넘파이는 리스트보다 속도가 빠르다, 여러 딥러닝 자료분석에 많이 사용한다.
아나콘다 설치시 자동으로 인스톨 된다. 따로 인스톨 받아도 된다.

* numpy 공식페이지 참고.
https://docs.scipy.org/doc/numpy/index.html

Numpy 모듈
import numpy as np

1. array(배열) 함수로 생성
  x = np.array([1,2,3,4,5]) : 1차원 백터 생성 : 행
  print(x)
  print(x.shape) //모양확인

  y = np.array([[2,3,4,5],[1,2,5,4]]) : 행,열 / 개수가 다르면 list로 만든다.
  print(y)
  ptirn(y.shape)
  print(type(y)) : type확인 해보기, 잘 사용하지 않는다.
  print(y.shape[0], y.shape[1]) 0:행, 1:열을 가르킨다.

 z = np.array([[[0,1,2], [3,4,5]], [[0,1,2], [3,4,5]]]) : 면,행,열
z
z.shape (2,2,3)
z.shape[2] (3)

2. np.arange 함수로 생성
  np.arange(10) : 0~9까지의 array생성
  np.arange(1,10)
  np.arange(1,10,2)
  np.arange(1,10).reshape(3,3)

3. np.ones, np.zeros 생성하기
  np.ones((4,5)) : 모든원소가 1인 행렬을 만든다.(2차원)
  np.ones((2,3,4)) : 3차원의 행렬생성
  np.zeros((2,3,4)) : 모든원소가 0인 행렬 생성

4. np.empty, np.full 로 생성
  np.empty((3,4)) : 3행4열을 만들지만 쓰레기값으로 들어간다. 초기화가 되지 않는다.(이상한값이 들어간다)
  np.full((3,4), 7) 모든 원소를 7로 초기화한다.

5. np.eye로 생성
  단위행렬생성(대각선 값이 1이고 나머지는 0)
  np.eye(5) : 3, 6 7 로 변경해 본다.

6. np.linspace로 생성 (균등하게 눈금줄 때 많이 사용)
  np.linspace(1,10,3) : 1~10을 포함 전체를 3개가 되게 간격을 나누어라.
  np.linspace(1,10,4)
  np.linspace(1, 10, 5)

7. reshape 함수 : 다양한 차원의 행렬로 변경가능
  x = np.arange(1,16)
  print(x) :1차원행렬
  x.reshape(3,5) : 2차원으로 변경
  x.reshape(5,3)
  x.reshape(5,3,1) :3차원으로 변경

** 주의: 변경할 개수가 같아야 한다. 모자라면 안된다.
a1 = np.arange(12)
print(a1, a1.shape)

a2 = a1.reshape(3,4)
print(a2, a2.shape)

a3 = a1.reshape(-1, 6) :행의 요소수를 자동으로 계산,2
print(a3, a3.shape) : (2,6)
바꿔서 해보기

a2 = a1.reshape(4, -1) : 열의 요소수를 자동으로 계산,
print(a2, a2.shape) :(4,3)
바꿔서 해보기

a3 = a1.reshape(2,-1,2) :3차원으로 행을 자동으로 생성
print(a3, a3.shape) : (2,3,2)
바꿔서 해보기

** numpy 안에는 random서브모듈이 있다.
>> random, randn, randint
1. random 함수
  np.random.rand(2)
  np.random.rand(2,3) : 2행 3열의 0~1사이의 랜덤수(할때마다 다르다)
  np.random.rand(4,2,3)

2. randn 함수
  np.random.randn(3,4) : 3행4열 짜리 정규분포값(음수도 나온다.)
  np.random.randn(3,4,2)

3. randint 함수 (shift + tab 으로 내용확인)
  np.random.randint(1,100) :1~100중 하나의 정수값

np.random.randint(1,100, size=(3,5)) :1~100사이의 정수를 3행5렬짜리 행렬
np.random.randint(1,100, size=(5,)) :1차원 5개 ‘,’ 안해도 같다.

** 랜덤값이 계속 바뀌어나오면 분석할 때 정확한 분석이 어렵다. 실행때 마다 같은 값이 나오도록 해야 한다.

4. seed함수 (랜덤한 값을 동일하게 다시 생성하고자 할 때 사용)
np.random.seed(100) :정수 아무거나 넣어도 됨
np.random.randn(3,4) : 값이 동일한 것을 알 수 있다.

5. choice 함수(주어진 1차원 ndarray로부터 랜덤으로 샘플링)
np.random.choice(100) : 하나만 가져온다.
np.random.choice(100, size = (3,4)) : 0~100 사이의 랜덤한 수를 2차원으로
np.random.choice(100,size =(3,4)).reshape(3,-1)

x= np.array([1,2,3,1,5, 2.6, 4.9])
np.random.choice(x, size=(2,2)) : x에서 2행2렬의 랜덤수 가져오기.
np.random.choice(x, size=(2,2), replace = False) : 기본은 True(같은값허용)
np.random.choice(x, size=(2,2), replace = True)

>> numpy_example을 이용하여 복습하고 본인정리할시간 주기

>> ppt 슬라이도 ~5까지 그래프 그리기 하고 다음 진도 나가기

6. 확률분포에 따른 ndarray 생성
참고사이트:
https://rfriend.tistory.com/284

np.random.uniform(1.0, 3.0, size=(4,5)) //균일 분포, 최소1.0에서 최대 3.0의 사이의 임의의수
np.random.normal(1.0, 3.0, size=(3,4)) : 정규분포(가우시안)
np.random.randn(3,4) 와 동일

7. ndarray 인덱싱
1차원>
인덱싱 : d[행번호, 열 번호]
x = np.arange(10) : 0~9인 10개원소생성
x[0] : 첫 번째
x[9] or x[-1] : 마지막
x[0]= 100 : 값 변경

2차원>
x = np.arange(10).reshape(2,5) : 2차원으로 변경
print(x, x.shape)
print(x[0]) : 1행의 값(하나의 vector출력)
print(x[0][0]) : 0 old style
print(x[0,2]) : 0행의 2번째 열 , 파이썬 스타일

3차원
  x = np.arange(36).reshape(3,4,3) : 3차원으로 변경
  print(x, x.shape)
  print(d[0][0][0]) : old style
  print(x[0]) : python style
  print(x[1,2]) : python style
  print(x[1,2,2])

7.1 fancy 인덱싱: 정수나 불린 값을 가지는 다른 numpy배열을 이용하여 배열을 인덱싱할 수 있는 기능
인덱싱은 동일 메모리 값을 가져와서 반환한다.

>> 정수값사용해서 가져오기와 다른 배열을 사용하여 가져오기

1차원
>>위치값이용하기
a = np.array([10, 21, 22, 31, 41, 55, 67, 78, 80])
print(a, a.shape)
print(a[[1,3,5,2,3]]) : [21, 31, 55, 22, 31]
print(a[[0,-1,-3,-2]] : [10, 80, 67, 78]

>>불린값 이용하기
index = np.array(a>50)
print(a[index])

2차원
c = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
d = np.array([0, 2])
e = np.array([1, 2])
print(c[d, e]) : [1 8] //(0,1) (2,2)값

a= np.arange(10,100,10).reshape(3,3) :
print(a, a.shape)
a[[0,2]] : 0행, 2행
print(a[[0,2],[0,1]) : 0,0과 2,1 의 값이 나온다. [10 80]

print(a[[0,-1,-3,-2]]
[[10 20 30]
 [70 80 90]
 [10 20 30]
 [40 50 60]]

>> 복습.정리 하기

8. ndarray 슬라이싱.

x = np.arange(10) : 1차원
print(x, x.shape)
x[1:7] :1부터가져오고 7은 포함하지 않는다.
x[1:]
x = np.arange(10).reshape(2,5) : 2차원
print(x, x.shape)
x[:, 1:4] : 모든행을 가져오고 1에서 3까지 가져오기
x[:, :2] : 모든행을 가져오고 2개가져오기
x[0, :2] : 행열이 아니라 크기가 2인 벡터,창원이 줄어든다.
x[:1, :2] : 차원은 안줄었다 1행2열
인덱싱은 차원이 준다, 슬라이싱은 차원은 유지하면서 자른다.

ex1)
a = np.arange(10)**2
print(a, a.shape)
# Slicing
print(a[2:5])
print(a[:4:2]) # equal to a[0:4:2]
print(a[4::2]) # eqaul to a[4:len(a):3]
print(a[::-1]) # reversed array //끝에서 모두 가져온다.

ex2)
d = np.arange(12).reshape(3,4)
print(d, d.shape)
print(d[0]) : [0 1 2 3]
print(d[0, :]) : [0 1 2 3] //0행의 처음~끝
print(d[:, :-1]) : (2,3) 마지막열 제외 //모든행, 각행은 마지막제외
‘ :: -> 0:10:2 (0에서 9까지 2개 건너가져오라)
print(d[::2, ::2]) : (2,2)
//행 :처음~끝 2개씩 건너니 0행,2행 가져온다. 열: 처음부터끝까지 2개씩 건너니 0열 2열 가져온다.
d[0] :0번행
d[0] =20 : 0번 행의모든 요소 변경
print(d)
print(d[::-1]) : 행의 순서를 거꾸로
print(d[:, ::-1]) : 열의 순서를 거꾸로(','가 행 열 구분해 준다.
print(d[::-1, ::-1]) : 행도 열도 순서를 거꾸로

3차원
x = np.arange(54).reshape(2,9,3) :9행3열짜리 2개인 3차원
x[:1, :2, :] :차원을 유지하면서 가져온다.
x[0, :2, :] : 2차원행렬로 변경된다.(인덱싱이여서 차원이 준다.)

d만들기(0~12)
print(d[:, :, :) : (2,2,3) 모두출력
print(d[:, : -1, :]) : 모든면, 끝에서하나앞행, 모든열

비교하기 -> d[:,::-1,::-1] :모든면, 행,열 거꾸로
print(d[:, 1:, 1:) : 모든면, 1에서 끝행, 1에서 끝열까지
print(d[:, :, -1) : 모든면, 모든행, 끝열들
print(d[:, -1, -1) : 모든면, 끝행들, 끝열들

>> 복습.정리 하기

9. ndarray 데이터형태를 바꿔보기(차원을 바꾸거나 차원의 값을 바꾼다.)

다차원을 일차원으로 펴는 것 ravel함수
>> ravel 은 np에 함수로도 존재하고, ndarray에 함수로도 있다.
x = np.arange(30).reshape(2,3,5) 3차원을 1차원으로
y =np.ravel(x)
print(y)

y = np.ravel(x)
x.ravel()
x값은 보존된다.

>> order 값
np.ravel(x, order = 'C') : 행기준으로
np.ravel(x, order = 'F') : 열기준으로 펴준다.

>> flatten함수 : ravel과 차이점은 복사본을 만들어 처리한다.
flatten() 넘파이는 1차원으로 keras는 2차원으로 만든다.

y = x.flatten() : ravel 과 차이점이 없어보인다.

>> 차이점 비교
x_r[0] =100
print(x_r)
x_r[0] =100
print(x_r)
print(x) : x의 값이 변경된 것을 볼수 있다.
** 메모리 공유해서 사용

x_f = x.flatten()
print(x_f)
x_f[0] = 100
print(x_f)
print(x) :y의 값은 그대로 유지한다. 즉 x_f에 복사해서 사용한다. 메모리 분리

# argsort() : sort 된 후 해당값의 인덱스(원본기준)를 반환
a = np.array([9,2,8,3,1,4,5,6])
print(a, a.shape)

```
sort_i = np.argsort(a)    //오름차순
print(type(sort_i))
print(sort_i)

for i in range(0,len(a)):
    print(a[sort_i[i]])  #인덱스로 a에서 빼온다

sort_i2 = np.argsort(-1*a)  :내림차순
print(sort_i2)

for i in range(0,len(a)):
    print(a[sort_i2[i]])
```

```
# transpose, dtype,  ndim, flat, T
>> transpose() :전치행렬, 행과 열을 서로 맞바꾸는 함수
d = np.arange(12).reshape(3,4)
print(d, d.shape)
t = d.transpose()
print(t, t.shape)
print(d.T) : transpose()와 동일

임시로 1차원으로 변경하여 데이터 사용후 다시 원래의 차원으로 돌아간다.
d.flat = 0
print(d, d.shape)


d = np.arange(12).reshape(3,4)
print(d, d.shape)
print(d.dtype)
print(d.ndim) 차원을 알려준다.
```

>> 복습.정리 하기

10. numpy의 여러 가지 연산 함수
add, substract, multiply, divide

```
x = np.arange(15).reshape(3,5)
y = (np.arange(15)*2).reshape(3,5)
print(x)
print(y)

np.add(x,y)  : shape가 같은 두 행렬을 합한다.
np.subtract(x,y)
```

np.multiply(x,y)
np.divide(x,y) //nan(Not a Number) 연산이 불가능할때 NaN값 반환

x = np.arange(15).reshape(3,5)
y = np.random.rand(16).reshape(4,4)
print(x)
print(y)

np.add(x,y) : shape이 다른 두 행렬은 에러가 난다. (뒤에서 다시 가능한 경우 보자)

>> x, y 중 하나를 수정하여 연산이 가능하도록 해 보시오.

>> 연산자(+ - * /)를 바로사용가능하다.
>> mean, sum var, median, std, cumsum

np.mean(y) or y.mean() : 평균. 모든함수가 두가지 모두 가능한 것은 아닌다.웬만한건 np안에 있다.
np.max(y)
np.argmax(y) : y에서 가장 큰 값의 위치
np.var(y), np.median(y), np.std(y) : 분산, 중간값, 표준편차

np.sum(y) <-> sum(y)와 비교해보기
>>옵션 axis (축을 의미, axis를 따라서 연산한다) 액시스
np.sum(y, axis=None) : 전체합, None,
np.sum(y, axis=0) :각 열의 합(아래로 누른다고생각)
np.sum(y, axis=1) : 각 행의 합(오른쪽으로 민다생각)
np.cumsum(y) :누적합계(그래프그릴 때 유용)

>>3차원
z = np.arange(36).reshape(3,4,3) : 3차원
print(z)
np.sum(z)
np.sum(z, axis=0) : 3개의 상자를 포갯다고 생각
np.sum(z, axis=1) : 위에서 아래로 누른다 생각
np.sum(z, axis=2) : 오른쪽으로 민다 생각 (axis=-1도 가능)

>> any, all함수
z = np.random.rand(10) //정규분포
print(z)
np.any(z > 0) : 원소 중 조건이 하라나도 맞으면 true.
z > 0 로 확인해보기
np.all(z > 0) : 모두 참일 경우 참.

where함수
np.where(조건, 참값, 거짓값)

np.where(z > 0.5, z, 0) : 거짓은 0으로 나머지는 그대로.

//제외하기
np.sum(z, axis =(0,1)) : asix 에 튜플을 줘서 열만남게 되다.
np.sum(z, axis =(0,2)) : 행만남게된다.
//

>> 복습.정리 하기

12. Boolean indexing

x = np.random.randint(1, 100, size =10)
boolean array를 만들어
x % 2 == 0 : x의 모든 원소에대하여 boolean 값 얻어온다.
even_mask = x % 2 == 0
print(even_mask)
x[even_mask] : true인 값만 필터링 가능하다.
x[x % 2 == 0]
x[x > 30]

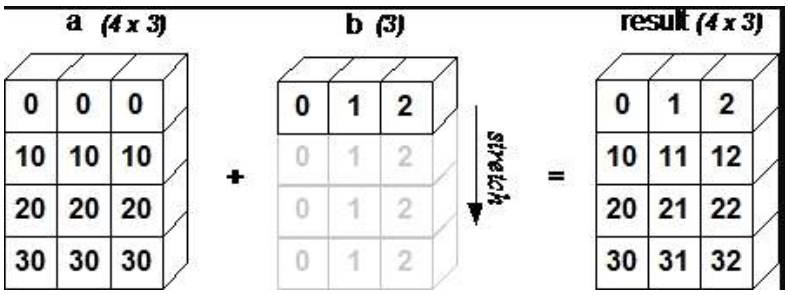
x[(x % 2 == 0) & (x < 30)] : 다중조건
x[(x < 30) | (x > 50)] : 다중조건

문제1) 20도에서 39도까지 31개의 랜덤수 배열 만들기
temp = np.random.randint(20,40, size=31)
len(temp) //길이확인
temp //내용확인

문제2) 25도이상의 날의 개수구하기
len(temp[temp>=25])

문제3) 25도이상의 평균구하기
temp[temp>=25]
np.mean(temp[temp>=25])
round(np.mean(temp[temp>=25]),1)

13. broadcasting : shape이다른 것도 연산이 가능하다. 뒤에서부터 비교한다. 차원 중 값이 1인 것이 존재하면 가능. 차원을 뒤에서부터 비교하여 shape이 같거나 차원중에 값이 1인 것이 존재하면 연산가능.



b(3) = b(1 x 3)

▶ shape이 같은경우

```
x = np.arange(15).reshape(3,5)
y = np.random.rand(15).reshape(3,5)
print(x)
print(y)
```

x+y :각 인덱스가 매칭되는 값들과 더해준다.

▶ shape이 다른 경우(상수)

```
x = np.arange(15).reshape(3,5)
```

x + 2 : 모든원소에 2가 더해진다. (*, **, %2==0 바꿔보기)

▶ shape이 다른 경우

ex1)

```
a = np.arange(12).reshape(4,3)
b = np.arange(100,103)
print(a, a.shape)
print(b, b.shape)
print(a+b)
print(a-b)
print(a*b)
print(a/b)
```

그림그려 설명하기 // b가 1차원이고 shape의 끝이 동일하다. 연산이 잘 된다.

ex2)

```
a = np.arange(12).reshape(4,3)
```

b = np.arange(1000,1004) : 크기가 4인 벡터

```
print(a, a.shape)
print(b, b.shape)
print(a+b)
print(a-b)
print(a*b)
print(a/b)
```

b가 1차원이지만 shape의 끝이 다르다. 계산이 되지 않는다.

ex3)

```
a = np.arange(12).reshape(4,3)
b = np.arange(100,103)
```

d = b.reshape(1,3) : 1행 3열의 2차원 행렬로 변경했다.
a+d : 연산가능 (뒤에서부터 비교해도 맞고, 행렬의 하나는 1이면 가능)
즉, 행열이 같으면 문제없다. 다를 경우, 1차원이거나 맨 뒤가 일치하는 1행인 행열은 가능하다.

>> 행령의 내적의 곱_ dot
>>같은 차원

```
a = np.array([1,2,3])
b = np.array([1,2,3]) +1
print(a)
print(b)
print(a*b)
dot_ab = np.dot(a, b)  #내적곱 (sumproduct) 각 항목을 곱해서 모두 더한값
print(dot_ab)
print(a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
```

결과)

```
[1 2 3]
[2 3 4]
[ 2  6 12]
20
20
```

>>다른차원

```
a = np.arange(1,7).reshape(2,3)
b = np.arange(7,13).reshape(3,2)
print(a)
print(b)
#print(a*b)  #shape이 달라 계산 안됨
dot_ab = np.dot(a,b)
print(dot_ab)
# (2,3) * (3,2) = (2,2)
# (m,n) * ( i,j) = (m,j)
```

결과 >>

```
[[1 2 3]
 [4 5 6]]
[[ 7  8]
 [ 9 10]
 [11 12]]
[[ 58  64]
 [139 154]]
```

58 = 1*7+2*9+3*11
64 = 1*8+2*10+3*12
139 = 4*7+5*9+6*11
154 = 4*8+5*10+6*12

>> 복습.정리 하기
>> ppt나머지 다하기

수업에서 제외>>
14. linalg 서브모듈을 이용하여 선형대수연산
▶ 역행렬은 np.linalg.inv 이용: 모든차원의 값이 같아야 함
  x = np.random.rand(3,3) : 3,3 정방행렬만 가능하다.
  print(x)
  np.linalg.inv(x)
  x @ np.linalg.inv(x) : 행렬의 곱은 @, 대각선은 1, 나머지는 0에 가까움
  or Anp.matmul(x, np.linalg.inv(x))

▶ np.linalg.solve

- Ax = B 형태의 선형대수식 솔루션을 제공
- 예제) 호랑이와 홍합의 합 : 25 호랑이 다리와 홍합 다리의 합은 64
  - x + y = 25
  - 2x + 4y = 64,

$$\begin{pmatrix} 1 & 1 \\ 2 & 4 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 25 \\ 64 \end{pmatrix}$$

```
  A = np.array([ [1,1], [2,4]])
  B = np.array( [25,64])
  x = np.linalg.solve(A,B)
  print(x)

  검산
  np.allclose(A@x, B)
//

*** 이미지 그래프로 표현하기 예제 넣기

import scipy.misc
import matplotlib.pyplot as plt
face = scipy.misc.face()
print(face.shape)
f1= face.copy()
f2= face.copy()
f3= face.copy()
f4= face.copy()
f1[:,:,0] = 0 # red ->0
f2[:,:,1] = 0 # green -> 0
f3[:,:,2] = 0
plt.subplot(221)
plt.imshow(f1)
plt.subplot(222)
plt.imshow(f2)
plt.subplot(223)
plt.imshow(f3)
```

```
plt.subplot(224)
plt.imshow(face)
plt.show()

xmax = face.shape[0]
print(xmax)
ymax = face.shape[1]
print(ymax)
zmax = face.shape[2]
print(zmax)
for i in range(xmax):  #768
    for j in range(ymax):  #1024
        for k in range(zmax): #3
            if face[i][j][k]<100:
                f4[i][j][k] = 0
```