Chapter 04.
Numpy

목차

Numpy 기본문법

- numpy
  - ✓ Numerical Python의 줄임말로 파이썬에서 과학적 계산을 위한 핵심 라이브러리
  - ✓ 고성능 다차원 배열 객체와 이들 배열과 함께 작동하는 도구 제공
- 배열(ndarray)
  - ✓ List, tuple 보다 생성 방법이 다양
  - ✓ numpy의 배열은 모두 동일한 자료형 grid
  - ✓ 생성: numpy.array(컬렉션객체)
  - ✓ shape 속성은 각 차원의 크기를 알려주는 정수 튜플
  - ✓ 각 데이터의 자료형은 dtype으로 확인 가능
  - ✓ 배열객체[인덱스]를 이용해서 각각의 데이터 접근 가능
  - ✓ Print 함수를 이용해서 출력하면 데이터 각각을 순서대로 출력
  - ✓ 배열의 크기 변경은 numpy.ndarray.reshape 함수로 가능한데 파라미터는 튜플

❖ import

```
import numpy # 넘파이의 모든 객체를 numpy.obj 형식으로 불러 사용할 수 있다.
import numpy as np  # 넘파이의 모든 객체를 np.obj 형식으로 불러 사용할 수 있다.
from numpy import * # 넘파이의 모든 객체를 내장 함수 (객체) 처럼 사용 가능.
```

```
import numpy as np
import datetime
li = range(1,10000)
s = datetime.datetime.now()
print("리스트 작업 시작 시간:", s)
for i in li:
    i = i * 10
s = datetime.datetime.now()
print("리스트 작업 종료 시간:", s)
ar = np.arange(1,10000)
s = datetime.datetime.now()
print("ndarray 작업 시작 시간:", s)
ar = ar * 10
s = datetime.datetime.now()
print("ndarray 작업 종료 시간:", s)
```

```
리스트 작업 시작 시간: 2017-04-12 07:53:34.114300
리스트 작업 종료 시간: 2017-04-12 07:53:34.116300
ndarray 작업 시작 시간: 2017-04-12 07:53:34.116300
ndarray 작업 종료 시간: 2017-04-12 07:53:34.116300
```

```
import numpy as np
ar = np.array([1, 2, 3])
print(type(ar))
print (ar.shape)
print (ar[0], ar[1], ar[2])
ar[0] = 5
print (ar)
br = np.array([[1,2,3],[4,5,6]])
print (br.shape)
print (br[0, 0], br[0, 1], br[1, 0])
```

```
<class 'numpy.ndarray'>
(3,)
1 2 3
[5 2 3]
(2, 3)
1 2 4
```

- 배열 생성
  - ✓ 배열을 입력하는 가장 기본적인 함수는 array()인데 첫 번째 인자로 리스트를 받는데 이것으로 array객체를 생성
  - ✓ 파이썬의 내장 함수인 range()와 유사하게 배열 객체를 반환해주는 arange()
  - ✓ 시작점과 끝점을 균일 간격으로 나눈 점들을 생성해주는 linspace()
  - ✓ 생성된 배열은 reshape() 함수를 이용하여 행수와 열수를 조절 가능

```
import numpy as np
ar = np.array([1, 2, 3])  # 1차 배열 (벡터) 생성
print(ar)
ar = np.array([[1, 2, 3], [4, 5, 6]]) # 2x3 크기의 2차원 배열 (행렬) 생성
print(ar);
ar = np.arange(10)  # 1차원 배얼 [0,1,2, ... ,9] 생성
print(ar);
ar = np.linspace(0, 1, 6)  # start, end, num-points
print(ar);
ar = np.linspace(0, 1, 5, endpoint=False)
print(ar);
ar = np.arange(10)  # 1차원 배열 [0,1,2, ... ,9] 생성
print(ar);
ar = ar.reshape(2, 5)  # 같은 요소를 가지고 2x5 배열로 변형
print(ar);
```

```
[1 2 3]
[[1 2 3]
 [4 5 6]]
[0 1 2 3 4 5 6 7 8 9]
[ 0.   0.2  0.4  0.6  0.8  1. ]
[ 0.   0.2  0.4  0.6  0.8]
[0 1 2 3 4 5 6 7 8 9]
[[0 1 2 3 4]
 [5 6 7 8 9]]
```

- 특수 행렬 생성
  - ✓ zeros(), ones() 라는 함수들은 각각 0과 1로 채원진 배열을 생성하는데 차수를 정수 혹은 튜플로 받음
  - ✓ 대각행렬을 생성하기 위한 eye() 함수와 diag() 함수도 존재
  - ✓ eye() 함수는 항등행렬을 생성하고 diag()는 주어진 정방행렬에서 대각요소만 뽑아내서 벡터를 만들거나 반대로 벡터요소를 대각요소로 하는 정방행렬 생성
  - ✓정의 : eye(N, M=, k=, dtype=), M은 열 수, k는 대각 위치(대각일 때가 0), dtype은 데이터형
  - ✓ empty() 라는 함수는 크기만 지정해 두고 각각의 요소는 초기화 시키지 않고 배열을 생성하는데 이 함수로 생성된 배열의 요소는 가비지 값이 채워진다.

```
import numpy as np
b1 = np.ones( 10 ) # 1로 채원진 10 크기의 (1차원) 배열 생성
print(b1)
b2 = np.zeros( (5,5) ) # 0으로 채원진 5x5 크기의 배열 생성
print(b2)
ar = np.eye(2, dtype=int)
print(ar)
ar = np.eye(3, k=1)
print(ar)
ar = np.arange(9).reshape((3,3))
print(ar)
br = np.diag(ar)
print(br)
cr = np.diag(ar,k=1)
print(cr)
dr = np.diag(ar, k=-1)
print(dr)
aar = np.empty( (2,2) ) # 가비지 값으로 채원진 2x2 크기의 배열 생성
print(ar)
```

```
[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
[[ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]]
[[1 0]
 [0 1]]
[[ 0.  1.  0.]
 [ 0.  0.  1.]
 [ 0.  0.  0.]]
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[0 4 8]
[1 5]
[3 7]
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

- **난수 생성**
  - ✓ numpy는 효율적으로 무작위 샘플을 만들 수 있는 numpy.random 모듈을 제공
  - ✓ np.random.normal(size=개수 또는 shape): size를 생략하면 1개의 데이터만 리턴하고 개수를 입력하면 그 개수에 해당하는 데이터를 배열로 리턴하고 shape에 해당하는 배열을 리턴
  - ✓ np.random.seed(seed=시드번호): 시드번호를 가지고 난수를 생성
  - ✓ np.random.binomial(n, p, size): 이항분포로부터의 무작위 추출 함수
    n은 0부터 나올 수 있는 숫자의 범위로 정수 p는 확률이고 size는 개수

```
import numpy as np
print(np.random.normal(size=5)) #5개의 난수 생성 - 랜덤
print()
print(np.random.normal(size=(2, 3))) #2행 3열의 난수 생성
print()
np.random.seed(seed=100) #시드 설정
print(np.random.normal(size=5)) #5개의 난수 생성 - 실행할 때마다 동일한 값
```

```
[-0.24961316 -0.48887513 -0.36998486  0.74862078 -0.27312433]

[[ 1.40273795  0.84421724 -0.13975186]
 [ 0.00408551  0.90391068 -0.88485027]]

[-1.74976547  0.3426804   1.1530358  -0.25243604  0.98132079]
```

- ndarray의 자료형
  - ✓ 객체를 생성할 때 dtype이라는 파라미터를 이용해서 설정 가능하고 dtype이라는 속성을 이용해서 확인 가능 : np.array([xx, xx], dtype=np.Type)
  - ✓ 서로 다른 데이터 타입의 데이터를 생성할 때 대입하면 자동 형 변환을 해서 배열이 생성
  - ✓ 정수와 실수가 혼합되어 있다면 실수, 숫자와 문자열이 혼합되어 있다면 문자열로 생성
  - ✓ astype 함수를 이용해서 데이터의 타입 변경
  - ✓ 자료형의 종류

정수 자료형: int8, 16, 32, 64, uint8, uint16, uint32, uing64

실수 자료형: float16, 32, 64, 128

복소수 자료형: complex64, 128, 256

boolean: bool

객체: object

문자열: string_

유니코드: unicode_

```
import numpy as np
ar = np.array([1,2,3])
print("타입 확인:",ar.dtype);
ar = np.array(['12', '2' ,'3'])
print("타입 확인:",ar.dtype);
ar = np.array([3, 2.9, 4])
print("타입 확인:",ar.dtype);
ar = np.array([3, 2.9, '4'])
print("타입 확인:",ar.dtype);
ar = np.array([3, 2, '4'], dtype=np.int32)
print("타입 확인:",ar.dtype);
ar = ar.astype(np.string_)
print("타입 확인:",ar.dtype);
```

```
타입 확인: int32
타입 확인: <U2
타입 확인: float64
타입 확인: <U32
타입 확인: int32
타입 확인: |S11
```

- ndarray의 산술 연산
  - ✓ 배열과 숫자 데이터와의 연산은 배열의 모든 요소에 숫자 데이터를 연산한 결과를 리턴
  - ✓ 동일한 크기를 갖는 배열간의 산술 연산은 동일한 위치의 데이터끼리 연산을 수행한 후 결과를 리턴
  - ✓ 동일한 크기를 갖는 배열 간의 비교 연산은 동일한 위치의 데이터를 비교해서 True 또는 False로 리턴해서 배열로 만듬 – equal, not_equal, greater, greater_equal, less, less_equal
  - ✓ 동일한 크기를 갖는 배열 간의 할당 연산: Add AND (m += n), Subtract AND (m -= n), Multiply AND (m *= n), Divide AND (m /= n), Floor Division (m //= n), Modulus AND (m %= n), Exponent AND (m **= n)
  - ✓ 배열간의 논리 연산
    - ✓ np.logical_and(a, b) : 두 배열의 원소가 모두 '0' 아니면 True 반환
    - ✓ np.logical_or(a, b) : 두 배열의 원소 중 한개라도 '0' 아니면 True 반환
    - ✓ np.logical_xor(a, b) : 두 배열의 원소가 모두 '1' 아니면 True 반환
  - ✓ 소속 여부 판단 연산 : in, not in
  - ✓ 배열 (혹은 리스트)에 특정 객체가 들어있으면 True를 반환, 안들어 있으면 False를 반환

```
import numpy as np
ar = np.array([1,2,3])
br = np.array([4,5,6])
cr = np.array([[6,7,8], [10,20,30]])
result = ar * 2 #배열의 모든 요소에 2를 곱한 결과
print(result)
result = ar + br; #배열 간의 덧셈: 동일한 위치간의 덧셈을 한 결과
print(result)
```

```
import numpy as np
ar = np.array([1,2,3])
br = np.array([4,2,6])
print(np.equal(ar, br))
print(np.not_equal(ar, br))
print(np.greater(ar, br))
print(np.greater_equal(ar, br))
print(np.less(ar, br))
print(np.less_equal(ar, br))
```

- ndarray의 산술 연산 - broadcasting
  - ✓ 동일하지 않은 크기의 배열끼리의 연산은 브로드캐스팅 연산을 수행
  - ✓ 브로드캐스팅은 기준 축의 데이터 개수가 동일한 경우에만 수행하며 모든 행에 대해서 연산을 수행

```
import numpy as np
a = np.array([10,20,30])
b = np.arange(12).reshape((4, 3))
print(a + b) #b의 각 행에 a의 데이터를 더한 결과
a = np.array([10,20,30]).reshape(3,1)
b = np.arange(12).reshape((3, 4))
print(a + b)
```

- 인덱스
  - ✓ 1차원 배열의 인덱스는 list와 유사
  - ✓ 인덱스에 음수를 사용하면 뒤에서부터 계산
  - ✓ 2차원 이상의 배열에서 인덱스는 [행번호][열번호]의 형태로 입력해도 되지만 [행번호, 열번호]를 이용
  - ✓ 2차원 이상의 배열에서 인덱스를 생략하는 경우에는 생략된 인덱스의 데이터 전체를 의미
  - ✓ 2차원 이상의 배열에서 인덱스를 리스트 형태로 입력하면 리스트에 해당하는 데이터만 리턴
  - ✓ 리스트의 리스트로 대입하는 것도 가능

[[0,1],[2,3]] => [0,2], [1,3]

[[0,1]][[:,[1,2]] => [0,1],[0,2],[1,1],[1,2]

```
import numpy as np
ar = np.empty((10, 5)) #gabage 값을 갖는 10행 5열의 배열을 생성
for i in range(10):
    ar[i] = i   #각 행의 모든 데이터를 i 값으로 채움
#print(ar)
br = ar[[1,3,5,7]] #1,3,5,7 행만 선택
#print(br)
cr = ar[[0,1], [3,4]] #[0,3],[1,4] 만 선택
#print(cr)
dr = ar[[0,1]][:,[3,4]] #0번 행의 3번째와 4번째 1번행의 3번째와 4번째 원소 선택
print(dr)
```

2차원 배열 인덱스(index)


```
# 그림에 해당하는 정보 추출
print a[0,3:5]
print a[4:,4:]
print a[:,2]
print a[2::2, ::2]
```

```
[3 4]
[[44 45]
 [54 55]]
[ 2 12 22 32 42 52]
[[20 22 24]
 [40 42 44]]
```

- 슬라이싱
  - ✓ 슬라이싱을 하면 복제가 안됨
  - ✓ 복제를 하고자 하면 copy 메서드를 호출해서 리턴
  - ✓ [시작위치:끝위치]의 형태로 슬라이싱 가능
  - ✓ 2차원 이상의 배열의 경우는 [행의 슬라이싱, 열의 슬라이싱]의 형태로 슬라이싱 가능

```
import numpy as np
ar = np.array([1,2,3,4,5])
br = ar[0:3]
cr = ar[0:3].copy()
ar[0]=10
print(br)
print(cr)
br = ar[-2:]
print(br)
```

```
[10  2  3]
[1 2 3]
[4 5]
```

- 슬라이싱
  - ✓ 특정 조건을 만족하는 배열의 모든 열을 선별하기 : ==
  - ✓ 특정 조건을 만족하지 않는 배열의 모든 열을 선별하기 : !=, ~(==)
  - ✓ 여러 조건을 가지고 작업하기: & (and), | (or)
  - ✓ 조건에 맞는 데이터를 선택한 후 = 을 이용해서 특정한 값을 할당

```
import numpy as np
ar = np.arange(20).reshape(5, 4)
br = np.array(['A', "B", 'C', 'A', 'C'])
print(br == 'A') #데이터가 A인 경우는 True 그렇지 않으면 False
print(ar[br == 'A']) #True 인 행만 반환
print(ar[br == 'A', 2]) #열을 2번째만 반환
print(ar[br == 'A', 0:2]) #열을 0부터 2앞까지 반환
ar[br == 'A'] = 100
print(ar)a
```

```
[ True False False  True False]
[[ 0  1  2  3]
 [12 13 14 15]]
[ 2 14]
[[ 0  1]
 [12 13]]
[[100 100 100 100]
 [  4   5   6   7]
 [  8   9  10  11]
 [100 100 100 100]
 [ 16  17  18  19]]
```

- 슬라이싱
  - ✓ 특정 조건을 만족하는 배열의 모든 열을 선별하기 : ==
  - ✓ 특정 조건을 만족하지 않는 배열의 모든 열을 선별하기 : !=, ~(==)
  - ✓ 여러 조건을 가지고 작업하기: & (and), | (or)
  - ✓ 조건에 맞는 데이터를 선택한 후 = 을 이용해서 특정한 값 할당
- Fancy Indexing
  - ✓ 정수 배열을 indexer로 사용해서 다차원 배열로 부터 Indexing
  - ✓ Fancy Indexing은 copy를 만듬
  - ✓ 특정 순서로 다차원 배열의 행(row)과 열(column)을 Fancy Indexing
    - 특정 순서로 행(row)을 fancy indexing 합니다. 그런 후에 전체 행을 ':'로 선택하고, 특정 칼럼을 순서대로 배열을 사용해서 indexing을 한번 더 해주는 것
    - np.ix_ 함수를 사용해서 배열1 로 특정 행(row)을 지정, 배열2 로 특정 열(column)을 지정해주는 것

```
import numpy as np
ar = np.arange(20).reshape(5, 4)
print(ar)
print(ar[[1,2]])#1행과 2행만 선택하기
print(ar[[-1, -2]])# 뒤에서 2개행 선택하기
print(ar[[0, 2, 4]][:, [0, 2]])
print(ar[np.ix_([0, 2, 4], [0, 2])])
```

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

- 범용함수(Universal function)
  - ✓ python에서 제공하는 math module은 실수(real number)에 대해서만 범용함수를 지원하며, cmath module 은 복소수(complex number) 까지 범용함수를 지원
  - ✓ numpy module은 실수, 복소수, 복소수 행렬 (complex matrix)의 원소 간 범용 함수를 모두 지원하므로 사용 범위가 가장 넓어 매우 유용
  - ✓ 배열의 원소간 연산을 위해 numpy의 Universal function 함수는 매우 유용
  - ✓ numpy 범용 함수는 몇 개의 배열에 대해 적용 여부에 따라 분류
    - 1개의 배열에 적용하는 Unary Universal Functions (ufuncs)
    - 2개의 배열에 대해 적용하는 Binary Universal Functions (ufuncs)

- 기본 통계 함수
  - ✓ sum: 배열에 있는 모든 원소의 합을 계산해서 리턴
  - ✓ mean, median, std, var: 평균값, 중간값, 표준편차, 분산

```
ar = np.array([1, 2, 4, 5, 5, 7, 9, 10, 13, 18, 21])
print(np.sum(ar))
print(np.mean(ar))
print(np.median(ar))
print(np.std(ar))
print(np.var(ar))
```

```
95
8.63636363636
7.0
6.1388883695
37.6859504132
```

- 소수 관련 함수
  - ✓ np.around(a) : 0.5를 기준으로 올림 혹은 내림
  - ✓ np.round_(a, N) : N 소수점 자릿수까지 반올림
  - ✓ np.rint(a) : 가장 가까운 정수로 올림 혹은 내림
  - ✓ np.fix(a) : '0' 방향으로 가장 가까운 정수로 올림 혹은 내림
  - ✓ np.ceil(a) : 각 원소 값보다 크거나 같은 가장 작은 정수 값 (천장 값)으로 올림
  - ✓ np.floor(a) : 각 원소 값보다 작거나 같은 가장 큰 정수 값 (바닥 값)으로 내림
  - ✓ np.trunc(a) : 각 원소의 소수점 부분은 잘라버리고 정수 값만 남김

```
import numpy as np
ar = np.array([-4.62, -2.19, 0, 1.57, 3.40, 4.06])
print(np.around(ar))
print(np.round_(ar, 1))
print(np.rint(ar))
print(np.fix(ar))
print(np.ceil(ar))
print(np.floor(ar))
print(np.trunc(ar))
```

```
[-5. -2.  0.  2.  3.  4.]
[-4.6 -2.2  0.   1.6  3.4  4.1]
[-5. -2.  0.  2.  3.  4.]
[-4. -2.  0.  1.  3.  4.]
[-4. -2.  0.  2.  4.  5.]
[-5. -3.  0.  1.  3.  4.]
[-4. -2.  0.  1.  3.  4.]
```

- 배열 통계
  - ✓ np.prod(): 2차원 배열의 경우 axis=0 이면 같은 열(column)의 위*아래 방향으로 배열 원소 간 곱하며, axis=1 이면 같은 행(row)의 왼쪽*오른쪽 원소 간 곱
  - ✓ np.sum(): keepdims=True 옵션을 설정하면 1 차원 배열로 배열 원소 간 합을 반환
  - ✓ np.nanprod(): NaN (Not a Numbers) 을 '1'(one)로 간주하고 배열 원소 간 곱
  - ✓ np.nansum(): NaN (Not a Numbers)을 '0'(zero)으로 간주하고 배열 원소 간 더하기
  - ✓ np.cumprod(): axis=0 이면 같은 행(column)의 위에서 아래 방향으로 배열 원소들을 누적으로 곱해 나가며, axis=1 이면 같은 열(row)에 있는 배열 원소 간에 왼쪽에서 오른쪽 방향으로 누적으로 곱해 나감
  - ✓ np.cumsum(): axis=0 이면 같은 행(column)의 위에서 아래 방향으로 배열 원소들을 누적으로 합해 나가며, axis=1 이면 같은 열(row)에 있는 배열 원소 간에 왼쪽에서 오른쪽 방향으로 누적으로 합해 나감
  - ✓ np.diff(): 배열 원소 간 n차 차분 구하기

```
import numpy as np
b = np.array([1, 2, 3, 4])
c = np.array([[1, 2], [3, 4]])
print(np.prod(b)) # 1*2*3*4
print(np.prod(c, axis=0))  # [1*3, 2*4]
print(np.prod(c, axis=1))  # [1*2, 3*4]
print(np.sum(b)) # [1+2+3+4]
print(np.sum(b, keepdims=True))
print(np.sum(c, axis=0))  # [1+3, 2+4]
print(np.sum(c, axis=1) ) # [1+2, 3+4]
```

```
24
[3 8]
[ 2 12]
10
[10]
[4 6]
[3 7]
```

```
import numpy as np
b = np.array([1, 2, 3, 4])
c = np.array([[1, 2], [3, 4]])
print(np.cumprod(b))# [1, 1*2, 1*2*3, 1*2*3*4]
print(np.cumprod(c, axis=0))  # [[1, 2], [1*3, 2*4]]
print(np.cumprod(c, axis=1))  # [[1, 1*2], [3, 3*4]]
print(np.cumsum(b))# [1, 1+2, 1+2+3, 1+2+3+4]
print(np.cumsum(c, axis=0))  # [[1, 2], [1+3, 2+4]]
print(np.cumsum(c, axis=1))  # [[1, 1+2], [3, 3+4]]
```

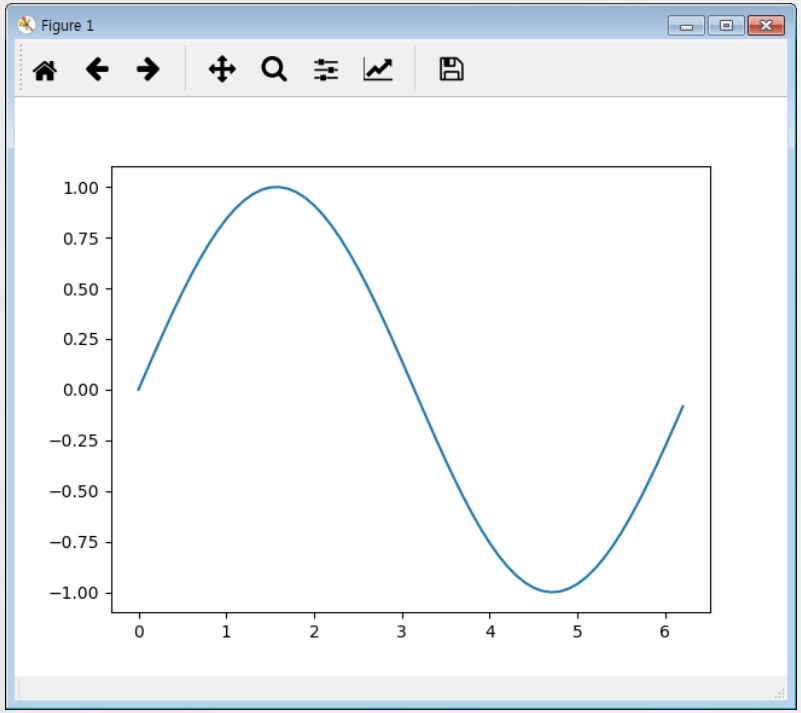
```
[ 1  2  6 24]
[[1 2]
 [3 8]]
[[ 1  2]
 [ 3 12]]
[ 1  3  6 10]
[[1 2]
 [4 6]]
[[1 3]
 [3 7]]
```

```
import numpy as np
g = np.array([1, 2, 4, 10, 13, 20])
print(np.diff(g))  # [2-1, 4-2, 10-4, 13-10, 20-13]
print(np.diff(g, n=2))  # [2-1, 6-2, 3-6, 7-3]
```

```
[1 2 6 3 7]
[ 1  4 -3  4]
```

- 지수 로그 함수
  - ✓ np.exp() 함수는 밑(base)이 자연상수 e 인 지수함수 로 변환
  - ✓ np.log(x), np.log10(x), np.log2(x), log1p(z): 지수함수의 역함수인 로그함수는 밑이 자연상수 e, 혹은 10, 또는 2 이냐에 따라서 np.log(x), np.log10(x), np.log2(x) 를 구분해서 사용
- 삼각 함수
  - ✓ np.sin(), np.cos(), np.tan(): 각도는 라디언을 사용하기 때문에 degree * np.py/180으로 연산을 해서 작업을 수행
  - ✓ np.arcsin(), np.arccos(), np.arctan(): 역삼각함수

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0, 2* np.pi, 0.1)
y = np.sin(x)
plt.plot(x, y)
plt.show()
```



- 숫자 처리 함수
  - ✓ np.abs(x), np.fabs(x): 배열 원소의 절대값 (absolute value)
  - ✓ np.sqrt(y): 배열 원소의 제곱근(Square Root)
  - ✓ np.square(y): 배열 원소의 제곱값 (square value) 범용 함수
  - ✓ np.modf(z): 배열 원소의 정수와 소수점을 구분하여 2개의 배열 반환
  - ✓ np.sign(x): 열 원소의 부호 판별 함수 -> 1 (positive), 0(zero), -1(negative)
- 논리 함수
  - ✓ np.isnan(x): NaN(Not a Number) 포함 여부 확인
  - ✓ np.isfinite(x): 유한수(finite number) 포함 여부 확인
  - ✓ np.isinf(x): 배열에 무한수(infinite number) 포함 여부 확인
  - ✓ np.logical_not(condition): 배열 원소가 조건을 만족하지 않는 경우 참 반환

```
import numpy as np
b = np.array([10, 1, 2, 3, 4])
print(np.logical_not( b <= 2 ))
c = b[np.logical_not( b <= 2 )]
print(c)
```

```
[ True False False  True  True]
[10  3  4]
```

- 이항 함수: 배열 2개를 가지고 작업하는 함수
  - ✓ add, subtract, multiply, divide, floor_divide
  - ✓ power
  - ✓ maximum, fmax, minimum, fmin
  - ✓ mod
  - ✓ copysign
  - ✓ greater, greater_equal, less, less_equal, equal, not_equal
  - ✓ logical_and, logical_or, logical_xor
- where(boolean 배열, True일 때 선택할 데이터, False일 때 선택할 데이터)

```
import numpy as np
ar = np.array([1, 2, 3, 4])
br = np.array([5, 6, 7, 8])
print(ar + br)
cond = [True, False, False, True]
print(np.where(cond, ar, br))
```

```
[ 6  8 10 12]
[1 6 7 4]
```

- 이항 함수: 배열 2개를 가지고 작업하는 함수
  - ✓ numpy.concatenate: 여러 개의 배열을 한 개로 합치는 함수로 X축과 Y축 방향으로 합치는 두 가지 방법이 있으며 axis 라는 파라미터를 통해 제어함.
    - axis = 0: Y축 (세로 방향) 으로 설정
    - axis = 1: X축 (가로 방향) 으로 설정

```
import numpy as np
ar = [1,2,3,4]
br = [5,6,7,8]
cr = np.concatenate((ar, br))
print(cr)
ar = np.array([[1, 2], [3, 4]])
br = np.array([[5, 6], [7, 8]])
cr = np.concatenate((ar, br), axis = 0)
print(cr)
cr = np.concatenate((ar, br), axis = 1)
print(cr)
```

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

- numpy.split
  - ✓ 배열을 여러 개의 크기로 나누어주는 함수
  - ✓ 나누는 방법은 X축, 그리고 Y축을 기준을 나누는 두 가지의 방법이 있으며 numpy.concatenate의 axis와 동일하게 작동
  - ✓ 또한 두 번째 파라미터에 숫자 N을 넣으면 배열을 N개의 동일한 크기의 배열들로 나누고 리스트를 넣는다면 리스트 안의 숫자들 번째 인덱스에서 배열을 나눈다.
- ndarray.sort()
  - ✓ 배열의 데이터를 정렬
  - ✓ numpy.sort()는 정렬한 결과를 리턴
  - ✓ axis 매개변수는 정렬할 축 번호

```
import numpy as np
ar = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13 ,14 ,15, 16]])
print(ar)
slice_Y_equal_size = np.split(ar, 2, axis = 0) #x축 방향으로 2개로 나눔
print(slice_Y_equal_size[0])
print(slice_Y_equal_size[1])
slice_X_different_sizes = np.split(ar, [2, 3], axis = 1) #2번째 앞까지 나누고 3번째 앞까지 나누고 나머지
print(slice_X_different_sizes[0])
print(slice_X_different_sizes[1])
print(slice_X_different_sizes[2])
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
[[1 2 3 4]
 [5 6 7 8]]
[[ 9 10 11 12]]
[[ 1  2]
 [ 5  6]
 [ 9 10]
 [13 14]]
[[ 3]
 [ 7]
 [11]
 [15]]
[[ 4]
 [ 8]
 [12]
 [16]]
```

```
import numpy as np
ar = np.array([90, 40, 30, 78])
ar.sort()
print(ar)
br = np.sort(ar)
print(br)

ar = np.array([[90, 40, 50], [30, 78,27]])
ar.sort(axis=0)
print(ar)
```

```
[30 40 78 90]
[30 40 78 90]
[[30 40 27]
 [90 78 50]]
```

```
import numpy as np
ar = np.array([90, 40, 30, 78])
ar.sort()
print(ar[0:int(0.5 * len(ar))]) #하위 50%
print(ar[int(0.5 * len(ar))*-1:]) #상위 50%
```

```
[30]
[90]
```

- 집합 관련 함수
  - ✓ unique(): 중복을 제거
  - ✓ intersect1d(): 교집합
  - ✓ union1d(): 합집합
  - ✓ in1d(): 데이터의 존재 여부를 boolean 배열로 리턴
  - ✓ setdiff1d(): 차집합
  - ✓ setxor1d(): 한쪽에만 있는 데이터의 집합

```
import numpy as np
ar = np.array([90, 40, 30, 78, 30])
print(np.unique(ar))
br = np.array([30, 45, 76, 90])
print(np.intersect1d(ar, br))
print(np.union1d(ar, br))
print(np.in1d(ar, br))
print(np.setdiff1d(ar, br))
print(np.setxor1d(ar, br))
```

```
[30 40 78 90]
[30 90]
[30 40 45 76
78 90]
[ True False
True False  True]
[40 78]
[40 45 76 78]
```

Numpy 파일 입출력

❖numpy.save(파일경로, 데이터): raw 데이터의 형태로 저장 시 확장자(.npy) 생략 시 자동으로 삽입

❖읽을 때는 load(파일경로): 저장할 객체 그대로 리턴

❖savez('파일명', 키=배열, 키=배열): 디셔너리 형식으로 데이터를 저장하는데 이 때는 확장자가 npz

❖위의 경우는 dict 형식으로 저장되어 있으므로 데이터를 전부 읽은 후 ['키']를 이용해서 읽음

❖loadtxt('파일경로', delimiter='구분자')를 이용해서 파일을 읽어 올 수 있다.

```
import numpy as np
ar = [100, 300, 200]
np.save('ar', ar)
br = np.load('ar.npy')
print(br)
cr = ar+br
np.savez('dic', a=ar, b=br, c=cr)
result = np.load('dic.npz')
print(result)
print(result['c'])
```

```
[100 300 200]
<numpy.lib.npyio.NpzFile object at 0x0000000003400C50>
[200 600 400]
```

```
import numpy as np
ar = np.arange(30)
np.savetxt('test.csv', ar)
br = np.loadtxt('test.csv', delimiter='₩t')
print(br)
```

```
[ 0.  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.]
```

Numpy 모양 변경

- reshape
  - ✓ 1차원 배열을 여러 크기의 2차원 배열로 변환
  - ✓ 2차원 배열을 다른 크기로 변형
- T : 전치행렬(행렬변경)
- transpose : 원하는 축 순서대로 변환
- swapaxes : 축 변경

```
import numpy as np
ar = np.array([[1,2,3], [4,5,6]])
print(ar)
print()
print(ar.T)
```

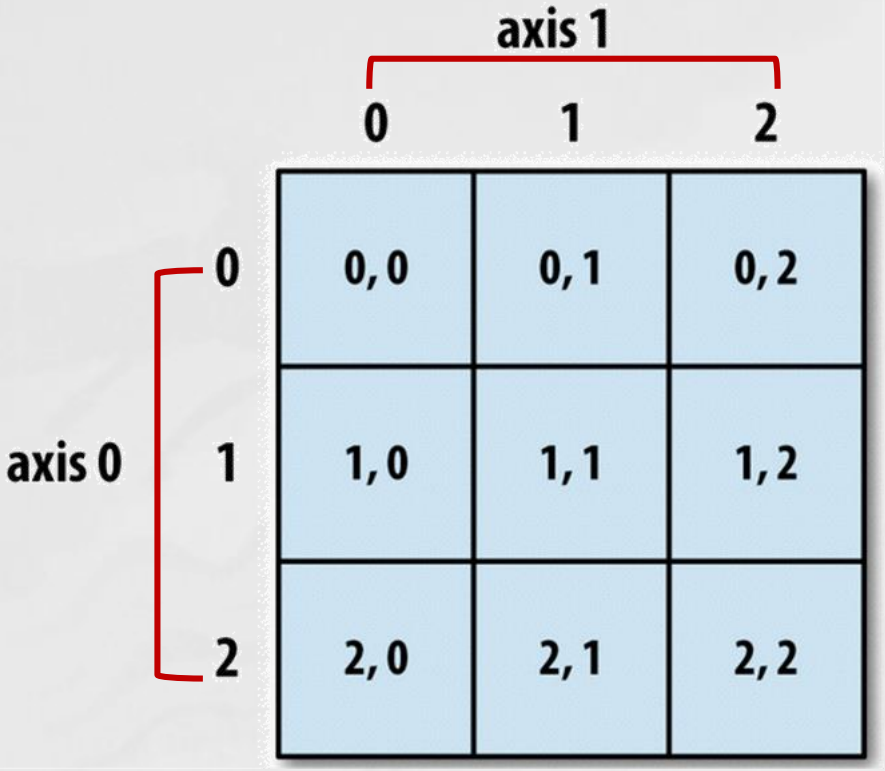
```
[[1 2 3]
 [4 5 6]]

[[1 4]
 [2 5]
 [3 6]]
```

```
import numpy as np
ar = np.array([[[1,2],[3,4],[5,6]],[[71,72],[73,74],[75,76]]])
print()
print(ar.transpose())
print()
print(ar.transpose(2,1,0))
print()
print(ar.transpose(1,0,2))
```

```
[[[ 1 71]
  [ 3 73]
  [ 5 75]]

 [[ 2 72]
  [ 4 74]
  [ 6 76]]]

[[[ 1 71]
  [ 3 73]
  [ 5 75]]

 [[ 2 72]
  [ 4 74]
  [ 6 76]]]

[[[ 1  2]
  [71 72]]

 [[ 3  4]
  [73 74]]

 [[ 5  6]
  [75 76]]]
```

2차원 배열에서 축(axis)

행 축 : 동일한 열의 모음
열 축 : 동일한 행의 모음




```
arr = np.arange(20).reshape(5, 4)
print(arr)

print('축(axis) 연산')
print(arr.sum(axis=0)) # 행축 합계
print(arr.sum(axis=1)) # 열축 합계
```

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

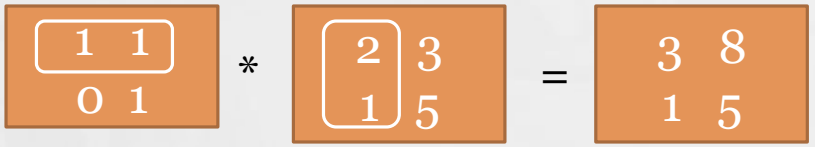
축(axis) 연산
[40 45 50 55]
[6 22 38 54 70]

Numpy 행렬곱(행렬 내적)

❖ 행렬곱

✓ 다차원 배열 구조를 갖는 데이터 끼리 행과 열 단위로 곱과 합 연산에 의해서 또 다른 행렬의 결과를 나타내는 연산

```
array1 = np.array([[1, 1], [0, 1]])
array2 = np.array([[2, 3], [1, 5]])
print("행렬 곱 구하기 : ")
print(array1.dot(array2))
print(np.dot(array1, array2))
'''
[[3 8]
 [1 5]]
'''
```



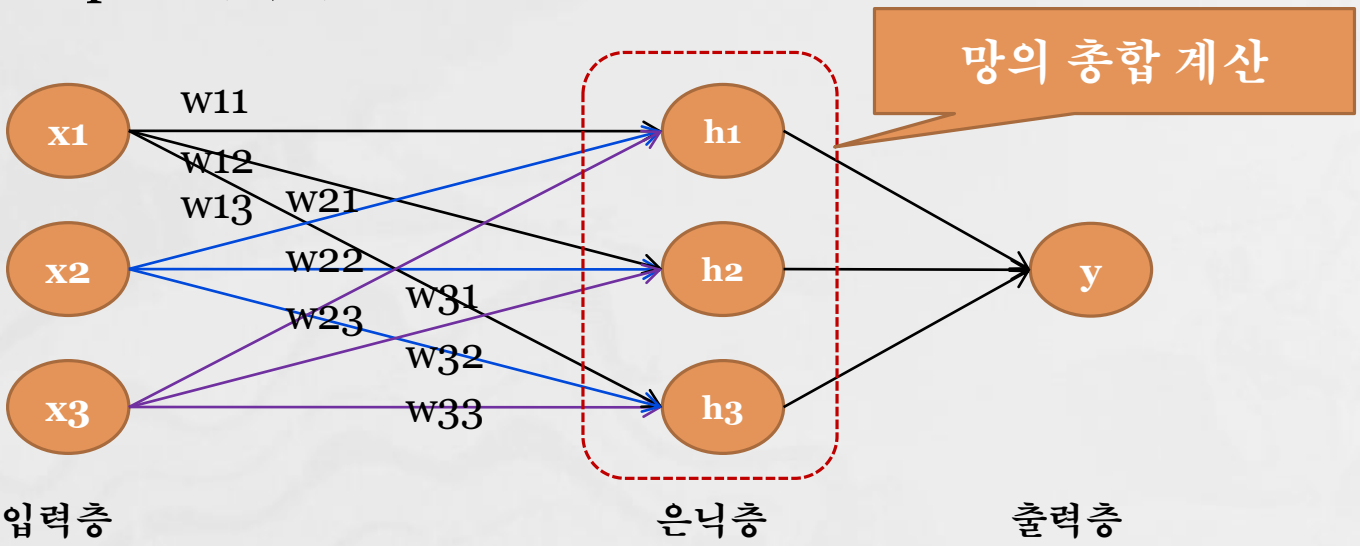
LinearRegress 행렬곱 사용 예

- 회귀방정식

```
X = score_arr[:, 1:]
Y = score_arr[:, 0]
slope = np.array([[0.376966], [2.992800]]) # 기울기
intercept = 25.229141 # 절편

matmul = np.dot(score_X, slope)
fitted_values = matmul + intercept
```

ANN에서 행렬곱 사용 예

- np.dot(w, x)



계산을 위한 위치 변경

w11	w21	w31
w12	w22	w32
w13	w23	w33

3 x 3

*

x1
x2
x3

3 x 1

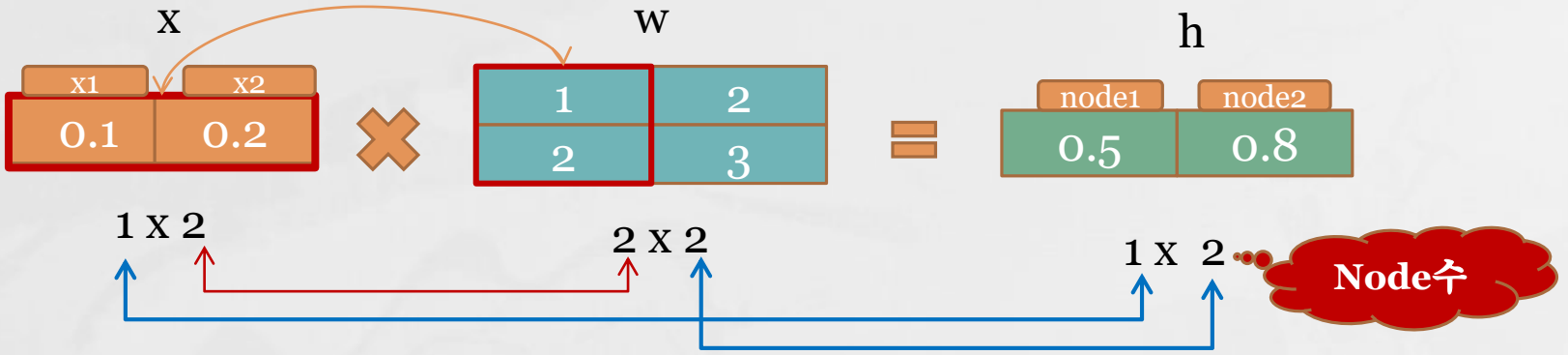
=

h1	n1
h2	n2
h3	n3

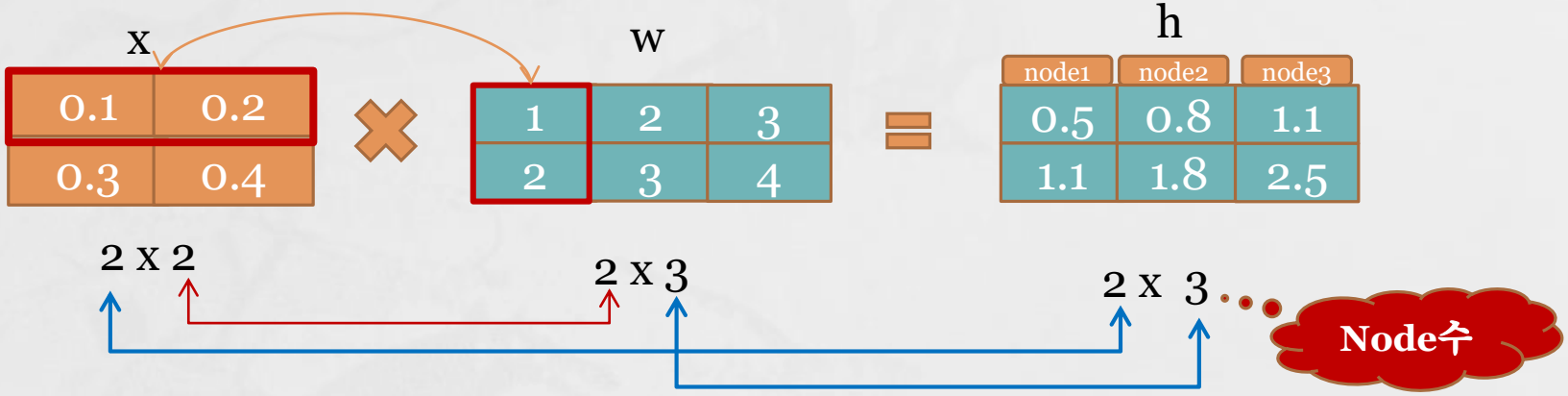
3 x 1

Numpy 행렬 곱(입력 x 가중치)

1) 1개 관측치 : x(1,2) * w(2,2) = h(1,2)

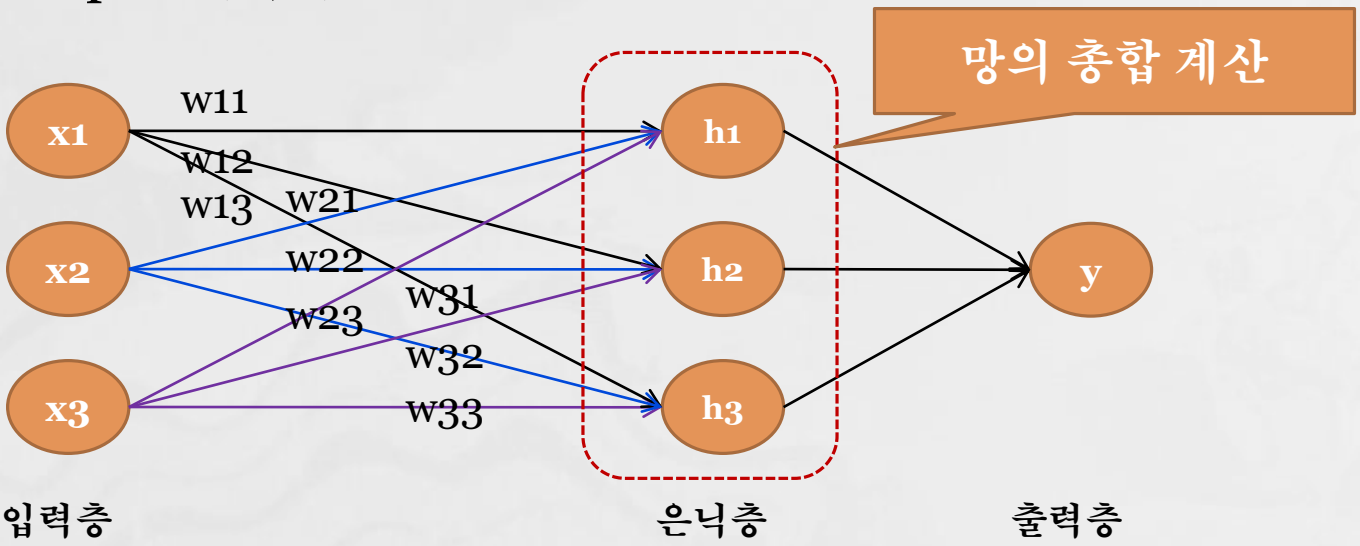


2) 2개 관측치 : x(2,2) * w(2,3) = h(2,3)



ANN에서 행렬곱 문제

- np.dot(w, x)



0~1 사이 난수

w11	w21	w31
w12	w22	w32
w13	w23	w33

W(3 x 3)

*

1
2
3

X(3 x 1)

=

?	node1
?	node1
?	node1

H(3 x 1)