CHAPTER 1

Table of contents

1 . Python Basics

1) Variables

a) Strings (str)

파이썬에서 작은 따옴표나 큰 따옴표를 이용하여 문자열을 만들 수 있다. 따옴표를 1번 이용하거나 3번 이용하여 문자열을 만들 수 있는데 일반적으로 문자열 데이터를 만드는 경우 따옴표를 1번 이용하며, 주석처리를 하는 경우 따옴표를 3번 이용한다.

```
'a' == "a"

True
```

```
a = 'Hello World'
print(a)

Hello World
```

```
'''
주석처리를 하는 경우 이런식으로 code information 을 첨부한다
'''
```

b) Integer and Float (int, float)

소수점 이하의 값이 존재하지 않으면 정수형(int), 존재하면 실수형(float)이다.

```
x = 1
print(type(x))

<class 'int'>
```

```
x = 0.1
print(type(x))

<class 'float'>
```

c) Boolean (bool)

참이면 TRUE, 거짓이면 FALSE라고 하여 참과 거짓을 표시하는 데이터형이다.

```
5 == 5

True
```

NOTE)
type() 함수를 사용하면 데이터가 어떤 형인지 출력할 수 있다.

2) Basic Arithmetic

a) Math operation

intger와 float형의 데이터에 대하여 사칙연산이 가능하다.

+ (addition), - (subtraction), * (multiplication), ** (Exponentiation), /(division), //(floor division), %(Modulus)

```
# addition
print(1+1)

# subtraction
print(5-2)

# multiplication
print(2*3)

# Exponential
print(2**3)

# Division
print(6/3) # 0 으로 나누는 경우 error 가 발생한다

# Modulus
print(5%3)
```

```
2
3
6
8
2.0
2
```

NOTE)
print를 이용하지 않으면 마지막 줄의 결과만 나온다.

```
1+1
5-2
2*3
2**3
```

```
8
```

b) Boolean expressions

Boolean형에서 keywords는 True와 False이다. 모델을 정의하는 과정에서 option에 True, False를 입력하는 경우 R과는 다르게 앞글자만 대문자로 입력하며 T, F로 약어 사용이 불가능하다.

==(equals), !=(does not equal), >(greater than), >=(greater than or equal), similarly, < and <=

```
# equals
print(5 == 5)

# does not equal
print(5 != 5)
```

```
# greater than
print(5 > 4)

# greater than or equal
print(5 >= 5)
```

```
True
False
True
True
```

c) Logical operations

여러가지 조건문을 연결하는 경우 논리 연산자를 이용한다.

- and : 왼쪽 값과 오른쪽 값이 모두 True인 경우에 True를 출력
- or : 왼쪽 값과 오른쪽 값 중 하나라도 True이면 True를 출력
- not : 값이 True인 경우 False를 값이 False인 경우 True를 출력

```
(1 == 1) and (2 == 2)
```

```
True
```

```
(1 == 1) or (2 != 2)
```

```
True
```

```
not 5 == 5.0
```

```
False
```

d) Strings

문자열을 출력하는 경우 + operation을 이용하여 여러개의 문자열을 연결할 수 있다. 또한, f-string을 이용하면 문자열을 편리하게 출력할 수 있다.

```
str1 = "Hello"
str2 = "World"
str3 = str1 + str2
str3
```

```
'HelloWorld'
```

```
# f-string
x = 1886
str1 = f"Since {x}"
str1
```

```
'Since 1886'
```

```
# (Optional) Old School String Format
str1 = "a: %s" % "string"
print(str1)
str2 = "b: %f, %s, %d" % (1.0, 'hello', 5)
print(str2)
str3 = "c: {}".format(3.14)
```

```
print(str3)

a: string
b: 1.000000, hello, 5
c: 3.14
```

3) Control Flow

a) if문

if와 else를 사용한 기본구조는 다음과 같다.

Syntax1

```
if expression:
    statement(s)
  else:
    statement(s)
```

- expression에 참 혹은 거짓을 판단할 조건문을 입력하고, 조건문을 테스트 후 참이면 if문 아래의 문장들을 수행하고, 거짓이면 else문 다음 문장들을 수행하게 된다. 이때, else문은 if문 없이 독립적으로 사용할 수 없다.
- if문을 만들 때는 **"if 조건문:"** 바로 아래 문장부터 if문에 속하는 모든 문장에 들여쓰기(identation)를 해주어야 한다.
- 조건문 다음에 콜론(:)을 반드시 붙이도록 한다.

```
# Ex 1

# input 으로 받은 입력값을 integer 로 변환
age = int(input("What is your age? : "))

if age > 18:
    print('You can drive')
else:
    print('''Sorry, you can't drive''')

What is your age? : 25
You can drive
```

Syntax2

```
if expression1:
    statement(s)
  elif expression2:
    statement(s)
  elif expression3:
    statement(s)
  else:
    statement(s)
```

- 조건문을 여러개로 만들고 싶은 경우 elif를 추가한다. 이때, else는 생략가능하다.

```
# Ex 2

x = 3
y = 2
```

```
if x == y:
    print("The value of x is the same as value of y")
elif x == 3:
    print("I am here")
    print("x is 3")
else:
    print("x is something else")

print("Finished")
```

```
I am here
x is 3
Finished
```

```
# Ex 3

N = int(input("입력하는 숫자는 6 과 20 사이의 짝수인가?:"))

if N % 2 == 0:    # 첫번째로 짝수와 홀수를 판단

    # 두번째로 6 과 20 사이의 숫자인지 판단
    if N >= 2 and N <= 5:
        print("No")
    elif N >=6 and N <= 20:
        print("Yes")
    elif N > 20:
        print("No")
else:
    print("No")
```

```
입력하는 숫자는 6 과 20 사이의 짝수인가?:14
Yes
```

b) for문

같은 계산을 반복해서 수행하는 경우에는 for문을 사용한다. 이때, 횟수 지정은 숫자 리스트를 자동으로 만들어주는 range 함수를 사용한다. range(0,n)으로 함수를 호출하면 0부터 시작해서 (n-1)까지 정수를 출력한다. 즉, 0, 1, 2이 된다.

```
print(list(range(2,8)))
```

```
[2, 3, 4, 5, 6, 7]
```

NOTE) print 명령 이전에 list로 설정하지 않으면 range(2,8)이 그대로 출력된다.

```
print(range(2,8))
```

```
range(2, 8)
```

Syntax
for val in sequence: loop body

- sequence에는 리스트나 튜플, 문자열이 올 수 있다.

```
# Ex 1

for i in range(5):
    print(i**2)

0
1
4
9
16
```

```
# Ex 2
# 정수를 입력받은 후 1 부터 입력값까지의 합을 구하기

# input integer
n = int(input("정수를 입력하시오. : ")) + 1

# set initial value
mysum = 0

# calculate sum
for i in range(0, n):
    mysum = mysum + i

print(mysum)

정수를 입력하시오. : 10
55
```

Syntax2
[expression for val in sequence if condition]

- 리스트 안에 for 문을 포함하는 것도 가능하다.

```
# Ex 3

mylist = [1, 2, 3, 4]

res = [num*3 for num in mylist if num % 2 == 0] # mylist 에서 짝수만 추출한 후 3 을 곱하여 출력

print(res)

[6, 12]
```

여러 리스트에서 요소와 인덱스를 동시에 얻으려면 enumerate(), zip() 함수를 함께 사용한다. enumerate()는 for 문 결과 출력 시 인덱스를 출력해주는 함수이며, zip()은 여러 리스트를 함께 for 문에 돌리는 경우 쓰는 함수이다.

```
# Ex 4

names = ['Song', 'Kim', 'Lee']
ages = [24, 50, 18]

for i, (name, age) in enumerate(zip(names, ages)):
    print(i, name, age)
```

```
0 Song 24
1 Kim 50
2 Lee 18
```

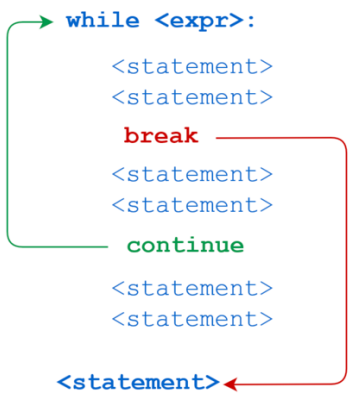
c) while 문

일반적으로 반복문을 어느정도 반복해야 하는지 알 수 없을 때 while문을 많이 이용한다.

Syntax
while test_expression: Body of while

- while 문은 조건문이 참인 동안에 while 문 아래의 문장이 반복해서 실행된다.
- 루프가 break 를 만나게 되면 해당 루프는 실행이 종료 되고 while 문 바로 뒤의 코드를 실행한다.
- 루프가 continue 를 만나게 되면 해당 루프는 실행이 종료되고 루프가 시작된 지점부터 다시 루프를 실행한다. (if 문, for 문, while 문 모두 해당)

```
# Ex 1

i = 1
while i < 100:
    print(i**2)
    i += i**2    # a += b is short for a = a+b

1
4
36
1764
```



제어문에서 break 와 continue 명령문을 이용하여 루프의 경로를 변경할 수 있다. 위의 그림을 보면, break 바로 위의 명령문이 True 일 경우 루프를 종료하고 continue 바로 위의 명령문이 True 일 경우 루프의 시작 지점으로 이동한다.

```
# Ex 2

i = 1

while True:
    print(i)
    i = i + 1 # update

    if(i>5):  # i 가 5 보다 큰 경우 종료
        break
```

```
1
2
3
4
5
```

```
# Ex 3

a = 1
while a<5:
    a += 1

    if a == 3:
        continue # a 가 3 일 경우 시작지점으로 이동
    print(a)
```

```
2
4
5
```

2. Packages

python 을 이용하여 통계분석을 하기 위해서는 몇가지 module 들이 필요하다. module 을 설치하는 방법에 대해 소개한다.

NOTE)
R의 패키지를 python서는 module이라고 부른다.

1) Install package

관리자 프롬프트를 열고 다음을 실행한다.

```
pip install 패키지이름
```

예를 들어 numpy 라는 패키지를 설치하려면 아래 코드를 실행하면 된다.

```
pip install numpy
```

NOTE)
cmd에 들어가지 않고 Jupyter에서도 설치가 가능하다. Jupyter에서는 "!" 이후에 나오는 내용들을 cmd창에 입력하는 것과 동일하게 처리한다. 다음 코드를 Jupyter에서 실행한다.

```
!pip install numpy
```

NOTE)
설치된 패키지들을 보기 위해선 다음을 실행하면 된다.

```
pip list
```

2) Some packages

a) numpy, pandas

numpy 와 pandas 는 외부 패키지이다. R 개발자들이 파이썬으로 통계분석을 하기 위하여 만든 패키지로, 데이터를 읽어 가공하거나 통계량을 계산하는 등의 기능을 제공하는 패키지이다.

- numpy 는 배열(ndarray) 데이터를 다루는 클래스를 주로 사용하며 특히 행렬 연산에 강하다.
- pandas 는 데이터프레임(dataframe) 데이터 관리에 강한 클래스를 사용하기 때문에 통계분석에서는 필수적인 패키지이다.

```
import numpy as np
import pandas as pd
```

b) matplotlib, seaborn

mathplotlib, seaborn 은 데이터를 시각화하는데 사용되는 패키지이다.

```
import matplotlib.pyplot as plt
import seaborn as sns
```

c) Sklearn, Tensorflow

Sklearn, Tensorflow 은 머신러닝을 위한 패키지이다.

- Sklearn 은 classification(분류), regression(회귀), clustering(클러스터링), sufficient dimension (차원축소) 등 다양한 머신러닝 모델링을 지원한다.
- Tensorflow 는 구글에서 제공하는 머신러닝 및 딥러닝 개발을 위한 패키지이다.

```
import tensorflow as tf
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

3. Objects

파이썬에서 통계 분석을 하기 위해선 데이터를 불러들인 후 가공하거나 메모리를 위해 필요한 부분만을 추출하는 등 데이터를 다루는데 익숙해질 필요가 있다. 데이터의 기본 형태를 만드는 방법과 슬라이싱 방법에 대해 소개한다.

1) List

a) List

리스트는 복수의 데이터를 모아서 묶어든 것으로 파이썬의 표준 데이터 형식이다. 데이터를 대괄호 [ ]로 감싸주면 리스트를 만들 수 있으며, 리스트의 내용을 표시하기 위해선 아래처럼 변수명만 쓰면 된다.

```
sample_list = [1, 2, 3, 4, 5]
sample_list

[1, 2, 3, 4, 5]
```

b) Indexing and Slicing

b-1) Indexing

R에서의 인덱싱과는 다르게 파이썬에서 리스트의 인덱싱은 0부터 시작한다. 아래의 예시를 통해 알아보도록 한다.

```
sample_list[0]

1
```

sample_list[0]은 첫 번째 요솟값이 출력되는 것을 확인할 수 있다. 따라서, 리스트의 인덱스는 0부터 시작해 "리스트의 길이 -1"까지가 끝이다. 인덱스에는 양수뿐만 아니라 음수도 들어갈 수 있다.

```
print(sample_list[-1],sample_list[-2])   # sample_list[-1] is last item

5 4
```

음수로 인덱싱을 하는 경우 오른쪽에서 부터 요소를 찾아감을 알 수 있다. 정리하자면, 길이가 N인 리스트의 인덱스 범위는

양수 : 0 ~ N-1
음수 : -N ~ -1

이다.

b-2) Slicing

Syntax	
a[start:stop]	items start through stop-1
a[start:]	items start through the rest of the list
a[:stop]	items from the beginning through stop-1
a[:]	a copy of the whole list
a[start:stop:step]	start through not past stop, by step
a[-1]	last item in the list
a[-2:]	last two items in the list
a[:-2]	everything except the last two items
a[::-1]	all items in the list, reversed
a[1::-1]	the first two items, reversed
a[:-3:-1]	the last two items, reversed
a[-3::-1]	everything except the last two items, reversed

NOTE) 콜론은 모든 인덱스를 의미한다.

NOTE) a[start:stop:step]는 a[slice(start:stop:step)]과 같다

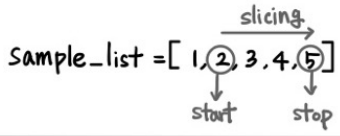
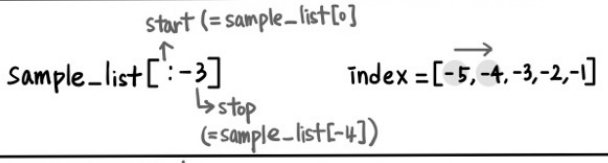
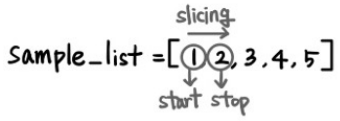
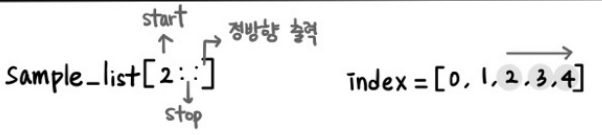
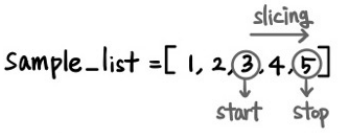
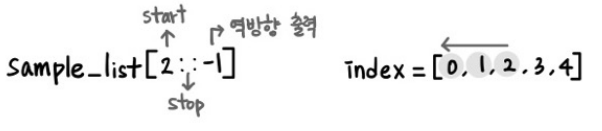
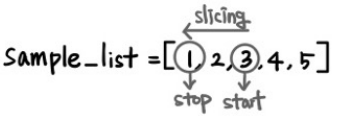
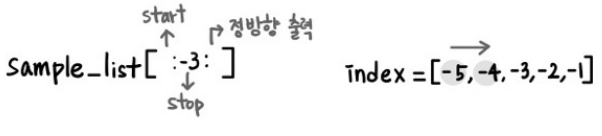
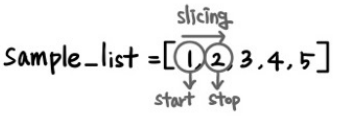
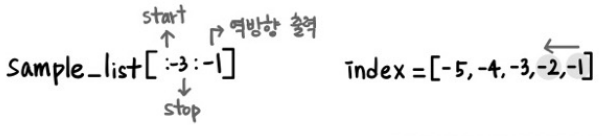
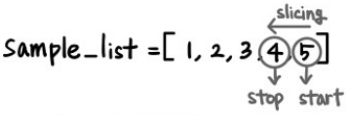
NOTE)
R에서의 인덱싱과 비교해보자.
R의 경우 sample_list[c(1,3,5)]이 과정을 통해 해당 인덱스의 값만을 추출할 수 있는데, 이와 동일한 파이썬의 문법은 다음과 같다.

```
> print(sample_list[i] for i in [0,2,4])
```

```python
# list slicing example

# 1:2 는 1 부터 (2-1)까지 인덱싱을 하므로 sample_list[1]만을 출력하게 된다.
a=sample_list[1:2]
print(f'sample_list[1:2] : {a}')

# 모든 item 을 출력한다.
a=sample_list[:]
print(f'sample_list[:] : {a}')

# sample_list[-4]부터 끝까지 출력한다.
a=sample_list[-4:]
print(f'sample_list[-4:] : {a}')

# 처음부터 sample_list[-4]까지만 출력한다.
a=sample_list[:-3]
print(f'sample_list[:-3] : {a}')

# 0,2,4 번째 인덱스에 해당하는 값을 출력한다.
a=sample_list[0:5:2]
print(f'sample_list[0:5:2] : {a}')

# 0,2,4 번째 인덱스에 해당하는 값을 출력한다.
a=[sample_list[i] for i in [0,2,4]]
print(f'[sample_list[i] for i in [0,2,4]] : {a}')

# sample_list[2]를 기준으로 오른쪽의 모든 값들을 출력한다.
a=sample_list[2::]
print(f'sample_list[2::] : {a}')

# sample_list[2]를 기준으로 왼쪽의 모든 값들을의 순서를 뒤집어서 출력한다.
a=sample_list[2::-1]
print(f'sample_list[2::-1] : {a}')

# sample_list[-4]을 기준으로 왼쪽의 모든 값들을 출력한다.
a=sample_list[:-3:]
print(f'sample_list[:-3:] : {a}')

# sample_list[-2]을 기준으로 오른쪽의 모든 값들의 순서를 뒤집어서 출력한다.
a=sample_list[:-3:-1]
print(f'sample_list[:-3:-1] : {a}')
```

```
sample_list[1:2] : [2]
sample_list[:] : [1, 2, 3, 4, 5]
sample_list[-4:] : [2, 3, 4, 5]
sample_list[:-3] : [1, 2]
sample_list[0:5:2] : [1, 3, 5]
[sample_list[i] for i in [0,2,4]] : [1, 3, 5]
sample_list[2::] : [3, 4, 5]
sample_list[2::-1] : [3, 2, 1]
sample_list[:-3:] : [1, 2]
sample_list[:-3:-1] : [5, 4]
```

cf. list indexing

c) List operation

- len() : 리스트의 길이를 구하는 함수
- del() : 리스트의 특정 요소 혹은 리스트의 특정 범위를 삭제하는 함수
- 덧셈 : 리스트를 이어 붙임
- 곱셈 : 리스트를 반복

```
# Ex1

# len()
print(len(sample_list))

# del()
del(sample_list[1])
print(sample_list)

5
[1, 3, 4, 5]
```

```
# Ex2

list1 = [1, 2, 3]
list2 = [4, 5]

# 덧셈
list3 = list1 + list2
print(list3)

# 곱셈
print(list1*2)
5
[1, 3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 1, 2, 3]
```

NOTE)
len()와 del()는 파이썬 내장 함수이다.

d) List method

Syntax	
list.append(x)	리스트의 맨 뒤에 값 추가
list.insert(a, b)	리스트의 a(index)위치에 b 값을 추가하는 함수
list.remove(x)	리스트에서 특정 값 제거
list.pop()	리스트 맨 마지막 값 반환 후 삭제
list.extend(list2)	리스트에 다른 리스트 2 연결
list.copy()	리스트 복사
list.reverse()	last two items in the list
list.sort()	everything except the last two items
list.count(x)	all items in the list, reversed
list.index(x)	the first two items, reversed
list.clear()	the last two items, reversed

NOTE)
sort()의 경우 리스트 내부 요소의 데이터 타입이 동일해야 한다.


```
# Ex

list1 = [1, 2, 3, 4, 5]
list2 = [6, 7, 8]

# append
list1.append(9)
print(f' append :{list1}')

# insert
list1.insert(1, 1.5)
print(f' insert :{list1}')

# remove
list1.remove(9)
print(f' remove  :{list1}')

# pop
list1.pop()
print(f' pop :{list1}')

# extend
list1.extend(list2)
print(f' extend :{list1}')

# copy
list3 = list1.copy()
print(f' copy :{list3}')

# reverse
list3.reverse()              # list3.sort(reverse = True)
print(f' reverse :{list3}')
```

```
# sort
list3.sort()
print(f' sort :{list3}')

# count
list3.count(1)
print(f' count :{list3}')

# index
list3.index(3)
print(f' index :{list3}')

# clear
list3.clear()
print(f' clear :{list3}')
```

```
append :[1, 2, 3, 4, 5, 9]
insert :[1, 1.5, 2, 3, 4, 5, 9]
remove  :[1, 1.5, 2, 3, 4, 5]
pop :[1, 1.5, 2, 3, 4]
extend :[1, 1.5, 2, 3, 4, 6, 7, 8]
copy :[1, 1.5, 2, 3, 4, 6, 7, 8]
reverse :[8, 7, 6, 4, 3, 2, 1.5, 1]
sort :[1, 1.5, 2, 3, 4, 6, 7, 8]
count :[1, 1.5, 2, 3, 4, 6, 7, 8]
index :[1, 1.5, 2, 3, 4, 6, 7, 8]
clear :[]
```

NOTE)
list.remove(x)에서 x가 list에 여러개가 존재하는 경우 첫번째 x를 제거한다.

NOTE)
sort의 기본값은 오름차순 정렬이며, reverse 옵션 True는 내림차순 정렬이다. => a.sort(reverse=True)

NOTE)
append 안에 원소뿐만이 아니라 리스트를 넣는 것도 가능하다.

2) Array

a) Array

파이썬에서 많이 이용되는 object type 중 하나는 np.array이다. array를 만들기 위해서는 파이썬의 library인 numpy를 먼저 import해야한다. 다음으로 list를 이용하여 array를 만들 수 있다.

```
# Load package
import numpy as np

# 1D array 생성
sample_array = np.array([1, 2, 3, 4, 5])
print(sample_array)

# 2D array 생성
sample_array2 = np.array([
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10]
])
print(sample_array2)

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

array의 차원 및 크기는 ndim과 shape으로 알 수 있다.

```
print(sample_array.ndim)
print(sample_array2.shape)

1
(2, 5)
```

파이썬에서 np.array의 2D array는 R의 matrix 형태와 비슷하게 다뤄진다. $A_{ij} = i \times j$의 형태를 가지는 2D array를 만드는 방법은 다양하며 아래에서 몇가지 예시를 소개한다.

```
import numpy as np

np.array([[i*j for i in range(5)] for j in range(5)])

array([[ 0,  0,  0,  0,  0],
       [ 0,  1,  2,  3,  4],
       [ 0,  2,  4,  6,  8],
       [ 0,  3,  6,  9, 12],
       [ 0,  4,  8, 12, 16]])
```

initializing function을 이용하여 numpy array를 만드는 방법도 있다.

- np.zeros
- np.ones
- np.arange

이 함수들은 reshape을 이용하여 for loop 없이 array를 만들 수 있다. numpy에서 reshape 함수는 array의 차원과 모양을 바꿔준다.

```
x = np.zeros((10))
x

array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
np.zeros((10,1))

array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

reshape 함수를 이용하는 경우 reshape(n, -1)과 같은 예시를 많이 볼 수 있다. -1은 n의 크기에 맞추어 원본 데이터 개수가 유지되도록 차원을 정해주는 역할을 한다. 아래의 예시에서 arange를 이용하여 20개의 정수를 발생시킨 후 reshape(4,-1)을 이용하여 데이터의 차원을 변형하게 되면 20개의 데이터 개수가 유지되도록 -1은 5로 할당된다.

```
a2 = np.arange(20)
print(a2.reshape(4, -1))     # 20 = 4 X 5(-1)
print(a2.reshape(2, 5, -1))  # 20 = 2 X 5 X 2(-1)

[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
[[[ 0  1]
  [ 2  3]
  [ 4  5]
  [ 6  7]
  [ 8  9]]

 [[10 11]
  [12 13]
  [14 15]
  [16 17]
  [18 19]]]
```

b) Indexing and Slicing

array 의 인덱싱과 슬라이싱은 list 와 동일하다. 단, array 의 차원에 따라서 인덱싱의 의미가 달라진다.

```
print(sample_array[0:2])
print(sample_array2[0:2])

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

1차원 array인 sample_array의 경우 인덱스는 원소들에 부여가 되지만, 2차원 이상의 경우 인덱스는 1차원 벡터를 의미함을 알 수 있다. 2차원 이상의 경우 각 열마다 인덱싱이나 슬라이싱 지정이 가능하다.

```
print(sample_array2[0,0])
print(sample_array2[0:2, 1:3])

1
[[2 3]
 [7 8]]
```

c) Vectorization and Broadcasting

ndarray(N-dimensional Array) 클래스는 list와 비슷한 구조처럼 보이지만 많은 차이가 있다. list의 경우 여러가지 타입의 원소들이 들어갈 수 있는 반면, array의 경우 모든 원소들이 같은 자료형이어야 한다. 이런 제약을 가지는 대신에 반복문을 사용하지 않고 벡터화(vectorization) 연산이 가능하며 계산 속도가 향상된다.

```
arr = np.array([[1., 2., 3.],[4., 5., 6.]])
arr

array([[1., 2., 3.],
       [4., 5., 6.]])
```

```
# type 확인
print(arr.dtype)

# 곱하기
print(arr*arr)

# 빼기
print(arr-arr)

# 나누기
print(1/arr)

float64
[[ 1.  4.  9.]
 [16. 25. 36.]]
[[0. 0. 0.]
 [0. 0. 0.]]
[[1.         0.5        0.33333333]
 [0.25       0.2        0.16666667]]
```

크기가 다른 배열간에도 연산이 가능하다. 이러한 연산을 브로드캐스팅(broadcasting)이라고 한다. 단, 차원의 길이가 같은 경우만 가능하다.

```
arr1 = np.array([1,2,3])    # 1x3
arr2 = np.array([[4,5,6],
                 [7,8,9]])  # 2x3
arr3 = np.array([1,2,3,4])  # 1x4

print(arr1+arr2)
print(arr1+arr3)  # Error

[[ 5  7  9]
 [ 8 10 12]]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-31-85cdf2de1705> in <module>
      5
      6 print(arr1+arr2)
----> 7 print(arr1+arr3)  # Error

ValueError: operands could not be broadcast together with shapes (3,) (4,)
```

1차원 벡터의 길이가 다른 경우는 배열간 연산이 불가능하다.

3) DataFrame

a) Series and DataFrame

Pandas는 파이썬에서 tabular data를 다루기 위한 패키지이며 R의 dataframe과 동일하다. Pandas의 dataframe과 numpy의 array는 파이썬에서 기본적으로 제공하는 데이터형인 list와 비슷하지만, array는 integer와 float같은 숫자형 타입만 원소로 정의할 수 있는 반면 dataframe은 숫자형과 string 등 다른 형태의 타입도 정의할 수 있다. 따라서, R에 익숙한 경우 dplyr library를 이용하여 데이터를 다루는 것처럼 pandas의 dataframe을 통해 missing data를 찾아내거나 데이터를 가공하는 작업을 할 수 있다. 마지막으로 pandas의 가장 중요한 특징 중 하나는 데이터를 프로그램으로 불러들이는 작업을 가능하게 한다는 것이다. (File I/O)

Pandas에서 다루는 두가지 object로 Series와 DataFrame이 있다.

Name	Dimensions	Description
pd.Series	1	1D labeled homogenously-typed array
pd.DataFrame	2	General 2D labeled, size-mutable tabular structure

Series는 vector와 label이 합쳐진 개념이다. Series는 vector와 index가 항상 함께 출력되는데 index는 자료형에 관계없이 자유롭게 지정이 가능하기 때문에 label처럼 이용된다.

```
# Load package
import pandas as pd
import numpy as np
```

```
mys = pd.Series([1,2,4,8,16,32,64])
mys
```

```
0     1
1     2
2     4
3     8
4    16
5    32
6    64
dtype: int64
```

index를 지정하지 않으면 default로 0부터 시작하는 정수형 index가 출력된다.

```
s = pd.Series(np.random.randn(5), index = ['a', 'b', 'c', 'd', 'e'])
s
```

```
a   -0.530376
b    0.519920
c    0.909843
d    0.581598
e   -0.736578
dtype: float64
```

NOTE) Series Attributes

- s.index : get the index
- s.values : get the values
- s.shape : find the shape

다음으로 DataFrame을 정의하는 방법에 대해 알아보도록 하자. Dataframe을 정의하는 간단한 코드는 다음과 같다.

```
df = pd.DataFrame(data, index = index, columns = columns)
```

pandas의 데이터프레임을 만드는 방법은 여러 가지가 있지만 배열이나 리스트를 이용해서 만드는 방법이 보편적이다. (이때, 중괄호를 반드시 써야한다.)

```
# list 이용
df1 = pd.DataFrame([[1,2,3], [4,5,6], [7,8,9]])
df1
```

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

```
# dictionary 이용
data = {                # dictionary 생성
    'age' : [20, 23, 48],
    'height' : [183, 192, 175],
    'weight' : [77, 83, 65],
    'gender' : ["M", "M", "F"]  #character vector
}
indexName = ['A', 'B', 'C']  # index 생성

df2 = pd.DataFrame(data, index = indexName)
df2
```

	age	height	weight	gender
A	20	183	77	M
B	23	192	83	M
C	48	175	65	F

```
# array 이용
sample_array = np.array([1,2,3,4,5])  # array 생성

df3 = pd.DataFrame({
    'col1' : sample_array,
    'col2' : sample_array*2,
    'col3' : ["A", "B", "C", "D", "E"],
})
df3
```

	col1	col2	col3
0	1	2	A
1	2	4	B
2	3	6	C
3	4	8	D
4	5	10	E

NOTE)
데이터프레임을 출력할 때 print() 메서드를 이용하지 않으면 표 형태로 출력이 된다.

NOTE) dictionary 자료형이란?

딕셔너리의 기본형은 다음과 같다.

{key1:Value1, Key2:Value2, Key3:Value3,...}

이때, 딕셔너리는 리스트처럼 순차적으로 해당 요솟값을 구하는 것이 아닌 key값을 이용하여 value를 얻는다. 따라서, key에는 변하지 않는 값을 사용하고, value에는 변하는 값과 변하지 않는 값을 모두 사용할 수 있다. 통계 분석시에는, key는 x,y를 할당하고 value는 x,y에 해당하는 컬럼을 배치한다.

NOTE) DataFrame Attributes

- df.index : the row index of the df
- df.columns : the columns of the df
- df.shape : the shape of the df
- df.values : numpy array of values

다음으로 난수(random number)를 발생하여 DataFrame 을 생성해보자. 이는 numpy 에서 제공하는 np.random 모듈의 기능을 이용한다.

```
# Load package
import numpy as np
import pandas as pd

# seed 설정
np.random.seed(22)

# 난수 생성
df = pd.DataFrame(np.random.randint(1, 10, size=(3, 5)))
df
```

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

난수 생성법은 다음과 같다.

난수 생성	
np.random.rand	0 부터 1 사이의 균일분포
np.random.randn	가우시안 표준 정규 분포
np.random.randint	균일 분포의 정수 난수 (np.random.randint(low, high, size))

b) Indexing

데이터프레임의 크기가 커질 경우 원하는 조건에 해당하는 데이터만을 추출하고 가공하는 과정이 필요하다. 데이터프레임에서 특정한 데이터만을 골라내는 인덱싱에 대해 소개한다.

Operation	Syntax	Result
Select Column	df[col]	Series
Select Row by Label	df.loc[label]	Series
Select Row by Integer Location	df.iloc[idx]	Series
Slice rows	df[5:10]	DataFrame
Select rows by boolean	df[mask]	DataFrame

우선 예시로 사용할 데이터프레임을 만들어보자.

```
# Load package
import numpy as np
import pandas as pd

# 난수 시드 설정
np.random.seed(1886)

# dataframe 생성
df = pd.DataFrame(np.random.randint(10, 22, (4,5)),
                  index = ["a", "b", "c", "d"],
                  columns = ['A', 'B', 'C', 'D', 'E'])
df
```

	A	B	C	D	E
a	16	13	11	19	18
b	14	14	13	15	17
c	20	13	18	20	16
d	18	21	16	13	21

데이터를 살펴보기 위해 처음 3 개의 행만을 가져온다.

```
df.head(n=3)
```

	A	B	C	D	E
a	16	13	11	19	18
b	14	14	13	15	17
c	20	13	18	20	16

데이터에 대한 기본 정보를 본다.

```
df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, a to d
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       4 non-null      int64
 1   B       4 non-null      int64
 2   C       4 non-null      int64
 3   D       4 non-null      int64
 4   E       4 non-null      int64
dtypes: int64(5)
memory usage: 192.0+ bytes
```

NOTE)
df.info() method는 매우 유용하며 통계 분석과정에서 데이터를 관찰하기 위해 첫단계에서 필수적으로 이용하는 코드이다.

데이터 프레임의 구조를 살펴보자.

```
df.shape

(4, 5)
```

이 데이터프레임에서 행 기준 인덱스는 0-3 까지 이며, 열 기준 인덱스는 0-4 까지 이다.

b-1) df[]

다음으로 대괄호([ ])를 이용하여 데이터프레임의 데이터를 추출하는 방법에 대해 알아보자. 대괄호를 이용하여 열을 추출하기 위해선 []안에 추출하고자하는 컬럼명을 입력한다.
"A" 열을 추출하는 방법은 다음과 같다.

```
df['A']

a    16
b    14
c    20
d    18
Name: A, dtype: int64
```

복수의 열을 추출하는 것도 가능하다. 이때, 컬럼명을 리스트 형식으로 넣어준다.

```
df[["A", "C"]]
```

	A	C
a	16	11
b	14	13
c	20	18
d	18	16

열을 추출하기 위해선 점(.)을 이용하는 방법도 가능하다.

```
df.B
```

```
a    13
b    14
c    13
d    21
Name: B, dtype: int64
```

지정한 열만 빼고 데이터를 추출하는 방법도 있다.

```
df.drop("A", axis = 1)
```

	B	C	D	E
a	13	11	19	18
b	14	13	15	17
c	13	18	20	16
d	21	16	13	21

NOTE) axis=0은 행 기준이며 axis=1은 열 기준이다.

조건문을 이용하여 데이터를 추출해보자.

```
df[df["A"]>15]
```

	A	B	C	D	E
a	16	13	11	19	18
c	20	13	18	20	16
d	18	21	16	13	21

조건이 2 개 이상인 경우도 가능하다.

```
df[(df["A"] > 15) | (df["B"] > 17)]
```

	A	B	C	D	E
a	16	13	11	19	18
c	20	13	18	20	16
d	18	21	16	13	21

조건을 둘다 만족해야 할 때 (and)	df[(조건 1) & (조건 2)]
조건 중 하나라도 만족하면 될 때 (or)	df[(조건 1) | (조건 2)]

b-2) df.query()

query method 를 이용하여 데이터를 추출하는 방법도 있다. 첫번째 행을 추출한다.

```
df.query('index == "a"')
```

	A	B	C	D	E
a	16	13	11	19	18

query 함수는 여러가지 조건을 지정해서 데이터를 꺼내오는게 가능하다. 예를 들어, C 열에서 값이 13 인 행을 찾고 싶다면 아래와 같이 코드를 작성한다.

```
df.query('C == 13')
```

	A	B	C	D	E
b	14	14	13	15	17

복수의 조건을 지정할 수도 있다. (|) 연산자는 '또는' 혹은 OR 조건이며, (&) 연산자는 '그리고' 혹은 AND 조건이다.

```
# OR 연산자
df.query('A == 14 | D == 20')
```

	A	B	C	D	E
b	14	14	13	15	17
c	20	13	18	20	16

```
# AND 연산자
df.query('B == 13 & index == "c"')
```

	A	B	C	D	E
c	20	13	18	20	16

b-3) df.loc

loc 는 주로 label 를 기반으로 이용하며 허용되는 input 은 다음과 같다.

- A single label
- A list of labels
- A boolean array

만약 loc 를 사용하면서 인덱스를 하나만 넣으면 행(row)을 선택한다.

```
df.loc["a"]
```

```
A    16
B    13
C    11
D    19
E    18
Name: a, dtype: int64
```

슬라이스 형식도 가능하다.

```
df.loc["b":"c"]
```

	A	B	C	D	E
b	14	14	13	15	17
c	20	13	18	20	16

NOTE)
위의 경우 loc를 쓰지 않고 나타낼 수도 있다.

```
df["b":"c"]
```

	A	B	C	D	E
b	14	14	13	15	17
c	20	13	18	20	16

리스트 형태를 이용하여 원하는 행만을 추출하는 것도 가능하다.

```
df.loc[["a", "d"]]
```

	A	B	C	D	E
a	16	13	11	19	18
d	18	21	16	13	21

NOTE)
위의 경우 loc를 쓰지 않으면 KeyError 오류가 발생한다.

loc 를 이용하여 조건문을 넣을 수 있다.

```
df.loc[df.A > 15]
```

	A	B	C	D	E
a	16	13	11	19	18
c	20	13	18	20	16
d	18	21	16	13	21

행과 열 모두 지정하고 싶다면 df.loc[행 인덱스, 열 인덱스]와 같은 형태로 사용한다.

```
df.loc["a", "A"]
```

```
16
```

인덱싱 값으로 슬라이싱 또는 리스트를 이용할 수 있다.

```
df.loc[["a", "b"], "C":"E"]
```

	C	D	E
a	11	19	18
b	13	15	17

조건문도 가능하다.

```
df.loc[df.A >10, ["C", "D"]]
```

	C	D
a	11	19
b	13	15
c	18	20
d	16	13

b-4) df.iloc

iloc 는 loc 와 다르게 정수(integer)만 입력할 수 있다. 허용되는 input 은 다음과 같다.

- An integer
- A list of integers
- A slice
- A boolean array

NOTE)
loc 와 마찬가지로 인덱스가 하나만 들어가면 행을 선택한다.

```
df.iloc[1]

A    14
B    14
C    13
D    15
E    17
Name: b, dtype: int64
```

```
df.iloc[:2, 2]

a    11
b    13
Name: C, dtype: int64
```

c) Dataframe method

Syntax	
df.info()	데이터프레임에 관련된 기본 데이터를 반환
df.head()	데이터프레임의 첫 5 열을 반환(default 는 5 줄)
df.tail(7)	데이터프레임의 마지막 7 열을 반환
df.axes	데이터프레임의 축을 나타내는 목록을 반환
df.index	데이터프레임의 인덱스(행 레이블) 반환
df.columns	데이터프레임의 열 레이블 반환
df.values	데이터프레임을 Numpy 형식으로 반환
df.ndim	차원을 나타내는 int 를 반환
df.shape	데이터프레임의 크기를 나타내는 튜플을 반환
df.size	데이터프레임의 크기를 나타내는 int 를 반환
df.dtypes	데이터프레임에서 dtype(데이터 타입)을 반환
df.count	데이터프레임에서 데이터의 개수를 반환(옵션으로 axis=0, axis=1 은 각각 컬럼기준, 열기준을 의미)

```
# Ex

# count : 결측치의 개수를 파악하는데 유용
print(df.count(axis=0))
print(df.count(axis=1))

# shape
print(df.shape)
```

```
A    4
B    4
C    4
D    4
E    4
dtype: int64
a    5
b    5
c    5
d    5
dtype: int64
(4, 5)
```

R 에서의 summary 함수는 컬럼별 통계량을 반환하여 데이터의 구조를 파악하는데 유용하다. 파이썬에서 동일한 함수는 pandas 의 describe 함수이다.

```
df.describe()
```

	A	B	C	D	E
count	4.000000	4.00000	4.000000	4.000000	4.000000
mean	17.000000	15.25000	14.500000	16.750000	18.000000
std	2.581989	3.86221	3.109126	3.304038	2.160247
min	14.000000	13.00000	11.000000	13.000000	16.000000
25%	15.500000	13.00000	12.500000	14.500000	16.750000
50%	17.000000	13.50000	14.500000	17.000000	17.500000
75%	18.500000	15.75000	16.500000	19.250000	18.750000
max	20.000000	21.00000	18.000000	20.000000	21.000000

4. File I/O

Pandas 는 데이터 파일을 읽어 데이터프레임을 만들 수 있다. 다음처럼 여러가지 포맷을 지원한다.

- CSV
- Excel
- HTML
- JSON
- HDF5
- SAS
- STATA
- SQSL

널리 사용되는 파일 형식은 CSV(Comma Separated Value), excel, sql 등이 있으며 import-output function 은 다음과 같다.

- pd.read_csv/pd.read_csv
- pd.read_excel/pd.to_excel
- pd.read_sql/pd.ro_sql

NOTE)
CSV 파일 포맷은 데이터 값이 쉼표(Comma)로 구분되는 텍스트 파일이다.

NOTE)
샘플 데이터로 사용한 CSV 파일을 **%%writefile** 매직 명령으로 만들어보자. 이 명령은 셀에 서술한 내용대로 텍스트 파일을 만드는 명령이다.

```
%%writefile sample1.csv
c1, c2, c3
1, 1.11, one
2, 2.22, two
3, 3.33, three

Writing sample1.csv
```

1) File import

CSV 파일로부터 데이터를 읽어 데이터프레임을 만들 때는 pandas.read_csv 함수를 사용한다. 이때, **작업 중인 폴더에 파일을 복사해두고** 함수의 입력값으로 파일 이름을 넣는다.

```
# import library
import pandas as pd

pd.read_csv('sample1.csv')
```

	c1	c2	c3
0	1	1.11	one
1	2	2.22	two
2	3	3.33	three

파일경로를 입력하여 파일을 불러올 수 있다.

```
data = pd.read_csv('파일경로/파일이름.csv')
```

NOTE)
윈도우에서 파일경로를 찾는 방법 참고
https://ko.wikihow.com/윈도우에서-파일-경로-찾는-방법
맥에서 파일경로를 찾는 방법 참고
https://ko.stealthsettings.com/cum-putem-afla-locatia-exacta-path-a-unui-fisier-sau-folder-in-mac-os-x.html

Parameter	
read_csv 는 여러가지 파라미터를 통해서 옵션을 줄 수 있다.	
filepath/buffer	파일경로/파일이름.csv 을 입력하여 파일을 불러온다.
sep/delimiter	default 는 comma(,)이며 만약 데이터의 분리 기준이 쉼표(,)가 아닌 경우 기준이 되는 값을 입력하면 된다.
header	default 는 0 으로 컬럼명으로 사용할 행의 번호를 입력한다.
names	사용할 변수명을 입력한다. 파일에 변수명이 없다면 header 를 None 으로 설정해야 한다.
index_col	데이터의 인덱스로 사용할 열의 번호를 입력한다.
skiprows	첫 행을 기준으로 데이터를 얼마나 건너뛰고 읽어올지를 정한다.
nrows	파일을 읽어올 행의 수를 입력한다.
date_parser	시계열 타입으로 변환할 변수를 입력한다.

NOTE)
더 많은 옵션은 오른쪽의 판다스 공식 문서를 참고한다.
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

확장자가 csv 가 아닌 파일 즉, 데이터를 구분하는 구분자가 쉼표가 아닌 예시를 보자. 길이가 정해지지 않은 공백이 구분자인 경우에는 sep = '\s+' 정규식(regular expression) 문자열을 사용한다.

```
%%writefile sample2.txt
c1         c2         c3         c4
0.179181 -1.538472  1.347553  0.43381
1.024209  0.087307 -1.281997  0.49265
0.417899 -2.002308  0.255245 -1.10515

Writing sample2.txt
```

```
pd.read_csv('sample2.txt', sep='\s+')
```

	c1	c2	c3	c4
0	0.179181	-1.538472	1.347553	0.43381
1	1.024209	0.087307	-1.281997	0.49265
2	0.417899	-2.002308	0.255245	-1.10515

2) File output

지금까지와 반대로 파이썬의 데이터프레임 값을 csv 파일로 출력하고 싶으면 to_csv 메서드를 사용한다.

```
df.to_csv('sample3.csv')
```

리눅스나 맥에서는 cat 셸 명령으로 파일의 내용을 확인할 수 있다. 윈도우에서는 type 함수를 사용한다.

NOTE) 느낌표(!)는 셸 함수를 사용하기 위한 아이파이썬(IPython) 매직 명령이다.

```
!cat sample3.csv      # 윈도우에서는 !type sample3.csv 함수를 사용
```

```
,A,B,C,D,E
a,16,13,11,19,18
b,14,14,13,15,17
c,20,13,18,20,16
d,18,21,16,13,21
```

파일을 읽을 때와 마찬가지로 출력할 때도 sep 인수로 구분자를 바꿀 수 있다.

```
df.to_csv('sample4.txt', sep = '|')
```

```
!cat sample4.txt        # 윈도우에서는 !type sample4.txt 함수를 사용

|A|B|C|D|E
a|16|13|11|19|18
b|14|14|13|15|17
c|20|13|18|20|16
d|18|21|16|13|21
```

3) Load data from url

웹상에는 다양한 파일이 CSV 파일 형태로 제공된다. read_csv 명령 사용시 파일 패스 대신 URL을 지정하면 Pandas가 직접 해당 파일을 다운로드하여 읽어들인다.

```
math = pd.read_csv("http://home.ewha.ac.kr/~josong/dm/mathcat.data", sep = '\s+')
# 구분자가 공백이므로 옵션으로 지정한다.
```

4) Check data

데이터의 수가 많을 경우, 데이터프레임을 보여줄 행과 열의 수를 지정할 수 있다.

다음 메서드는 데이터 전부를 보여주지 않고 데이터의 상단부분(head)와 하단부분(tail)만 출력하여 보여준다. 기본값으로 5줄을 보여주며 옵션으로 이를 지정할 수 있다.

- head
- tail

우선, 데이터의 크기를 확인한다.

```
math.shape
```

```
(394, 6)
```

394개의 데이터가 있고 변수는 6개이다.

```
math.head(n=4)
```

	hsgpa	hsengl	hscalc	course	passed	outcome
1	78.0	80	Yes	Mainstrm	No	Failed
2	66.0	75	Yes	Mainstrm	Yes	Passed
3	80.2	70	Yes	Mainstrm	Yes	Passed
4	81.7	67	Yes	Mainstrm	Yes	Passed

```
math.tail(n=4)
```

	hsgpa	hsengl	hscalc	course	passed	outcome
391	77.0	79	Yes	Mainstrm	Yes	Passed
392	80.7	70	Yes	Mainstrm	Yes	Passed
393	80.7	81	Yes	Mainstrm	Yes	Passed
394	82.2	86	Yes	Catch-up	Yes	Passed

pandas를 이용하여 앞, 뒤의 일부분만 보여주는 방법도 있다. 보여줄 행의 수는 display.max_rows 옵션으로 정할 수 있다.

```
pd.set_option("display.max_rows", 6) # 앞뒤로 모두 6 개의 행만 보여준다.
Math
```

	hsgpa	hsengl	hscalc	course	passed	outcome
1	78.0	80	Yes	Mainstrm	No	Failed
2	66.0	75	Yes	Mainstrm	Yes	Passed
3	80.2	70	Yes	Mainstrm	Yes	Passed
...	...	...	...	...	...	...
392	80.7	70	Yes	Mainstrm	Yes	Passed
393	80.7	81	Yes	Mainstrm	Yes	Passed
394	82.2	86	Yes	Catch-up	Yes	Passed

394 rows × 6 columns

```
pd.set_option("display.max_column", None)   # 모든 컬럼을 다 보여준다.
Math
```

	hsgpa	hsengl	hscalc	course	passed	outcome
1	78.0	80	Yes	Mainstrm	No	Failed
2	66.0	75	Yes	Mainstrm	Yes	Passed
3	80.2	70	Yes	Mainstrm	Yes	Passed
...	...	...	...	...	...	...
392	80.7	70	Yes	Mainstrm	Yes	Passed
393	80.7	81	Yes	Mainstrm	Yes	Passed
394	82.2	86	Yes	Catch-up	Yes	Passed

394 rows × 6 columns

5. Function

함수는 하나의 관련 작업을 수행하는 데 사용되고 체계적이고 재사용 가능한 코드 블록이다. 함수는 응용 프로그램의 모듈성을 높이고 코드 재사용 수준을 높인다. 파이썬은 print() 등과 같은 많은 내장 built-in 함수를 제공하지만 사용자가 직접 함수를 만들 수도 있는데, 이를 사용자 정의 user-defined 함수라고 한다.

모듈을 사용하면 논리적으로 파이썬 코드를 구성 할 수 있다. 관련 코드를 모듈로 그룹화하면 코드를 더 쉽게 이해하고 사용할 수 있다. 모듈(.py)은 파이썬 코드로 구성된 파일이며 함수, 클래스 및 변수를 정의 할 수 있다.

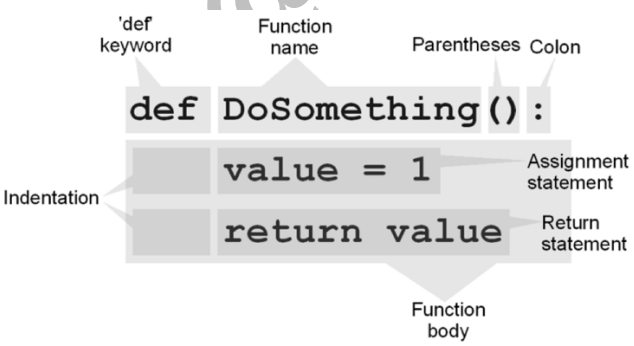
NOTE)
함수<모듈<패키지

NOTE)
클래스는 프로그램의 원형으로 변수 정의, 함수 등으로 구성되어 있고 함수는 원하는 반복 작업을 위한 입력 -> 실행 -> 외부 출력으로 이루어진 코드 묶음으로 생각할 수 있다.

Syntax1

```
def your_function_name(parameters):
    statements
    return expression
```

- 시작은 define의 약어인 def로 시작하고 함수명(입력값 지정, 생략 가능), 그리고 콜론(:)으로 끝난다.
- 원하는 작업 실행이 가능한 코드를 적는다.
- 출력되는 함수 결과값을 지정한다.




```
# Ex 1
# 두 개의 숫자를 입력받아 합을 구하는 함수

def mysum1(a, b):
    res = a + b
    return res

mysum1(1,2)

3
```

```
# Ex 2
# 입력값까지의 합을 구하는 함수
```

```
def mysum(n):
    res = 0
    for i in range(1, n+1):
        res = res + i
    return res

mysum(10)
```

```
55
```

여러 개의 입력값을 받는 함수를 만드는 경우 매개변수 이름 앞에 * 을 붙이면 입력값을 전부 모아서 튜플로 만들어 준다.

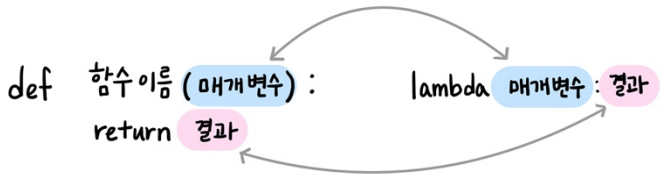
```
# Ex 3

def mysum3(*args):
    res = 0
    for i in args:
        res = res + i
    return res

mysum3(1,3,5,7)
```

```
16
```

파이썬에서는 람다함수를 통해 이름이 없는 함수를 만들 수 있다. 람다함수를 통해 코드를 간결하게 쓸 수 있으며 메모리를 절약할 수 있다.

Syntax2
lambda arguments : expression

- 람다함수는 lambda 라는 키워드를 통해 생성할 수 있다. 익명함수라는 이름처럼 lambda 함수는 함수의 이름을 지정하지 않는다.
- 함수에 이름이 없고, 저장된 변수가 없기 때문에 다시 사용할 수는 없다.




```
# Ex 4

cube = lambda x: x**3

print(cube(4))
```

```
64
```

map()함수와 lambda 함수를 동시에 이용할 수 있다.

```
# Ex 5

in1 = [1, 3, 5, 7]
in2 = [2, 4, 6, 8]
list(map(lambda x,y : x+y, in1, in2))

[3, 7, 11, 15]
```

NOTE)
map( ) 함수는 두 개의 인수를 가지는 함수이다.

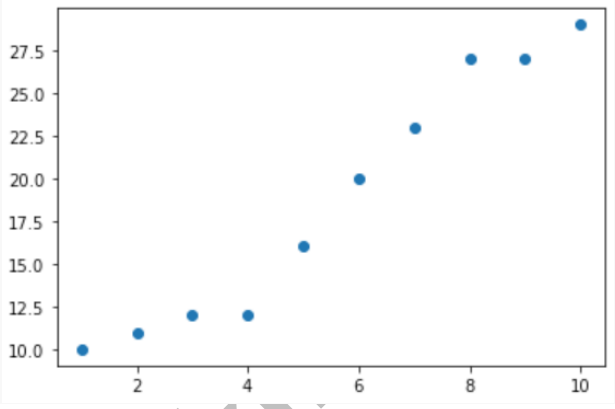
6. Plotting data

A picture is worth a thousand words

데이터 분석에서 데이터의 시각화(visualization)는 필수적인 단계이다. 시각화를 통해 data 에 대한 insight 를 발견하거나 분석의 결과를 직관적으로 나타내는데 효과적이기 때문이다. Python 에서 데이터의 시각화를 제공하는 대표적인 package 는 matplotlib 이다. matplotlib.pyplot 의 plot() function 은 R 에서의 plot function 과 비슷하다.

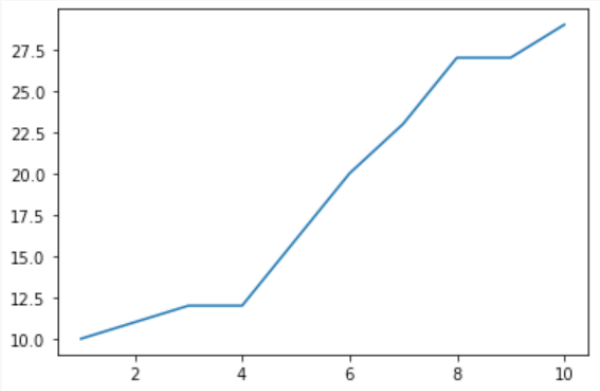
우선, 가장 기본적인 scatter plot 을 그려보도록 한다. Scatter plot 은 입력된 x, y 값을 2 차원 평면에 점으로 나타내는 그림이다. 그림을 그리기 위한 데이터 중 x 값은 1 부터 10 까지 정수이며 y 값은 10 부터 30 까지 정수 중 10 개를 임의로 추출한 뒤 오름차순으로 정렬하였다.

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

np.random.seed(123)

x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
y = np.random.randint(10, 30, 10)

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



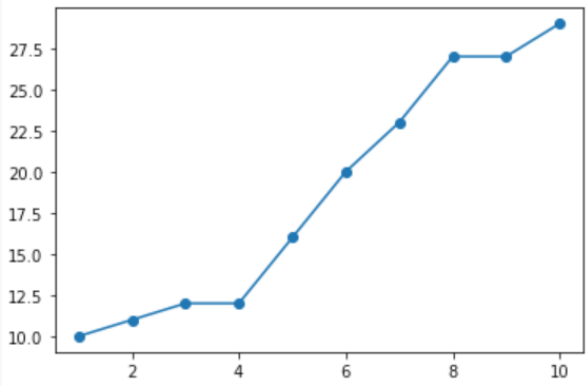
선으로 연결된 interpolation function 을 그리기 위해선 .plot() 함수를 이용한다.

```
plt.plot(x, np.sort(y))
plt.show()
```



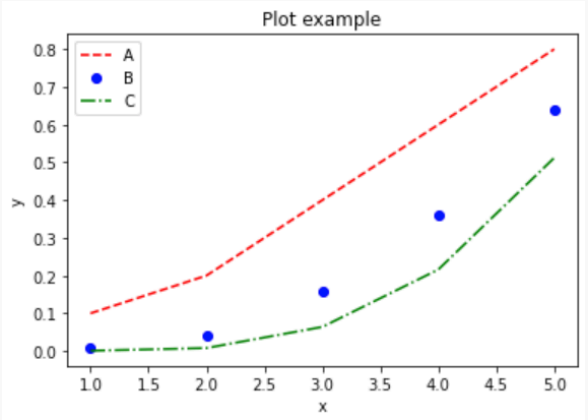
마지막으로 두 함수를 겹쳐 그리기 위해선 scatter plot 을 먼저 정의한 후 intepolation function 을 정의한다.

```
plt.scatter(x, np.sort(y))
plt.plot(x, np.sort(y))
plt.show()
```



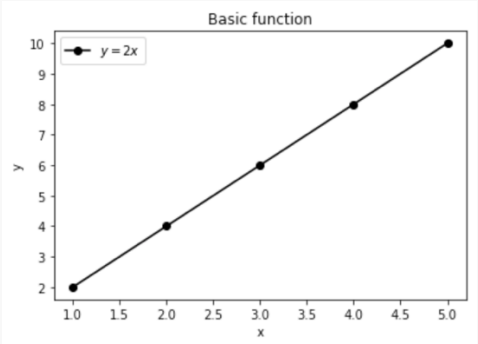
plot 을 그리는 경우 그림의 title, x 와 y label, 범례, 선 종류과 색상 지정은 반드시 알아야둬야 할 옵션들이다. 그림의 title 은 plt.title()을 이용하여 설정한다. 다음으로, plt.xlabel(), plt.ylabel()을 이용하여 x, y 축의 레이블을 설정할 수 있다. 선 종류는 solid, dashed, dotted, dashdot 총 4 가지 종류가 있다. 이 중 solid(실선)는 default 로 다른 종류로 변경하는 방법은 아래의 예제를 참고하면 된다. 마지막으로 범례를 설정하기 위해선 plt.plot()에 keyword 로 "label="을 설정한 후 plt.legend() 함수를 추가하면 된다. 범례 위치를 변경하기 위해선 "loc=" keyword 로 설정하면 된다.

```
# Ex 1

x = np.array([1, 2, 3, 4, 5])
y = np.array([0.1, 0.2, 0.4, 0.6, 0.8])

plt.plot(x,y, 'r--', label = "A")
plt.plot(x, y**2, 'bo', label = "B")
plt.plot(x, y**3, 'g-.', label = "C")
plt.title("Plot example")
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

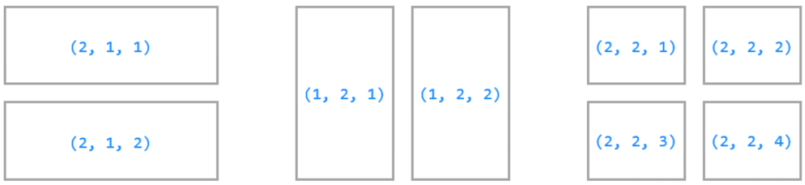


.plot()의 keyword 인 'marker='를 이용하여 scatter plot 과 interpolation plot 을 겹쳐그릴 수도 있다.

```
# Ex 2

x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])

plt.plot(x, y, color = 'black', marker = 'o', label = "$y = 2x$")
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.title("Basic function")
plt.show()
```

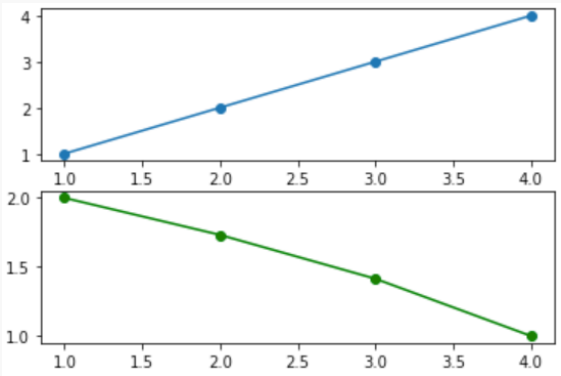


matplotlib.pyplot 모듈의 subtplot() 함수를 이용하여 여러개의 그래프를 겹치지 않게 그릴 수 있다. 아래의 그림은 subplot()을 이용하여 그림의 개수와 위치를 결정하는 방법을 나타낸다.




```
plt.subplot(row, column, index)
```

```
myx = np.array([1, 2, 3, 4])

plt.subplot(2, 1, 1)
plt.plot(myx, myx, 'o-')

plt.subplot(2, 1, 2)
plt.plot(myx, np.sqrt(-myx+5), 'go-')

plt.show()
```



7. module(.py)

1) module?

프로그램이 길어짐에 따라 디버깅과 유지를 쉽게 하려고 여러 개의 파일로 나누어 코딩하거나 각 프로그램에서 썼던 편리한 함수를 각 프로그램에 정의를 복사하지 않고도 사용하고 싶을 때 사용한다.

파이썬은 정의들을 파일에 넣고 스크립트나 인터프리터의 대화형 모드에서 사용할 수 있는 방법을 제공하는데 그런 파일을 모듈이라고 한다. 모듈은 파이썬 정의와 문장들을 담고있는 파일로 확장자 .py 를 붙인다.

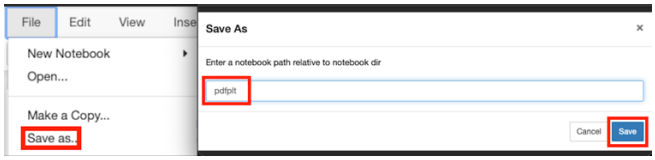
2) creating module

정규분포와 감마분포의 PDF를 그리는 모듈을 작성해 보자. 새로운 주피터 노트북을 열고 아래 코드를 입력한다.

```
def normplt(m,s):   # m(mean), s(standard devation)

    import numpy as np
    import matplotlib.pyplot as plt
    from scipy.stats import norm

    x = np.arange(m-3*s,m+3*s,0.1)
    plt.plot(x,norm.pdf(x,m,s),'b')
    plt.title('Normal(Mean=' + str(m) +',STD='+str(s)+')PDF')
    plt.show()

def gamplt(a,b):

    import numpy as np
    import matplotlib.pyplot as plt
    from scipy.stats import gamma

    x = np.arange(0,a*b+6*a*b*b,0.1)
    plt.plot(x,gamma.pdf(x,a,b),'b')
    plt.title('Gamma(Alpha='+str(a)+',Beta='+str(b)+')PDF')
    plt.show()
```

method 1

a) 주피터 노트북에서 **File** => **Save as** 메뉴를 선택하여 폴더 지정 없이 바로 파일 이름을 적는다.



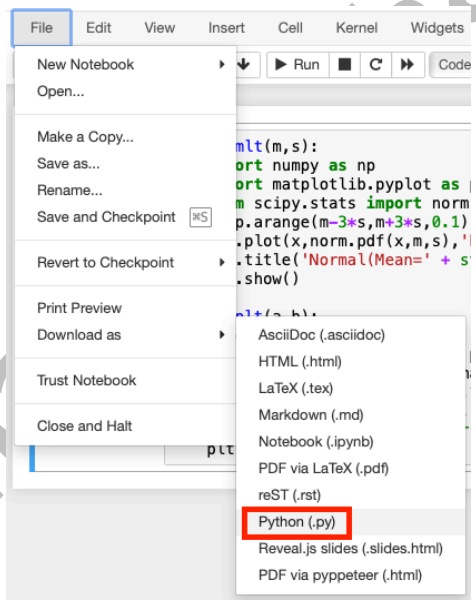
b) 탭이름이 'untitled'에서 저장된 이름 'pdfplt'으로 바뀌고 Home 폴더에는 pdfplt.ipynb 파일이 저장된다. (확장자 *.ipynb는 자동으로 부여된다.)

c) 주피터 노트북 실행 후 In 라인에 아래의 코드를 입력하면 Home 폴더에 pdfplt.py 모듈이 만들어진다.

```
!jupyter nbconvert --to python pdfplt.ipynb
```

method 2

주피터 노트북에서 **File** => **Download as** => **Python(.py)** 메뉴를 선택하면 다운로드 폴더에 pdfplt.py 모듈이 만들어진다. 작업중인 노트북이 있는 폴더로 이동시킨다.



3) using module

Syntax
from 모듈이름 import 함수이름1, 함수이름2

pdfplt.py를 이용하여 정규분포와 감마분포를 그려본다.

```
from pdfplt import gamplt,normplt

import matplotlib.pyplot as plt
plt.subplot(2,1,1)          # nrows=2, ncols = 1, index = 1
gamplt(2,1)
plt.subplot(2,1,2)          # nrows=2, ncols = 1, index = 2
normplt(0,1)
```