Python DataType

안 화 수

1. 수치형

- 정수형
  - ✓ 음의 정수, 0, 양의 정수
  - ✓ python에서는 메모리가 허용하는 한, 무한대의 정수를 다룰 수 있음.
  - ✓ 하지만 CPU 레지스터로 표현할 수 있는 크기보다 큰 정수를 다루는 경우 연산 속도는 느려집니다.
  - ✓ 일반적인 정수는 10진 정수로 간주
  - ✓ 0o로 시작하는 정수는 8진수
  - ✓ 0x, 0X로 시작하는 정수는 16진수
  - ✓ 0b로 시작하는 정수는 2진수
  - ✓ 문자열을 정수로 변환할 때는 int(문자열) 또는 int(문자열, 진수)를 이용
  - ✓ 문자열 대신에 실수를 대입하면 소수를 버립니다.
  - ✓ 실수로 된 문자열이나 복소수는 정수로 변환이 되지 않습니다.

1. 수치형

❖ 예제

```
a = 10
print("a =", a)
a = 0o12
print("a =", a)
a = 0xA
print("a =", a)
a = 0b1010
print("a =", a)
a = int('10')
print("a =", a)
a = int('20', 5)
print("a =", a)
a = int(10.8)
print("a =", a)
```

❖ 결과

```
a = 10
a = 10
a = 10
a = 10
a = 10
a = 10
a = 10
```

1. 수치형

- 정수형
  - ✓ 정수형 데이터를 바이트 열로 변환 가능

(정수데이터).to_bytes(표현할 바이트 수, byteorder='big' | 'little' [,signed=True|False])

  - ✓ 바이트 열에서 정수형으로 변환

int.from_bytes(바이트열, byteorder='big' | 'little'

  - ✓ 예제

```
print((1024).to_bytes(2, byteorder='big'))
print((-1024).to_bytes(2, byteorder='big', signed=True))
print(int.from_bytes(b'□x04□x00', byteorder='big'))
```

```
b'□x04□x00'
b'□xfc□x00'
1024
```

1. 수치형

- 실수형
  - ✓ 파이썬에서는 실수를 지원하기 위해 부동 소수형을 제공
  - ✓ 부동 소수형은 소수점을 움직여서 소수를 표현하는 자료형
  - ✓ 부동 소수형은 8바이트만을 이용해서 수를 표현
  - ✓ 한정된 범위의 수만 표현
  - ✓ 디지털 방식으로 소수를 표현해야 하기 때문에 정밀도의 한계가 있음
  - ✓ 소수점을 표현하면 실수(1.2)
  - ✓ 지수(e)를 포함해도 실수(3e3, -0.2e-4)
  - ✓ 실수의 가장 큰 값과 작은 값은 sys 모듈의 float_info 또는 info.max, info.min으로 확인 가능
  - ✓ 무한대의 수는 float('inf' | '-inf')
  - ✓ is_integer 메소드를 통해서 정수로 변환시의 오차 문제 확인 가능
  - ✓ 실수를 정수로 변경할 때 사용할 때 사용할 수 있는 함수
  - ✓ int, round, math.ceil, math.floor

1. 수치형

❖ 예제

```
import sys
print("실수 정보:", sys.float_info)
a = 1.2
b = 3e3
c=-0.2e-4
print(a, b, c)
if a.is_integer():
    print("a는 정수로 변환하면 오차가 발생합니다.")
else:
    print("a는 정수로 변환하면 오차가 발생하지
      않습니다.")
a = int(1.7)
print("a:", a)
a = round(1.7)
print("a:", a)
import math
a = math.ceil(1.7)
print("a:", a)
a = math.floor(1.2)
print("a:", a)
```

❖ 결과

```
실수 정보:
    sys.float_info(max=1.79769313486231
    57e+308, max_exp=1024,
    max_10_exp=308,
    min=2.2250738585072014e-308,
    min_exp=-1021, min_10_exp=-307,
    dig=15, mant_dig=53,
    epsilon=2.220446049250313e-16,
    radix=2, rounds=1)
1.2 3000.0 -2e-05
a는 정수로 변환하면 오차가 발생하지
    않습니다.
a: 1
a: 2
a: 2
a: 1
```

1. 수치형

- 실수의 진법 변환 오류:소수 중에는 정확하게 2진수로 표현할 수 없는 수가 있으므로 실수 연산은 오차가 발생할 수 있습니다.

```
sum = 0
for i in range(0,1000):
    sum += 1
print("정수 1을 1000번 더한 결과:" , sum)

sum = 0.0
for i in range(0,1000):
    sum += 0.1
print("실수 0.1을 1000번 더한 결과:" , sum)
```

- 결과

```
정수 1을 1000번 더한 결과: 1000
실수 0.1을 1000번 더한 결과: 99.9999999999986
```

1. 수치형

- 복소수
  - 실수부와 허수부로 숫자를 표현
  - 허수부는 j를 추가해서 표현
  - 복소수에서 real 속성을 호출하면 실수부만 리턴
  - imag를 호출하면 허수부만 리턴
  - 정수 2개를 가지고 복소수 생성 가능 – complex(실수부, 허수부)
  - conjugate()를 호출해서 켤레 복소수 리턴

```
c = 3 + 4j
print(c)
a = 3;
b = 4;
print(complex(3,4))
d = c.conjugate();
print(d)
```

1. 수치형

- fractions 모듈을 이용한 분수 표현
  - ✓ fractions 모듈을 이용해서 분수 표현 가능
  - ✓ 5/7을 표현하고자 할 때 fractions('5/7') 또는 fraction(5,7)
  - ✓ 그리고 만들어진 데이터의 numeratior 와 denominator를 이용해서 분자와 분모를 따라 추출 가능

```
from fractions import Fraction
a = Fraction(5,7) + Fraction('2/5')
print(a)
```

1. 수치형

- decimal 모듈을 이용한 숫자 표현
  - ✓ Decimal(정수| 실수| 문자열)을 이용하면 정밀한 숫자를 표현 가능
  - ✓ Decimal('0.1')을 이용하면 정확하게 숫자 표현 가능
  - ✓ Decimal 객체는 Decimal 객체나 정수 데이터와 연산은 가능하지만 실수 객체와의 연산은 수행되지 않음

```
from decimal import *
sum = Decimal(0.0)
delta = Decimal('0.1')
for i in range(0,1000):
    sum += delta
print("sum:" , sum)
```

1. 수치형

- 수치 연산 함수
  - ✓ abs(숫자): 절대 값 리턴
  - ✓ divmod(x, y): x//y, x%y의 값을 쌍으로 리턴
  - ✓ pow(x, y): x의 y승을 구합니다.

- math 모듈
  - ✓ pi, e 와 같은 상수
  - ✓ sqrt(숫자), sin(값)과 같은 함수 내장

1. 수치형

- and와 or 연산에서의 주의 사항
  - ✓ True 또는 False 가 아닌 value의 and와 or 연산에서는 결과가 True 또는 False로 리턴되지 않습니다.
  - ✓ and 연산의 경우 앞의 연산이 True이면 뒤의 연산의 결과가 리턴되며 False 인 경우 앞의 연산 결과가 리턴됩니다.
  - ✓ or 연산의 경우는 and와 반대의 결과를 가져옵니다.

```
b = 1>2 and 3
print(b)
b = 0 and 3
print(b)
b = 1<2 and 3
print(b)
b = 0 or 3
print(b)
b = 1 or 3
print(b)
```

```
False
0
3
3
1
```

1. 수치형

- 연습문제
  - ✓ 1 and 2 or 3 and 4의 결과가 얼마인지 확인
  - ✓ n 번째 피보나치 수열을 구하는 코드를 작성하시오

(1,1,2,3,5,8,13.. f(n) = f(n-1) + f(n-2) 단f(1)=1, f(2)=1)

2. Sequential Type

- Sequential Type
  - ✓ 객체를 순서대로 저장하는 자료형
  - ✓ 문자열, list, tuple
  - ✓ 문자열은 ‘’ 또는 “” 로 묶인 문자들의 모임
  - ✓ 리스트는 [] 안에 묶인 데이터의 모임
  - ✓ 튜플은 ()안에 묶인 데이터의 모임
  - ✓ Sequential Type의 공통 연산
    - 인덱싱: [인덱스] – 인덱스 번째 데이터 리턴, 음수는 뒤에서부터 진행
    - 슬라이싱: [s:t:p] – s번째 부터 t번째 까지 p 간격으로 가져오기로 t가 생략되면 s이후 전체, p를 생략하면 순서대로 하나씩
    - 연결하기: +연산자
    - 반복하기: * 연산자
    - 멤버 검사: in 연산자
    - 길이 정보: len()

2. Sequential Type

- 예제

```
str = "Korea"
print(str[0]) #0번째 글자
print(str[-2]) #뒤에서 2번째 글자
print(str[1:3]) #1번째부터 3번째 글자 앞까지
print(str[0:5:2]) #0부터 5번째글자 앞까지 2개씩 건너뛰면서
print(str[:-1])#가장 마지막 글자를 제외하고
print(str[::-1])#반대로 출력
str = "Hello" + "World"
print(str)
str = str * 4
print(str)
```

3. str

- str
  - ✓ 파이썬의 문자열
  - ✓ 한 줄 문자열: ‘ 또는 “ 안에 기재
  - ✓ 여러 줄 문자열:‘‘‘ ‘‘‘ 또는 “”” “”” 안에 기재
  - ✓ escape 문자(□를 이용해서 표현)
    - □enter: 다음 줄과 연속
    - □□: □문자
    - □’, □”: 작은 따옴표나 큰 따옴표
    - □b: 백스페이스
    - □n 또는 □012: 줄 변경
    - □t: 탭
    - □0xx: 8진수 코드 xx
    - □xXX: 16진수 코드 XX
    - □e: esc

3. str

- 문자열은 근본적으로 데이터를 변경할 수 없음
- 문자열 내의 특정 문자의 값을 변경할 수 없습니다.

str = “hello”

str[0] = ‘f’=> 이런 문장은 에러

- 문자열을 추가하거나 삭제할 때는 슬라이싱과 연결하기를 이용해야 합니다.

- Hardward and Shell 문자열에서 and 앞에 Kernel을 추가하기

```
str = "Hardware and Shell"
str = str[:8] + ' Kernel ' + str[8:]
print(str)
```

- 위의 문자열에서 and 제거하기

```
str = str[:15] + str[20:]
print(str)
```

3. str

- 문자열의 서식 지정
  - ✓ print() 함수를 사용하면 출력 문자열의 서식을 자유롭게 지정 가능
  - ✓ 단순하게 문자열이나 데이터를 , 를 이용해서 출력할 수 있지만 양식을 만들어두고 빈칸에 필요한 내용을 채워서 문서를 자유롭게 채워나갈 수 있음
  - ✓ 이전 방식은 문자열 안에 서식 문자를 지정하고 튜플을 이용해서 값을 채움
  - ✓ 서식 문자는 %와 함께 표현

```
int_val = 23
float_val =  45.9876
print("int_val:%3d, float_val:%0.2f" % (int_val, float_val))
```

  - ✓ 파이썬 3.0에서는 format(데이터, '서식')을 이용해서 서식을 지정하기를 권장

```
int_val = 23
float_val =  45.9876
print("int_val:", format(int_val,"3d"), "float_val:", format(float_val, "0.2f"))
```

3. str

- 문자열의 서식 지정
  - ✓ 문자열의 format 메소드를 이용
  - ✓ 문자열 안에 {}를 사용하고 .format(데이터 나열)을 이용하면 나열된 데이터가 순차적으로 대입
  - ✓ {인덱스}를 이용하면 format에서 인덱스 번째 데이터가 대입
  - ✓ {인덱스:서식}을 이용하면 인덱스 번째의 데이터를 서식에 맞추어 출력

```
str = [1,3,4]
print('최대값:{0:5d}□t최소값:{1:5d}'.format(max(str), min(str)))
```

  - ✓ 서식 자료형
    - d는 정수, f는 실수, s는 문자열
  - ✓ 서식에 사용할 수 있는 보조 서식 문자
    - 숫자: 숫자만큼의 최소자리 확보
    - 숫자,숫자: 앞의 숫자만큼 자리를 확보하고 뒤의 숫자만큼 소수 표현
    - <: 왼쪽 맞춤
    - >: 오른쪽 맞춤
    - 공백: 양수 일 때 앞에 공백 출력
    - +: 양수 일 때 + 출력
    - 0: 왼쪽의 빈공간을 0으로 채움

3. str

- 문자열의 대소문자 변환 메소드
  - upper(): 대문자로 변환
  - lower(): 소문자로 변환
  - swapcase(): 대소문자 스위치
  - capitalize(): 첫글자만 대문자로 변환
  - title: 각 단어의 첫글자를 대문자로 변환
- 문자열의 검색 관련 메소드
  - count(문자열): 문자열의 횟수
  - find(문자열): 문자열이 있을 때 오프셋 반환
  - find(문자열, 숫자): 숫자 위치부터 문자열을 검색해서 오프셋 반환
  - 찾는 문자열이 없을 때는 -1을 반환
  - rfind(문자열): 뒤에서부터 검색
  - index(문자열): 문자열이 없을 경우 예외 발생
  - rindex(문자열): 문자열을 뒤에서부터 검색하고 없으면 예외 발생
  - startswith(문자열): 문자열로 시작하는지 여부를 검사해서 bool로 리턴
  - endswith(문자열): 문자열로 끝나는지 여부를 검사
  - startswith(문자열, 숫자): 숫자에 해당하는 위치가 문자열로 시작하는지 여부를 검사해서 bool로 리턴
  - endswith(문자열, 숫자1, 숫자2): 숫자1부터 숫자2까지의 문자열이 문자열로 끝나는지 여부를 검사

3. str

- 문자열의 편집과 치환 메소드
  - ✓ strip(): 좌우 공백 제거
  - ✓ rstrip(): 오른쪽 공백 제거
  - ✓ lstrip(): 왼쪽 공백 제거
  - ✓ strip('문자열'): 좌우의 문자열을 제거
  - ✓ replace(문자열1, 문자열2): 앞의 문자열을 뒤의 문자열로 치환
- 문자열의 분리와 결합 관련 메소드
  - ✓ split(): 공백으로 문자열을 분리해서 리스트로 반환
  - ✓ split('문자열'):문자열로 분리해서 리스트로 반환
  - ✓ split('문자열', 숫자):문자열로 분리하는데 숫자만큼만 불리해서 리스트로 반환
  - ✓ '문자열'.join(문자열 리스트): 문자열 리스트를 문자열을 각 요소 별로 문자열을 추가해서 결합
  - ✓ splitlines(): 줄 단위로 분리
- 맞춤 관련 메소드
  - ✓ center(숫자): 숫자 만큼의 자리를 확보하고 가운데 맞춤
  - ✓ ljust(숫자): 왼쪽 맞춤
  - ✓ rjust(숫자): 오른쪽 맞춤
  - ✓ center(숫자, 문자열): 숫자 만큼의 자리를 확보하고 가운데 맞춤한 후 빈자리는 문자열로 채움

3. str

- 예제

```
str = "c:□□data□□data.txt"
#test의 존재여부
print(str.find('test'))
#위의 문자열에서 파일의 확장자만 추출하기
list = str.split(".")
print("확장자:", list[len(list)-1])
print(str.replace('data', 'python'))
```

3. str

- 탭 문자 변환 메소드
  - ✓ expandtabs(숫자): 숫자를 생략하면 탭을 8자 공백으로 변경하고 숫자를 대입하면 숫자만큼의 공백으로 변경
- 문자열 확인 메소드
  - ✓ isdigit()
  - ✓ isnumeric()
  - ✓ isdecimal()
  - ✓ isalpha()
  - ✓ isalnum()
  - ✓ islower()
  - ✓ isupper()
  - ✓ isspace()
  - ✓ istitle()
  - ✓ isidentifier()
  - ✓ isprintable()

3. str

- 문자열 매핑 메소드
  - ✓ maketrans()와 translate()를 이용하면 문자열 매핑 가능
  - ✓ maketrans()를 이용해서 치환할 문자열을 만들고 translate의 매개변수로 대입

```
instr = 'abcdef'
outstr = '123456'
trans = ''.maketrans(instr, outstr)
str = 'hello world'
print(str.translate(trans))
```

3. str

- 문자열 인코딩
  - ✓ 파이썬의 문자열은 기본적으로 utf-8 인코딩을 사용하는데 기본 아스키 코드는 1바이트를 사용하고 나머지는 2바이트를 사용하는 방식의 인코딩입니다.
  - ✓ 문자열을 바이트 코드로 변환할 때는 encode()를 이용하고 특정 코드로 변환하고자 하는 경우에는 encode('인코딩방식') 메소드를 이용합니다.
  - ✓ 바이트 코드는 문자열 함수를 거의 사용이 가능하지만 문자열과 직접 연산은 불가능합니다.
  - ✓ 바이트 코드를 문자열로 변환할 때는 decode('인코딩방식')을 이용해서 문자열로 변환할 수 있습니다.
  - ✓ ord('문자'):문자를 코드 값으로 변환
  - ✓ chr(유니코드): 유니코드를 문자로 변환

```
b = '파이썬'.encode('utf-8')
print(b)
b = '파이썬'.encode('ms949')
print(b)
str = b.decode('ms949') #utf-8로 인코딩 하면 예외 발생
print(str)
```

3. str

- 유니코드에서 한글 자소 분리 및 자소를 글자로 변환
  - ✓ 한글 유니코드는 조성 19개, 중성 21개, 종성 28개로 구성
  - ✓ 한글 유니코드는 0xAC00에서 시작
  - ✓ 한글 유니코드 구하기

0xAC00 + ((초성순서 * 21)+중성순서)*28 + 종성순서

  - ✓ 한글 한 글자의 초성 중성, 종성의 위치 찾아내기

```
c = '한'
code = ord(c) - 0xAC00;
chosung = code // (21*28)
jungsung = (code - chosung*21*28)//28
jongsung = (code - chosung*21*28 - jungsung*28)
print('초성:', chosung, "번째 글자")
print('중성:', jungsung, "번째 글자")
print('종성:', jongsung, "번째 글자")
```

3. str

- 영타로 입력한 내용 한글로 변경하기

```
cho_list_eng = [ "r", "R", "s", "e", "E", "f", "a", "q", "Q", "t", "T", "d", "w", "W", "c", "z", "x", "v", "g"]
jung_list_eng = ["k", "o", "I", "O", "j", "p", "u", "P", "h", "hk", "hO", "hl", "y", "n", "nj", "np", "nl", "b",
    "m", "ml", "l"]
jong_list_eng = ["", "r", "R", "rt", "s", "sw", "sg", "e", "f", "fr", "fa", "fq", "ft", "fx", "fv", "fg", "a", "q",
    "qt", "t", "T", "d", "w", "C", "z", "x", "v", "g"]

str='wnd'
cho = cho_list_eng.index(str[0])
jung = jung_list_eng.index(str[1])
jong = jong_list_eng.index(str[2])
code = 0xac00 + ((cho*21) + jung)*28 + jong
print(chr(code))
```

3. str

- 연습문제
  - ✓ 하나의 문자열을 입력받고 입력받은 문자열을 while을 이용해서 뒤집어서 출력해보기
  - ✓ 하나의 문자열을 입력받고 입력받은 문자열을 for를 이용해서 뒤집어서 출력해보기
  - ✓ 다음의 경로명에서 경로명과 파일명을 분리

  s = 'usr/local/bin/python'

  dic = '/usr/local/bin'

  filename = 'python'
  - ✓ 문자열이 주어졌을 때 특정 문자열이 몇번 나오는지 세어보기

  (단 대소문자는 구별하지 않고)

  s = 'Kia KIA kia nexen Nexen Dusan KiA'

  KIA는 4번

4. List

- List
  - ✓ 순서를 갖는 객체의 집합
  - ✓ 내부 데이터를 변경 가능한 자료형이고 시퀀스 자료형의 특성을 지원하며 데이터의 크기를 동적으로 변경할 수 있으며 내용을 치환하여 변경할 수 있습니다.
  - ✓ 대괄호 [ ]로 표현
  - ✓ 항목 교체
    - 리스트[인덱스] = 값
    - 리스트[시작범위:종료범위] = [데이터 나열]
    - 리스트[시작범위:종료범위] = 값
  - ✓ 데이터 삭제
    - 리스트[시작범위:종료범위] = [ ]
    - del 리스트[인덱스]
  - ✓ 데이터 중간에 삽입
    - 리스트[시작범위:시작범위] = [데이터]
  - ✓ range()를 리스트로 변환
    - list(range(범위))

4. List

- 예제

```
list = list(range(4)) #range를 이용한 list 생성
print(list)
del list[::2]#짝수번째 데이터 삭제
print(list)
list[1:1] = [2] #1번째 자리에 2를 추가
print(list)
list[0:0] = [0] #0번째 자리에 0을 추가
print(list)
#모든 객체 순회
for i in list:
    print(i)
```

4. List

- 중첩 List
  - ✓ 리스트 안에 다른 리스트가 포함된 경우
  - ✓ 리스트는 다른 객체를 직접 저장하지 않고 객체들의 참조만을 저장합니다.
  - ✓ 따라서 다른 리스트를 참조하는 경우 참조된 리스트의 내용이 변경되면 포함하고 있는 리스트의 내용도 변경됩니다.

```
innerlist = list(range(4))
outerlist = ['begin', innerlist, 'end']
print(outerlist)
innerlist[0] = 200
print(outerlist)
```

4. List

- List 메소드
  - ✓ append(값): 마지막에 데이터 추가
  - ✓ insert(인덱스, 데이터): 인덱스 위치에 데이터를 삽입
  - ✓ index(데이터): 데이터의 위치를 검색
  - ✓ count(): 요소의 개수 리턴
  - ✓ sort(): 리스트를 정렬
  - ✓ reverse(): 리스트의 순서를 변경
  - ✓ remove(데이터): 데이터의 첫번째 것을 삭제
  - ✓ pop(인덱스): 인덱스를 생략하면 가장 마지막 데이터를 삭제하고 리턴하고 인덱스를 대입하면 인덱스 번째를 삭제하고 리턴
  - ✓ extend(리스트): 리스트를 추가

4. List

- Stack
  - ✓ 나중에 삽입한 데이터를 먼저 꺼내도록 해주는 메모리 구조
  - ✓ LIFO(Last In First Out)구조의 메모리
  - ✓ 데이터를 삽입하는 연산을 push라 하고 꺼내는 연산을 pop이라고 합니다.
  - ✓ 리스트는 스택으로 사용할 수 있게 설계되었습니다.
  - ✓ push 연산은 append()를 이용하면 되고 pop 연산은 pop() 메소드를 이용하면 됩니다.

```
stack = [10,20,30,40,50]
stack.append(60) #push
print(stack)
data = stack.pop() #pop
print(data)
print(stack)
```

4. List

- Queue
  - ✓ 먼저 삽입한 데이터를 먼저 꺼내도록 해주는 메모리 구조
  - ✓ FIFO(First In First Out)구조의 메모리
  - ✓ 리스트는 큐로 사용할 수 있게 설계되었습니다.
  - ✓ push 연산은 append()를 이용하면 되고 pop 연산은 pop(0) 메소드를 이용하면 됩니다.

```
queue = [10,20,30,40,50]
queue.append(60) #push
print(queue)
data = queue.pop(0) #pop
print(data)
print(queue)
```

4. List

- 정렬
  - ✓ sort()를 이용하면 데이터를 내부적으로 정렬
  - ✓ 값을 리턴하지 않음
  - ✓ 매개변수로 reverse=True를 대입하면 내림차순 가능

```
data = [30,50,10,40,20]
data.sort()
print("오름차순:", data)
data.sort(reverse=True)
print("내림차순:",data)
```

4. List

- 정렬
  - ✓ 문자열을 정렬할 때 대소문자 구분없이 정렬 가능
  - ✓ 매개변수로 key라는 속성에 비교할 함수이름을 전달

```
data = ['Morning', 'Afternoon', 'evening' ,'Night' ]
data.sort()
print("대소문자 구분해서 정렬:", data)
data.sort(reverse=True)
print("내림차순 정렬:", data)
data.sort(key=str.lower, reverse=True)
print("내림차순 정렬:", data)
```

4. List

- 정렬
  - ✓ 사용자 정의 함수를 이용한 정렬

```
def strlen(st):
    return len(st)

data = ['Morning', 'Evening', 'Afternoon', 'Night']
data.sort(key=strlen)
print(data)
```

4. List

- 정렬
  - ✓ 리스트를 정렬하지 않고 정렬한 결과를 리스트로 리턴하는 함수는 sorted()
  - ✓ key와 reverse 키워드를 동일하게 지원
  - ✓ 이 메소드는 튜플이나 디셔너리에서도 사용 가능
  - ✓ reverse()를 이용하면 데이터를 역순으로 배치

```
data = [30,50,10,40,20]
print("오름차순:", sorted(data))
print("내림차순:",sorted(data, reverse=True))
```

```
data = [30,50,10,40,20]
data = sorted(data)
print("오름차순:", data)
data.reverse()
print("내림차순:",data)
```

5. Tuple

- Tuple
  - ✓ 임의 객체들이 순서를 가지는 모음으로 리스트와 유사
  - ✓ 차이점은 변경이 불가능하다는 것
  - ✓ 튜플은 시퀀스 자료형이므로 인덱싱과 슬라이싱, 연결하기, 반복하기, 길이 정보등의 일반적인 연산을 가짐
  - ✓ 튜플은 ()안에 표현
  - ✓ ()를 사용하지 않고 ,로 데이터를 구분하면 튜플로 처리
  - ✓ ()를 하는 경우 데이터가 1개 일 때는 ,를 마지막에 추가
  - ✓ list()와 tuple()를 이용해서 서로 간에 변환 가능
  - ✓ 포함된 데이터 개수를 리턴해주는 count 소유
  - ✓ 포함된 데이터 위치를 리턴해주는 index 소유

```
t = (1,2,3,2,2,3)
print("2의 개수:", t.count(2))
print("2의 위치:", t.index(2))
print("2의 위치:", t.index(2,3))
```

5. Tuple

- Tuple
  - ✓ 여러 개의 데이터를 한꺼번에 대입하는 것이 가능
  - ✓ 변수, 변수 = 데이터, 데이터
  - ✓ 위의 방법을 이용하면 swap이 쉽게 가능

```
x,y=1,2
x,y=y,x
print(x, y)
```

6. Set

- Set
  - ✓ 데이터를 순서와 상관없이 중복 없이 모아놓은 자료형으로 set과 frozenset 두 가지 자료형을 제공
  - ✓ 빈 객체 생성은 set()
  - ✓ 처음부터 데이터를 가지고 있는 경우에는 {데이터 나열}
  - ✓ 데이터가 없는 경우에는 { } 대신 set()으로 생성하는 이유는 디셔너리와의 혼동의 문제
  - ✓ 튜플, 문자열, 리스트 및 디셔너리로부터 생성이 가능
  - ✓ set의 원소로 변경이 가능한 데이터 타입은 불가능합니다.

```
hashset = set()
print(hashset)
hashset = {1,2,3}
print(hashset)
hashset = set((1,2,3,2))
print(hashset)
hashset = set('hello world')
print(hashset)
hashset = set([1,3,2])
print(hashset)
```

6. Set

- Set의 연산
  - ✓ 데이터의 추가는 add(데이터), update[데이터 나열])
  - ✓ 데이터의 복사는 copy()
  - ✓ 데이터의 전체 제거는 clear()
  - ✓ 데이터의 한 개 제거는 discard(데이터) – 없으면 그냥 통과
  - ✓ 데이터의 한 개 제거는 remove(데이터) – 없으면 예외
  - ✓ pop(): 1개 제거

- Set의 집합 연산
  - ✓ union(다른 set), intersection(다른 set), difference(다른 set), symmetric_difference(다른 set)의 메소드가 제공되는데 결과를 리턴합니다.
  - ✓ update(), intersection_update(), difference_update(), symmetric_difference_update()는 호출하는 객체의 데이터를 변경

6. Set

- 포함 관계 연산자
  - ✓ 데이터 in set : 앞의 요소가 포함되어 있는지 리턴
  - ✓ 데이터 not in set: 앞의 요소가 포함되어 있지 않은지 리턴
  - ✓ isupperset(다른 Set): 다른 Set을 포함하고 있는지 여부 리턴
  - ✓ issubset(다른 set): 다른 set의 서브셋인지 여부 리턴
  - ✓ isdisjoint(다른 set): 교집합이 공집합인지 여부 리턴
- 예제

```
a = {1,2,3,4,5}
b = {4,5,6,7,8}
print(a.union(b))
print(a.intersection(b))
print(a.difference(b))
print(a.symmetric_difference(b))
```

7. Dictionary

- Dictionary
  - ✓ Key와 Value를 쌍으로 저장하는 자료형
  - ✓ {키:데이터, 키:값….}로 생성가능
  - ✓ Key의 순서는 없으며 Key의 값은 중복될 수 없습니다.
  - ✓ Key의 자료형에 제한은 없지만 거의 str
  - ✓ Key 값에 해당하는 데이터를 가져올 때는 디셔너리[키]
  - ✓ 디셔너리[키] = 데이터 를 이용하면 키의 데이터가 없으면 삽입이 되고 있으면 수정
  - ✓ 없는 키의 값을 호출하면 에러
  - ✓ len(디셔너리)는 키의 개수
  - ✓ del 디셔너리[키]는 키의 데이터 삭제

```
member = {'baseball':9, 'soccer':11,"volleyball":6}
print(member)
print(member['baseball'])
print(member['basketball'])
```

7. Dictionary

- Dictionary
  - 생성하는 다른 방법은 dict(키=데이터, 키=데이터)로 가능
  - 키의 리스트나 값의 리스트가 존재하는 경우 dict(zip(키의 리스트, 값의 리스트))로 생성 가능
  - 사전을 for에 사용하면 키의 모든 리스트가 순서대로 대입됩니다.

```
keys = ['one', 'two', 'three']
values = (1,2,3)
dic = dict(zip(keys, values))
for key in dic:
    print(key,":", dic[key])
```

7. Dictionary

- Dictionary
  - keys(): 키에 대한 모든 데이터를 리턴
  - values(): 값에 대한 모든 데이터를 리턴
  - items(): 모든 데이터 리턴
  - list 메소드의 매개변수로 위 3개의 메소드 리턴값을 넣으면 list로 변환
  - clear(): 모두 삭제
  - copy(): 복사
  - get(key [,x]): 값이 존재하면 값을 리턴하고 없으면 x 리턴
  - setdefault(key [,x]):get 과 동일하지만 값이 존재하지 않으면 x로 설정
  - update(디셔너리): 디셔너리의 모든 항목을 설정
  - popitem(): 마지막 하나의 튜플을 반환하고 제거
  - pop(key): key에 해당하는 데이터를 반환하고 제거

7. Dictionary

- OrderedDict
  - 키의 순서를 유지하는 디셔너리를 만들고자 할 때는 collections 모듈의 OrderedDict를 사용
  - Dictionary와 동일하게 동작하지만 데이터가 추가된 순서를 기억해서 데이터를 순서대로 처리할 수 있도록 해주는 자료형

```
from collections import OrderedDict
keys = ['one', 'two', 'three']
values = (1,2,3)
dic = dict(zip(keys, values))
for key in dic:
    print(key,":", dic[key])
print("======================")
dic = OrderedDict(zip(keys, values))
for key in dic:
    print(key,":", dic[key])
```