11.

```
a=[[1],[2,3],[4,5,6]]
n=0
for b in a:
    for d in b :
        n+=1
print(n)
```

6

12.13.

```
n=[1,2,3,4]
for I in range(len(n)):
     print(i)
```

6번의 함수 실행

18.

```
t=0
for p in range(3):
    if p <=1:          0,1
     for q   in range(4):            0,1,2,3
         t+=q(t=t+q)
     for r in range(2):               0,1
          t +=r(t=t+r)
print(t*r)
```

14

10/31

숫자들의 나열은 벡터의 형태가 된다.

행렬은 벡터는 아니지만, 길게 만들면, 한줆한줄 벡터화할 수 있다.

(이미지는 행렬이다. 그걸 길게 눌려놓은 것을 벡터화한다라고 말한다.)

모든 데이터는 벡터형태로 했을 때 다루기가 쉽다.

칼라 이미지는 3차원, 동영상은 4차원

소리도 벡터화 된다.

텍스트도 벡터화시킨다. 1첫째 단어 1000000000000 (0을 5만개)

numpy는 라이브러리

```
a=[1,3,5]
b=[2,4,6]
```

c=a+b
c # 곱하기, 더하기 안되는데, 이것을 해주는 것이 numpy이다.
→ [1,3,5,2,4,6]

import numpy # import numpy as np A=np.array(a), B=np.array(b)
A=numpy.array(a)
B=numpy.array(b) #array는 계산가능한 것을 바꾸어준다.
A+B
→ array([3, 7, 11])

type(A)
→ numpy. narray

import numpy as np
A=np.array(a) ; B=np.array(b)
X=np.array([[1,2,3],[4,5,6]])
X
→ array([[1,2,3],
[4,5,6]])
X.shape
→(2,3)

11.01
1. numpy.A.D.f
2. import numpy
numpy.A.D.f
3. import numpy
from numpy import A
A.B.f
4. import numpy
from numpy import A.D

#numpy 안에 A와 B가 부분집합으로, A안에 D
#import numpy numpy.A.D.f
#from numpy import A
A.D.f
#from numpy import A.D

import numpy as np
import matplotlib.pyplot as plt # from metaplotlib import pyplot as plt

. 은 포함관계이다.

np.empty([2,3], dtype='int')
→ array([[0, 1072168960, 0],
[1072168960, 0, 0]])

np.empty([2,3])
→array([[4.64583300e-308, 4.66664552e-308, -2.57703408e+010],
[-4.70231646e-064, 2.26262303e-319, 1.46914166e+195]])

np.zeros([2,3])
→ array([[0., 0., 0.],
[0., 0., 0.]])

[[0,0,0],[0,0,0]] #list를 만든 것, 그런데 계산 안 된다. 계산되는 array로 만드는 방법이 밑에,
np.array([[0,0,0], [0,0,0]]) #np.zeros([2,3])과 똑같음
그냥 [[0,0,0], [0,0,0]] 은 벡터일 뿐이다.
np.ones([2,3]) #1이 6개 만들어진다. ones라는 함수는 데이터타입을 float로 하는구나 추측, 왜? 결과에 1. 이 돼서 = 따라서, np.ones([2,3], dtype='int')
#float64라고 쓰면 조금의 오차도 만들지 않으려고, 단점은 메모리를 차지한다. 정확도와 데이터 쓰는 양과 메모리가 반비례

np.arange(5) #for loop에서 range는 계산은 안 됨, 여기서는 계산이 됨.
→array([0,1,2,3,4])

np.arange(0,10,2)
→array([0,2,4,6,8])

np.arange(0,10,2, dtype='float64')
→ array([0., 2., 4., 6., 8.])

np.linspace(0,10,6) #0부터 10까지 6개의 숫자로 나눠라. linearspace에서 linear은 늘 똑같다. 첫 번째에서 두 번째로 가는, 두 번째에서 세 번째로 가는 차이가 다 똑같다.
→ array)[0., 2., 4., 6., 8., 10.])

X=np.array([4,5,6])
X
→ array([4,5,6]) #벡터가 만들어진다.

X=np.array([[1,2,3],[4,5,6]]) #2by3

```
X=np.array([[1,2],[4,5],[8,9)])  #3by2
X
→array([[1,2],
        [4,5],
        [8,9]])
```

X=np.array([[1,2], [4,5], [8,9]]) #그냥 벡터는 1차원, 직사각형 매트릭스 2차원, volume이 있으면 3차원, 3차원으로 표기도 가능하다. 대괄호가 두 개 나오면 2차원, 대괄호가 세 개 나오면 3차원이다.
X=np.array([[[1,2], [4,5], [8,9]], [[1,2], [4,5], [8,9]]]) #2차원 행렬이 두 개 있으면 3차원이네

```
X
→ array([[[1,2],
          [4,5].
          [8,9]].
         [[1,2],
          [4,5],
          [8,9]]])

X.ndim
→3
X.shape
→(2,3,2)  # 3x2 가 2개 있다.

X.dtype
→ dtype('int32')

X.astype(np.float64)
→ array([[[1., 2.],
          [4., 5.],
          [8., 9.]],

         [[1., 2.],
          [4., 5.],
          [8., 9.]]])

np.zeros_like(X) # X*0 과 같다.
→ array([[[0, 0],
          [0, 0],
          [0, 0]],
```

[[0, 0],
[0, 0],
[0, 0]]])

data= np.random. normal(0,1,100) #정규분포 모양의 데이터를 만들어준다. 평균0, 표준편차1, 100(100개의 데이터를 만들어라.) random 하게 만들어봐라.

print(data)

data.shape

→(100,)

data.ndim

→1

data=np.random.normal(0,1,100)

print(data)

plt.hist(data, bins=10) #matplot이라는 라이브러리 안에 있는 plt라는 subset안에 있는 hist
바구니가 10개

plt.show() # 그래프의 y축에는 0을 포함한 정수값만 나올 수 있다. 전체 값은 몇 개 인가? 100개이다. (시험)

X=np.ones ([2,3,4]) #직사각형은 2차원, 3차원은 그게 여러 장 쌓인 것 [3,2,4] 이런 식으로 24개에 맞춰서 바꿀 수 있다. 그런데 reshape(-1,3,2) 는 -1은 니가 알아서 해라.

X

→ array([[[1., 1., 1., 1.],
[1., 1., 1., 1.],
[1., 1., 1., 1.]],

[[1., 1., 1., 1.],
[1., 1., 1., 1.],
[1., 1., 1., 1.]]])

2by3by4 에 들어있는 element의 개수 24개 – 바꾸더라도 24개가 되도록 바꿀 수 있다. 안되면 reshape은 error message가 뜬다.

Y=X.reshape(-1,3,2) #4인데, 4인지 뭔지 모르겠다. 그러면 -1넣을 수 있다. 4넣어도 똑같은 값이 나온다.

Y

→array([[[1., 1.],
[1., 1.],
[1., 1.]],

```
       [[1.,  1.],
        [1.,  1.],
        [1.,  1.]],

       [[1.,  1.],
        [1.,  1.],
        [1.,  1.]],

       [[1.,  1.],
        [1.,  1.],
        [1.,  1.]]])
```

np.allcose(X.reshape(-1,3,2),Y)
true
#allclose는 똑같은지 아닌지 판단하는 함수

a=np.random.randint(0,10,[2,3])
b=np.random.random([2,3])
np.savez("test",a,b) ---- 파일로 저장

! ls-al test* --- 실제 저장이 됬는지 안됬는지 보는 것이다.

del a,b #메모리 상에서 지우고 싶으면,
%who #variable중에 뭐가 available한 지가 나옴.

npzfiles = np.load("test.npz") #저장한 후 파일을 불러오고 싶을 때 (앞에 저장했으니)
npzfiles.files
→['arr_0', 'arr_1']
npzfiles['arr_0']
→array([1,0,3],
       [4,0,6]))

Comma Separated Values (CSV) (엑셀에서도 부를 수 있음)
data=np.loadtxt("regression.csv", delimiter=",",skiprows=1, dtype={'names': ("X", "Y"),'formats': ('f','f')}) #데이터를 파일로 주는 방법이다, delimiter 분리, skiprows는 첫 번째 row(X, Y)는 빼겠다. f는 float라는 것, integar하고 싶으면, I쓰면 됨.

np.savetxt("regression_saved.csv", data, delimiter=",")
!ls -al regression_saved.csv : 저장하는 것

```
arr= np.random.random([5,2,3])

print(type(arr))
print(len(arr))
print(arr.shape)
print(arr.ndim)
print(arr.size)
print(arr.dtype)
→ <class 'numpy.ndarray'>
5
(5, 2, 3)
3
30
float64

a=np.arange(1,5)  #index만들어내라 1,2,3,4
b=np.arange(9,5,-1) #index 만들어내라 9,8,7,6
-- 계산 가능한 상태가 되었기 때문에,
print(a-b)
print(a *b) 계산 가능하다.
a,b 모두 numpy array이기 때문이다.

matrix는 다음에

a=np.arange(1,10).reshape(3,3) #reshape 해서 2차원으로 만들자
b=np.arange(9,0,-1).reshape(3,3)
→[[1 2 3]
   [4 5 6]
   [7 8 9]]
  [[9 8 7]
   [6 5 4]
   [3 2 1]]

a==b
→array([[False, False, False],
        [False,  True, False],
        [False, False, False]])
a>b
→ array([[False, False, False],
         [False, False,  True],
         [ True,  True,  True]])
```

#비교할려면 dimension과 shape이 같아야 한다.

a.sum() #a가 numpy로부터 만들어진 산물이다. a라는 자체가 numpy이다. numpy가 만들어냈다. 함수니까 괄호친다. 자기 자신을 sum하는 건데 ()뭐 안써도 된다.
np.sum(a) #위의 것과 같다. np속에 있는 sum을 쓸건데, 뭘 sum을 하냐? a

a.sum(axis=0), # 3x3 의 2차원, 첫 번째 차원은 1,2,3, 4,5,6, 7,8,9 두 번째 차원은 1,4,7 2,5,8 3,6,9 // a.sum(axis=0)은 행의 관점에서 sum을 해라
→ (array([12, 15, 18])
 np.sum(a, axis=0) #행의 관점에서 합을 하는 것이다. 첫 번째 차원

a.sum(axis=1), np.sum(a, axis=1)
→ array([6, 15, 24]), array([6, 15, 24]))

#reshape이 브로드캐스팅이다. (broadcasting)
a=np.arange(1,25). reshape(4,6) #2차원으로 4X6
→ 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]])

a + 100
→
array([[101, 102, 103, 104, 105, 106],
       [107, 108, 109, 110, 111, 112],
       [113, 114, 115, 116, 117, 118],
       [119, 120, 121, 122, 123, 124]])

b = np.arange(6)
b
→ array([0, 1, 2, 3, 4, 5]) #a는 4by 6, b는 그냥 6

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

#sound가 continous wave이지만, 컴퓨터에서는 하나하나 값들로 담김. 1초동안 간격은 같지만, 얼마나 듬성듬성, 빽빽이 담을 것인가?=sampling rate.
#sampling rate를 만으로 한다고 정의한다면. 1초동안 만 개의 숫자를 담을 거다. 4만 4천 백이 CD , 그 이상은 사람의 귀로는 구분이 안되는데, 4만 4천 백이 인간이 구분할 수 있는 최대 음질,
1초를 표현하는 데 숫자의 개수가 4만 4천 백 개 있다. 아 - 소리 내는데 기록하는데 19만 ++ sive wave를 19만 2천개의 숫자로 표현도 되고, 3백 개도 되고, 이게 sampling rate이다. 1초에 얼마이다. HZ를 반드시 붙인다. 1초에 몇 번 떨리느냐 1초 당이라는 말이 들어가는 순간 모든 경우에서 Hz를 쓴다. 1초에 몇번 왔다갔다 하는가 pitch표현할 때도,
#sampling rate