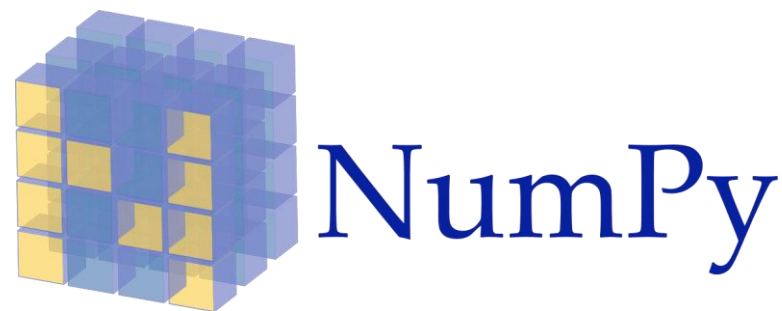


AI Programming

Lecture 20

Preview

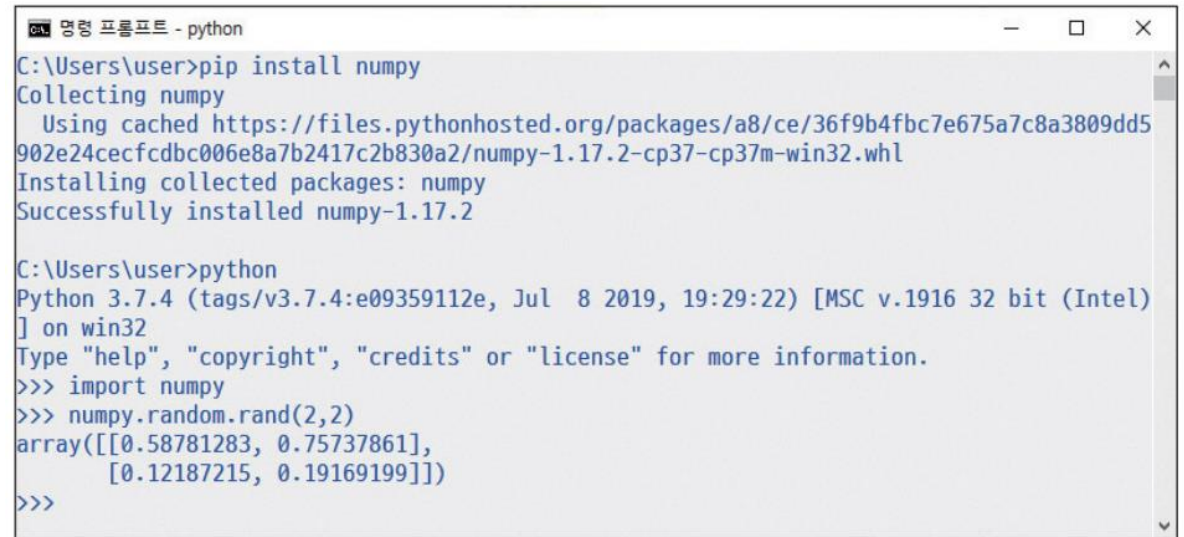
- NumPy ([link](#))
 - 1. 넘파이 라이브러리
 - 2. ndarray의 메소드와 주요 함수
 - 3. ndarray의 연산
 - 4. ndarray의 생성
 - 5. ndarray의 재구성



1 넘파이 라이브러리

NumPy

- NumPy library
 - Matrix & vector calculations
 - Installation
 - Using pip at command
 - Anaconda ([link](#))
 - Colab



```
C:\Users\user>pip install numpy
Collecting numpy
  Using cached https://files.pythonhosted.org/packages/a8/ce/36f9b4fbc7e675a7c8a3809dd5902e24cecfcdabc006e8a7b2417c2b830a2/numpy-1.17.2-cp37-cp37m-win32.whl
Installing collected packages: numpy
Successfully installed numpy-1.17.2

C:\Users\user>python
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> numpy.random.rand(2,2)
array([[0.58781283, 0.75737861],
       [0.12187215, 0.19169199]])
>>>
```

[그림 11-1] pip를 이용한 numpy 설치화면과 간단한 테스트 코드

NumPy

```
import numpy as np
a = np.array([1, 2, 3]) # 넘파이 ndarray 객체 생성
a
array([1, 2, 3])
```

```
a.shape          # 객체 a의 형태 (shape)
(3,)
```

```
a.ndim           # 객체 a의 차원
1
```

```
a.dtype          # 객체 a의 내부 자료형
dtype('int64')
```

```
a.itemsize       # 객체 a의 내부 자료형이 차지하는 메모리 크기 (byte)
8
```

```
a.size           # 객체 a의 크기 (항목의 수)
3
```

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

1	2	3
---	---	---

```
a.shape : (3,)
a.ndim  : 1
a.dtype : dtype('int32')
a.size  : 3
a.itemsize : 4
```

[그림 11-2] 넘파이의 ndarray a와 그 속성들

NumPy

속성	설명
ndim	배열 축 혹은 차원의 갯수.
shape	배열의 차원으로 (m, n) 형식의 튜플 형이다. 이 때, m과 n은 각 차원의 원소의 크기를 알려주는 정수 값이다.
size	배열의 원소의 갯수이다. 이 갯수는 shape내의 원소의 크기의 곱과 같다. 즉 (m, n) shape 배열의 size는 $m*n$ 이다.
dtype	배열내의 원소의 형을 기술하는 객체이다. numpy는 파이썬 표준 형을 사용할 수 있으나 numpy 자체의 자료형인 bool_, character, int_, int8, int16, int32, int64, float, float8, float16_, float32, float64, complex_, complex64, object_ 형을 사용할 수 있다.
itemsize	배열내의 원소의 크기를 바이트 단위 로 기술한다. 예를 들어 int32 자료형의 크기는 $32/8=4$ 바이트가 된다.
data	배열의 실제 원소를 포함하고 있는 버퍼.

NumPy

• NumPy and data types

```
a = np.array([1, 2, 3], dtype='int32')  
b = np.array([4, 5, 6], dtype='int64')  
a.dtype
```

```
dtype('int32')
```

```
b.dtype
```

```
dtype('int64')
```

```
c = a + b  
c.dtype
```

```
dtype('int64')
```

```
d = np.array([1, 2, 3], dtype=np.float32)  
d
```

```
array([1., 2., 3.], dtype=float32)
```



NOTE : 넘파이 배열의 데이터 타입을 지정하는 두 가지 방법

넘파이 배열의 데이터 타입을 지정하는 방법에는 두 가지가 있다.

1. dtype = np.int32 와 같이 np의 int32 속성 값으로 지정하기

```
>>> a = np.array([1, 2, 3, 4], dtype = np.int32)
```

2. dtype = 'int32' 와 같이 문자열 형식으로 속성 값 지정하기

```
>>> a = np.array([1, 2, 3, 4], dtype = 'int32')
```

NumPy



주의 : ndarray 배열을 생성할 때 주의할 점

1. 넘파이의 배열 ndarray을 생성할 때, 반드시 대괄호를 사용하여 리스트 형식의 데이터를 만들어서 array() 함수의 인자로 넣어야 한다.

```
>>> a = np.array([1, 2, 3, 4])
```

만일 리스트 형식으로 하지 않고 쉼표로 구분해서 입력할 경우 다음과 같은 오류가 발생된다.

```
>>> a = np.array(1, 2, 3, 4) # 잘못된 입력
```

```
...
```

```
ValueError: only 2 non-keyword arguments accepted
```


NumPy

2. 넘파이의 ndarray는 리스트와는 달리, 서로 다른 자료형의 값을 원소로 가질 수 없다.

```
>>> a = np.array([1, 'two', 3, 4], dtype = np.int32)
...
ValueError: invalid literal for int() with base 10: 'two'
```

만일 다음과 같이 자료형을 명시하지 않을 경우 'two'라는 원소의 자료형인 str 형으로 자동 형 변환이 일어난다. 이 경우 모든 원소들은 문자열 형이되어 정수의 덧셈, 뺄셈 등의 연산을 사용할 수 없다.

```
>>> a = np.array([1, 'two', 3, 4])
>>> a
array(['1', 'two', '3', '4'], dtype = '<U21')
```

NumPy

• Exercise



LAB 11-1 : ndarray 객체 생성하기 그리고 속성 알아보기

1. 0에서 9까지의 정수 값을 가지는 ndarray 객체 a를 넘파이를 이용하여 작성하여 다음과 같이 출력하여라.

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

2. range() 함수를 사용하여 0에서 9까지의 정수 값을 가지는 ndarray 객체 b를 만들고 문제 1의 결과와 같이 나타나도록 하여라.

3. 문제 2의 코드를 수정하여 0에서 9까지의 정수 값 중에서 다음과 같이 짝수를 가지는 ndarray 객체 c를 출력하여라.

```
array([0, 2, 4, 6, 8])
```

```
a = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])  
a
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
b = np.array(range(10))  
b
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
c = np.array(range(0, 10, 2))  
c
```

```
array([0, 2, 4, 6, 8])
```

NumPy

- (Cont'd)

4. 문제 3번 ndarray 객체 c의 shape, ndim, dtype, size, itemsize를 다음과 같이 출력하여라.

```
c.shape = (5,)
c.ndim = 1
c.dtype = int64
c.size = 5
c.itemsize = 8
```

```
print(c.shape)
print(c.ndim)
print(c.dtype)
print(c.size)
print(c.itemsize)
```

```
(5,)
1
int64
5
8
```

NumPy

- Multi-dimensional arrays

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

```
(3, 2)  
2
```

```
b = np.array([[1, 2, 3]])  
print(b.shape)  
print(b.ndim)
```

```
(1, 3)  
2
```

```
c = np.array([[1], [2], [3]])  
print(c.shape)  
print(c.ndim)
```

```
(3, 1)  
2
```

```
d = np.array([[[1, 2], [3, 4], [5, 6]], [[7, 8], [9, 10], [11, 12]]])  
print(d.shape)  
print(d.ndim)
```

```
(2, 3, 2)  
3
```

2 ndarray의 메소드와 주요 함수

Methods

- **Max, min, mean, flatten**

```
a = np.array([1, 2, 3])  
print(a.max())  
print(a.min())  
print(a.mean())
```

```
3  
1  
2.0
```

```
a = np.array([[1, 1], [2, 2], [3, 3]])  
a.flatten()
```

```
array([1, 1, 2, 2, 3, 3])
```

Methods

- Axis (축)

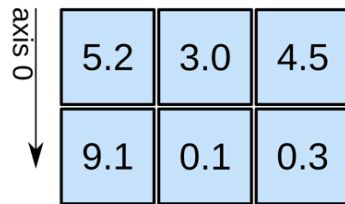
1D array



axis 0 →

shape: (4,)

2D array

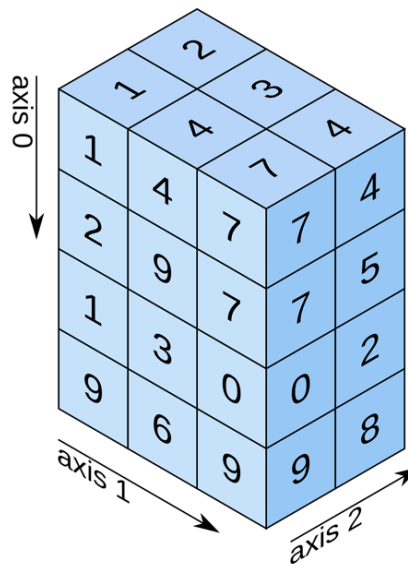


axis 0 ↓

axis 1 →

shape: (2, 3)

3D array



axis 0 ↓

axis 1 ↘

axis 2 ↗

shape: (4, 3, 2)

Functions

- `append(a, b, axis)`
 - NumPy array (혹은 Python list) `a`와 `b`를 `axis` 번째 차원을 기준으로 병합
 - `axis` 입력을 명시하지 않으면 flatten된 NumPy array 반환

```
a = np.array([1, 2, 3])  
b = np.array([[4, 5, 6], [7, 8, 9]])  
np.append(a, b)
```

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
np.append(b, a)
```

```
array([4, 5, 6, 7, 8, 9, 1, 2, 3])
```

```
np.append([1, 2, 3], [4, 5, 6])
```

```
array([1, 2, 3, 4, 5, 6])
```

```
a = np.array([[1, 2, 3]])  
b = np.array([[4, 5, 6], [7, 8, 9]])  
np.append(a, b, axis=0)
```

```
array([[1, 2, 3],  
       [4, 5, 6],  
       [7, 8, 9]])
```

```
np.append(a, b, axis=1)
```

ValueError: all the input array dimensions for the concatenation axis must match exactly, but along dimension 0, the array at index 0 has size 1 and the array at index 1 has size 2

Functions

- (Cont'd)

```
np.append([[1, 2], [3, 4]], [[5, 6], [7, 8]], axis=0)
```

```
array([[1, 2],  
       [3, 4],  
       [5, 6],  
       [7, 8]])
```

```
np.append([[1, 2], [3, 4]], [[5, 6], [7, 8]], axis=1)
```

```
array([[1, 2, 5, 6],  
       [3, 4, 7, 8]])
```

```
np.append([[1, 2], [3, 4]], [[5, 6]], axis=0)
```

```
array([[1, 2],  
       [3, 4],  
       [5, 6]])
```

```
np.append([[1, 2], [3, 4]], [5, 6], axis = 0)
```

Functions

- Random number generation

```
np.random.rand(3, 3)  # (3, 3) shape의 random array 생성
```

```
array([[0.22329133, 0.03950058, 0.28256727],  
       [0.41382097, 0.92093592, 0.97253715],  
       [0.22575063, 0.5673712 , 0.28169122]])
```

```
np.random.randint(0, 10, size=10)  # 0에서 10사이의 10개 random number 생성
```

```
array([1, 7, 9, 5, 6, 3, 1, 7, 3, 5])
```

```
np.random.randint(0, 10, size=(3, 3))
```

```
array([[7, 4, 7],  
       [7, 7, 4],  
       [6, 6, 0]])
```

Functions

• Exercise



LAB 11-2 : ndarray 객체의 메소드와 함수

1. [23, 45, 67, 7, 2, 30, 34, 82]의 정수 값을 가지는 ndarray 객체 a를 생성하여 다음과 같이 출력하여라.

```
a = array([23 45 67 7 2 30 34 82])
```

- 이 ndarray의 max(), min(), mean() 메소드를 이용하여 최댓값, 최솟값, 평균을 다음과 같이 출력하여라.

```
최댓값 : 82  
최솟값 : 2  
평균 : 36.25
```

```
a = np.array([23, 45, 67, 7, 2, 30, 34, 82])  
print("최대값: %d" % a.max())  
print("최소값: {0:d}".format(a.min()))  
print("평균: {}".format(a.mean()))
```

```
최대값: 82  
최소값: 2  
평균: 36.25
```

Functions

• (Cont'd)

2. `numpy.random.randint()` 함수를 사용하여 0에서 99까지의 정수 값 10개를 랜덤하게 생성하십시오. 이 10개의 값을 `b`라는 `ndarray`에 넣고 `ndarray`에서 최댓값, 최솟값, 평균을 다음과 같이 출력하십시오(`b` 값은 랜덤하게 생성되므로 다음 화면의 값과 일치하지 않음).

```
b = [25 2 9 86 93 73 15 53 67 20]
```

최댓값 : 93

최솟값 : 2

평균 : 44.3

3. 위의 문제 1의 `a` 배열과 문제 2의 `b` 배열을 `append()` 하여 다음과 같은 배열 `c`를 생성하여 출력하십시오.

```
c = [23 45 67 7 2 30 34 82, 25 2 9 86 93 73 15 53 67 20]
```

```
b = np.random.randint(0, 99, 10)
print(b)
print("최대값: %d" % b.max())
print("최소값: {0:d}".format(b.min()))
print("평균: {}".format(b.mean()))
```

[45 96 94 73 11 64 79 97 44 47]

최대값: 97

최소값: 11

평균: 65.0

3 ndarray의 연산

Calculations

- Addition

```
a = np.array([1, 2, 3])  
b = np.array([4, 5, 6])  
c = a + b  
c
```

```
array([5, 7, 9])
```

1	2	3
+		
4	5	6
=		
5	7	9

Calculations

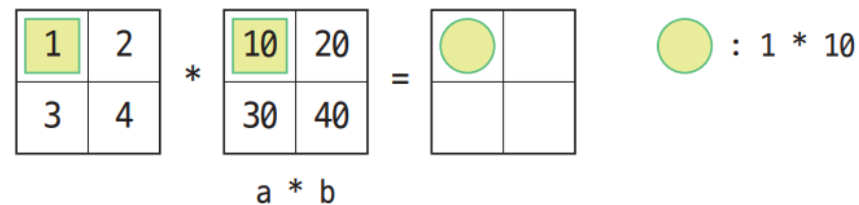
- Arithmetic operations of 2D arrays

```
a = np.array([[1, 2], [3, 4]])  
b = np.array([[10, 20], [30, 40]])  
a + b
```

```
array([[11, 22],  
       [33, 44]])
```

```
a - b
```

```
array([[ -9, -18],  
       [-27, -36]])
```



[그림 11-4] 넘파이의 a * b 연산

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix} = \begin{bmatrix} 1*10 & 2*20 \\ 3*30 & 4*40 \end{bmatrix}$$

```
a * b    # element-wise multiplication
```

```
array([[ 10,  40],  
       [ 90, 160]])
```

```
a / b    # element-wise division
```

```
array([[0.1, 0.1],  
       [0.1, 0.1]])
```

Calculations

- Matrix multiplication

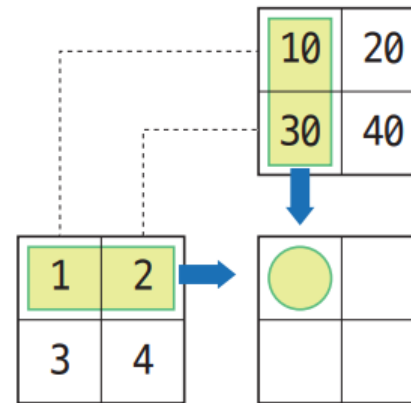
```
np.matmul(a, b)
```

```
array([[ 70, 100],  
       [150, 220]])
```

```
a @ b
```

```
array([[ 70, 100],  
       [150, 220]])
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} @ \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix} = \begin{bmatrix} 1*10+2*30 & 1*20+2*40 \\ 3*10+4*30 & 3*20+4*40 \end{bmatrix}$$



a @ b

○ : $1 * 10 + 2 * 30$

Calculations

- Shapes

```
a = np.array([1, 2])
b = np.array([4, 5, 6])
c = a + b
c
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-61-324e1dce6f11> in <module>()
      1 a = np.array([1, 2])
      2 b = np.array([4, 5, 6])
----> 3 c = a + b
      4 c
```

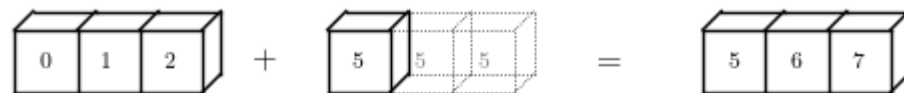
ValueError: operands could not be broadcast together with shapes (2,) (3,)

Calculations

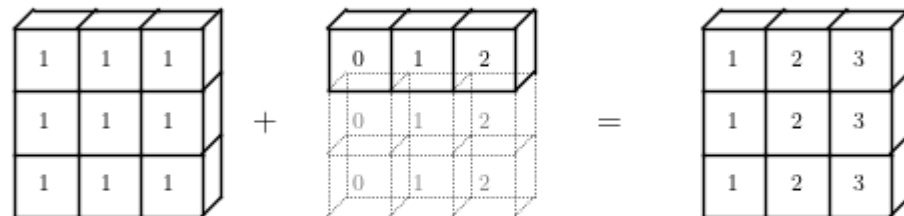
- **Broadcasting**

- 특정 조건이 만족되면 shape이 다른 ndarray 연산 수행 가능

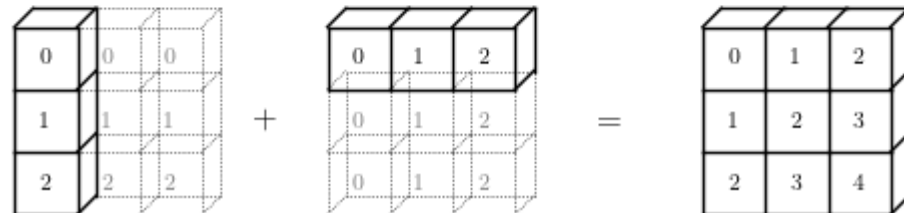
`np.arange(3) + 5`



`np.ones((3, 3)) + np.arange(3)`



`np.arange(3).reshape((3, 1)) + np.arange(3)`



Calculations

- Examples

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

```
a + 1
```

```
array([[2, 3],  
       [4, 5]])
```

```
a - 1
```

```
array([[0, 1],  
       [2, 3]])
```

```
a * 100
```

```
array([[100, 200],  
       [300, 400]])
```

```
a / 100
```

```
array([[0.01, 0.02],  
       [0.03, 0.04]])
```

Calculations

- **Broadcasting 조건**

- Scalar

- 원소가 하나인 ndarray (1,)

- `np.array([[1, 2], [3, 4]]) + np.array([3]) → [[4, 5], [6, 7]]`

- 1차원 ndarray

- `np.array([[1, 2], [3, 4]]) + np.array([1, 2]) → [[2, 4], [4, 6]]`

- 차원의 짝이 맞음

- `np.array([[1, 2]]) + np.array([[1], [2]]) → [[2, 3], [3, 4]]`

- `np.array([[1], [2]]) + np.array([[1, 2]]) → [[2, 3], [3, 4]]`

Calculations

```
a = np.array([[1, 2, 3], [2, 5, 6], [8, 9, 10]])
b = 1
c = np.array([1])
d = np.array([3, 3, 3])
e = np.array([[4, 5, 6]])

print(a + b, end="\n\n") # (3, 3) + scalar
print(a + c, end="\n\n") # (3, 3) + (1,)
print(a + d, end="\n\n") # (3, 3) + (3,)
print(e + d, end="\n\n") # (1, 3) + (3,)
```

```
[[ 2  3  4]
 [ 3  6  7]
 [ 9 10 11]]
```

```
[[ 2  3  4]
 [ 3  6  7]
 [ 9 10 11]]
```

```
[[ 4  5  6]
 [ 5  8  9]
 [11 12 13]]
```

```
[[7 8 9]]
```

4 ndarray의 생성

Generation

- Zeros, ones, full, eye

```
np.zeros((2, 3))
```

```
array([[0., 0., 0.],  
       [0., 0., 0.]])
```

```
np.full((2, 3), 10)
```

```
array([[10, 10, 10],  
       [10, 10, 10]])
```

```
np.zeros((3,))
```

```
array([0., 0., 0.])
```

```
np.zeros((1, 3))
```

```
array([[0., 0., 0.]])
```

```
np.ones((3, 2))
```

```
array([[1., 1.],  
       [1., 1.],  
       [1., 1.]])
```

```
np.eye(3)
```

```
array([[1., 0., 0.],  
       [0., 1., 0.],  
       [0., 0., 1.]])
```

```
np.zeros((3, 1))
```

```
array([[0.],  
       [0.],  
       [0.]])
```

```
np.eye(2, 3)
```

```
array([[1., 0., 0.],  
       [0., 1., 0.]])
```

Generation

- Arange

```
np.arange(0, 10)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
np.arange(0, 10, 2)
```

```
array([0, 2, 4, 6, 8])
```

```
np.arange(0, 10, 3)
```

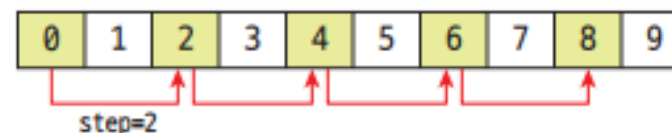
```
array([0, 3, 6, 9])
```

arange(0, 10)

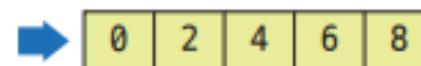
0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

[그림 11-6] arange(0,10) 실행 결과

arange(0, 10, 2)



[그림 11-7] arange(0, 10, 2) 실행과 스텝 값



[그림 11-8] arange(0, 10, 2) 실행 결과

Generation

- Arange

```
np.arange(0.0, 1.0, 0.2)  
  
array([0. , 0.2, 0.4, 0.6, 0.8])
```

```
range(0.0, 1.0, 0.2)
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-108-f0723f02f946> in <module>()  
----> 1 range(0.0, 1.0, 0.2)
```

```
TypeError: 'float' object cannot be interpreted as an integer
```

Generation

- Linspace

```
np.linspace(0, 10, 5)
```

```
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

```
np.linspace(0, 10, 4)
```

```
array([ 0.          ,  3.33333333,  6.66666667, 10.          ])
```

Generation

• Exercise



LAB 11-4 : 행렬의 생성

1. `full()` 함수를 이용하여 모든 원소의 값이 2인 3x3 크기의 행렬 `a1`을 생성하여라.

```
a1 = [[2 2 2]
      [2 2 2]
      [2 2 2]]
```

2. `arange()` 함수를 사용하여 1에서 12까지 12개의 원소를 가지는 1차원 행렬 `a2`를 생성하여라.

```
a2 = [ 1 2 3 4 5 6 7 8 9 10 11 12]
```

```
a1 = np.full((3, 3), 2)
a1
```

```
array([[2, 2, 2],
       [2, 2, 2],
       [2, 2, 2]])
```

```
a2 = np.arange(1, 13)
a2
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

Generation

- (Cont'd)

3. `arange()` 함수의 `step` 값을 이용하여 1에서 50까지 정수 중에서 3의 배수만을 가지는 행렬 `a3`을 생성하여라.

```
a3 = [ 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48]
```

4. 0에서 20 사이에서 동일한 간격의 값을 가지는 원소 5개를 가진 `a4` 행렬을 생성하여라.

```
a4 = [ 0. 5. 10. 15. 20.]
```

```
a3 = np.arange(3, 50, 3)  
a3
```

```
array([ 3,  6,  9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48])
```

```
a4 = np.linspace(0, 20, 5)  
a4
```

```
array([ 0.,  5., 10., 15., 20.])
```

Summary

- **NumPy array**

- Attributes: shape, ndim, size,...
- Methods: max, min, mean, flatten
- Functions: append, random, zeros, ones, full, eye, arange
- Axis
- Broadcasting