넘파이(NUMPY)

In [ ]:
```
# 기존 리스트의 '+'연산은 두 리스트 연결!

list1 = [1,3,5,7,9]
list2 = [2,4,6,8,10]

print(list1 + list2)
```

In [ ]:
```
# 넘파이의 다차원 배열(ndarray)은 연산이 가능함!
import numpy as np

a = [1,2,3]
b = [5,6,7]
x = np.array(a)
y = np.array(b)

print(a+b)
print(x+y)
print(a * 3)
print(x * 3)
total = x + y
print('합계 : ', total)
print('평균 : ', total/2)
```

In [ ]:
```
# 연산 결과를 예측해 보자

a = np.array(range(1,11))
b = np.array(range(10,101,10))
print('a = ', a)
print('b = ', b)

print("a + b = ", a + b)
print("a - b = ", a - b)
print("a / b = ", a / b)
```

다차원배열(ndarray)의 속성

- ndim : 배열 축 혹은 차원의 개수
- shape : 배열의 차원으로 (m,n)형식의 튜플 형. m,n은 각 차원의 원소 크기(정수)
- dtype : 배열내의 원소의 형
- itemsize : 배열내 원소의 크기를 바이트 단위로 기술
- data : 배열의 실제 원소를 포함하고 있는 버퍼

In [ ]:
```
# 다차원배열 속성 실습

c = np.array([2,4,10])        # 1차원
d = np.array([[1,2,3],[4.7,5,6]]) # 2차원
e = np.array([[[20, 40],
               [6, 7],
               [2,2]],
              [[1, 4],
               [26, 37],
               [21,20]]]) # 3차원
```

```
print(c.shape, d.shape, e.shape)
print(c.ndim, d.ndim, e.ndim)
print(c.dtype, d.dtype, e.dtype)
print(c.itemsize, d.itemsize, e.itemsize)
print(c.size, d.size, e.size)
```

In [ ]:
```
# 연산 결과 예측해 보자.
a = np.array(range(1, 11)) + np.array(range(10,101,10))

# 몇 차원 배열이간?
print(a.ndim)

# 몇 행 몇 열의 배열인가?
print(a.shape)

# 배열 원소의 갯수는?
print(a.size)
```

[도전] 넘파이의 다차원 배열 생성 실습문제

1. 0~9까지의 정수값을 가지는 ndarray 객체 array_a 생성하기
2. range()함수를 사용하여 0~9까지의 정수 값을 가지는 ndarray 객체 array_b 생성하기
3. 2번 코드를 수정하여 짝수를 가지는 array_c 생성하기
4. array_c의 shape, ndim, dtype, size, itemsize를 출력하기

넘파이 배열 연산

- 넘파일 배열에는 수학적인 연산자 얼마든지 적용 가능 함.

In [ ]:
```
# 기존 월급에서 50만원씩 올려준다면...
salary = np.array([220, 250, 230])
salary += 50
salary
```

In [ ]:
```
# 월급을 2.1배 올려준다면....
salary = salary * 2.1 # 곱셈연산에서 실수값을 적용할 수도 있음
salary
```

[도전] BMI(체질량 지수) 구하기

- 다수의 키(m)와 몸무게(kg)값로 한꺼번에 BMI 구하기
- BMI = 체중(kg) / (키(m) * 키(m))
- 다수의 키 = [1.83, 1.76, 1.69, 1.81, 1.60, 1.73]
- 다수의 체중 = [86, 74, 59, 95, 61, 68]

넘파이 인덱싱과 슬라이싱

- 넘파일 배열에서 특정 요소를 추출하려면 인덱스를 사용. 리스트와 같음. 0부터 시작.
- 마지막 요소에 접근하려면 리스트와 같이 인덱스를 -1을 주면 됨.

- 슬라이싱도 리스트와 동일

In [ ]:
```
scores = np.array([88,72,94,89,78,99])

print("scores[2] = ", scores[2])
print("scores[-1] = ", scores[-1])
print("scores[1:4] = ", scores[1:4])   # 인덱스 1 ~ 3까지 슬라이싱 됨!!
print("scores[3:] = ", scores[3:])    # 종료 인덱스를 생략하면 항목 끝까지임!!
print("scores[4:-1] = ", scores[4:-1])  # 종료 인덱스를 -1로 쓰면, -1의 앞에서 끝남!
```

넘파이 배열 논리적인 인덱싱

- 리스트는 논리적인 인덱싱 불가!
- 넘파이 배열은 논리적인 인덱싱 가능~!

In [ ]:
```
a = [18,19,25,30,28]
b = np.array(a)
print("a 타입 : ", type(a), " b 타입 : ", type(b))

#c = a > 20   #리스트는 논리연산 안됨.
#print(c)

c = b > 20
print("c = ", c)

print("20 초과 항목만 출력 : ", b[b>20])    # b배열에서 20초과 항목만 추출함.
print("30 이상 항목만 출력 : ", b[b>=30])   # b배열에서 30이상 항목만 추출함.
```

2차원 배열 인덱싱

- 리스트는 단순 항목 나열. 행렬과 비슷하지만 연산처리 안됨.
- 넘파일 2차원 배열은 행렬에 해당하는 연산 가능.
- 배열명[행번호][열번호] 또는 배열명[행번호, 열번호]

In [ ]:
```
np_a = np.array([[1,2,3],
                 [4,5,6],
                 [7,8,9]])
print(np_a[0][2])   # 0행 2열
print(np_a[0,2])    # 0행 2열 둘다 사용가능. 리스트는 이렇게 인덱싱 할 수 없음!
print(np_a[2,-1])   # 2행의 마지막 열 항목
print(np_a[-1,-1])   # 마지막 행의 마지막 열의 값 추출
```

In [ ]:
```
# 배열의 요소를 변경할 수 있다.

np_a[0,0] = 12
np_a[-1,-1] = 123
np_a
```

In [ ]:
```
# 넘파일 배열은 모든 항목이 동일한 자료형을 가짐.

np_a[2,2] = 1.234  #이미 정수형 배열이라서 실수를 넣으면 에러는 없으나, 자동으로 정수
np_a
```

In [ ]:
```
# 2차원 배열 슬라이싱

np_b = np.array([[1,2,3,4],
```

```
                [5,6,7,8],
                [9,10,11,12],
                [13,14,15,16]])
print(np_b[0], "₩n") # 0행만 추출
print(np_b[:, 2], "₩n") # 모든행의 2열만 추출

print(np_b[0:2, 2:4], "₩n") # 행렬 일부만 추출. 0행~1행에서 2열~3열 항목만 추출(종료

print(np_b[1, 1:3], "₩n")   # 1행에서  1~2열 항목만 추출
print(np_b[::2, ::2], "₩n") # 모든 행, 모든 열에서 처음부터 두개씩 건너뛰어라~!
print(np_b[1::2, 1::2], "₩n") # 1행, 1열에서 시작하여 두개씩 건너뛰어라~!
```

In [ ]:
```
# 파이썬 리스트 슬라이싱과 넘파이 스타일의 슬라이싱은 적용했을때 다른 결과 나옴.

print(np_b[::2][::2], "₩n") # 첫 슬라이싱 : 0행, 2행 선택, 두번째 슬라이싱 : 그 중 0
print(np_b[::2][1], "₩n") # 첫 슬라이싱 : 0행, 2행 선택, 두번째 슬라이싱 : 그 중 1행
```

In [ ]:
```
import numpy as np
np_b = np.array([[1,2,3,4],
                [5,6,7,8],
                [9,10,11,12],
                [13,14,15,16]])

print(np_b[:, ::2], "₩n") #

# 첫 슬라이싱 : 모든행에서 첨부터 2열씩 건너(0열, 2열) 항목 선택, 두번째 슬라이싱 :
print(np_b[:, ::2][::2], "₩n")
```

In [ ]:
```
# 리스트

list1 = list([[1,2,[3,4]], [5,6], 7, 8])
print(list1[0][2][1])

# list1[1,1] # 오류 남!
# print(list1[1][1]) # 이것은 가능.
```

2차원 배열에서 논리적인 인덱싱 실습

- 배열에서 조건을 주어 원하는 값들만 추려 낼 수 있음.

In [ ]:
```
np_c = np.array([[1,2,3],
                [4,5,6],
                [7,8,9]])

print(np_c > 5)  # 연산의 결과가 진리값으로 표시
```

In [ ]:
```
print(np_c[np_c > 5]) # 조건에 맞는 값만 추출
```

In [ ]:
```
# 세번째 열의 값이 5초과하는 모든 행 추출하려면

print(np_c[:,2]) # 세번째 열만 추출
print()

print(np_c[:,2] > 5) # 조건 결과 확인
print()

print(np_c[ np_c[:,2] > 5]) # 조건에 해당하는 행 모두 추출
```

In [ ]:
```
# 배열에서 2의 배수 항목만 추출해보자

print(np_c, "₩n")
print(np_c[np_c % 2 == 0])
```

In [ ]:
```
# 2차원 배열 더하기
d = np_c + np_c
d
```

np.dot() 함수 - 행렬곱 함수

In [ ]:
```
# 행렬 A와 B 정의
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

# 행렬 곱셈
C = np.dot(A, B)

# 결과 출력
print(C)
```

In [ ]:
```
print(np_c)

D = np.dot(np_c, np_c)
D
```

In [ ]:
```
# 2차원 배열 그냥 곱한 값과 비교해 보자.

d = np_c * np_c
d
```

[도전] 2차원 배열에서 특정 조건을 만족하는 행만 추출하기

- 선수들의 키와 몸무게가 하나의 리스트를 구성하고 있으며, 또 이들의 리스트로 이루어진 데이터 player가 있다.
- player = [[170, 76.4],[183, 86.2],[181, 78.5],[176, 80.1]]
- 이것으로 넘파이 2차원 배열을 만들고, 선수들 가운데 몸무게가 80을 넘는 선수들만 골라서 정보를 추출해 보자.
- 또한, 키가 180이상인 선수들의 정보도 추추해 보자.

arange()함수 - range()와 같이 넘파이 배열을 만들고 싶을때 사용

In [ ]:
```
# range()함수로 리스트 만들기
list3 = list(range(0,14,2))
list3
```

In [ ]:
```
np_e = np.arange(0,5,2) # arange()함수로 바로 넘파일 베일 만들 수 있음.
print(np_e)
```

```
np_f = np.array(range(0,5,2)) # 같은 결과
print(np_f)
```

linspace() 함수 - 선형 간격의 벡터

- 그래프 축 간격 만드는데 많이 사용.
- 시작값부터 끝값(포함)까지 균일한 간격으로 숫자 추출 됨

In [ ]:
```
y = np.linspace(0,1250,10)
y
```

reshape() 함수 - 배열의 형태 변경

- 데이터의 개수는 유지한 채로 배열의 차원과 형태를 변경 함.

In [ ]:
```
y = np.arange(12)   # 1차원
print(y)

g = y.reshape(3,4)    # 3행 4열의 2차원 배열로 변경
print(g)
print(g.shape)

# g = y.reshape(3,3)  # 항목갯수가 안 맞으면 에러 남!
```

In [ ]:
```
# 만약, 어쨌든 5행으로 만들어 달라, 또는 3열로 만들어 달라 할 경우,
# 인수를 -1로 지정하면 자동으로 배열형태가 지정됨.

y = np.arange(1, 21)   # 1차원
print(y)
print()

g = y.reshape(5, -1)    # 5행은 고정 열은 자동으로 4열이 됨
print(g)
print(g.shape)
print()

h = y.reshape(-1, 2)   #2열은 고정, 행은 자동으로 10행이 되어 배열이 생성됨.
print(h)
print(h.shape)
```

넘파이 난수 생성

- 넘파이에서 정규 분포 난수 생성 가능 함. --> randn()함수

In [ ]:
```
# 넘파이에서 난수의 시드(seed)값을 설정하는 문장
# 시드값을 설정하면 매번 같은 값이 나옴

#np.random.seed(100)
np.random.rand(5)  # 0~1사이의 실수 중 임의로 5개 추출
```

In [ ]:
```
np.random.rand(5,3) # 5행 3열, 임의대로 총 15개 실수 추출
```

In [ ]:
```
np.random.randint(1, 7, size = 100) # 1~6까지의 정수 중 100번 임의대로 추출
```

In [ ]:
```
np.random.randint(1, 7, size = (5,3)) # 1~6까지의 정수를 임의대로 추출하여 5행 3열 ㅂ
```

randn()함수 - 정규 분포 난수 생성

- 표준 정규 분포 : 평균값이 0이고, 표준편차가 1.0인 분포

In [ ]:
```
# 정규 분포 난수 5개 생성
np.random.randn(5)
```

In [ ]:
```
# 정규 분포 난수 5행 4열 생성

np.random.randn(5,4)
```

In [ ]:
```
# 평균이 175cm이고, 표준편차가 10인 정규분포를 따르는 10000명의 키를 난수로 생성해 ㅂ

np.set_printoptions(precision = 2) # 넘파일 배열 소숫점 자리수 지정

m = 175
sigma = 10
height = m + sigma * np.random.randn(10000)

print(height)
```

In [ ]:
```
# 위 데이터에서 평균 계산 해보기
np.mean(height)
```

In [ ]:
```
# 위 데이터에서 중앙값 계산 해보기
# 중앙값이란 데이터를 크기순으로 나열했을때, 가운데 위치한 값을 말함.
np.median(height)
```

상관관계 계산하기 - corrcoef(x,y)함수

- x와 y 변수간의 관계를 나타내는 지표
- 값은 -1에서 1까지
- 0에 가까울수록 두 변수간의 관계가 약하다
- 양의 상관관계란 한 변수 증가시 다른 변수도 증가
- 음의 상관관계란 한 변수가 증가할 때 다른 변수는 감소 함
- 결과 [[xx xy], [yx yy]] 2행 2열 행렬로 반환

In [ ]:
```
x = np.arange(1,100)
x
```

In [ ]:
```
y = x * x
y
```

In [ ]:
```
# x값과 x의 제곱값인 y의 상관관계 구하기

result = np.corrcoef(x, y)
result                    #양의 상관관계가 있다
```