NUMPY

- 파이썬기초 -

NUMPY

Numerical Python - 행열 연산이나 다차원 배열을 편리하게 처리

배열(array) 단위로 벡터, 행렬 연산등을 파이썬의 기본 리스트에 비해 빠르고 적은양의 메모리로 연산

브로드캐스트 지원 - 형태(차원)가 다른 행렬끼리의 계산

https://numpy.org/

numpy cheat sheet

소스코드
99-1 Numpy.ipynb

NUMPY 사용하는 이유?

matrix_1 = [[1, 2],[3, 4]]
matrix_2 = [[5, 6],[7, 8]]
행열합 구하기

matrix_1 = [[1, 2],[3, 4]]
에 숫자 1씩 더하기

소스코드
99-1 Numpy.ipynb

NUMPY 사용하는 이유?

matrix_1 = [[1, 2],[3, 4]]
matrix_2 = [[5, 6],[7, 8]]
행렬합 구하기

```
for i in range(len(matrix_1)):
    tmp = []
    for j in range(len(matrix_2)):
        tmp.append(matrix_1[i][j]+matrix_2[i][j])
    matrix_result.append(tmp)

print(matrix_result)
```

```
matrix_result = np.array(matrix_1) + np.array(matrix_2)
print(matrix_result)
```

소스코드
99-1 Numpy.ipynb

NUMPY 사용하는 이유?

matrix_1 = [[1, 2],[3, 4]]
에 숫자 1씩 더하기

```
matrix_1 = [[1, 2],[3, 4]]

for i in range(len(matrix_1)):
    for j in range(len(matrix_2)):
        matrix_1[i][j] = matrix_1[i][j] + 1

print(matrix_1)
```

```
matrix_1 = [[1, 2],[3, 4]]

matrix_1 = np.array(matrix_1) + 1
```

소스코드
99-1 Numpy.ipynb

NUMPY 사용하는 이유?

```
import time

size = 10000000

#list
x = list(range(size))
y = list(range(size))

start_time = time.time()
z = [x[i]+x[i] for i in range(size)]
print("리스트 걸린시간", time.time()-start_time)

#adlist
x = np.arange(size)
y = np.arange(size)
start_time = time.time()
z = x + y
print("넘파이 걸린시간", time.time()-start_time)
```

결과 :

리스트 걸린시간 **1.85866379737854**
넘파이 걸린시간 **0.17046713829040527**

소스코드
99-1 Numpy.ipynb

NUMPY 자료형

int(8bit, 16bit, 32bit, 64bit)
부호가 있음
비트수 만큼 크기를 가지는 정수형

uint(8bit, 16bit, 32bit, 64bit)
부호가 없음
비트수 만큼 크기를 가지는 정수형

float(16bit, 32bit, 64bit, 128bit)
부호가 있음
비트수 만큼 크기를 가지는 실수형

복소수형
complex64 : 두개의 32비트 부동소수점으로 표시되는 복소수
complex128 : 두개의 64비트 부동소수점으로 표시되는 복소수

bool
True, False

소스코드
99-1 Numpy.ipynb

ndarray

넘파이 리스트 ndarray 만들기
np.array(데이터)
np.float32(데이터)
np.int_(데이터)

타입설정
np.array(데이터, **dtype**=np.float32)

형변환
np.int32(ndarray)

타입체크
ndarray.dtype
np.issubdtype(ndarray.dtype, np.floating)

```
x = np.uint(32)
print(x.dtype) #uint64

x = np.array([1, 2, 3, 4])
print(x.dtype) #int64

x = np.float32([1, 2, 3, 4])
print(x.dtype) #float32

x = np.array([1, 2, 3, 4], dtype=np.float32)
print(x.dtype) #float32

x = np.int32(x)
print(x.dtype)
np.issubdtype(x.dtype, np.int32) #float32
```

소스코드
99-1 Numpy.ipynb

다차원 ndarray

0차원
np.array(1)

1차원
np.array([1, 2])

2차원 혹은 그 이상
np.array([[1, 2, 3],[4, 5, 6]])

차원확인
ndarray.shape #행렬구조 ex (2, 3)
ndarray.ndim #몇차원인지
ndarray.size # 원소의 개수

```
x = np.array(1)
print(x.shape) # ()
print(x.ndim) # 0
print(x.size) # 1

x = np.array([1, 2])
print(x.shape) # (2, )
print(x.ndim) # 1
print(x.size) # 2

x = np.array([[1, 2, 3],[4, 5, 6]])
print(x.shape) # (2, 3)
print(x.ndim) # 2
print(x.size) # 6
```

소스코드
99-1 Numpy.ipynb

arange

파이썬의 range 와 비슷

np.arange(10)
np.arange(10.0)
np.arange(시작값, 끝값, 증가값)
np.arange(1, 10, 0.5)
np.arange(10, 1, -0.5)

```
x = np.arange(10)
print(x)
```
결과 : [0 1 2 3 4 5 6 7 8 9]

```
x = np.arange(10.0)
print(x)
```
결과 : [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]

```
x = np.arange(1, 10, 2)
print(x)
```
결과 : [1 3 5 7 9]

```
x = np.arange(1, 10, 0.5)
print(x)
```
결과 : [1. 1.5 2. 2.5 3. 3.5 4. 4.5 5. 5.5 6. 6.5 7. 7.5 8. 8.5 9. 9.5]

```
x = np.arange(10, 1, -0.5)
print(x)
```
결과 : [10. 9.5 9. 8.5 8. 7.5 7. 6.5 6. 5.5 5. 4.5 4. 3.5 3. 2.5 2. 1.5]

소스코드
99-1 Numpy.ipynb

linspace

균일한 간격으로 리스트 크기만큼의 리스트생성

np.linspace(시작값, 끝값, 벡터크기)

끝값 포함여부
np.linspace(시작값, 끝값, 벡터크기, endpoint=False)

```
x = np.linspace(1, 20, 5)
print(x)
```

결과 :

```
[ 1.    5.75 10.5  15.25 20.  ]
```

```
x = np.linspace(1, 20, 10, endpoint=False)
print(x)
```

결과 :

```
[ 1.   2.9  4.8  6.7  8.6 10.5 12.4 14.3 16.2 18.1]
```

소스코드
99-1 Numpy.ipynb

reshape

데이터를 유지하면서 차원의 형태를 변경

ndarray.reshape(3, 3)
ndarray.reshape(2, 3, 4)
ndarray.reshape(2, 2, 2, 2)

np.array([[1,2],[3,4],[5,6]]).reshape(2,3)

-1 일경우 자동으로 맞춰서 생성
ndarray.reshape(3,-1)

```
x = np.arange(9).reshape(3, 3)
print(x) # 2차원

x = np.arange(24).reshape(2, 3, 4)
print(x) # 3차원

x = np.arange(16).reshape(2, 2, 2, 2)
print(x) # 4차원

x = np.array([[1,2],[3,4],[5,6]])
print(x)
print(x.reshape(2,3)) # 데이터를 유지하고 형태변경

x = np.arange(9).reshape(3, -1)
print(x) #개수에 맞춰 생성
```

소스코드
99-1 Numpy.ipynb

slicing

a[처음값:끝값:증가값]

1차원뿐만 아니라 다차원 슬라이싱 가능
x[1:3, 1:3]
x[:3, :3, :3]

```
# 1차원 슬라이싱
x = np.arange(20)
print(x[1:3])

# 2차원 슬라이싱
x = np.arange(20).reshape(4,5)
print(x)
print(x[1:3])
print(x[1:3, 1:3])

# 3차원 슬라이싱
x = np.arange(30).reshape(2,5,3)
print(x)
print(x[:,3:5,1])
```

소스코드
99-1 Numpy.ipynb

indexing & boolean indexing

다차원리스트 접근
x[1][1]
x[1,1]

한번에 여러값 인덱싱
x[[1,1],[1,2]]

indexing 에 비교, 논리연산자 사용가능
! - not, & - and, | - or
x[a > 3]
x[a == 1]
x[~(a == 1)]
x[(a > 3) & (a < 8)]

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

# indexing
print(a[1][1])
print(a[1,1])
print(a[[1,1],[1,2]])

# boolean indexing
print(a > 3)
print(a[a > 3])
print(a[a == 1])
print(a[~(a == 1)])
print(a[(a > 3)&(a < 8)])
```

소스코드
99-1 Numpy.ipynb

random

난수가 들어가있는 다양한형태의 데이터
np.random.rand(5,5)

정수난수
np.random.randint(1, 10)

정수형 들어가있는 다양한형태의 데이터
np.random.randint(1, 10, size=(5))
np.random.randint(1, 10, size=(5, 5))

소스코드
99-1 Numpy.ipynb

특별한 형태의 배열

1이 들어가있는 배열
np.ones([5, 5])

0이 들어가있는 배열
np.zeros([5, 5])

단위행렬 (행렬곱을 했을때 자기자신이 나오는)
np.eye(5)

행열 펼치기 (행기준)
ndarray.ravel()
ndarray.ravel(order='C')

열기준으로 펼치기
ndarray.ravel(order='F')

소스코드
99-1 Numpy.ipynb

concatenate

배열을 연결

np.concatenate([x, y])

열을기준으로

np.concatenate([x, y], axis=1)

```
x = np.arange(1, 4)
y = np.arange(4, 7)

np.concatenate([x, y])
```

```
x = np.arange(10).reshape(2, 5)
y = np.arange(10, 20).reshape(2, 5)

np.concatenate([x, y])
np.concatenate([x, y], axis=1)
```

소스코드

99-1 Numpy.ipynb

split

배열을 분해
np.split(x, 4)

열을기준으로
np.split(x, 2, axis=1)

```
x = np.arange(12)
np.split(x, 4)
```

```
x = np.arange(16).reshape(4, 4)
np.split(x, 2)
```

```
x = np.arange(16).reshape(4, 4)
np.split(x, 2, axis=1)
```

소스코드
99-1 Numpy.ipynb

broadcast (1)

형태(차원)가 다른 행렬끼리의 계산
다차원 + 숫자

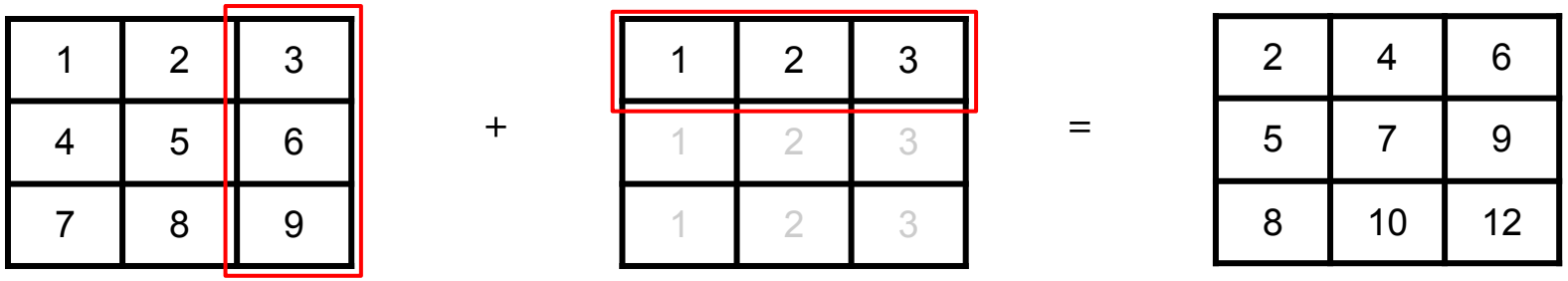

```
x = np.array([1,2,3])
x + 1
```

소스코드
99-1 Numpy.ipynb

broadcast (2)

형태(차원)가 다른 행렬끼리의 계산

2차원 + 1차원




```
x = np.arange(1, 10).reshape(3, 3)
y = np.arange(1, 4)
x + y
```

소스코드

99-1 Numpy.ipynb

broadcast (3)

형태(차원)가 다른 행렬끼리의 계산

3차원 + 1차원

10	10	10
20	20	20
30	30	30

+

1	2	3
1	2	3
1	2	3

=

11	12	13
21	22	23
31	32	33

```
x = np.array([10, 20, 30]).reshape(3, 1)
y = np.arange(1, 4)
x + y
```

소스코드

99-1 Numpy.ipynb

연산 및 집계함수

```
x = np.arange(4).reshape(2, 2)
y = np.arange(4).reshape(2, 2)

x.dot(y) # 행렬곱

np.transpose(x) # 전치행렬
np.linalg.inv(x) # 역행렬
np.linalg.det(x) # 행렬식

np.mean(x) # 평균
np.median(x) # 중간값
np.std(x) # 표준편차
np.var(x) # 분산
np.sum(x) # 합
np.sum(데이터, axis=1) # 합, 축변경
```

```
np.cumsum(x) # 누적합
np.cumprod(x) # 누적곱
np.min(x) # 최소값
np.argmin(x) # 최소값 위치
np.argmax(x) # 최대값 위치
np.any(x > 4) # 하나라도 참이어야 참
np.all(x > 4) # 모든 요소가 참이어야 참

np.where(x > 4) # 조건에 맞는위치
np.where(x > 4, x, -100) # 조건, True 일경우, False 일경우
```

소스코드
99-1 Numpy.ipynb

실습1

x = [1, 2, 3, 4, 5, 6, 7, 8, 9], y = [10, 20, 30, 40, 50, 60, 70, 80, 90]
위의 배열로 Numpy 를 사용하여 3 X 3 행렬을 만든뒤 행렬합과 행렬곱을 구해주세요

결과 :

```
[[11 22 33]
 [44 55 66]
 [77 88 99]]

[[ 300  360  420]
 [ 660  810  960]
 [1020 1260 1500]]
```

소스코드
99-1 Numpy.ipynb

실습2

1에서 20사이의 균일한 간격으로 30개의 숫자를 만들고
모든값에 숫자 10을 더해주세요

결과 :

```
[11.         11.65517241 12.31034483 12.96551724 13.62068966 14.27586207
 14.93103448 15.5862069  16.24137931 16.89655172 17.55172414 18.20689655
 18.86206897 19.51724138 20.17241379 20.82758621 21.48275862 22.13793103
 22.79310345 23.44827586 24.10344828 24.75862069 25.4137931  26.06896552
 26.72413793 27.37931034 28.03448276 28.68965517 29.34482759 30.        ]
```

소스코드
99-1 Numpy.ipynb

실습3

1~100까지 난수가 들어가있는 2x10x10 배열을 만들고
50보다 크거나 같은값은 1, 50보다 작은값은 2로 변환하고
2번째 배열에 아래와 같이 붉은색 위치만 뽑아주세요

```
[[[25 91 38 37 46 17 75 37 96 93]
  [15 70 77 26 89 61 31 38 18 22]
  [ 9 37  2 12 98 26 84 89 19 71]
  [33 38 30 28 11 78  2 67 86 85]
  [83 71 30 82 16 52 97 35 96 67]
  [66 60 56 71 88 66 51 22 46  3]
  [ 7 71 94 30 63 50 10 10 61 27]
  [14 89 79 72 41 73 61 51 26 42]
  [55 14 84 24 16 27 60 67 19 52]
  [97  9 30 98 87 73 85 27 51 19]]

 [[ 6 67 30  2 56 33 53 37 49 75]
  [78 56 88 40  4 99 48  4 59 36]
  [15 34 83 47 97 15 44 29 47 76]
  [70 14 61 18 11 36 72 28 22 64]
  [59 50 56 90 11 92  7 35 73 78]
  [38 46 56 48 68 17 29 63 29 42]
  [29 13 61  7 68 13 21 54 35 69]
  [47 74 60 93 71 19  7 94 76 11]
  [53 70 49 70 11 95 83 45 18 48]
  [ 3 66 53 61 36 77 68 31 50 10]]]
```

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

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

결과 :

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

소스코드
99-1 Numpy.ipynb