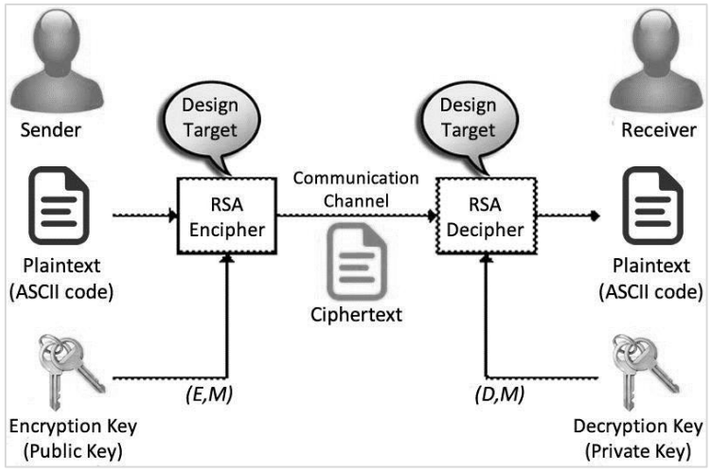
한양대학교

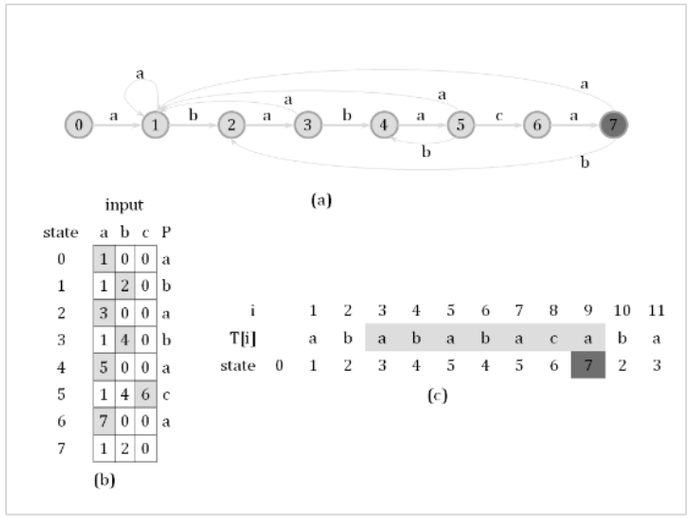
NumPy

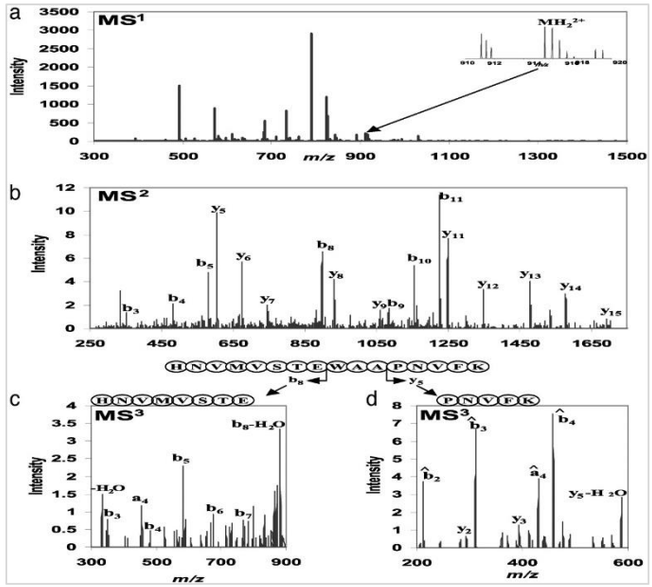
Hosung Jo

강사소개

- **조호성**
  - 한양대학교 SW융합원 SW교육전담교수
  - 한양대학교 전자컴퓨터통신 박사
    - 알고리즘 (정보보호, 문자열매칭, 생물정보학)



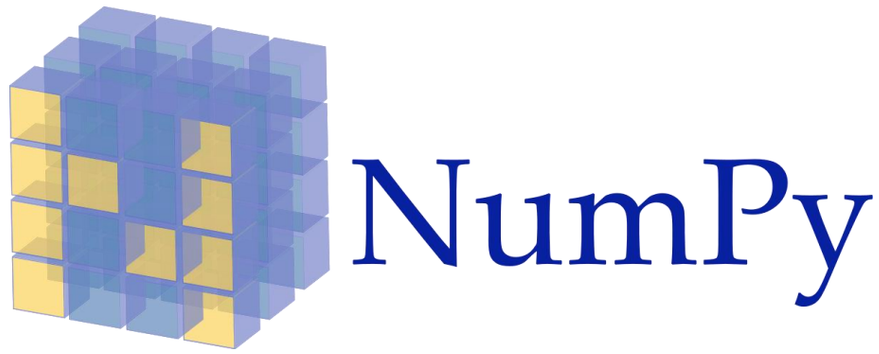




Day 2

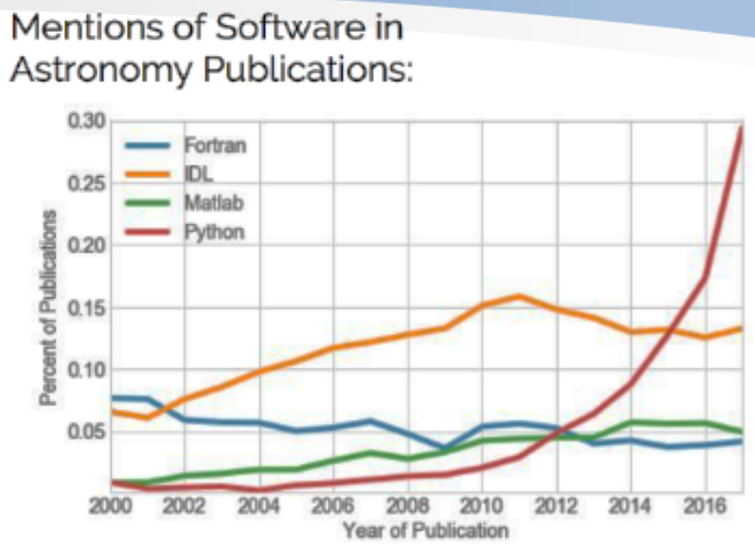
- **Review of Day 1**
  - Python, NumPy, and ndarray
  - NumPy Basics operations

- **NumPy 기능**



Python

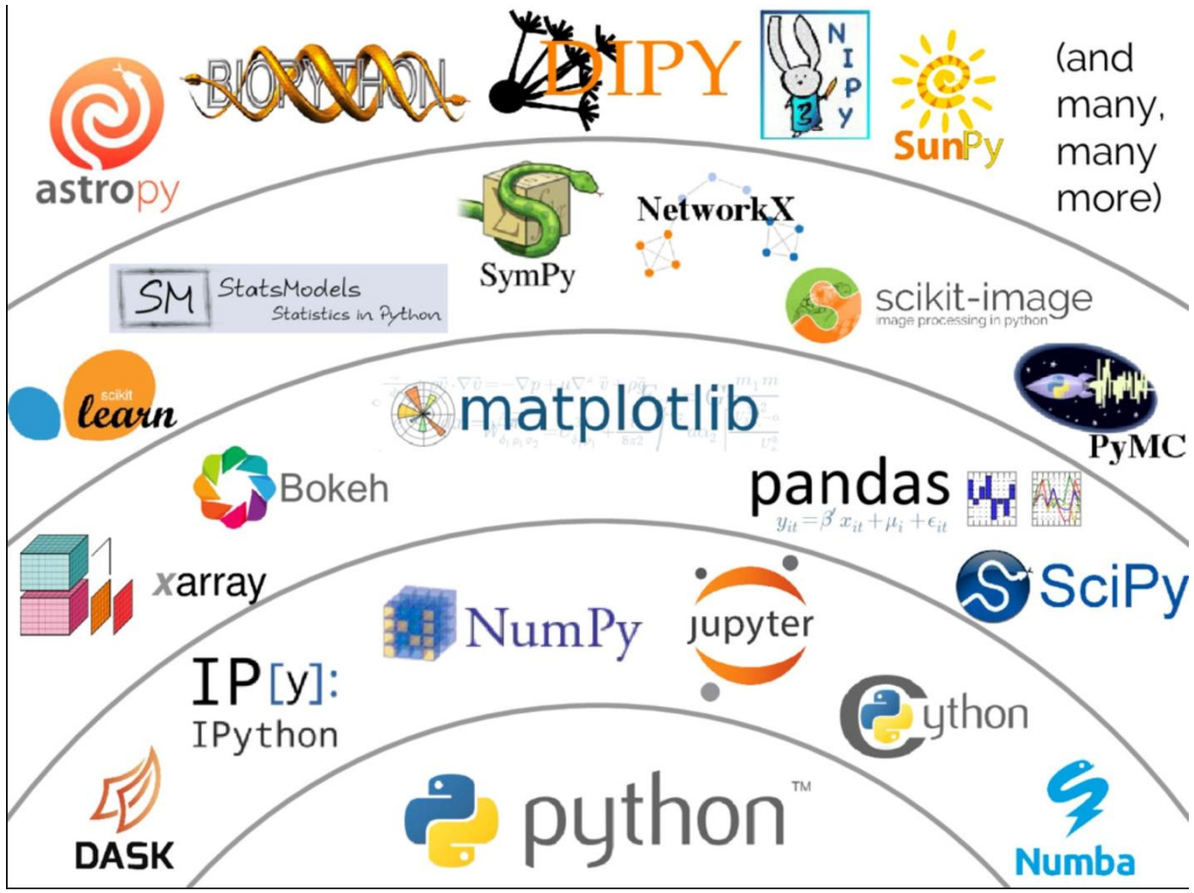
- **Python**
  - 오픈소스 기반의 하이레벨 컴퓨터 언어
  - 간결하여 이해가 쉽고, 활용 분야가 넓음



- **Python의 장점**
  - OpenSource 기반, 상호 운용성, 단순성과 동적 특성
  - **왠만한 건 다 있는 Third-party 라이브러리**

Python Library

- **Powerful Third-party Library**
  - **NumPy**, array 컴퓨팅 지원
  - **Ipython** 과 **Jupyter**,
    과학문서작성과 연구 정리
  - **Numba**, 분산컴퓨팅
  - **Pandas**, 데이타프레임
  - **Matplotlib**, 데이터 시각화
  - **Scikit-learn**, 기계학습



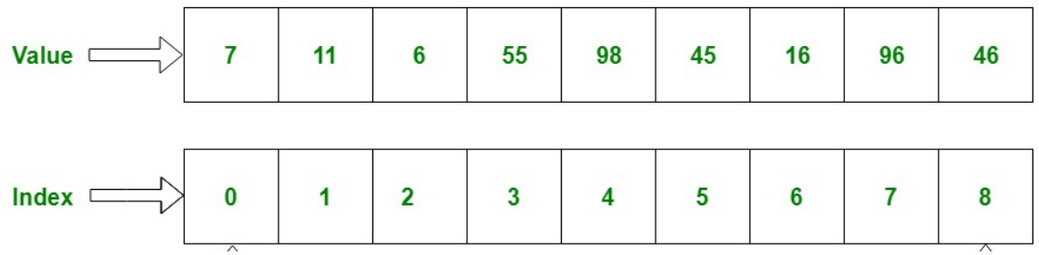
NumPy란?

NumPy = Number + Python

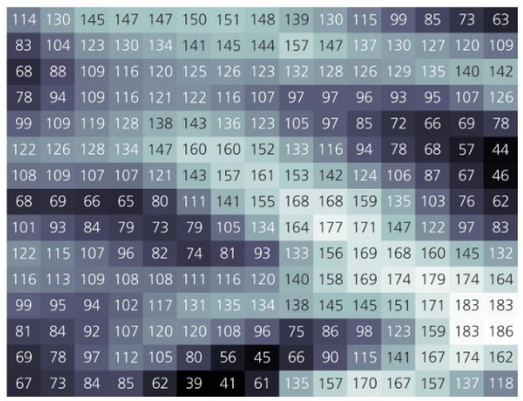
- **파이썬 라이브러리 (NumPy v1.18., http://www.numpy.org)**
- 행렬이나 다차원 배열 처리, 계산과학 분야의 복잡한 연산을 지원
- Scipy나 Matplotlib, Pandas 등으로 발전, 더 복잡한 연산을 쉽게 처리가능하도록 지원함.

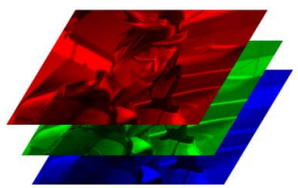
리스트, 벡터, 행렬, 배열

- 리스트, 스택, 큐 등의 기존의 자료 구조는 문자열 처리에 적합



- 행렬 형태의 사운드, 이미지, 영상은?





NumPy

- **ndarray class**
  - 배열을 이용한 벡터화연산
  - 배열에는 모두 동일한 자료형만 저장이 가능(★)
  - 메모리에 연속적으로 저장(★)
  - 일반적으로 리스트보다 빠른 것으로 간주

배열(Arrays) 생성

- 모듈 임포트

```
import numpy as np
```

- 1-D array 생성

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

- 2-D array 생성

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

- 3-D array 생성

```
ar = np.array([[[1,2,3], [4,5,6]],
               [[7,8,9], [10,11,12]]])
```

배열(Arrays) 생성

명령어	설명	사용법
np.ones	행렬을 1로 채워준다.	np.ones(3,6)
np.zeros	행렬을 0으로 채워준다.	np.zeros(3,6)
np.arange	주어진 범위의 숫자들로 행렬을 만들어준다.	np.arrange(1,100,5)
np.random.random	0~1사이의 실수로 행렬을 생성한다.	np.random.random(1,10)

배열(Arrays) 생성

명령어	설명	사용법
np.ones	행렬을 1로 채워준다.	np.ones(3,6)
np.ones_like	대상과 같은 shape의 행렬을 1로 채워준다.	np.ones_like(ar)
np.zeros	행렬을 0으로 채워준다.	np.zeros(3,6)
np.zero_like	대상과 같은 shape의 행렬을 0으로 채워준다.	np.zeros_like(ar)
np.arange	주어진 범위의 숫자들로 행렬을 만들어준다.	np.arrange(1,100,5)
np.linspace	주어진 범위를 개수만큼 분할한다.	np.linspace(1,100,5)
np.logspace	주어진 범위를 개수만큼 로그로 분할한다.	np.logspace(1,100,5)
np.random.random	0~1사이의 실수로 행렬을 생성한다.	np.random.random(1,10)

배열(Array) 명령어

명령어	설명	사용법
shape	행렬의 모양을 확인	np.shape(ar)
reshape	행렬의 모양을 변경	ar.reshape(3,5)
ndim	행렬의 차수(dimension, 깊이)를 확인	np.ndim(ar)
len	길이를 확인	len(ar), len(ar[0])
type	형식을 확인	type(ar)
T	행렬을 치환	ar.T
Slicing	1차원 배열: 리스트와 동일 N차원 배열: ‘,’ 로 구별하여 연결	ar[1:2] ar[1:, 2:, 3:]

NumPy

- **Vectorized operation**
  - 원소단위로 계산하지 않고 배열 단위로 계산을 지원
  - Loops 없이 연산하여 코딩의 양이 줄어듬 →병렬처리 가능 (multi-core)

- 배열 단위로 사칙연산, 논리연산, 비교연산 가능
  - 덧셈, 뺄셈, 곱셈, 나눗셈, >, <, ==, !=
- 지수함수, 로그함수 지원
  - Exp, **, log( )

Review Practice

- np.array를 이용하여 [3, 6, 8, 10, 11]로 행렬을 생성하시오.

- 임의의 실수 50개로 이루어진 ndarray A를 생성하시오
- A의 모양을 (2,5,5) 바꾸시오.

- 1~12의 정수로 이루어진 (3,4) A와
  linspace로 1~12의 정수 3개로 이루어진 (3,1) B를 생성하시오.
- C=A+B와 C'=B+A를 계산하고 C와 C'가 동일한 지 비교하시오.

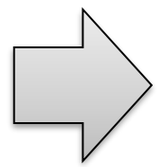
Arrays

min, max

```
ar = np.random.random((2,2,4))
ar
```

```
array([[[0.88931355, 0.37482078, 0.02377386, 0.51096842],
        [0.43388499, 0.38019536, 0.23263719, 0.48683875]],

       [[0.52503506, 0.06588319, 0.58253552, 0.48954106],
        [0.5845651 , 0.02081269, 0.37078975, 0.83175597]]])
```



```
np.min(ar[:1,])
```

```
0.023773856142962346
```

```
np.max(ar)
```

```
0.8893135456505813
```

- 배열에서 최소값과 최대값을 반환 (slicing 범위 조정)

Arrays

min, max

```
ar = np.random.random((2,2,4))
ar
```

```
array([[[0.90896646, 0.7524607 , 0.51694865, 0.24760385],
        [0.93605071, 0.13022272, 0.3655778 , 0.65051493]],

       [[0.43077222, 0.32467624, 0.82399023, 0.25208353],
        [0.74939392, 0.55698676, 0.30123888, 0.20338412]]])
```

```
np.argmin(ar)
```

```
5
```

```
np.argmax(ar)
```

```
4
```

- 배열에서 최소값과 최대값을 반환 (slicing 범위 조정)
- argmin 과 argmax 는 최대와 최소의 위치를 반환

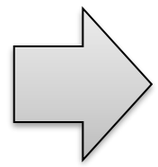
Arrays

min, max

```
ar = np.random.random((2,2,4))
ar
```

```
array([[[0.90896646, 0.7524607 , 0.51694865, 0.24760385],
        [0.93605071, 0.13022272, 0.3655778 , 0.65051493]],

       [[0.43077222, 0.32467624, 0.82399023, 0.25208353],
        [0.74939392, 0.55698676, 0.30123888, 0.20338412]]])
```



```
np.argmin(ar)
```

```
5
```

```
np.argmax(ar)
```

```
4
```

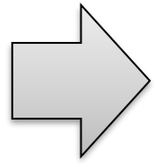
- 배열에서 최소값과 최대값을 반환 (slicing 범위 조정)
- argmin 과 argmax 는 최대와 최소의 위치를 반환
- np.random을 사용하여 (3, 5, 6)을 생성하고, 최대, 최소를 검색하시오. 또한 argmax와 argmin으로 위치를 확인하시오.

Arrays

sum, mean, median

```
ar=np.arange(1, 31, 2).reshape(3,5)
ar
```

```
array([[ 1,  3,  5,  7,  9],
       [11, 13, 15, 17, 19],
       [21, 23, 25, 27, 29]])
```



```
np.sum(ar)
```

```
225
```

```
np.mean(ar)
```

```
15.0
```

```
np.median(ar)
```

```
15.0
```

- 행렬전체합, 평균, 중간값을 계산

Arrays

sum, mean, median

```
ar=np.arange(1, 31, 2).reshape(3,5)
ar
```

```
array([[ 1,  3,  5,  7,  9],
       [11, 13, 15, 17, 19],
       [21, 23, 25, 27, 29]])
```

- 행렬전체합, 평균, 중간값을 계산
- sum은 axis를 기준으로 행과 열의 합계산이 가능

```
np.sum(ar, axis=0)
```

```
array([33, 39, 45, 51, 57])
```

```
np.sum(ar, axis=1)
```

```
array([ 25,  75, 125])
```

```
np.sum(ar, axis=0)
```

```
array([33, 39, 45, 51, 57])
```

```
np.sum(ar[1:], axis=0)
```

```
array([32, 36, 40, 44, 48])
```

Arrays

- **다음을 계산하시오.**
  - 전체 행렬의 합, 평균과 중간값
  - 2번째 행렬의 합, 최소값, 최대값
  - 1,4번째 열을 제외한 부분 행렬의 최대, 최소, 평균, 중간값

```
array([[[ 0.5,  2. ,  4. , -5. ],
        [ 0.7,  9. ,  3. ,  0.7],
        [ 8. ,  1. , -2. ,  0.3]],

       [[ 1.2,  4. ,  2. , -9. ],
        [ 8. ,  3. ,  2. ,  1. ],
        [ 2. ,  8. ,  5. , -1. ]],

       [[ 3. ,  9. ,  6. ,  0.8],
        [ 0.9, -1. ,  3. ,  4. ],
        [ 2. ,  5. ,  7. ,  9. ]]])
```

행렬곱 (내적)

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} = \begin{bmatrix} (a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}) & (a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32}) \\ (a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31}) & (a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32}) \\ (a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31}) & (a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32}) \\ (a_{41}b_{11} + a_{42}b_{21} + a_{43}b_{31}) & (a_{41}b_{12} + a_{42}b_{22} + a_{43}b_{32}) \end{bmatrix}$$

$$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \quad y = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

$$x^T y = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32$$

Arrays

- **행렬곱 (내적)**
  - 행렬과 행렬의 곱
    - 행렬과 스칼라의 곱과 구별
    - a*b
  - np.dot(a,b) 나 a@b 로 실행

```
a = np.arange(1,21,2).reshape(2,5)
b = np.arange(3,13,1).reshape(5,2)
print (a,'₩n', b)
a@b
```

```
[[ 1  3  5  7  9]
 [11 13 15 17 19]]
 [[ 3  4]
 [ 5  6]
 [ 7  8]
 [ 9 10]
 [11 12]]

array([[215, 240],
       [565, 640]])
```

Arrays

- 다음 행렬 곱을 손으로 계산한 후, dot 연산으로 계산하여 결과를 비교하시오.

(1)
$$\begin{bmatrix} 3 & 2 & -1 \\ 0 & 2 & 4 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ 3 & 4 \\ 5 & 1 \end{bmatrix}$$

(2)
$$\begin{bmatrix} 1 & 9 \\ 3 & 4 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} 1 & 4 & -1 \\ 7 & 1 & 4 \end{bmatrix}$$

- **행렬곱 (내적)**
  - 다음은 행렬연산의 규칙이다. 행렬A, B, C가 다음과 같이 주어졌을 때, 이 규칙이 올바르게 적용되는지 dot연산과 T연산을 이용하여 확인해보자.

$$AB \neq BA$$

$$A(B+C) = AB + AC$$

$$(A+B)C = AC + BC$$

$$(AB)^T = B^T A^T$$

행렬곱 (내적)

- 다음은 행렬연산의 규칙이다. 행렬A, B, C가 다음과 같이 주어졌을 때, 이 규칙이 올바르게 적용되는지 dot연산과 T연산을 이용하여 확인해보자.

$$AB \neq BA$$

$$A(B+C) = AB + AC$$

$$(A+B)C = AC + BC$$

$$(AB)^T = B^T A^T$$

A=
```
[[ 0  2  4  6]
 [ 8 10 12 14]
 [16 18 20 22]
 [24 26 28 30]]
```

B=
```
[[ 1  3  5  7]
 [ 9 11 13 15]
 [17 19 21 23]
 [25 27 29 31]]
```

C=
```
[[ 2  4  6  8]
 [10 12 14 16]
 [18 20 22 24]
 [26 28 30 32]]
```

- **행렬곱 (내적)**
  - 아래의 연산에 활용될 수 있음
    - 가중치 합 (Weighted Sum)
    - 가중치 평균 (Weighted average)
    - 유사도 검사 (Similarity check)

Arrays

- **행렬곱 (내적)**
  - 가중치 합 (Weighted Sum)
  - 요소들의 동일한 가치로 합산하는 것이 아니라 중요도에 따라 가중값을 곱해준 후 합산

$$w_1 x_1 + \cdots + w_N x_N = \sum_{i=1}^{N} w_i x_i$$

Arrays

- **행렬곱 (내적)**
  - 가중치 합 (Weighted Sum)
  - 요소들의 동일한 가치로 합산하는 것이 아니라 중요도에 따라 가중값을 곱해준 후 합산

$$w_1x_1 + \cdots + w_Nx_N = \sum_{i=1}^{N} w_ix_i \quad \Rightarrow \quad \begin{bmatrix} w_1 & w_2 & \cdots & w_N \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

  - ex) 사과, 바나나, 복숭아를 사는데 개수와 가격이 각각다르다

Arrays

- **행렬곱 (내적)**
  - 가중치 평균 (Weighted Average)
  - 각각의 가중치를 전체 가중치의 합으로 나누어 계산

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i = \frac{1}{N}\mathbf{1}_N^T x$$

  - ex) 성적환산시 학점별로 가중치를 곱해준 다음 평균을 계산

Arrays

- **행렬곱 (내적)**
  - 유사도 검사 (Similarity check)
  - 두 벡터가 비슷할 수록 높은 점수가 나옴
  - Cosine similarity 사용 가능

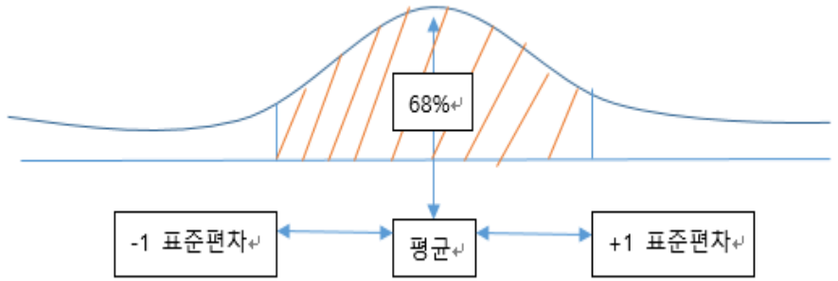
$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

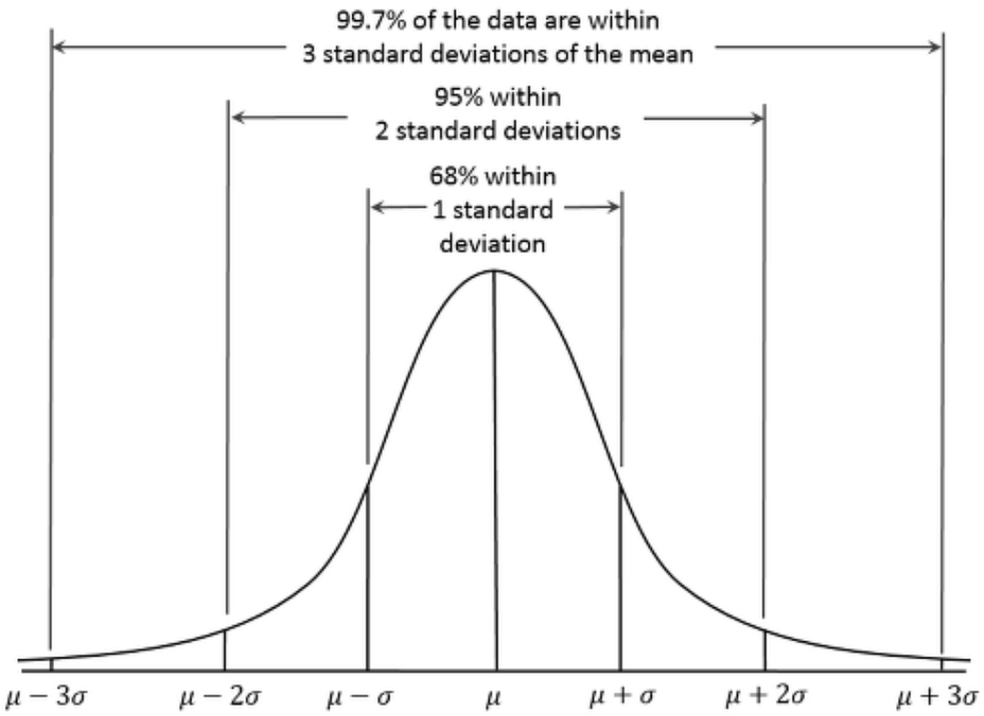
Arrays

- **var(분산), std(표준편차)**
  - 분산
    - 거리차의 제곱의 합,
    - 작을 수록 데이터가 뭉쳐있음
  - 표준편차
    - 분산의 제곱근,
    - 작을 수록 평균값에 뭉쳐있음

Arrays

- **var(분산), std(표준편차)**
  - 분산
    - 거리차의 제곱의 합,
    - 작을 수록 데이터가 뭉쳐있음
  - 표준편차
    - 분산의 제곱근,
    - 작을 수록 평균값에 뭉쳐있음





Arrays

```
array([ 1. ,  3. ,  8. , 12. , 14. , 17. , 20. , 21. , 24. , 30. , 33. ,
       34. , 35. , 38. , 42. , 43. , 44. , 45. , 46. , 47. , 48. , 49. ,
       50. , 51. , 51.5, 47.8, 48. ,  1. , 49. ,  9. , 45. ,  8. , 52. ,
       53. , 54. , 55. , 56. , 57. , 58. , 59. , 55. , 57.1, 61. , 69.1,
       54.3, 60. , 61. , 62. , 63. , 68. , 69. , 72. , 73. , 74. , 79. ,
       80. , 84. , 85. , 89. , 59. , 94. , 51. , 54. , 59. , 48. , 37. ,
       44. , 49. , 41. , 48. , 61. , 64. , 68. , 47. , 48. , 49. , 39. ,
       49. , 49. , 52. , 53. , 54. , 52. , 49. ])
```

```
np.var(ar)
```

```
385.9455725623583
```

```
np.std(ar)
```

```
19.645497513739 84
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
       69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
       86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

```
np.var(ar2)
```

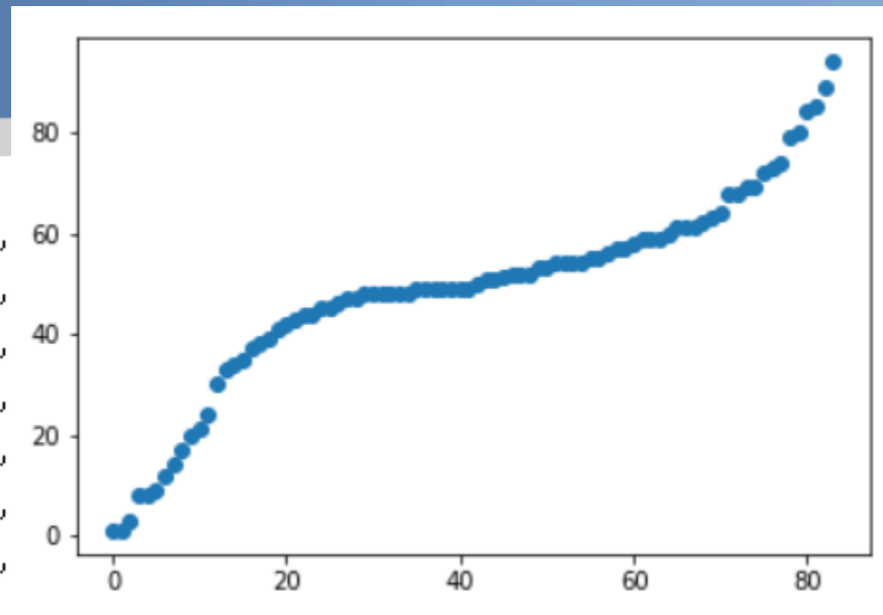
```
816.6666666666666
```

```
np.std(ar2)
```

```
28.577380332470412
```

Arrays

```
array([ 1. ,  3. ,  8. ,                                   , 33. ,
       34. , 35. , 38. ,                                    , 49. ,
       50. , 51. , 51.5,                                    , 52. ,
       53. , 54. , 55. ,                                    , 69.1,
       54.3, 60. , 61. ,                                    , 79. ,
       80. , 84. , 85. ,                                    , 37. ,
       44. , 49. , 41. ,                                    , 39. ,
       49. , 49. , 52. ,
```



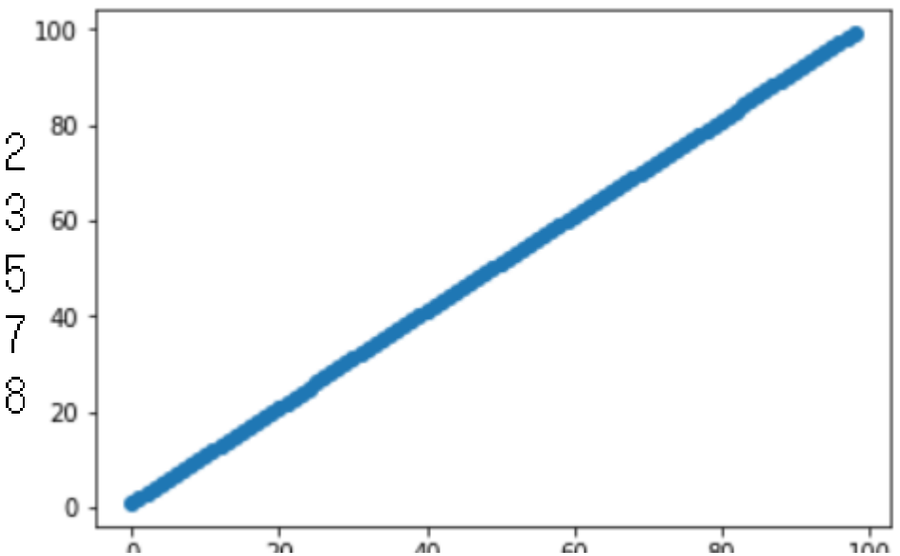

```
np.var(ar)
```

```
385.9455725623583
```

```
np.std(ar)
```

```
19.64549751373984
```

```
array([ 1,  2,  3,
       18, 19, 20, 2
       35, 36, 37, 3
       52, 53, 54, 5
       69, 70, 71, 7
       86, 87, 88, 8
```




```
3, 14, 15, 16, 17,
0, 31, 32, 33, 34,
7, 48, 49, 50, 51,
4, 65, 66, 67, 68,
1, 82, 83, 84, 85,
3, 99])
```

```
np.var(ar2)
```

```
816.6666666666666
```

```
np.std(ar2)
```

```
28.577380332470412
```

Arrays

sort

```
ar = np.arange(1,15)
np.random.shuffle(ar)
ar
```

```
array([ 7, 14, 12, 10, 13,  2,  3,  9,  4,  8,  1, 11,  5,  6])
```

```
print(np.sort(ar))
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14]
```

- 배열의 값을 오름차순으로 정렬
- sort in-place
  - np.random.shuffle()은 배열 shuffling 명령
- 2차원 이상인 경우 axis에 따라 정렬이 가능

Arrays

sort

```
ar=np.arange(1,16)
np.random.shuffle(ar)
ar=ar.reshape(3,5)
ar
```

```
array([[ 1, 14,  9, 12,  7],
       [11,  6,  3,  2,  4],
       [13, 10, 15,  5,  8]])
```

```
ar2=np.sort(ar)
ar2
```

```
array([[ 1,  7,  9, 12, 14],
       [ 2,  3,  4,  6, 11],
       [ 5,  8, 10, 13, 15]])
```

```
ar3=np.sort(ar, axis=0)
ar3
```

```
array([[ 1,  6,  3,  2,  4],
       [11, 10,  9,  5,  7],
       [13, 14, 15, 12,  8]])
```

- axis = -1. 1 이 기본, axis = 0,

Arrays

- **다음을 수행하시오.**
  - 1~100 의 정수로 (10,10) 행렬을 생성한 후, np.random.shuffle()을 이용하여 행렬을 뒤섞은 다음, 행렬을 정렬하시오.

  - 1~30 의 정수로 (3,10) 행렬을 생성한 후, np.random.shuffle()을 이용하여 행렬을 섞은 다음 axis 값을 바꾸어 가면서 정렬의 결과를 확인하시오.

Matplotlib