1장. 파이썬 기초

파이썬 문법 부분은 생략, 넘피에서 중요한 부분만 적는다.

정리 2023-05-17 (2013.05.13 부터 노트 필기하던 것)

In [2]:
```
import numpy as np
```

In [6]:
```
# 1차원 배열 (1x3)
x = np.array([1.0, 2.0, 3.0])
y = np.array([2.0, 4.0, 6.0])

# 2 차원 배열 (3x3)
X = np.array([
    [1,2,3],
    [4,5,6],
    [7,8,9],
])
Y = np.array([
    [1,2,3],
    [4,5,6],
    [7,8,9],
])
```

행렬의 사칙 연산

In [8]:
```
print(x+y)   # 각 요소를 더한 행렬
print(x-y)   # 각 요소를 뺀 행렬
print(x*y)   # 각 요소끼리 곱한 행렬
print(x/y)   # 각 요소끼리 나눈 행렬

print(X+Y)
print(X-Y)
```

```
print(X*Y)
print(X/Y)
```

```
[3. 6. 9.]
[-1. -2. -3.]
[ 2.  8. 18.]
[0.5 0.5 0.5]
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
[[0 0 0]
 [0 0 0]
 [0 0 0]]
[[ 1  4  9]
 [16 25 36]
 [49 64 81]]
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

브로드캐스트

행렬의 각 요소에 연산을 한다

In [12]:
```
z = np.array([2, 4, 6])

# 요렇게 하면
print(z/2)

# 요렇게 만들어준다
print(z/np.array([2,2,2]))
```

```
[1. 2. 3.]
[1. 2. 3.]
```

In [13]:
```
# 배열도 브로드캐스팅이 된다.
A = np.array([
    [1,2],
    [3,4]
])
```

```
B = np.array(
    [1,2]
)
print(A*B)
```

```
[[1 4]
 [3 8]]
```

1.5.5 브로드캐스트

넘파이에서는 형상이 다른 배열끼리도 계산할 수 있습니다. 앞의 예에서는 2×2 행렬 A에 스칼라값 10을 곱했습니다. 이때 [그림 1-1]과 같이 10이라는 스칼라값이 2×2 행렬로 확대된 후 연산이 이뤄집니다. 이 똑똑한 기능을 **브로드캐스트**broadcast라고 합니다.

그림 1-1 브로드캐스트의 예 : 스칼라값인 10이 2×2 행렬로 확대된다.

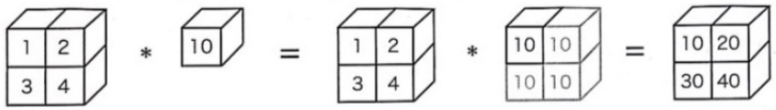
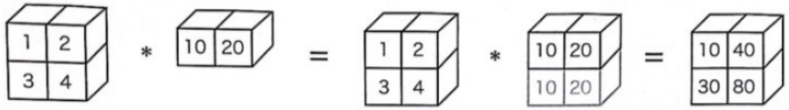


그림 1-2 브로드캐스트의 예 2



이처럼 넘파이가 제공하는 브로드캐스트 기능 덕분에 형상이 다른 배열끼리의 연산을 스마트하게 할 수 있습니다.

원소 접근

In [34]:
```
print('-- A --')
print(A)
print('-------\n')
```

```
print(A[0],   '\t<--- A[0]   첫 행')              # 0부터 시작이니까
print(A[0,1], '\t<--- A[0,1] 첫 행, 두번째 요소')
```

```
-- A --
[[1 2]
 [3 4]]
-------

[1 2]   <--- A[0]   첫 행
2       <--- A[0,1] 첫 행, 두번째 요소
```

In [35]:
```
# 한줄로 만들기
C = A.flatten()
print(C)
```

```
[1 2 3 4]
```

In [36]:
```
print(C[np.array([0,2])])                          # 인자에 np.array를 주면, 여러 원소에 한꺼번에 접근 가능
print(C[np.array([True, True, False, False])])  # T,F의 배열을 줄 수 도있다.
```

```
[1 3]
[1 2]
```

In [38]:
```
# 조건식으로 접근 (가져오기)
print(C>2)              # 연산 결과로 행렬 만들기
print(C[C>2])           # 그걸로 필터링 하기
```

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

In [ ]:
```

```