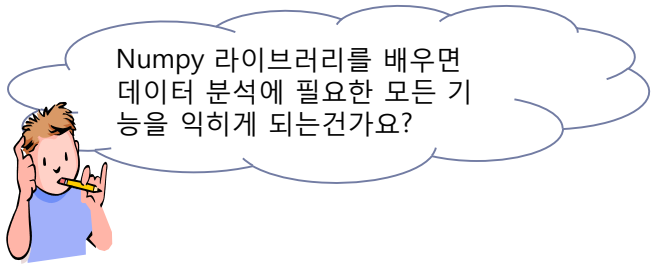
# 4. numpy 라이브러리

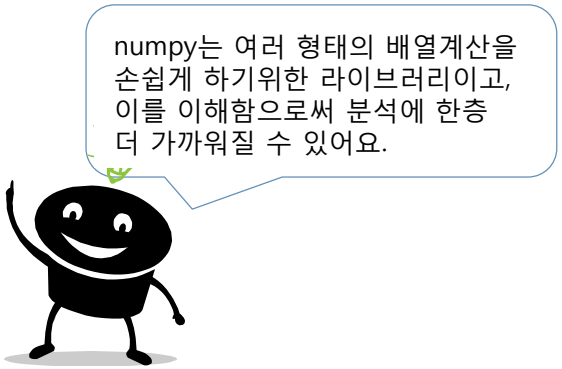

이번시간에 학습할 내용은..

배열구조 데이터의 유연한 변환 및 빠르고 정확한 연산을 제공하는 numpy에 대해 합습

# 1. Numpy 란?

- 고차원 과학계산 및 데이터 분석을 위한 기본 패키지
- 다차원 배열 객체와 이를 다루기위한 다양한 도구를 제공




++ 참고 numpy를 배우기 위해서는 기본 자료구조형태를 먼저 알고있어야 해요

✓ 리스트 - 배열과 달리 크기 변경이 가능하고 서로 다른 자료형으로도 데이터 구성이 됨

```
In [3]: a_data=[23,32,37]
        b_data=('길동','길순','길자')
        a_data.append(b_data)
        a_data
Out[3]: [23, 32, 37, ('길동', '길순', '길자')]

In [4]: new_data=list(b_data)
        new_data
Out[4]: ['길동', '길순', '길자']
```

append 메소드를 통해 리스트 끝에 새로운 값들을 추가할 수 있어요.

tuple도 list로 변환될 수 있어요.

리스트 일부분에만 색인을 이용하여 접근할 수 있어요-슬라이스기능

```
In [11]: out_1=a_data[:2]
         print(out_1)
         [23, 32]

In [14]: out_2=a_data[1:2]
         print(out_2*2)
         [32, 32]
```

2번 인덱스 이전값 까지만 표시

1번 인덱스만 2번 반복하여 출력

✓ 튜플(tuple) – 요소들 간 순서가 있으며 값이 변하지 않는 리스트

```
In [5]: a_tuple=tuple(a_data)
        a_tuple
Out[5]: (23, 32, 37, ('길동', '길순', '길자'))

In [6]: b_1=tuple('python')
        b_1
Out[6]: ('p', 'y', 't', 'h', 'o', 'n')
```

list의 각 요소를 인덱스와 함께 접근하고자 할 경우 –enumerate함수 이용

```
In [37]: friend_list=['경아','희정','수연','성심']
         for num,myfriend in enumerate(friend_list):
             print (num + 1,myfriend)

1 경아
2 희정
3 수연
4 성심
```

## ◆ Numpy 의 배열 –ndarray 생성

- 동일한 자료타입을 갖는 값들이 n개의 차원으로 구성된 형태

numpy 배열 생성관련 함수

| 함수 | 기능 |
|---|---|
| array | 리스트,튜플,배열과 같은 순차 데이터를 ndarray로 변환 |
| arange | range함수와 비슷. (ndarray 데이터타입 반환) |
| zeros | 배열 요소 모두 0으로 초기화한  배열 생성 |
| ones | 배열 요소 모두 I로 초기화한  배열 생성 |
| empty | 새로운 배열 생성시 값을 초기화 하지 않음 |
| eye / identity | n행 n열의 단위행렬 생성 |

## ◆ array함수 이용한 배열 생성

```
In [1]: import numpy as np
        num_list=[10,20,30]
        arr=np.array(num_list)
        arr

Out[1]: array([10, 20, 30])
```

리스트형 구조를 변환하여 numpy배열 생성

```
In [3]: arr2=np.array([[1,2,3],[2,4,8]])
        arr2

Out[3]: array([[1, 2, 3],
               [2, 4, 8]])
```

다차원 배열 arr2 생성

```
In [12]: arr2.shape

Out[12]: (2, 3)
```

배열의 구조를 표시

```
In [13]: arr2.dtype

Out[13]: dtype('int32')
```

배열 요소의 데이터 타입을 표시

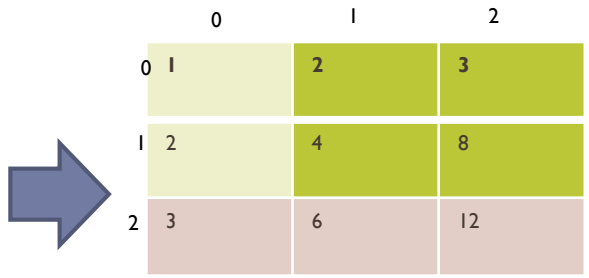
```
In [14]: arr3=np.array([10,20,30],dtype=np.float64)
         arr3.dtype

Out[14]: dtype('float64')
```

배열 생성시 자료형을 지정하면 메모리에서 지정된 데이터 타입으로 데이터를 해석

## ◆ 배열 접근 및 연산작업

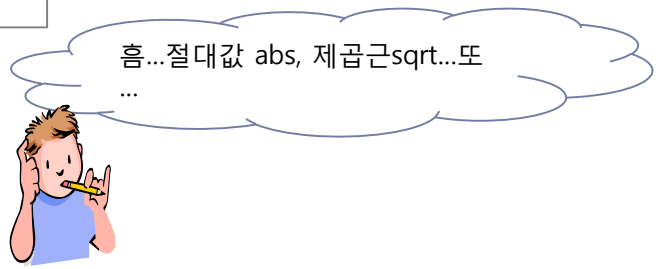
- numpy에서 배열과 배열간의  연산작업은 요소단위로 이루어짐
- 리스트에서  슬라이싱을 이용하여 각 부분에 접근했던 방법과 같이 접근

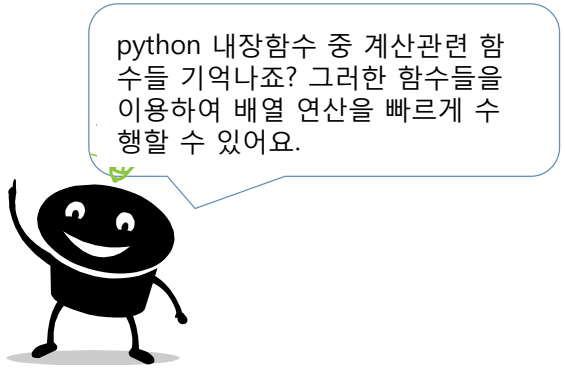
```
In [4]: import numpy as np
        arr=np.array([[1,2,3],[2,4,8],[3,6,12]])
        arr_c=arr[:2,1:]
        arr_c

Out[4]: array([[2, 3],
               [4, 8]])
```




```
In [7]: arr_c[arr_c ==0]=5
        arr_c

Out[7]: array([[2, 5],
               [4, 5]])
```

해당 배열의 요소값에 대해 조건비교 연산자를 적용할 수 있어요.
➔ 요소 0에 대해 5를 대입했어요.



## 연산을 위한 함수

| 함수 | 기능 | 함수 | 기능 |
|---|---|---|---|
| Abs/fabs | 절대값 계산 | Add /subtract | 두 배열의 같은 위치의 요소끼리 덧셈/뺄셈 |
| sqrt | 제곱근계산 | Multiply | 배열 원소끼리 곱셈 |
| square | 제곱계산 | divide | 첫번째 배열의 원소에서 두 번째 배열 원소로 나눔 |
| exp | 지수계산 | power | 첫번째 배열의 원소에 두 번째 배열의 원소만큼 제곱 |
| Ceil /floor | 소수자릿수 올림/소수자릿수 내림 | maximum | 비교하는 두 원소 중 큰 값 |
| rint | 소수자릿수 반올림 | mod | 첫번째 배열의 원소에서 두 번째 배열 원소로 나눈 나머지 |
| modf | 원소의 몫과 나머지를 각 배열로 변환 | | |
| isnan | 각 원소가 숫자인지 아닌지를 나타내는 불리언 배열 리턴 | | |

```
In [18]: arr=np.array([12,13,21,24,25])
         arr2=np.array([5,3,33,23,28])
         np.sqrt(arr)
Out[18]: array([ 3.46410162,  3.60555128,  4.58257569,  4.89897949,  5.        ])
```

arr 배열의 모든 요소값을 제곱근으로 계산하여 보여줌

```
In [19]: np.maximum(arr,arr2)
Out[19]: array([12, 13, 33, 24, 28])
```

두 배열의 요소값 들 끼리 비교하여 큰 값을 보여줌

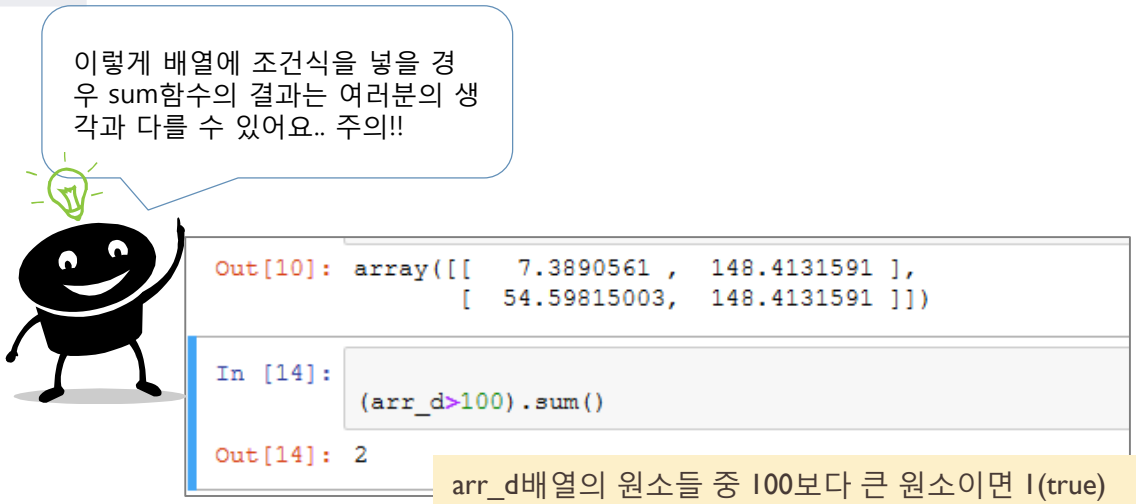
## 배열 관련 기본 통계 메소드

| 함수 | 기능 |
|---|---|
| sum | 배열의 전체 혹은 일부원소의 합 계산 |
| mean | 산술 평균 계산 |
| Std / var | 표준편차 계산 / 분산계산 |
| Min | 최소 값 계산 |
| Cumsum | 각 원소의 누적값 |
| cumprod | 각 원소의 누적곱 |

```
In [15]: ar=np.random.randn(3,3)
         ar

Out[15]: array([[-0.40207962, -0.12587151,  0.18681062],
                [ 1.29637821, -0.41248384, -1.63131452],
                [ 0.14304985, -0.56108339, -0.30251941]])

In [16]: np.mean(ar)

Out[16]: -0.20101262258860342

In [17]: np.sum(ar)

Out[17]: -1.8091136032974307
```

배열 전체요소 대상으로 평균계산

배열 전체요소 대상으로 합계계산




```
Out[10]: array([[   7.3890561 ,  148.4131591 ],
                [  54.59815003,  148.4131591 ]])

In [14]: (arr_d>100).sum()

Out[14]: 2
```

arr_d배열의 원소들 중 100보다 큰 원소이면 1(true) 그렇지않으면 0(false)으로 취급!! 따라서, 큰원소가 2개 있으므로 (즉, true값 개수) 결과 2가 출력돼요.

## ◆ 난수생성

➢ 파이썬 내장 random모듈보다 더 빠르게 더 많은 난수 표본 데이터를 생성할 수 있음

numpy.random에 포함된 주요함수

| 함수 | 기능 |
|---|---|
| shuffle | 리스트나 배열의 순서를 뒤섞음 |
| rand | 균등분포에서 표본 추출 |
| randint | 최소,최대 범위내에서 임의의 난수 추출 |
| randn | 표준편차1, 평균값이 0인 정규분포에서 표본추출 |
| normal | 정규분포에서 표본추출 |

```
In [6]: import numpy as np
        arr=np.random.normal(size=(3,3))
        arr

Out[6]: array([[-0.32555084,  1.19204773,  0.68402213],
               [ 1.09256599,  0.56935178,  0.41688875],
               [ 0.13825296,  0.27481216,  1.18990366]])
```

```
In [4]: arr1=np.random.randn(3,3)
        arr1

Out[4]: array([[-0.85796829, -0.61243938, -0.2975528 ],
               [ 0.55879012,  0.425622  , -0.71001031],
               [-0.78202528, -0.03397111,  0.09671411]])
```