셋(set) : 중복되지 않는 데이터를 만들기 위해 사용하는 자료구조

딕셔너리(dictionary) : 키 데이터와 관련된 데이터를 연결한 자료구조

* 파이썬에 제공하는 반복형 자료 구조
* 데이터의 중복을 허용하지 않음

**1. 셋 생성 및 조작법**

**셋 생성**

셋 : 중괄호{} 안에 서로 다른 자료형의 유일한 값을 콤마(,)로 구분해 하나 이상 저장할 수 있는 컬렉션 자료형 중의 하나

* 인덱스를 제공하지 않음
* 순서의 개념이 없음
* 중복을 허용하지 않음

data\_set = {10, 20, ‘파이썬’, ‘파이썬’}

print(data\_set) => {‘파이썬’, 10, 20} -> 중복을 허용하지 않으므로

data\_str = “Better Tomorrow”

data\_set = set(data\_str)

print(data\_str) => {‘ ‘, ‘o’, ‘w’, ‘T’, ‘m’, ‘t’, ‘e’, ‘r’, ‘B’} -> 정해진 순서 없이 출력, 중복 문자는 하나만 저장됨

**셋 조작법**

**셋 기본연산**

교집합(intersection) &, 합집합(union) |, 차집합(difference) –

**셋 항목 추가**

data\_set.add(2) -> 2가 추가 됨

data\_set.update({4,5,6}) -> 4, 5, 6이 추가 됨

**셋 항목 제거**

data\_set.remove(9)

data\_set.pop() -> 첫번째 항목 제거

data\_set.clear() -> 모든 객체 제거

* {}은 딕셔너리의 리터럴이기 때문에, 비어있는 셋은 set()으로 표시함

**셋 항목 확인**

3 in date\_set -> 셋 객체에 해당 항목이 있는지 검사

6 not it date\_set -> 셋 객체에 해당 항목이 없는지 검사

data\_set1 = {1,2,3,4,5}

data\_set2 = {2,3}

issuperset : a가 b를 전부 포함하는 집합인지 확인

ex) data\_set1.issuperset(data\_set2) => True

issubset : a가 b에 전부 포함되는 집합인지 확인

ex) data\_set2.issubset(data\_set1) => True

**for 문을 이용한 셋 항목 접근**

data\_set = set(range(0,11,2))

for idx, item in enumerate(date\_set) -> 이뉴머레이션 객체로 변환, 인덱스와 항목 정보를 매 반복마다 변수 idx와 item에 저장

print(‘[{0}] => {1}’.format(idx, item))

[결과]

[0] => 0

[1] => 2

[2] => 4

[3] => 6

[4] => 8

[5] => 10

**2. 셋 내포의 특징**

data\_set2 = {item for item in data\_set1 if item % 2 == 0} -> 반복 가능한 자료형의 경우 해당 리터럴 안에서 for 문을 사용하면 내포 기능을 사용할 수 있음

data\_set5 = {x\*y for x in data\_set1 if x%2 == 1

for y in data\_set2 if y%2 == 0}

-> 셋 내포는 for 문의 중첩 구조 사용 가능

-> If 조건문을 추가해 조건을 만족하는 항목만 선택해 연산의 결과로 항목을 만들 수 있음

-> 중첩된 for 문보다 셋 내포를 사용하는 것이 간단함

**리스트, 튜플, 셋의 변환**

data\_str = “Hello”

data\_list = list(data\_str) [‘H’, ‘e’, ‘l’, ‘l’, ‘o’]

data\_tuple = tuple(data\_list) (‘H’, ‘e’, ‘l’, ‘l’, ‘o’)

data\_set = set(data\_tuple) {‘H’, ‘e’, ‘l’, ‘o’}

data\_list = list(data\_set) [‘H’, ‘e’, ‘l’, ‘o’]

**3. 딕셔너리 생성 및 조작법**

딕셔너리 : 중괄호{} 안에 키 : 값의 형식을 가진 유일한 데이터를 콤마(,)로 구분해 하나 이상 저장할 수 있는 컬렉션 자료형 중의 하나

* 인덱스를 제공하지 않음
* 순서의 개념이 없음
* 중복을 허용하지 않음

data\_dict1 = {

“홍길동”: 20,

“이순신”: 45,

“강감찬”: 35

}

data\_dict2 = dict(홍길동=20, 이순신=45, 강감찬=35) -> 매개변수 목록을 기술해 딕셔너리 객체 생성

* ‘키=값’형식의 매개변수 목록에서 키를 문자열로 작성하지 않도록 주의

data\_tuple1 = ((“홍길동”, 20), (“이순신, 45), (“강감찬”, 35))

data\_dict3 = dict(data\_tuple1)

data\_list1 = [(“홍길동”, 20), (“이순신”, 45), (“강감찬”, 35)]

data\_dict4 = dict(data\_list1)

data\_set1 = {(“홍길동”, 20), (“이순신”, 45), (“강감찬”, 35)}

data\_dict5 = dict(data\_set1)

[결과]

data\_dict1, data\_dict2, data\_dict3, data\_dict4, data\_dict5 모두 다

{‘홍길동’: 20, ‘이순신’: 45, ‘강감찬’: 35}로 반환됨

**딕셔너리 항목 접근**

print(data\_dict1[“홍길동”] => 20

**딕셔너리 항목 추가**

data\_dict1[‘을지문덕’] = 40

data\_dict1.update({‘신사임당’: 50, ‘유관순’: 16})

**딕셔너리 항목 변경**

data\_dict1[‘강감찬’] = 38

data\_dict1.update({‘홍길동’: 25, ‘이순신’: 48}) -> 키가 동일할 때 기존 항목 변경

**딕셔너리 항목 제거**

del data\_dict1[‘강감찬’]

data\_dict1.pop(‘이순신’)

data\_dict1.cler() -> 모든 항목 제거

**딕셔너리 항목 확인**

‘홍길동’ in data\_dict1 -> 딕셔너리 항목이 있는지 검사

‘신사임당’ not in data\_dict1 -> 딕셔너리 항목이 없는지 검사

**for 문을 이용한 딕셔너리 항목 접근**

data\_dict1.items() -> 튜플로 구성

data\_dict1.key() -> 문자열로 구성

data\_dict1.values() -> 정수로 구성

**4. 딕셔너리 내포의 특징**

{item[0]: item[1] for item in data\_dict1.items()}

{key: value for key, value in data\_dict1.items()}