**Q1. 다음 코드의 실행결과의 형태를 예측하시오.**

***NL.eig(a1_1)***
>> a1_1 행렬의 eigen value 와 normalized eigen vector 를 리턴.

***NL.eig(a1_2)***
>> error! non-square matrix 는 eigen value 를 구할 수 없다.

***NL.svd(a1_1, full_matrices=True)***
>> a1_1 을 svd 하여 u, s, v 로 분해하여 리턴.
u.shape=(2,2) / s.shape=(2,) / v.shape=(2,2)

***NL.svd(a1_1, full_matrices=False)***
>> svd 하려는 행렬이 정방행렬일 경우 full_matrices 여부와 관련없이 똑같은 값 리턴.

***NL.svd(a1_2, full_matrices=True)***
>> a1_2 를 svd 함. u.shape=(m,m)=(3,3) / s.shape=(n,)=(2,) / v.shape=(n,n)=(2,2)

***NL.svd(a1_2, full_matrices=False)***
>> a1_2 가 정방행렬이 아니므로 mxn 행렬 A 에서 구한 k=min(m,n) 에 의해
u.shape=(m,k)=(3,2) / s.shape=(2,) / v.shape=(k,n)=(2,2)

---

**Q2. 주어진 코드와 실행 결과가 다음과 같을 때, 결과 값 x, y 의 의미와 그 관계에 대해 설명하시오.**

***a2 = np.array([[7,1],[1,7]])***
***x, y = NL.eig(a2)***
>> x 는 eigenvalues, y 는 eigenvectors.
정의에 의해서 a2 * v = ƛ * v 이므로, 각각의 eigenvalue 와 eigenvector 에 대하여...
np.matmul(a2, y[:0]) == x[0]*y[:,0]
np.matmul(a2, y[:1]) == x[1]*y[:,1]

**Q3. 주어진 코드의 실행 결과인 U, S, V 의 형태를 예측하고 , 각 결과 값이 의미하는 바를 쓰시오.**

***a3 =***
***np.array([[1,1,1,0,0],[3,3,3,0,0],[4,4,4,0,0],[5,5,5,0,0],[0,2,0,4,4],[0,0,0,5,5],[0,1,0,2,2]])***
***U,S,V = NL.svd(a3, full_matrices=False)***

>> a3.shape=(m,n)=(7,5) 이고, full_matrices=False 이므로 Reduced SVD 하면..
U.shape=(m, min(m,n))=(7,5)
S.shape=(n,)=(5,)
V.shape=(min(m,n), n)=(5,5)

>> U 는 orthogonal matrix, 각 column 은 $AA^T$ 의 eigenvectors.
S 는 diagonal matrix Σ 의 0이 아닌 대각성분으로 1차 행렬이고, 각 요소는 eigenvalue.
V 는 orthogonal matrix, 각 column 은 $A^TA$ 의 eigenvectors.

---

**Q4. 주어진 1 차원 ndarray 를 A 와 같이 다차원 형태로 변환하는 코드를 적고 , 변환된 값을 활용하여 B 와 같이 출력하는 코드를 적어라.**

***a4 = np.array([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18])***

```
# answer A
a4_rs = a4.reshape(3,2,3)

# answer B
for dim1 in a4_rs.reshape(6,-1):
    print(dim1)
    print('----')
```

**Q5. 아래 코드의 실행 결과를 적고 , 두 함수의 차이점을 기술하시오.**

***a5 = np.array([[1,2],[3,4]])***
***b5 = np.array([[1,3],[2,4]])***
***np.dot(a5,b5)***
>> array([[5,11],
[11,25]])
행렬 곱 연산
***a5*b5***
>> array([[1,6],
[6,16]])
요소 곱 연산

---

**Q6. 다음 코드의 실행 결과를 보이시오.**

***a6_1 = np.zeros((5,5))***
***a6_1 += np.arange(5)***
***a6_1***
>> array([[0., 1., 2., 3., 4.],
[0., 1., 2., 3., 4.],
[0., 1., 2., 3., 4.],
[0., 1., 2., 3., 4.],
[0., 1., 2., 3., 4.]])
(5x5) matrix 에 (5,) 를 더하기 위해 broadcasting 하여 (5,) -> (5,5) 로 변환하고 덧셈.

***a6_2 = np.full((4,5,3),3)***
***np.cov(a6_2)***
***a6_2***
>> error!
np.cov 는 1-D 혹은 2-D array 를 받는데, broadcasting 으로 array 차원 축소는 불가능.