

# **Big Data Analytics Programming**

**Week-03. String, Collection**

**Jungwon Seo, 2021-Spring**

# 배울 내용

## Week-02. Python Basic

- 문자열
- 그룹형 데이터 타입

# Python Data Type - String

## 문자열

- 보통 한 글자일 경우 character, 그리고 2개 이상일 경우 string이라고 명명
- 하지만 Python에서는 개수에 상관없이 쌍따옴표("") 또는 따옴표(')로 감싸진 값은 모두 String
- 문자열의 특징
  - 대소문자가 다르게 인식된다: A != a
  - 숫자 1과 문자열 1은 다르다: 1 != '1'
  - + 연산자는 두 string을 연결하는 역할을 한다: 'he' + 'llo' = 'hello'
  - 문자열간의 \*, -, / 연산자는 지원되지 않는다.
  - 문자열 \* 숫자는 해당 문자를 숫자만큼 생성한다: 'a'\*5 = 'aaaaa'
  - 인덱스로 각각의 문자를 접근할 수 있다: 'hello'[3] = 'l'

0	1	2	3	4
H	E	L	L	O

# 영어를 배웠을 때 복수형을 배웠듯이..

s나 es를 붙인다.

y를 i로 바꾸고 es를 붙인다.

f나 fe를 v로 바꾸고 es를 붙인다

.....

# Python Data Type - Collection

## 묶음형 Data Type

- 단일 값만 저장 하는 것이 아니라, 값들을 저장
  - 복수 값 저장의 나쁜 예: `val1 = 1, val2 = 2, val3 = 3 .....`
  - 복수 값 저장의 좋은 예: `my_list = [1, 2, 3]`
- 목적에 따라서 다양한 묶음 형 데이터 타입을 사용
  - List: `[1,2,3,4,5,6]`
  - Dictionary: `{"name":"Jungwon", "score":100}`
  - Tuple: `(1,"b")`
  - Set : `{1,2,3}`
- 값들의 묶음이기 때문에, string은 list와 유사한 특징을 띠
  - `my_list = ['a','b','c']`
  - `my_str = 'abc'`

# Python Data Type - List

array? 배열? 인덱스?

- list를 표현 하는 기호는 대괄호(Brackets): [ ]
- 원소들을 구분하는 기호는 쉼표(comma): ,
  - [ ] : 원소가 없다
  - [ 1 ] : 원소가 1개
  - [ 1, 2 ] : 원소가 2개
  - [ 0 ] : 원소가 몇개?
- 원소는 **object** 또는 **function**이라면 가능
  - 단일 값: [1,2,3], ['a','abc','d']
  - 묶음 형: [ [1,2,3], [1,2,3] ] , [{"name":"손흥민"}, {"name":"류현진"}]
  - 함수 : [ function1, function2, function3 ]

# Python Data Type - List

## CRUD - Create, Read, Update, Delete

- Create

- `my_list = [ ]` 또는 `my_list = list( )`, 만약 초기값(initial value)를 제공한다면, `my_list = [1,2,3]`

- Read

- list 전체에 대한 접근은 list 명 그대로: `print(my_list)`, `temp_list = my_list`
- 원소에 대한 접근은: `my_list[index]`, 이때 index는 해당 원소의 위치 (0부터 시작, 정수)
- 예) `a = [1, 2, 3, 4]` 일 때 `a[3] = ?`

- Update

- list 전체를 변경하는 경우는 변수 재할당 : `my_list = ['a','b','c']`
- 원소에 대한 변경은: `my_list[index] = 'k'`
- 원소 추가는: `my_list.append('x')`
- Concatenation은 : `[1,2,3] + ['a','b','c'] = [1, 2, 3, 'a', 'b', 'c']`

- Delete

- list 전체에 대한 delete는: `del my_list`
- n번째 원소에 대한 delete는 : `del my_list[n]`

# Python Data Type - List

## Slicing

- Python의 List의 경우에는 Subset에 접근 가능
  - 예) `a = [1,2,3,4,5,6]` 일때 앞에 3개만 추출 하려는 경우?
- `list[start:end]`
  - start는 시작하려는 index, end는 끝나기 전의 index
  - `a = [1,2,3,4,5,6]` 일때, `a[0:3]`은 0,1,2 index에 대한 접근 => `[1,2,3]`
  - 만약 start 부분 또는 end 부분을 생략한다면, `list[ : ] == list [ 0 : len(list) ]`
- `list[-start:-end]`
  - 음수의 index를 넣는 경우, "뒤에서부터 n번째"의 의미
  - 예) `a[-1]`는 맨 뒤의 원소, `a[-3]`은 뒤에서 세번째 원소
  - 예) `a[1:-3]`은?





# Python Data Type - Dictionary

사전? Key-value? map?

- dictionary를 표현 하는 기호는 중괄호(brace): { }
- List가 index 기반으로 데이터를 Mapping 시킨다면, Dictionary는 key 기반으로 데이터를 Mapping 시킴
  - { key: value }
  - player = { "name": "손흥민", "team": "토트넘", "height": 183 }
  - my\_dict = { "a" : 3, 1: "c", (1,2,3): 4 }
- Key는 중복될 수 없지만, Value는 중복될 수 있음
  - { "a": 123, "a": 456 } => 🤔
  - { "a": 123, "b": 123 } => 😊
- Key와 value는 **object** 또는 **function**이라면 가능
  - 단 **mutable** 한 list, dictionary, set은 Key로 불가능

# Python Data Type - Dictionary

## CRUD - Create, Read, Update, Delete

- Create

- `my_dict = { }` 또는 `my_dict = dict( )`, 만약 초기값(initial key-value)를 제공한다면, `my_dict = { "a":1 }`

- Read

- dictionary 전체에 대한 접근은 dictionary 명 그대로: `print(my_dict)`, `temp_dict = my_dict`
- Value에 대한 접근은: `my_dict[key]`, 이때 key는 dictionary를 정의 할 때 사용한 Key

- Update

- dictionary 전체를 변경하는 경우는 변수 재할당 : `my_dict = { "a": 3 }`
- Value에 대한 변경은: `my_dict[key] = value`
- Key-Value 추가: `my_dict[new_key] = value`

- Delete

- dictionary 전체에 대한 delete는: `del my_dict`
- Key에 대한 delete는 : `del my_list[key]`
  - 이때, Mapping 된 Value도 같이 소멸

# Python Data Type - Dictionary

## Keys, Values, Items

- Dictionary의 전체 데이터에 대한 접근은 크게 세가지

- 전체 Key 출력

- `my_dict.keys()`
  - 출력: `dict_keys(['name', 'age', 'height', 'retired'])`

- 전체 Value 출력

- `my_dict.values()`
  - 출력: `dict_values(['손흥민', 29, 183, False])`

- 전체 Item 출력

- `my_dict.items()`
  - 출력: `dict_items([('name', '손흥민'), ('age', 29), ('height', 183), ('retired', False)])`

```
{  
    'age': 29,  
    'height': 183,  
    'name': '손흥민',  
    'retired': False  
}
```

# Python Errors - Part 2

## 에러의 종류

- Index Error

- list와 같은 index기반의 접근을 하는 데이터에 접근 가능한 범위 밖의 값을 접근하려는 경우
- a의 원소가 3개인데 100번째 원소는?

```
a = ["a", "b", "c"]  
a[100]
```

```
-----  
IndexError                                ]  
<ipython-input-1-268446cb9d24> in <module>  
      1 a = ["a", "b", "c"]  
----> 2 a[100]
```

```
IndexError: list index out of range
```

- Key Error

- Dictionary와 같이 Key기반으로 접근하는 데이터에 존재 하지 않는 Key로 접근하려는 경우
- key가 a,b,c 밖에 없을 때 d에 대한 값은?

```
a = {"a":1, "b":2, "c":3}  
a["d"]
```

```
-----  
KeyError                                ]  
<ipython-input-2-7f438077e339> in <module>  
      1 a = {"a":1, "b":2, "c":3}  
----> 2 a["d"]
```

```
KeyError: 'd'
```

# Python Data Type - Tuple

## Ordered, Immtuable

- Tuple을 표현하는 기호는 소괄호(Parentheses) : ( )
  - my\_tuple = (1,2,3)
  - Index 기반으로 접근 가능
- List와 비슷한 형태를 띄지만 몇가지 특징이 있음
  - Immutable 데이터 타입으로, 값의 변경이 불가
  - Immutable 데이터 타입이므로 Dictionary의 Key로 사용 가능
  - 함수에서 여러개의 값을 return 하는 경우에 사용됨
    - \* list, dict등은 여러 원소가 있는 것이지 return 되는 값은 1개

```
def hello():  
    return 1, 2  
  
print(type(hello()))  
  
<class 'tuple'>
```

# Python Data Type - Set

## 수학의 집합과 유사, Unordered and Unindexed

- Set을 표현하는 기호는 중괄호(brace) : { }
  - Dictionary와 같은 기호를 사용하지만, 원소가 key-value 형태로 저장되느냐에 의해 구별됨
- 몇가지 특징
  - Unordered: 순서가 보장되지 않음
  - Unindexed: index 기반으로 값이 접근 가능하지 않음
  - Unique Element: 원소가 중복되지 않음
- 자주쓰이는 사용처
  - List에서 Unique값 추출 시 (중복제거시)

```
my_list = [3,1,33,1,5,55]  
my_set = set(my_list)  
print(my_set)
```

```
{1, 33, 3, 5, 55}
```

**E.O.D**