Chapter. 02

파이썬의 주요 데이터 구조

리스트와 튜플

FAST CAMPUS
ONLINE
데이터 탐색과 전처리 I

강사. 안길승

I 개요

- 리스트(list)와 튜플(tuple) 모두 여러 데이터를 담는 **컨테이너형 변수**임
- 리스트의 정의: [Data 1, Data 2, …, Data n]
  - L1 = [1, 2, 3, 4, 5]
  - L2 = [‘a’, ‘b’, ‘c’, 1, 2]
  - L3 = [1, 2, [3, 4]]
- 튜플의 정의: (Data 1, Data 2, …, Data n)
  - T1 = (1, 2, 3, 4, 5)
  - T2 = (‘a’, ‘b’, ‘c’, 1, 2)
  - T3 = (1, 2, [3, 4])

I 공통점 1. 인덱싱과 슬라이싱

- 리스트와 튜플 모두 인덱싱과 슬라이싱이 가능함

- 인덱싱 방법
  - List[i], Tuple[i]: 앞에서 i번째 요소 (0부터 시작)
  - List[-i], Tuple[-i]: 뒤에서 i번째 요소 (-1부터 시작)

- 슬라이싱 방법: List[start:end:step], Tuple[start:end:step]
  - start 인덱스부터 end 인덱스까지 step으로 건너 뛴 부분 리스트 혹은 튜플을 반환
  - step은 default가 1로 입력하지 않아도 무방함, List[start:end]
  - start와 end도 입력하지 않아도 되나, 콜론(:)은 넣어야 함

I 공통점 1. 인덱싱과 슬라이싱

- L = [4, 5, 1, 2, 10, 6]

Index	0 (-6)	1 (-5)	2 (-4)	3 (-3)	4 (-2)	5 (-1)
Data	4	5	1	2	10	6

- L[2] = 1
- L[4] = 10
- L[-1] = 6
- L[0:3] = [4, 5, 1]
- L[:3] = [4, 5, 1]
- L[2:] = [1, 2, 10, 6]
- L[0:4:2] = [4, 1]

I 공통점 2. 순회 가능 (iterable)

- 리스트와 튜플 모두 for문을 이용하여 순회를 할 수 있음

```
for data in List:
for data in Tuple:
```

- 따라서 max, min 등의 순회 가능한 요소를 입력 받는 함수의 입력으로 사용할 수 있음

I 차이점 1. 가변과 불변

- 리스트의 요소는 바꿀 수 있으나, 튜플의 요소는 바꿀 수 없음

리스트의 요소 변경

```
L = [1, 2, 3, 4, 5]
L[0] = 10
L # [10, 2, 3, 4, 5]
```

튜플의 요소 변경 (Type error 발생)

```
T = (1, 2, 3, 4, 5)
T[0] = 10 # Error
```

- 따라서 리스트는 사전의 key로 사용할 수 없지만, 튜플은 사전의 key로 사용 가능함
  - Tip 1. 불변의 자료형(int, float, str 등)만 사전의 key로 사용할 수 있다
  - Tip 2. 조건 등을 입력으로, 해당 조건에 대응되는 값들을 출력으로 하는 사전 구축은 의외로 많은 데이터 전처리에서 사용한다.

I 차이점 2. 순회 속도

- 순회 속도는 리스트보다 **튜플이 약간 더 빠름**
- 따라서 요소를 변경할 필요가 없고, 요소에 대한 연산 결과만 필요한 경우에는 리스트보다 튜플이 적합함
- 데이터가 큰 경우에 한해서, 리스트로 작업 후, 튜플로 자료형을 바꾼 후 순회를 함
  - 리스트 → 튜플: tuple(리스트)
  - 튜플 → 리스트: list(튜플)

```python
# 리스트 순회 속도 측정
import time
start_time = time.time()
for val in large_L:
    pass
end_time = time.time()
print(end_time - start_time)
```

```
2.916229248046875
```

```python
# 튜플 순회 속도 측정
start_time = time.time()
for val in large_T:
    pass

end_time = time.time()
print(end_time - start_time)
```

```
2.6150364875793457
```

I 리스트 관련 함수: 요소 추가

- List.append(x): 새로운 요소 x를 맨 뒤에 추가

Index	0	1	2	3	4
Data	4	5	1	2	10

append(15) →

Index	0	1	2	3	4	5
Data	4	5	1	2	10	15

- List.insert(a, x): 새로운 요소 x를 a 위치에 추가 (기존에 있던 요소는 뒤로 한 칸씩 밀림)

Index	0	1	2	3	4
Data	4	5	1	2	10

insert(1, 7) →

Index	0	1	2	3	4	5
Data	4	7	5	1	2	10

I 리스트 관련 함수: 요소 제거

- List.remove(x): 기존 요소 x를 제거 (단, x가 여러 개면 맨 앞 하나만 지워지고, 없으면 오류 발생)

Index	0	1	2	3	4
Data	1	3	2	1	4

remove(1) →

Index	0	1	2	3
Data	3	2	1	4

- List.pop(): 맨 마지막 요소를 출력하면서 그 요소를 삭제 (stack 구조)

Index	0	1	2	3	4
Data	4	5	1	2	10

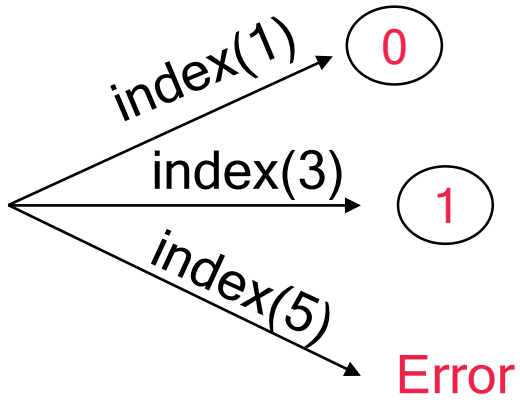
pop() → 10

Index	0	1	2	3
Data	4	5	1	2

I 리스트 관련 함수: 위치 찾기

- List.index(x): x의 위치를 반환 (단, x가 여러 개면 맨 앞 인덱스를 반환하고, 없으면 오류 발생)

Index	0	1	2	3	4
Data	1	3	2	1	4



I 리스트 관련 함수: 확장하기

- List1 + List2: 두 리스트를 그대로 이어 붙임 (튜플도 가능)

Index	0	1	2
Data	1	2	3

+

Index	0	1	2
Data	4	5	6

→

Index	0	1	2	3	4	5
Data	1	2	3	4	5	6

- List1.extend(List2): List1에 List2를 그대로 이어 붙임 (List1 +List2와 같음)

I 튜플 관련 함수

- 튜플은 **요소 변경이 불가능**하므로, 추가 및 제거 관련 함수를 지원하지 않음

- 튜플은 사실 소괄호를 쓰지 않아도 된다는 특징 덕분에 SWAP (값을 서로 변경), 함수의 가변인자 및 여러 개의 출력을 받는데 많이 사용함

Chapter. 02

파이썬의 주요 데이터 구조

사전

FAST CAMPUS
ONLINE
데이터 탐색과 전처리 I

강사. 안길승

I 개요

- 사전(dictionary)이란 키(key)와 값(value) 쌍으로 이루어진 해시 테이블 (hash table)임

Key	Value
key 1	value 1
key 2	value 2
key 3	value 3
key 4	value 4

{key 1: value 1, key 2: value 2, key 3: value 3, key 4: value 4}

- key는 불변의 값을 사용하며, value는 불변 혹은 가변에 상관없이 사용 가능함
- 사전은 다음과 같이 정의할 수 있음

```
dic = {key 1: value 1, key 2: value 2, key 3: value 3}
```

I 사전 요소에 접근하기 및 바꾸기

- 사전 요소에 접근하기: dict[key]
- 사전 요소 변경 및 추가: dict[key] = new value
- 사전 요소 삭제: del(dict[key])

Key	Value
1	1
2	4
3	9
4	16

dict

dict[1]

1

dict[5] = 20

Key	Value
1	1
2	4
3	9
4	16
5	20

dict[5] = 25

Key	Value
1	1
2	4
3	9
4	16
5	25

사전 관련 함수: 요소 확인하기

- 키 리스트 받기: .keys()
- 값 리스트 받기: .values()
- key, value 쌍 얻기: .items()
- 위 함수들은 주로 값을 **효율**적으로 **순회**하거나 **변경**할 때 주로 사용됨

Chapter. 02

파이썬의 주요 데이터 구조

반복문과 comprehension

FAST CAMPUS
ONLINE
데이터 탐색과 전처리 I

강사. 안길승

I 반복문 기초

- for문 기초 문법

```
for element in iterator:
    반복할 구문
```

순회 가능한 자료형(리스트, 튜플 등)의 요소를 순서대로 element에 저장하여 특정 구문을 반복함

- break: 현재 속한 반복문을 중지시키며, 보통 if문과 같이 사용

I 대표적인 이터레이터 객체 생성 함수

- 이터레이터 객체는 값을 차례대로 꺼낼 수 있는 객체를 의미 (리스트, 튜플)

- range, itertools 모듈에 있는 주요 함수 등을 통해서도 이터레이터를 생성할 수 있음
  - ➢ 단, 이러한 이터레이터는 리스트나 튜플 등이 아니라 순회만 가능한 객체임

I 대표적인 이터레이터 객체 생성 함수: range

- range(start, end, step)
  - start 인덱스부터 end 인덱스까지 step으로 건너 뛴 부분 이터레이터 객체를 반환
  - 값을 하나만 넣으면 end로 인식됨: range(end) = range(0, end, 1)
  - 값을 두 개를 넣으면 start와 end로 인식됨: range(start, end) = range(start, end, 1)

I 대표적인 이터레이터 객체 생성 함수: itertools 모듈 함수

- itertools 모듈은 다양한 종류의 이터레이터 객체를 생성하는 함수로 구성됨

- itertools.product(*L)
  - ➢ 순회 가능한 여러 개의 객체를 **순서대로 순회하는 이터레이터**를 생성

```
for v1, v2, v3 in itertools.product(L1, L2, L3):
```

```
for v1 in L1:
    for v2 in L2:
        for v3 in L3:
```

- itertools.combinations(p, r)
  - ➢ 이터레이터 객체 p에서 크기 r의 가능한 모든 **조합**을 갖는 이터레이터를 생성

- itertools.permutations(p, r)
  - ➢ 이터레이터 객체 p에서 크기 r의 가능한 모든**순열**을 갖는 이터레이터를 생성

I list comprehension

- list comprehension은 for문을 사용하여 한 줄로 리스트를 효과적으로 생성하는 방법임

```
[output for element in iterator if 조건]
```

- 예시

```
L = [x**2 for x in range(10) if x%2 == 0]
```



```
L = []
for x in range(10):
    if x%2 == 0:
        L.append(x**2)
```

- 조건문은 생략 가능함

dictionary comprehension

- dictionary comprehension은 for문을 사용하여 한 줄로 사전을 효과적으로 생성하는 방법임

```
{key: value for key, val in iterator if 조건}
```

- 예시

```
dic = {x:y**2 for x, y in zip(range(10), range(10)) if x%2 == 0}
```

↕

```
dic = dict()
for x, y in zip(range(10), range(10)) :
    if x%2 == 0:
        dic[x] = y ** 2
```

Chapter. 02

파이썬의 주요 데이터 구조

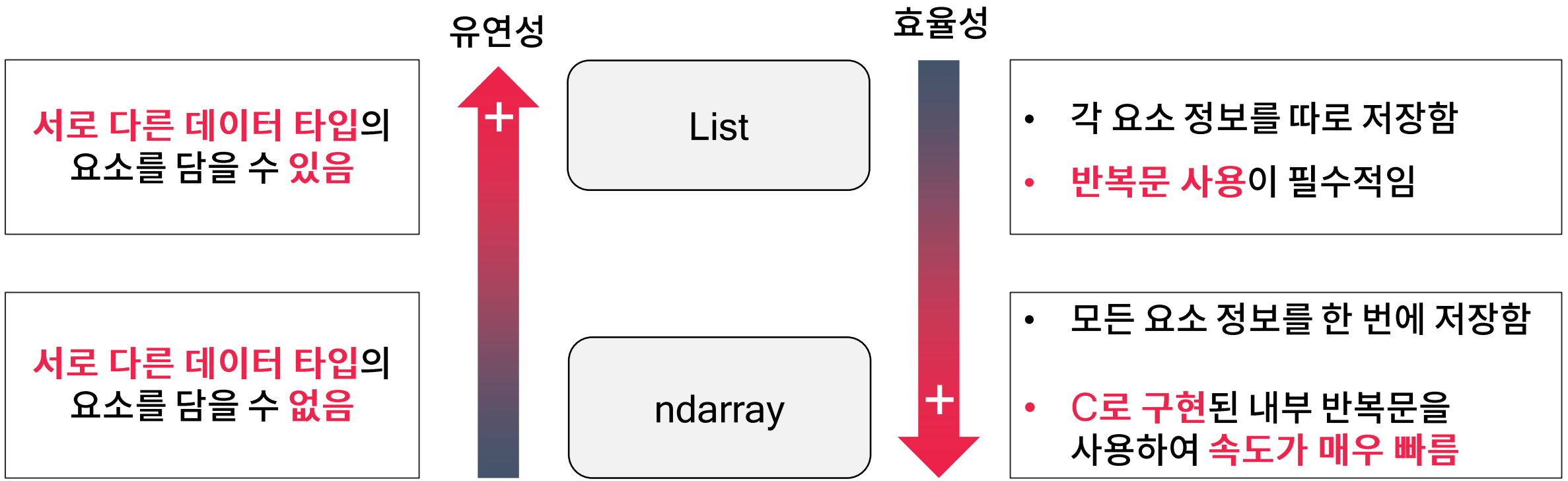
Numpy의 데이터 구조

FAST CAMPUS
ONLINE
데이터 탐색과 전처리 I

강사. 안길승

I 개요

- Numpy의 자료형은 ndarray로 효율적인 배열 연산을 하기 위해 개발되었음
- 리스트와 ndarray는 **유연성**과 **효율성**을 기준으로 비교할 수 있음



I 배열 만들기: np.array 함수

- np.array 함수를 사용하여 ndarray를 생성할 수 있음
  - 예시: np.array([1, 2, 3, 4])

배열 만들기: 다양한 함수

- 관련 함수를 사용해서 특정 패턴을 갖는 ndarray를 생성할 수 있음
  - ➢ np.zeros(shape)
    - shape (튜플) 모양을 갖는 영벡터/영행렬 생성
    - np.zeros((10, 2)): (10, 2) 크기의 영행렬 생성
  - ➢ np.arange(start, stop, step)
    - start부터 stop까지 step만큼 건너뛴 ndarray를 반환 (단, start와 step은 생략 가능)
    - np.arrange(1, 5, 0.1): ndarray([1, 1.1, 1.2, …, 4.9])
  - ➢ np.linspace(start, stop, num)
    - start부터 stop까지 num 개수의 요소를 가지는 등간격의 1차원 배열을 반환
    - np.linspace(0, 1, 9): ndarray([0., 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0])

I 인덱싱과 슬라이싱

- 기본적인 인덱싱과 슬라이싱은 리스트 자료형과 완전히 동일함
- 2차원 배열인 경우, X[i, j]는 i행 j열에 있는 요소를 나타냄 (c.f. X가 리스트라면, X[i][j]로 접근함)
- 부울 리스트도 인덱스로 사용할 수 있으며, True인 요소와 대응되는 요소만 가져옴

X	1	2	3	4	5
B	True	True	False	False	True

→ X[B] = [1, 2, 5]

- 여러 개의 인덱스를 리스트 형태로 입력받을 수도 있음

X	Index	0	1	2	3	4	5
	Data	1	2	3	4	5	6

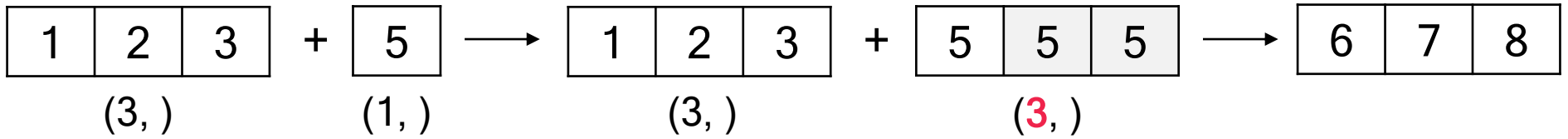
→ X[[0, 2, 3]] = [1, 3, 4]

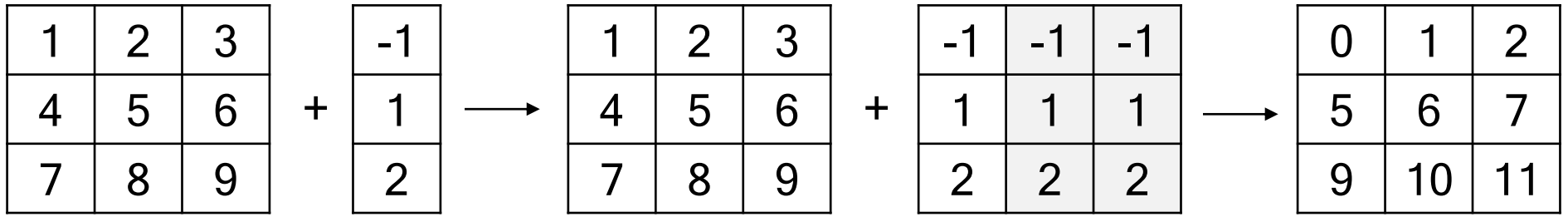
유니버설 함수

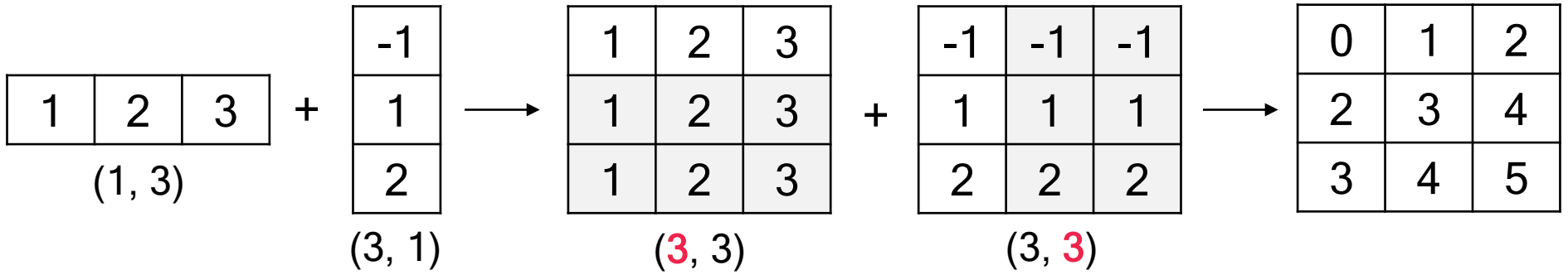
- 유니버설 함수는 ndarray의 개별 요소에 반복된 연산을 **빠르게** 수행하는 것을 주 목적으로 하는 함수
- ndarray x와 y에 대해, 덧셈, 뺄셈, 곱셈, 제곱 등 다양한 **배열 간 이항 연산**을 지원함
  - 벡터 간 덧셈: x + y = [x1 + y1, x2 + y2, ..., xn + yn]
  - 벡터 간 곱셈: x * y = [x1 * y1, x2 * y2, ..., xn * yn]
  - 벡터 간 제곱: x ** y = [x1 ** y1, x2 ** y2, ..., xn ** yn]
- 유니버설 함수는 단순 반복문에 비해, **매우 빠름**

I 브로드캐스팅

- **다른 크기**의 배열에 유니버설 함수를 적용하는 규칙 집합으로, 큰 차원의 배열에 맞게 작은 배열이 **확장**됨





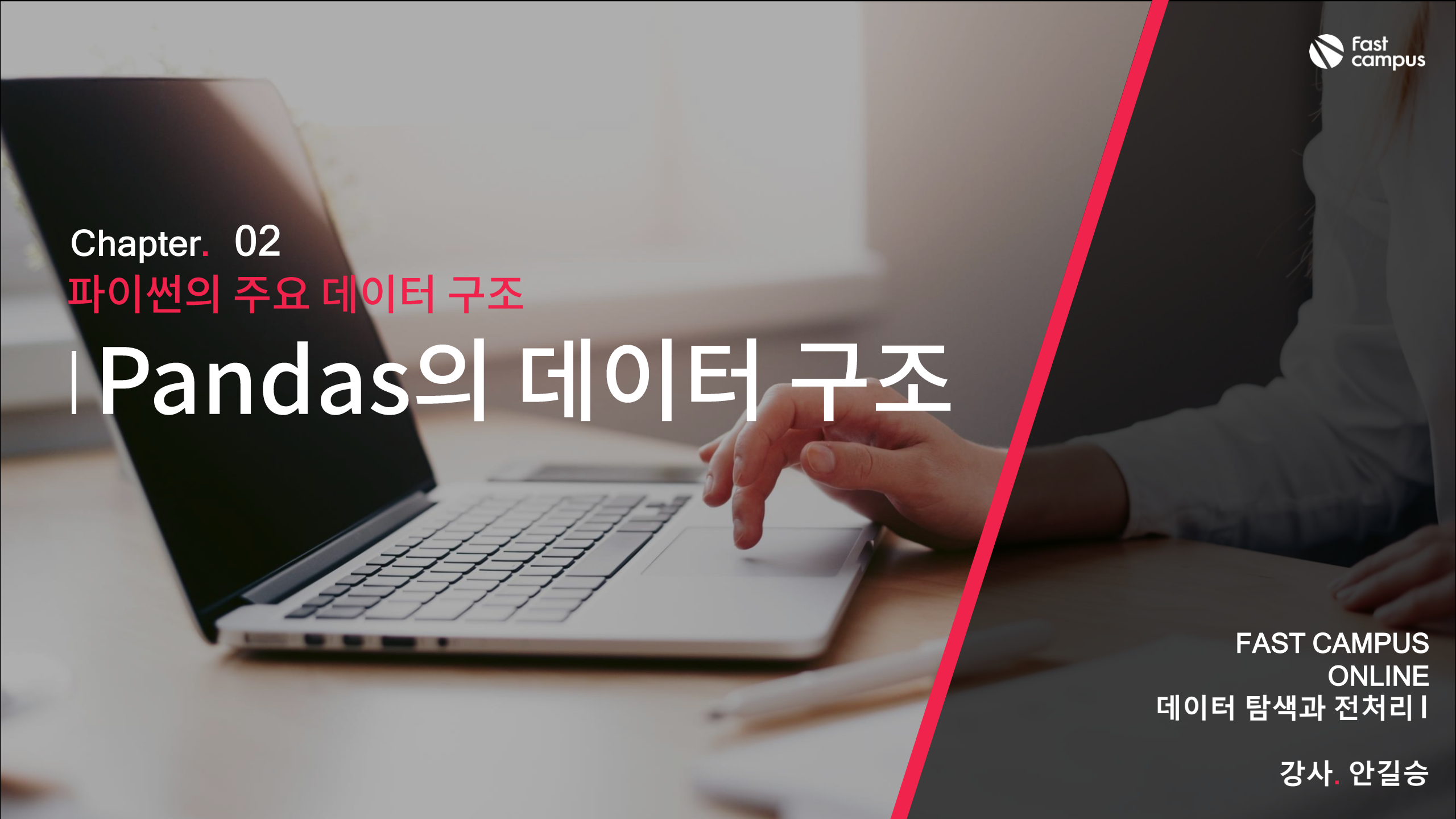


| 비교 연산자

- 비교 연산자의 결과는 항상 **부울 타입**의 배열임
  - [1, 2, 3, 4, 5] >= 3: [False, False, True, True, True]

- 따라서 비교 연산자의 결과를 바탕으로 **조건에 맞는 요소 탐색**에 활용할 수 있음

```
L = np.array([1, 2, 3, 4, 5])
cond = L >= 3
sum(cond) # 조건을 만족하는 요소의 개수
L[cond] # 조건을 만족하는 요소만 반환
```



Chapter. 02

파이썬의 주요 데이터 구조

Pandas의 데이터 구조

FAST CAMPUS
ONLINE
데이터 탐색과 전처리 I

강사. 안길승

I 자료형 1. Series

- Series는 1차원 배열 자료형으로 인덱스와 값의 쌍으로 구성

Index	Data
a	1
b	2
c	3
d	4

↔

pd.Series({'a': 1, 'b': 2, 'c': 3, 'd': 4}) 사전을 이용한 정의

pd.Series([1, 2, 3, 4], index = ['a', 'b', 'c', 'd']) 리스트를 이용한 정의

↓

ndarray

↓

- Series는 ndarray에 인덱스가 부여된 형태의 데이터
- Series에도 유니버설 함수와 브로드캐스팅 등이 적용됨

I 자료형 2. DataFrame

- DataFrame은 2차원 배열 자료형으로 값, 행 인덱스, 열 인덱스로 구성

Index	Col1	Col2
a	1	5
b	2	6
c	3	7
d	4	8

↔

```
pd.DataFame({'Col1': [1, 2, 3, 4], 'Col2': [5, 6, 7, 8]},
                index =  ['a', 'b', 'c', 'd'})
```
사전을 이용한 정의

```
pd.DataFame([[1, 5], [2, 6], [3, 7], [4, 8]],
                columns = ['Col1', 'Col2'],
                index =  ['a', 'b', 'c', 'd'})
```
데이터, 컬럼, 인덱스 따로 정의

↓

ndarray

↓

- DataFrame은 ndarray에 행과 열 인덱스가 부여된 형태의 데이터
- DataFrame은 하나 이상의 Series의 집합이라고도 볼 수 있음

인덱싱과 슬라이싱

- 판다스의 객체는 암묵적인 인덱스(위치 인덱스)와 명시적인 인덱스라는 두 종류의 인덱스가 있어, 명시적인 인덱스를 참조하는 loc 인덱서와 암묵적인 인덱스를 참조하는 iloc 인덱서가 존재함

암묵적 인덱스	명시적 인덱스	Data
0	a	1
1	b	2
2	c	3
3	d	4

S

→

S.loc['a'] = 1
S.iloc[2] = 3

S.loc['a':'c'] = [1, 2, 3] loc를 이용한 슬라이싱에서는 맨 뒤 값을 포함
S.iloc[1:3] = [2, 3] iloc를 이용한 슬라이싱에서는 맨 뒤 값을 포함 X

- 데이터 프레임의 컬럼 선택: df[col_name] or df[col_name_list]

값 조회하기 (1/2)

- 쥬피터 환경에서는 데이터 프레임 혹은 시리즈 자료를 갖는 변수를 출력할 수 있음

```
1 df
```

	Col1	Col2
a	1	5
b	2	6
c	3	7
d	4	8

```
1 s
```

```
a    1
b    2
c    3
d    4
dtype: int64
```

Tip 1. pd.set_option('display.max_rows', None)을 사용하여,모든 행을 보이게 할 수 있음
(None 자리에 숫자가 들어가면, 출력되는 행의 개수가 설정)

Tip 2. pd.set_option('display.max_columns', None)을 사용하여,모든 행을 보이게 할 수 있음
(None 자리에 숫자가 들어가면, 출력되는 열의 개수가 설정)

l 값 조회하기 (2/2)

- 데이터 크기 때문에, 아래 함수를 사용하여 데이터의 일부만 확인하거나 요약 정보를 확인하는 것이 바람직함

- DataFrame.head(n)
  - ➢ DataFrame의 맨 앞 n개 행을 보여줌 (default: 5)

- DataFrame.tail(n)
  - ➢ DataFrame의 맨 뒤 n개 행을 보여줌 (default: 5)

- DataFrame.columns
  - ➢ DataFrame을 구성하는 컬럼명 집합을 보여줌

- DataFrame.dtypes
  - ➢ DataFrame을 구성하는 컬럼별 데이터 타입을 보여줌

값 변경하기

- 인덱서를 사용하여 조회한 값을 직접 변경할 수 있음

Index	Col1	Col2
a	1	5
b	2	6
c	3	7
d	4	8

df

df.iloc[2, 1] = 10 →

Index	Col1	Col2
a	1	5
b	2	6
c	3	10
d	4	8

FAST CAMPUS
ONLINE
안길승 강사.