Introduction to Data Mining Lecture

Week 3: Numpy (1)

Joon Young Kim

Assistant Professor, School of AI Convergence
Sungshin Women's University

Week 3 Outline

- Week 2 Review
- Numpy Array Object
- Numpy ndarray
- Array Data types
- Structured Array
- Structured Datatypes
- Struct Array Assignment
- Struct Array Indexing and Slicing
- Conclusion

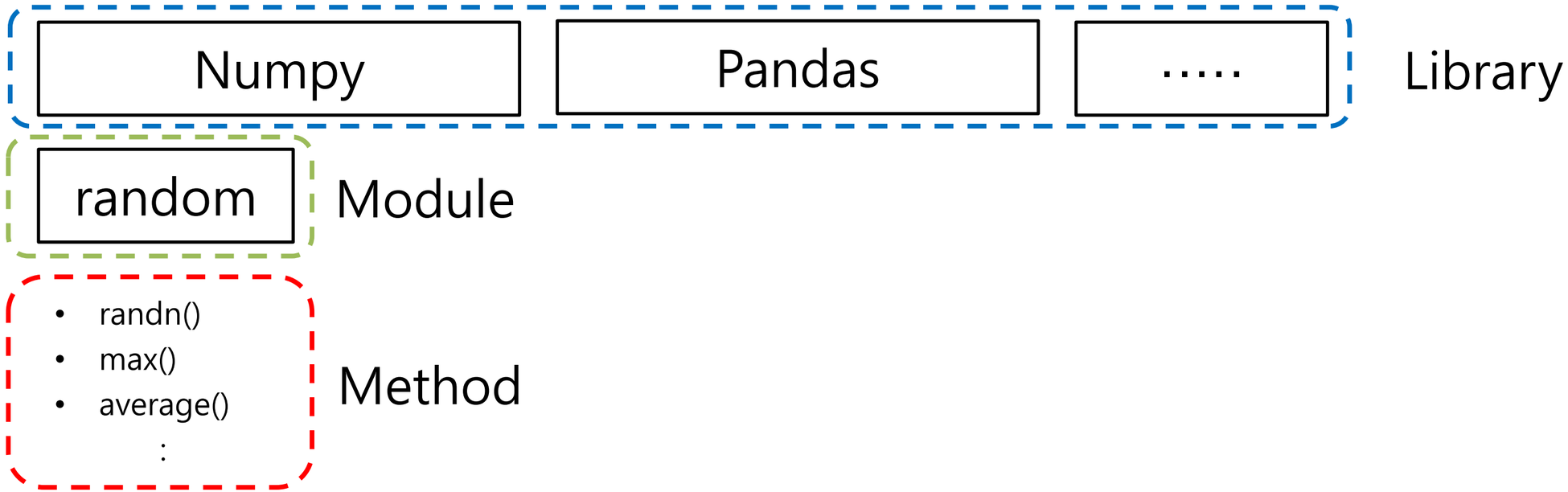
Week 3 Outline

- Week 2 Review
- Numpy Array Object
- Numpy ndarray
- Array Data types
- Structured Array
- Structured Datatypes
- Struct Array Assignment
- Struct Array Indexing and Slicing
- Conclusion

Week 2 Review

- 빅데이터를 위한 파이썬 개요

  → 파이썬/R 비교 및 함수, 메소드등 객체지향 언어 내용



Week 2 Review

■ 함수/메소드 실제 코드 실습

→ 함수와 메소드의 차이점 및 라이브러리부터 메소트까지 구조 학습

Difference between Python Methods vs Functions

METHODS	FUNCTIONS
Methods definitions are always present inside a class.	We don't need a class to define a function.
Methods are associated with the objects of the class they belong to.	Functions are not associated with any object.
A method is called 'on' an object. We cannot invoke it just by its name	We can invoke a function just by its name.
Methods can operate on the data of the object they associate with	Functions operate on the data you pass to them as arguments.
Methods are dependent on the class they belong to.	Functions are independent entities in a program.
A method requires to have 'self' as its first argument.	Functions do not require any 'self' argument. They can have zero or more arguments.

Week 3 Outline

- Week 2 Review
- Numpy Array Object
- Numpy ndarray
- Array Data types
- Structured Array
- Structured Datatypes
- Struct Array Assignment
- Struct Array Indexing and Slicing
- Conclusion

What is Numpy?

■ Numpy

→ 2006년 Travis E. Oliphant가 개발

→ 빠른 배열과 매트릭스의 처리

→ Pandas에서 데이터가 효율적으로 처리되는 역할 담당

→ 이미지 처리, 신호 처리, 선형 대수 등 어플리케이션에서 사용

■ 주요 기능

→ 다차원 배열 객체의 처리 & 배열 브로드캐스팅 기능

→ C/C++ 및 Fortran 코드를 통합하는 도구 & 수학적 연산

→ 빠른 이미지 및 컴퓨터 그래픽 처리

Numpy Array Object

■ Importing

```
import numpy as np
```

■ How to create array

→ 다른 파이썬 구조로부터 변환 : lists, tuples

→ 넘파이 배열 생성 함수 : arrange, ones, zeros, linspace

→ 저장 디스크로부터 배열을 읽어들임

→ Strings나 버퍼를 통한 raw bytes로부터 배열 생성

→ 특수한 라이브러리 함수 : random

Numpy Array Object

■ 1-dimentional array

```
import numpy as np
arr1 = np.array([0, 2, 5.5, 7])
print(arr1)
```

■ 2-dimentional array

```
arr2 = np.array([[1,2,3],[4,5,6]])
print(arr2)
```

Numpy Array Object

■ Zeros 1/2-dimentional array

```
arr = np.zeros(5)
type(arr)
arr.shape
arr.dtype
```

```
arr = np.zeros(shape=(6,2))
print(arr)
for i in range(6):
    arr[i] = (i,i)
Print(arr)
```

■ Array Initialization

zeros()
ones()
empty()

Numpy Array Object

■ empty array

```
arr = np.empty((2,2))
print(arr)
```

```
arr = np.empty((2,2), dtype=int)
print(arr)
```

■ eye array

```
arr = np.eye(3, dtype=int)
print(arr)
```

```
arr = np.empty(3, k=1)  # TypeError: empty() got an unexpected
print(arr)                keyword argument 'k'
```

Numpy Array Object

- **linspace array**

```
arr = np.linspace(2.0,3.0,nums=5)
print(arr)
```

```
arr = np.linspace(2.0,3.0,nums=5, endpoint=False)
print(arr)
```

Numpy Array Object

■ reshape

```
arr = np.zeros((2,3))
print(arr)
```

```
arr = arr.reshape(-1)
print(arr)
```

```
arr = np.zeros((2,3))
print(arr)
```

```
arr = arr.ravel(-1)
print(arr)
```

■ reshape with -1

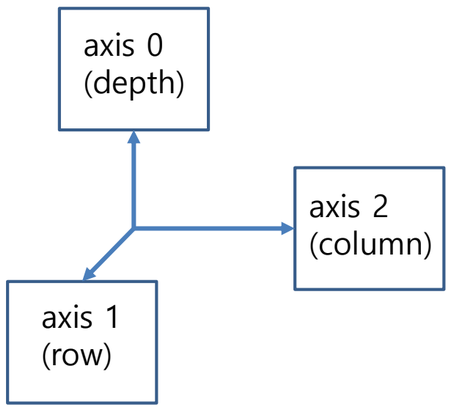
```
arr = arr.reshape(-1,1)
print(arr)
```

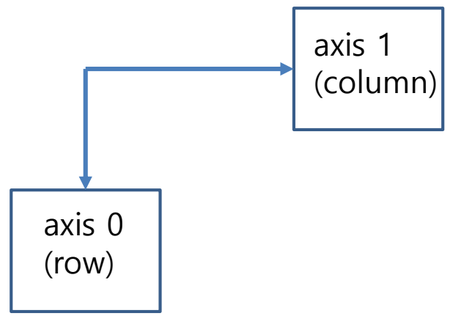
```
arr = arr.reshape(3,-1)
print(arr)
```

Numpy Array Object

- **reshape 3-dimentional array**

```
arr = np.arange(24).reshape(2,3,4)
print(arr)
```





Numpy Array Object

■ **reshape 3-dimentional array**

```
arr = np.arange(24).reshape(2,3,4)
print(arr)
```

■ **Read data format and create array**

→ HDF5와 FITS의 표준 바이너리 포맷

→ CSV파일의 ASCII 포맷

→ 기존의 바이너리 포맷

→ 기타 라이브러리

Week 3 Outline

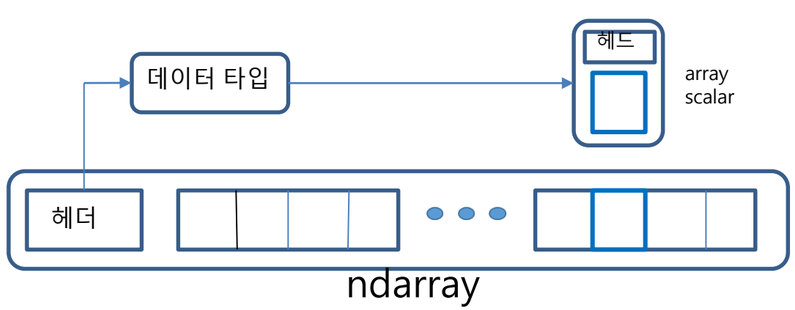
- Week 2 Review
- Numpy Array Object
- Numpy ndarray
- Array Data types
- Structured Array
- Structured Datatypes
- Struct Array Assignment
- Struct Array Indexing and Slicing
- Conclusion

Numpy ndarray

■ ndarray

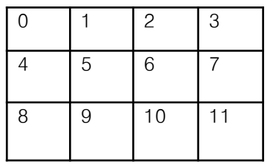
```
arr = array([[0, 1, 2], [3, 4, 5]])
print(type(arr))
print(arr.shape)
print(arr.dtype)
```

■ ndarray structure

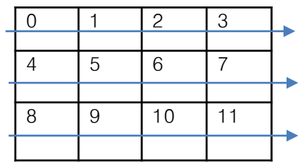


Numpy ndarray

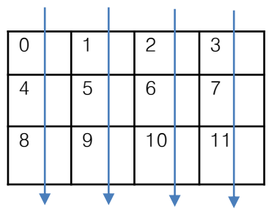
■ ndarray save direction



3x4 ndarray

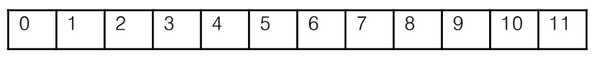


C 우선

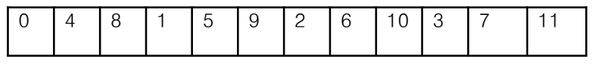


F 우선

C 우선



F 우선



Numpy ndarray

■ 3-dimentaional array

```
arr = np.array([[[0,  1,  2,  3],
                 [4,  5,  6,  7],
                 [8,  9, 10, 11]],
                [[12, 13, 14, 15],
                 [16, 17, 18, 19],
                 [20, 21, 22, 23]]])
print(arr.ndim)
print(arr.shape)
print(arr.size)
print(arr.dtype)
print(arr.itemsize)
print(arr.strides)
```

Week 3 Outline

- Week 2 Review
- Numpy Array Object
- Numpy ndarray
- Array Data types
- Structured Array
- Structured Datatypes
- Struct Array Assignment
- Struct Array Indexing and Slicing
- Conclusion

Array Data types

■ numpy.dtype의 적용

→ 데이터 타입 np.int16

```
import numpy as np
np.dtype(np.int16)
```

→ np.int16를 포함하고 필드이름 ‘f1’의 구조화된 타입

```
np.dtype([(‘f1’, np.int16)])
```

→ 1개의 필드가 있는 구조화 타입을 포함하고 필드이름이 ‘f1’의 구조화 타입

```
np.dtype([(‘f1’, [(‘f1’, np.int16)])])
```

Array Data types

■ numpy.dtype의 적용

→ 첫 번째 필드 unsigned int, 두 번째 필드는 int32의 구조화 타입

np.dtype([('f1', np.uint), ('f2', np.int32)])

→ Array-protocol type strings를 사용

np.dtype([('a', 'f8'), ('b', 'S10')])

→ Shape이 (2, 3)이고 comma-separated 포맷을 사용

np.dtype('i4, (2, 3)f8')

→ Int는 고정형, 3은 필드의 shape, void는 크기가 10인 가변형의 튜플을 사용

np.dtype([('hello', (int, 3)), ('world', np.void, 10)])

Array Data types

■ numpy.dtype의 적용

→ Int16을 x와 y의 2개의 int8로 분할하고 0과 1은 바이트로 된 옵셋

np.dtype((np.int16, {‘x’: (np.int8, 0), ‘y’: (np.int8, 1)}))

→ 2개의 필드가 ‘gender’와 ‘age’인 딕셔너리를 사용

np.dtype({‘names’:[‘gender’,‘age’],‘formats’:[‘S1’,np.uint8]})

→ 옵셋 0과 25바이트

np.dtype({‘surname’:(‘S25’,0), ‘age’:(np.uint8, 25)})

Array Data types

■ 데이터 타입 객체

→ 데이터 타입(정수, 실수, 파이썬 객체 등) 및 크기

- 데이터의 바이트 순서(little-endian 또는 big-endian)
- 데이터가 구조화되면 배열 요소의 데이터 타입에 대한 다음 사항을 기술
  - 구조화된 데이터의 필드의— 이름
  - 각각의 필드의 데이터 타입
  - 각각의 필드가 취하는 메모리 블럭의 어느 부분
- 데이터 타입이 sub-array인 경우에 그것의 shape와 데이터 타입

■ numpy 배열 스칼라 타입

넘파이의 배열 스칼라 타입	파이썬의 관련된 타입	비 고
Int_	IntType (파이썬2)	
float_	FloatType	
complex_	ComplexType	
bytes_	BytesType	
unicode_	UnicodeType	

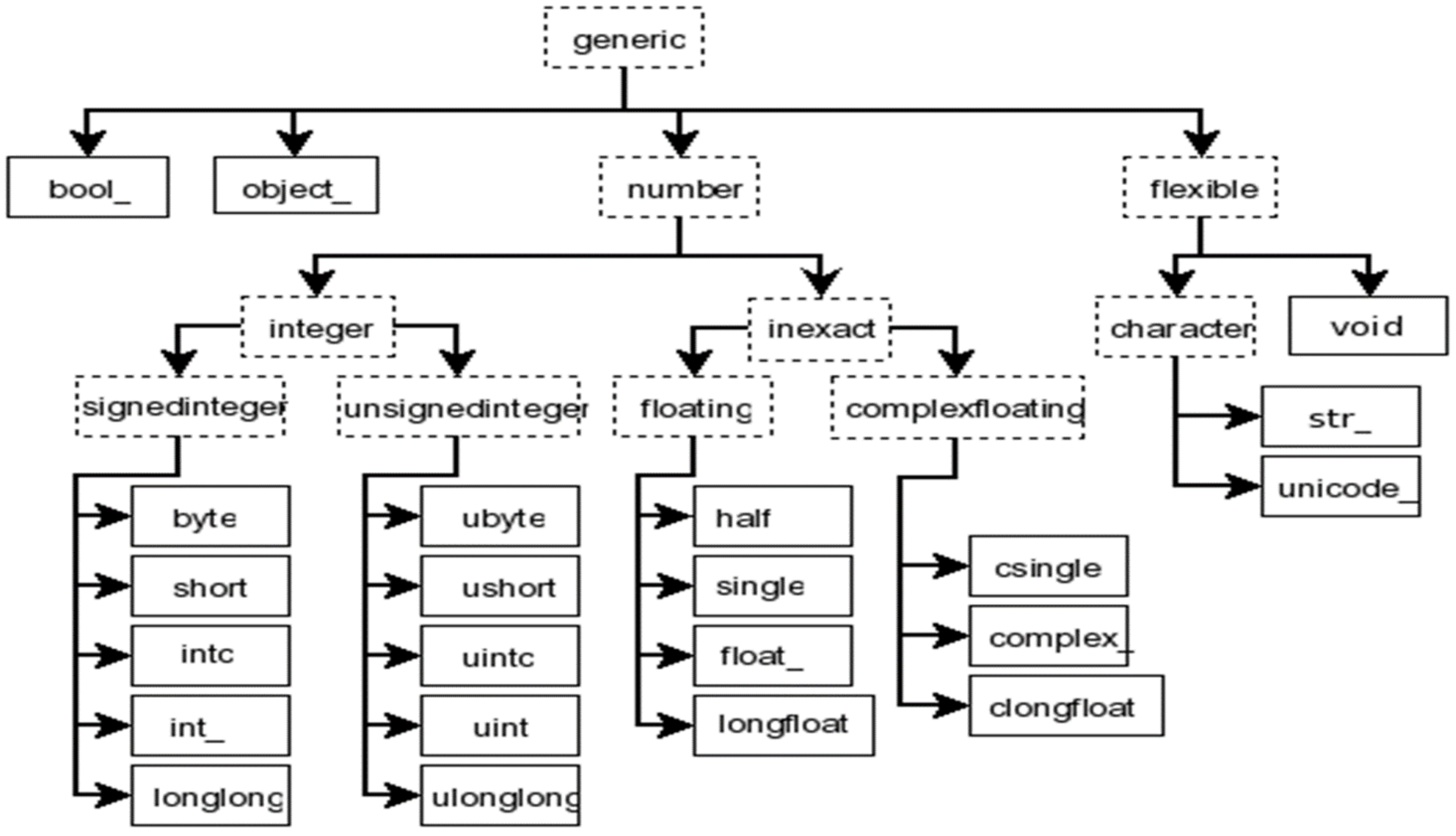
Array Data types

■ 넘파이 배열 스칼라 타입의 연산

```
a = np.float32(5.0)
type(a)
b = np.int_([1, 2, 3])
type(b)
c = np.arange(5, dtype=np.uint16)
type(c)
```

Array Data types

■ **배열 데이터 타입 객체의 분류 체계**



Array Data types

- **big-endian의 32비트의 정수를 가지는 데이터 타입**

```
dt = np.dtype(‘>i4’)
dt.byteorder
dt.name
dt.itemsize
dt.type is np.int32
```

- **16문자 string을 포함하는 구조화 데이터 타입과 2개의 64비트 sub_array**

```
dt = np.dtype([(‘name’, np.unicode_, 16), (‘grades’, np.float64, (2,))])
dt[‘name’]
dt[‘grades’]
```

Week 3 Outline

- Week 2 Review
- Numpy Array Object
- Numpy ndarray
- Array Data types
- Structured Array
- Structured Datatypes
- Struct Array Assignment
- Struct Array Indexing and Slicing
- Conclusion

Structured Array

■ 구조화배열 ndarray의 데이터 타입

→ 필드라 불리는 시퀀스로 조직화된 단순한 데이터 타입들을 모아 놓은 것

→ 필드는 구조화된 데이터 타입에서 각각의 서브 타입

→ 필드는 이름(string), 타입(dtype) 및 제목(title)을 의미

```
arr = np.array([('jin', 25, 67), ('suho', 18, 77)],
dtype=[('name', 'U10'), ('age', 'i4'), ('weight', 'f4')])
```

■ 구조화 배열 데이터 필드로의 접근 및 변경

```
arr[1]
arr['age'] = 20
arr
```

Week 3 Outline

- Week 2 Review
- Numpy Array Object
- Numpy ndarray
- Array Data types
- Structured Array
- Structured Datatypes
- Struct Array Assignment
- Struct Array Indexing and Slicing
- Conclusion

Structured Datatypes

■ 구조화된 데이터 타입의 생성

1) 1개의 필드에서 1개의 tuple로 구성하는 tuples의 list

→ tuple은 (필드이름, 데이터 타입, shape) 구조를 가지며 shape은 옵션

np.dtype([(‘a’, ‘f4’), (‘b’, np.float32), (‘c’, ‘f4’, (2,2))])

→ 필드이름이 빈 strin인 ‘’이면 f#의 형태를 가지고 #은 필드의 정수 인덱스

np.dtype([(‘a’, ‘f4’), (‘’, ‘i4’), (‘c’, ‘i8’)])

Structured Datatypes

■ 구조화된 데이터 타입의 생성

2) 컴마로 구분하는 dtype string

→ 필드의 바이트 옵셋과 itemsize는 자동으로 결정되고 필드이름은 f0, f1 등으로 주어진다

np.dtype('i8, f4, S3')
np.dtype('3int8, float32, (2,3)float64')

3) 필드 매개변수 배열의 딕셔너리

np.dtype({'names': ['col1', 'col2'], 'formats': ['i4', 'f4']})
np.dtype({'names': ['col1', 'col2'], 'formats': ['i4', 'f4'],
'offsets': [0, 4], 'itemsize': 12})

Structured Datatypes

■ 구조화된 데이터 타입의 생성

4) 필드 이름이 딕셔너리

→ 키는 필드이름이고 값은 타입과 옵셋을 기술하는 튜플

np.dtype({‘col1’: (‘i1’,0), ‘col2’: (‘f4’,1)})

■ 구조화된 데이터 타입의 조작 및 표시

→ 필드이름의 list는 dtype객체의 names속성에서 구한다

c = np.dtype([(‘a’, ‘i8’), (‘b’, ‘f4’)])
c.names

→ Dtype의 속성 fields의 키는 필드이름. 값은 dtype과 바이트 옵셋을 포함한 tuples

c.fields

Structured Datatypes

■ 자동 바이트 옵셋과 정렬(alignment)

```
def print_offsets(d):
        print("offsets:", [d.fields[name][1] for name in d.names])
        print("itemsize:", d.itemsize)
print_offsets(np.dtype('u1,u1,i4,u1,i8,u2'))
d=np.dtype('u1,u1,i4,u1,i8,u2')
print(d)
print(d.itemsize)
print(d.fields)
print(d.names)
print(d.fields['f0'])
print(d.fields['f0'][1])
print_offsets(np.dtype('u1,u1,i4,u1,i8,u2', align=True))
```

Structured Datatypes

■ 필드 제목(Field Titles)

```
x = np.dtype([((‘my title’, ‘name’), ‘f4’)])
print(x.fields)
print(x.names)
```

```
np.dtype({‘name’: (‘i4’, 0, ‘my title’)})
```

Week 3 Outline

- Week 2 Review
- Numpy Array Object
- Numpy ndarray
- Array Data types
- Structured Array
- Structured Datatypes
- Struct Array Assignment
- Struct Array Indexing and Slicing
- Conclusion

Struct Array Assignment

■ 파이썬 고유의 타입인 튜플로 할당

→ 구조화된 배열에 파이썬 튜플을 이용하여 값을 할당

→ 할당된 값은 배열에서 필드의 수와 같은 길이의 튜플이어야 함

```
a = np.array([(1, 2, 3), (4, 5, 6)], dtype='i8, f4, f8')
a[1] = (7, 8, 9)
print(a)
```

■ 스칼라 값으로 할당

```
a = np.zeros(2, dtype='i8, f4, ?, S1')
print(a)
a[:] = 7
print(a)
print(a[:] = np.arange(2)
print(a)
```

Struct Array Assignment

■ 다른 구조화된 배열로부터 할당

```
a = np.zeros(3, dtype=[('a', 'i8'), ('b', 'f4'), ('c', 'S3')])
b = np.ones(3, dtype=[('x', 'f4'), ('y', 'S3'), ('z', 'O')])
b[:] = a
b
```

Week 3 Outline

- Week 2 Review
- Numpy Array Object
- Numpy ndarray
- Array Data types
- Structured Array
- Structured Datatypes
- Struct Array Assignment
- Struct Array Indexing and Slicing
- Conclusion

Struct Array Indexing and Slicing

■ 기본 Syntax

→ 기본 자르기 신택스 - i:j:k, i는 시작 index, j-1은 종료 index, k는 증분

신택스 유형	설 명
arr[i :]	배열의 색인 i부터 끝까지의 요소 선택
arr[i:j:k]	i : 시작 색인, j −1 : 종료 색인, k : 증분(옵션) 시작하는 색인 i부터 증분 k를 더하여 색인 j-1까지의 요소를 선택
arr[:], arr[::]	배열의 모든 요소들을 선택
arr[::2]	배열의 첫번째 색인 0부터 끝 요소까지 선택하되 증분 2씩 더해서 마지막 요소 범위까지 선택
arr[−1]	배열이 마지막 요소
arr[:−1]	:를 중심으로 왼쪽은 공란이므로 첫번째 요소이고 오른쪽은 -1이므로 j-1이 되어 -2가 되므로 첫 번째부터 끝에서 두 번째까지의 요소를 선택
arr[−2:]	배열의 끝에서 두 번째부터 배열의 마지막 요소까지 선택
arr[:−2]	배열의 처음부터 끝에서 세 번째까지의 요소를 선택
arr[::−1]	배열의 모든 요소들을 마지막 요소부터 첫 번째 요소까지 역순으로 선택
arr[1::−1]	색인 1부터 마지막 요소를 선택하되 증분 -1을 더하므로 역순이 되므로 두 번째 요소와 첫 번째 요소를 차례로 선택
arr[−3::−1]	배열의 끝에서 3번째 요소부터 마지막 요소의 범위를 선택하되 증분이 -1이므로 역순으로 선택

Struct Array Indexing and Slicing

■ 배열 객체 요소 선택

```
arr1 = np.arange(10)
arr1[1]
arr2 = np.arange(9).reshape(3,3)
arr2
arr2[2]
arr2[2, 1]
arr3 = np.reshape(np.arange(24), (2,3,4))
arr3
arr3[1,1,2]
arr3[1][2][3]
```

Struct Array Indexing and Slicing

■ 기본 자르기(Basic Slicing)와 색인처리(indexing)

arr1[1]
arr1[:6]
arr1[0:5]
arr1[::2]
arr1[1::2]
arr1[1:7:2]

arr1[-3:9]
arr1[:-3]
arr1[-3:2:-1]

Struct Array Indexing and Slicing

■ 기본 자르기(Basic Slicing)와 색인처리(indexing)

arr1[5:2]

arr1[5:]
arr2[1:]

arr2
arr2[:2, :2]
arr2[:, ::-1]

arr2[:,:]
arr2[1, :]
arr2[1,2]

arr3[:2, 1:, :2]

Struct Array Indexing and Slicing

■ 고급 색인처리(Advanced Indexing)

→ 고급 색인처리는 copy를 반환하고 기본색인처리는 view를 반환

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

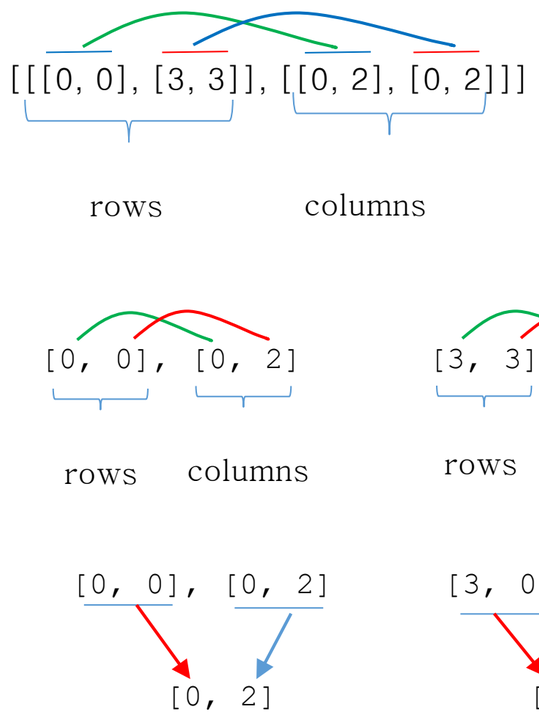
→ 4x3배열 객체에서 고급 색인처리의 사용

```
arr = np.array([[0,  1,  2], [3,  4,  5], [6,  7,  8], [9,  10,  11]])
rows = np.array([[0, 0], [3, 3]], dtype=np.intp)
columns = np.array([[0, 2], [0, 2]], dtype=np.intp)
arr[rows, columns]
```

Struct Array Indexing and Slicing

■ 고급 색인처리(Advanced Indexing)

→ 4x3배열 객체에서 고급 색인처리의 매핑 처리



Struct Array Indexing and Slicing

■ 고급 색인처리(Advanced Indexing)

→ 상수 numpy.newaxis의 사용

```
np.newaxis is None
arr = np.arange(25).reshape(5, 5)
arr.shape
arr_3d = arr[np.newaxis]
arr_3d.shape

arr = np.arange(10)
arr.shape
arr_row = arr[np.newaxis, :]
arr_row.shape
arr_col = arr[:, np.newaxis]
arr_col.shape
```

Struct Array Indexing and Slicing

■ 고급 색인처리(Advanced Indexing)

→ 상수 numpy.newaxis의 사용과 브로드캐스팅에 의한 절차의 단순화

```
arr = np.array([[0,   1,   2],
                [3,   4,   5],
                [6,   7,   8],
                [9,  10,  11]])
rows = np.array([0, 3], dtype=np.intp)
columns = np.array([0, 2], dtype=np.intp)
rows[:, np.newaxis]
arr[rows[:, np.newaxis], columns]
```

Struct Array Indexing and Slicing

■ 고급 색인처리(Advanced Indexing)

→ 상수 numpy.newaxis의 사용으로 차원 증가시켜 호환성 확보 후 브로드캐스팅

```
arr1= np.array([1, 2, 3, 4, 5])
arr2 = np.array([11, 12, 13])
arr1 + arr2
arr1_nx = arr1[:, np.newaxis]
arr2_nx = arr2[:, np.newaxis]
arr1_nx
arr2_nx
arr1_nx + arr2
arr2_nx + arr1
```

Struct Array Indexing and Slicing

■ 고급 색인처리(Advanced Indexing)

→ np.ix의 사용 – 입력 시퀀스가 Boolean인 경우

```
ix_ms = np.ix_([True, True], [2, 4])
arr[ix_mx]
ix_ms = np.ix_([True, True], [False, False, True, False, True])
arr[ix_mx]
```

→ 팬시 색인 – 정수 배열을 사용하여 색인 처리

```
arr = np.arange(1, 7).reshape(3, 2)
arr
arr[[0, 1, 2], [0, 1, 0]]
```

Struct Array Indexing and Slicing

■ Boolean 배열로 index 처리

```
arr = np.arange(12).reshape(3,4)
arr
ind = arr>5
ind
arr[ind]
```

→ 2차원 arr배열의 색인 배열로 처리된 결과는 2차원 배열

```
ind[:, 1]
arr[ind[:, 1]]
```

Struct Array Indexing and Slicing

■ 자르기와 색인배열 처리

→ 색인배열은 자르기와 함께 사용 가능

```
arr = np.arange(12).reshape(3,4)
arr[np.array([1, 2]), 1:3]
```

→ 브로드캐스팅 된 Boolean Index은 자르기와 함께 사용 가능

```
ind = arr > 5
ind
ind[:, 2]
arr[ind[:, 2], 1:3]
```

Struct Array Indexing and Slicing

■ 색인 반환의 numpy.nonzero와 numpy.transpose

```
arr = np.array([[1, 0, 0], [0, 2, 0], [1, 1, 0]])
arr
np.nonzero(arr)
```

→ 배열 요소가 0이 아닌 값들의 정렬 및 전치

```
arr[np.nonzero(arr)]
np.transpose(np.nonzero(arr))
```

Struct Array Indexing and Slicing

■ 색인 반환의 numpy.nonzero와 numpy.transpose

→ Nonzero함수를 불리언 조건으로 적용하여 True인 색인을 반환

```
arr = np.arange(9).reshape(3,3)
arr > 3
np.nonzero(arr > 3)
```

→ 불리언 배열의 nonzero메소드를 적용

```
(arr > 3).nonzero()
```

Struct Array Indexing and Slicing

■ 색인 반환의 numpy.nonzero와 numpy.transpose

→ 2, 3차원 배열의 전치

```
arr = np.arange(4).reshape((2, 2))
arr
np.transpose(arr)
arr1 = np.ones((1, 2, 3))
np.transpose(arr1, (1, 0, 2)).shape
```

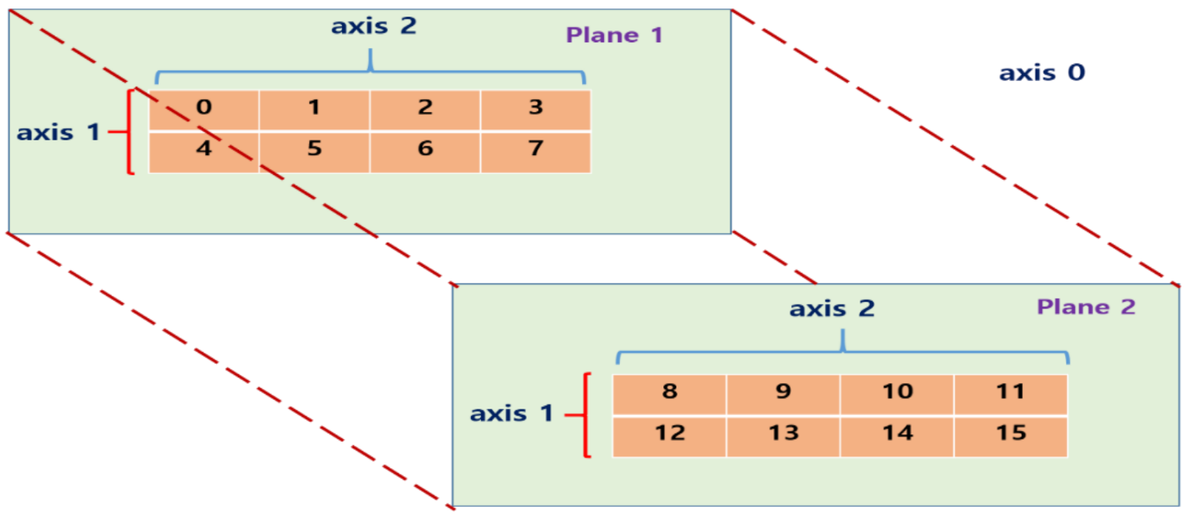
→ 1차원 배열은 전치처리가 되지 않는다?

```
a = [1, 2, 3]
b = np.array(a)
b
b.T
arr = np.array([a])
arr.transpose()
```

Struct Array Indexing and Slicing

■ 다차원의 넘파이 전치

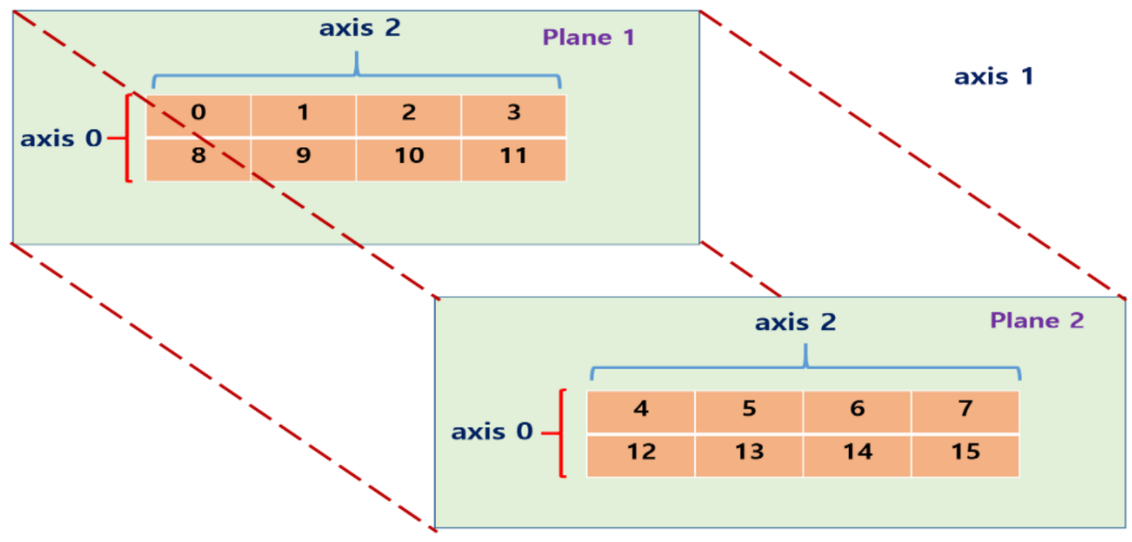
```
arr1 = np.array([[[0, 1,  2,  3], [ 4,  5,  6,  7]],
                 [[8, 9, 10, 11], [12, 13, 14, 15]]])
arr1
```



Struct Array Indexing and Slicing

■ 다차원의 넘파이 전치

```
arr2 = np.transpose(arr1, (1, 0, 2))
arr3 = arr1.transpose((1, 0, 2))
arr2
```



Struct Array Indexing and Slicing

■ 다차원의 넘파이 전치

→ arr1.T – 축 (0, 1, 2)가 축 (2, 1, 0)으로 치환

```
arr4 = arr1.T
arr4
```

Week 3 Outline

- Week 2 Review
- Numpy Array Object
- Numpy ndarray
- Array Data types
- Structured Array
- Structured Datatypes
- Struct Array Assignment
- Struct Array Indexing and Slicing
- Conclusion

다음 장에서는

- Another Numpy