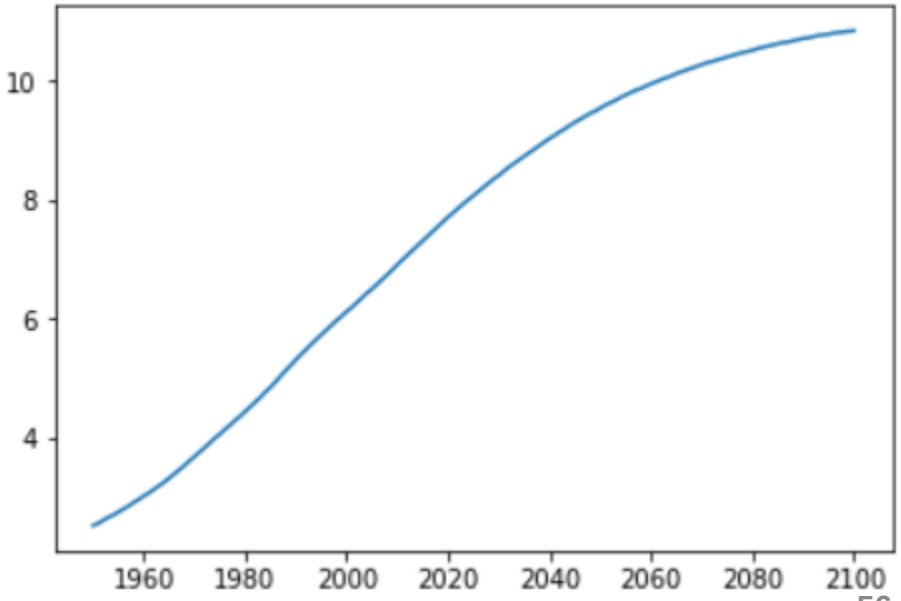


In [ ]:
```
# Histogram of life_exp, 15 bins
plt.hist(life_exp, bins = 15)

# Show and clear plot
plt.show()
plt.clf()

# Histogram of life_exp1950, 15 bins
plt.hist(life_exp1950, bins = 15)

# Show and clear plot again
plt.show()
plt.clf()
```

Customized Visualization (1)

- year = [1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100]
- population = [2.53, 2.57, 2.62, 2.67, 2.71, 2.76, 2.81, 2.86, 2.92, 2.97, 3.03, 3.08, 3.14, 3.2, 3.26, 3.33, 3.4, 3.47, 3.54, 3.62, 3.69, 3.77, 3.84, 3.92, 4.0, 4.07, 4.15, 4.22, 4.3, 4.37, 4.45, 4.53, 4.61, 4.69, 4.78, 4.86, 4.95, 5.05, 5.14, 5.23, 5.32, 5.41, 5.49, 5.58, 5.66, 5.74, 5.82, 5.9, 5.98, 6.05, 6.13, 6.2, 6.28, 6.36, 6.44, 6.51, 6.59, 6.67, 6.75, 6.83, 6.92, 7.0, 7.08, 7.16, 7.24, 7.32, 7.4, 7.48, 7.56, 7.64, 7.72, 7.79, 7.87, 7.94, 8.01, 8.08, 8.15, 8.22, 8.29, 8.36, 8.42, 8.49, 8.56, 8.62, 8.68, 8.74, 8.8, 8.86, 8.92, 8.98, 9.04, 9.09, 9.15, 9.2, 9.26, 9.31, 9.36, 9.41, 9.46, 9.5, 9.55, 9.6, 9.64, 9.68, 9.73, 9.77, 9.81, 9.85, 9.88, 9.92, 9.96, 9.99, 10.03, 10.06, 10.09, 10.13, 10.16, 10.19, 10.22, 10.25, 10.28, 10.31, 10.33, 10.36, 10.38, 10.41, 10.43, 10.46, 10.48, 10.5, 10.52, 10.55, 10.57, 10.59, 10.61, 10.63, 10.65, 10.66, 10.68, 10.7, 10.72, 10.73, 10.75, 10.77, 10.78, 10.79, 10.81, 10.82, 10.83, 10.84, 10.85]

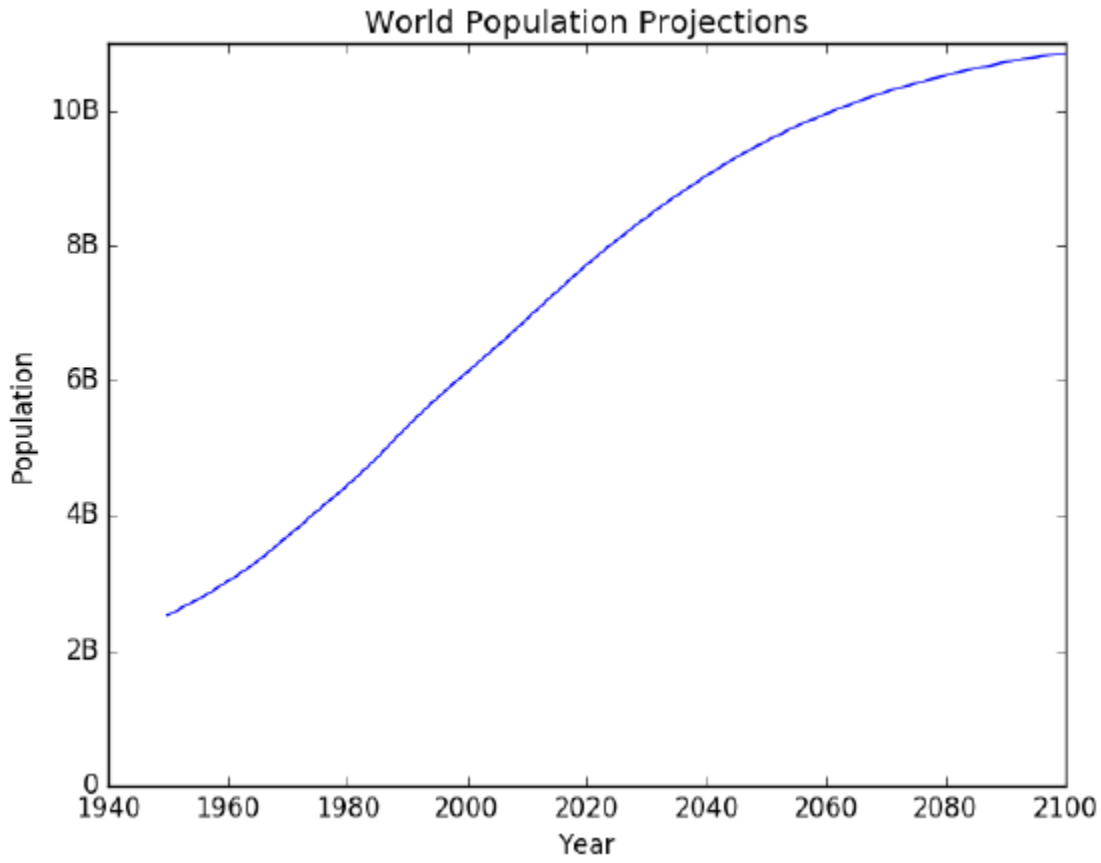
```
import matplotlib.pyplot as plt
plt.plot(year, population)
plt.show()
```



- *Ref) https://github.com/tomhostyn/DAT208x/blob/master/Week5.ipynb*

Customized Visualization (2)

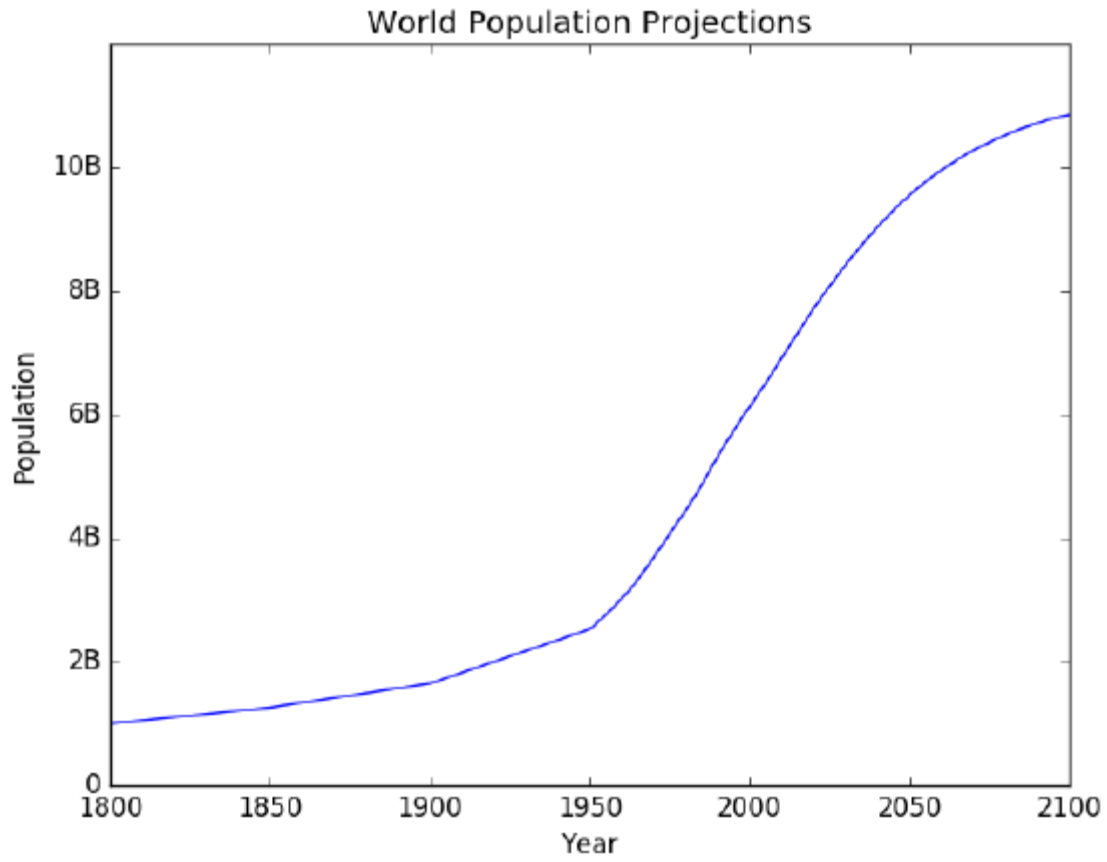
```
import matplotlib.pyplot as plt
year = ... # Implementation left out
population = ... # Implementation left out

plt.plot(year, population)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')
plt.yticks([0,2,4,6,8,10],
           ['0','2B','4B','6B','8B','10B']))

plt.show()
```



Customized Visualization (3)

- **Add historical data**

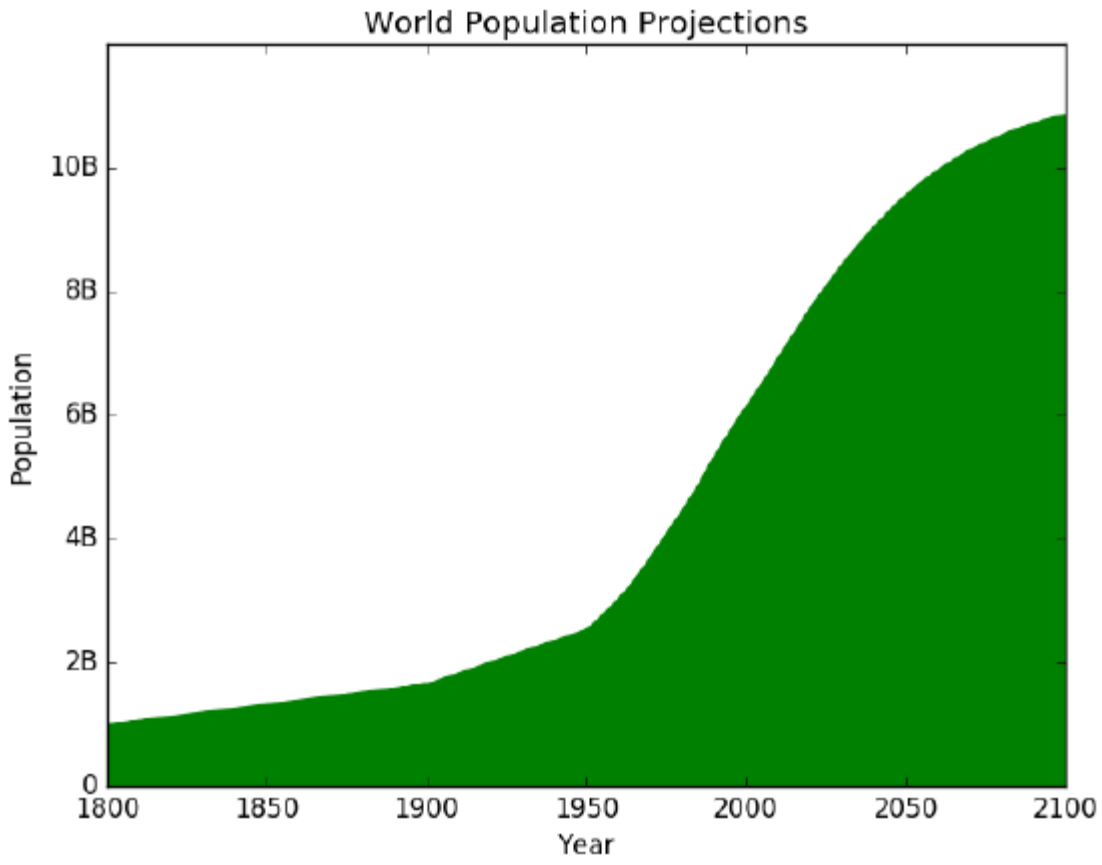
```
import matplotlib.pyplot as plt
year = ... # Implementation left out
population = ... # Implementation left out
population = [1.0,1.262,1.650] + population
year = [1800,1850,1900] + year

plt.plot(year, population)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')

plt.yticks([0,2,4,6,8,10],
           ['0','2B','4B','6B','8B','10B'])

plt.show()
```



Customized Visualization (4)

- **Fill graph**

```
import matplotlib.pyplot as plt
year = ... # Implementation left out
population = ... # Implementation left out
population = [1.0,1.262,1.650] + population
year = [1800,1850,1900] + year

plt.fill_between(year,population,0,color='green')

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')

plt.yticks([0,2,4,6,8,10],
           ['0','2B','4B','6B','8B','10B'])

plt.show()
```



파이썬 라이브러리 : 판다스

Python Libraries : Pandas

Pandas Overview

- Huge amounts of data are common
- 2D Numpy array?
  - Only one type possible
- Pandas
  - High-level data manipulation
  - `DataFrame`

Example : Brics (1)

brics.csv

```
,country,population,area,capital
BR,Brazil,200,8515767,Brasilia
RU,Russia,144,17098242,Moscow
IN,India,1252,3287590,New Delhi
CH,China,1357,9596961,Beijing
SA,South Africa,55,1221037,Pretoria
```

```
In [1]: brics = ... # declaration left out

In [2]: brics
Out[2]:
```

column labels

	country	population	area	capital
BR	Brazil	200	8515767	Brasilia
RU	Russia	144	17098242	Moscow
IN	India	1252	3287590	New Delhi
CH	China	1357	9596961	Beijing
SA	South Africa	55	1221037	Pretoria

row labels

Example : Brics (2)

- **CSV file -> DataFrame**

```
In [3]: import pandas as pd

In [4]: brics = pd.read_csv("path/to/brics.csv")

In [5]: brics
Out[5]:
  Unnamed: 0        country  population       area    capital
0         BR         Brazil         200    8515767   Brasilia
1         RU         Russia         144   17098242     Moscow
2         IN          India        1252    3287590  New Delhi
3         CH          China        1357    9596961    Beijing
4         SA   South Africa          55    1221037   Pretoria
```

Example : Brics (3)

- **CSV file -> DataFrame**

```
In [6]: brics = pd.read_csv("path/to/brics.csv", index_col = 0)

In [7]: brics
Out[7]:
         country  population      area    capital
BR        Brazil         200   8515767   Brasilia
RU        Russia         144  17098242     Moscow
IN         India        1252   3287590  New Delhi
CH         China        1357   9596961    Beijing
SA  South Africa          55   1221037   Pretoria
```

Example : Brics (4)

- **Column Access**

```
In [8]: brics["country"]
Out[8]:

BR          Brazil
RU          Russia
IN           India
CH           China
SA    South Africa
Name: country, dtype: object
```

```
In [9]: brics.country
Out[9]:

BR          Brazil
RU          Russia
IN           India
CH           China
SA    South Africa
Name: country, dtype: object
```

- **Add Columns**

```
In [10]: brics["on_earth"] = [True, True, True, True, True]
In [12]: brics["density"] = brics["population"] / brics["area"] * 1000000

In [13]: brics
Out[13]:

         country  population      area    capital on_earth     density
BR        Brazil         200   8515767   Brasilia     True   23.485847
RU        Russia         144  17098242     Moscow     True    8.421918
IN         India        1252   3287590  New Delhi     True  380.826076
CH         China        1357   9596961    Beijing     True  141.398928
SA  South Africa          55   1221037   Pretoria     True   45.043680
```

Example : Brics (5)

- **Row Access**

```
In [14]: brics.loc["BR"]
Out[14]:

country          Brazil
population          200
area            8515767
capital        Brasilia
density        23.48585
on earth           True
Name: BR, dtype: object
```

- **Element Access**

```
In [15]: brics.loc["CH","capital"]
Out[15]: Beijing

In [16]: brics["capital"].loc["CH"]
Out[16]: Beijing

In [17]: brics.loc["CH"]["capital"]
Out[17]: Beijing
```

Microsoft

INTRO TO PYTHON FOR DATA SCIENCE

Let's practice!