isidentifier() – 식별자가 적합한지 아닌지 판별하고자 할 때.

‘a-b’.isidentifier()는 false ‘a\_b’.identifier()는 true

문자열.zfilee(숫자) 이거는 숫자만큼 문자열을 확보하는데 모자른 공백들은 0으로 채움

파이썬은 동적 자료형

동적자료형

메모리 관리는 파이썬이 해줌

각 언어의 특징을 잘 알고 언어의 특징을 살려서 코딩

집합

순서가 없어서 인덱싱, 슬라이싱 불가

중복 데이터가 없음

set과 frozenset이 있음

set은 변경 가능, frozenset은 변경 불가

set은 {}로 묶음

set하고 dictionary는 둘 다 중괄호{}로 묵음

dictionary하고 차이점은???????????????????????????????

set함수는 형변환의 성격도 가지고 있음

튜플로 set을 만들 수 있고 문자열도 가능, list도 가능

set의 내용은 변경 불가능한 내용만 넣을 수 있음

>>> a=[1,2,3]

>>> b=[3,4,5]

>>> {a,b}

set은 변경 가능해서 넣으면 에러남

set([1,2,3])은 리스트 내용을 가지고 와서 set을 만드는거라 가능

변경 가능 - 리스트 셋 딕셔너리 바이트어레이 만 변경 가능

hashtable???????

집합이 hash기능 있는 것만 넣을 수 있음

.add와 .update의 차이점은 하나의 원소인지 하나의 객체인지

.add(숫자) .update([리스트]) 리스트가 아니어도 됨

add에 튜플은 추가 가능

>>> b.add([4,5,6])

Traceback (most recent call last):

File "<pyshell#67>", line 1, in <module>

b.add([4,5,6])

TypeError: unhashable type: 'list'

>>> b.update(4,5,6)

Traceback (most recent call last):

File "<pyshell#68>", line 1, in <module>

b.update(4,5,6)

TypeError: 'int' object is not iterable

>>> b.update([4,5,6])

>>> b.add((1,2,3))

>>> b

{4, 5, 6, 7, 8, (1, 2, 3)}

>>> a.remove(4);a

{1, 2, 5, 6, 7, 8, 9}

>>> a.remove(4);a

Traceback (most recent call last):

File "<pyshell#80>", line 1, in <module>

a.remove(4);a

KeyError: 4

>>> a.remove(4)

Traceback (most recent call last):

File "<pyshell#81>", line 1, in <module>

a.remove(4)

KeyError: 4

>>> a.pop()

1

>>> a.pop()

2

>>> a.pop()

5

>>> a.pop()

6

>>> a

{7, 8, 9}

remove는 원소가 없으면 예외 발생

discard는 원소가 없어도 통과

pop은 맨 앞에 값 반환하고 set에서 원소 제거

셋의 내용을 가지고 인덱싱이나 슬라이싱 하려면 다른 자료형을 ㅗ형변환 해서 해야함. 순서가 없는거니깐 순서잇는 애들로 변환. 순서? 인덱스?

변경가능한 자료형으로 하면 다른데서 변경되면 찾기 힘듦

frozenset은 내용변경이 안됨. 그래서 update같은 것들안됨

update는 내용을 바꾸는거라

사전

키를 가지고 값을 찾는거라 순서자체가 의미가 없음

hashtable기반으로 관리함. 검색에 특화

set과 dictionary사전의 차이는 사전은 반드시 key:value가 있음

데이터가 들어나도 검색속도가 일정하게 하기 위해서 hashmap활용

,

pop은 value값 popitem은 key와 value 반환

>>> d

{'one': 1, 'two': 2, 'three': 3, 'nine': 9}

>>> d.get('ten')

>>> d.setdefault('ten',10);d

10

{'one': 1, 'two': 2, 'three': 3, 'nine': 9, 'ten': 10}

get했을 때 없어도 에러 나지는 않음

>>> d={}.fromkeys('abcde',[]);d

{'a': [], 'b': [], 'c': [], 'd': [], 'e': []}

>>> d['a'].append(1);d

{'a': [1], 'b': [1], 'c': [1], 'd': [1], 'e': [1]}

[]리스트에 대한 append

>>> d={}.fromkeys('abcde',[]);d

{'a': [], 'b': [], 'c': [], 'd': [], 'e': []}

>>> d['a'].append(1);d

{'a': [1], 'b': [1], 'c': [1], 'd': [1], 'e': [1]}

이거는 한꺼번에 생성한거라 append하면 다 적용됨

>>> d2=dict((c,[])for c in 'abcde');d2;d2['a'].append(1);d2

{'a': [], 'b': [], 'c': [], 'd': [], 'e': []}

{'a': [1], 'b': [], 'c': [], 'd': [], 'e': []}

이거는 for로 하나하나 생성한거라 ㄱ한개에만 적용

enumerate() 는 인덱스하고 값 리턴

enumerate([‘one’,’two’,’three’]) 하면 0,’one’ 반환

zip은 인덱스들을 하나씩 꺼내서 묶어서 tuple로 만들어줌 보내줌

zip(a1,a2)

레퍼런스만 복사하고 레퍼런스가 가르키는 내용은 복사 안하는걸 얕은 복사

깊은 복사는 내용까지 복사하는거. 집합형태의 래퍼런스까지

얕은복사는 참조만 하는거고 칠판을 한 개 두고 위치만 알려주는거

깊은 복사는 참조하는 내용까지 복사 칠판을 여러 개 나눠주는거

>>> d[ [ 1, 2, 3 ] ] = ‘list’# error, 키가변경가능

>>> d[ { 1 : 2 } ] = 3# error,키가변경가능

?????????????110p