Python 기본문법

안 화 수

1. 식별자

- 파이썬 식별자는 변수, 함수, 모듈, 클래스 또는 객체를 식별하는데 사용되는 이름
  - ✓ 대소문자 구별함
  - ✓ 식별자는 문자 A~Z 또는 a~z과 언더바(_)로 시작
  - ✓ 식별자의 시작은 숫자(0~9)로 시작할 수 없음
  - ✓ 특수문자 @, $, %등은 식별자에 사용할 수 없음
  - ✓ 예를 들어 다음과 같은 것은 식별자가 될 수 없음 : 1abc, @file, %x

2. 변수와 상수

- 상수(Constant, Literal):값을 변경할 수 없는 데이터
  - ✓ number
    - 정수:10진수(107), 8진수(0107), 16진수(0x107, 0X107), 2진수(0b101), long(107L)
      =>파이썬 2.x 버전에서는 L을 붙여서 long 형을 만들 수 있지만 3.x에서는 정수 상수는 전부 int 형으로 만들기 때문에 L을 붙일 수 없습니다.
    - 실수:1.2, 0.12e1, 0.12E1 -> 주의: 지수부는 정수일 수 있으나 실수 일 수 없음
    - 복소수: 허수부 뒤에 j 나 J 사용 4+5j, 7-2J
  - ✓ str: ' 문자열` 또는 "문자열" 의 형태로 사용
  - ✓ list:[데이터 나열]
  - ✓ tuple:(데이터 나열)
  - ✓ dict:{키:값...}
  - ✓ bool :True(1), False(0)
  - ✓ 제어문자: □n(줄바꿈), □t(탭)
  - ✓ None: Types.NoneType 의 유일한 값으로 존재하지 않는 변수에 대입하여 이 변수에 아무런 값이 없다는 것을 나타내기 위해 활용

2. 변수와 상수

- 변수(Variable): 데이터를 저장할 수 있는 공간에 붙여놓는 이름
- Python은 변수를 선언할 때 자료형을 기재하지 않음
- 변수에 값을 할당 할 때 데이터 타입을 자동으로 설정
- 등호 (=)는 변수에 값을 할당하는 데 사용
- 파이썬에서 대입의 의미

  a = 1 : 1이라는 값을 저장한 공간의 주소를 a에 대입

- 변수 명명 규칙
  - ✓ 식별자 규칙을 적용
  - ✓ 예약어, 내장함수, 모듈 이름을 변수명으로 만드는 일이 없도록 해야 하는데 예약어를 변수명으로 사용하게 되면 원래의 기능을 잃어버리게 됩니다.

- 변수의 삭제는 **del 변수명**

2. 변수와 상수

```
counter = 100                # 정수 할당
miles   = 1000.0             # 부동 소수점
name    = "홍길동"           # 문자열

print(counter)
print(miles)
print(name)
```

3. 연산자

- 산술연산자
  - +연산자: 숫자의 경우는 덧셈을 하고 문자열이나 데이터의 모임은 결합을 수행
  - *연산자: 문자열의 경우는 반복을 나타내고 숫자의 경우는 곱셈
  - **연산자: 거듭제곱(3**3 은 3의 3제곱)
  - /연산자: 나눗셈을 한 결과 – 결과는 실수
  - //연산자: 몫을 정수로 구해줍니다.
  - x + y x더하기 y
  - x - y x빼기 y
  - x * y x 곱하기 y
  - x / y x 나누기 y의 몫을 구함 (실수형)
  - x // y x 나누기 y의 몫을 구함 (정수형)
  - x % y x를 y로 나눈 나머지
  - x ** y x의 y승

- 여러 개를 한번에 대입 가능

  a = b = 0
- 연속해서 여러 개의 변수에 값 할당 가능한데 순서대로 대입

  c, d = 3, 4

3. 연산자

- 샘플 코드

```
x=y=z=0
print(x)
print(y)
print(z)

c,d=3,4
c,d=d,c
print(c)
print(d)
```

3. 연산자

- 비교 연산자:연산의 결과가 True 또는 False
  - a = 10, b = 20 이라 가정
  - == : 값이 동일 (a == b) → False
  - != : 값이 동일하지 않음 (a != b) → True
  - > : 왼쪽 값이 오른쪽 값보다 크다 (a > b) → False
  - < : 왼쪽 값이 오른쪽 값보다 작다 (a < b) → True
  - >= : 왼쪽 값이 오른쪽 값보다 크거나 동일하다 (a >= b) → False
  - <= : 왼쪽 값이 오른쪽 값보다 작거나 동일하다 (a <= b) → True

- 비트 연산자: 정수 데이터끼리 비트 단위로 연산을 수행 한 후 결과를 10진 정수로 리턴하는 연산자
  - &: AND 연산으로 둘 다 1일 때만 1 -> (a & b) = 12 → 0000 1100
  - |: OR 연산. 둘 중 하나만 1이여도 1 -> (a | b) = 61 → 0011 1101
  - ^: XOR 연산. 두 개의 데이터가 다르면 1 -> (a ^ b) = 49 → 0011 0001
  - ~: 1의 보수 연산. (~a) = -61 → 1100 0011
  - <<: 왼쪽 시프트 연산자. 변수의 값을 왼쪽으로 지정된 비트 수 만큼 이동 a << 2 = 240 → 1111 0000
  - >>: 오른쪽 시프트 연산자. 변수의 값을 오른쪽으로 지정된 비트 수 만큼 이동 a >> 2 = 15 → 0000 1111

3. 연산자

- 논리 연산자: bool 식을 연산해서 결과를 bool 타입으로 리턴하는 연산자
  - a = True, b = False 이라 가정
  - and: 논리 AND 연산. 둘 다 참 일 때 만 참 (a and b) = False
  - or: 논리 OR 연산. 둘 중 하나만 참 이여도 참 (a or b) = True
  - not: 논리 NOT 연산. 논리 상태를 반전 not(a and b) = True

- 멤버 연산자: 데이터 모임의 멤버 인지 확인 가능한 연산자
  - a = 10, b = 10, list = [1, 2, 3, 4, 5] 라 가정
  - in: 컬렉션 내에 포함되어 있으면 참 (a in list) = False
  - Not in: 컬렉션 내에 포함되어 있지 않으면 참 (b not in list) = True

- 식별 연산자: 가리키고 있는 곳이 같은지 여부를 확인하는 연산자
  - A = 20, b = 20 이라 가정
  - Is: 개체메모리 위치나 값이 같다면 참 (a is b) = True
  - Is not: 개체메모리 위치나 값이 같지 않다면 참 (a is not b) = False

3. 연산자

- 확장 대입 연산자

  +=, -=, *=, /=, //=, %=,*=. **=, >>=, <<=, &=, ^=, &=, |=

  - ✓ += 왼쪽 변수에 오른쪽 값을 더하고 결과를 왼쪽변수에 할당 c += a → c = c + a
  - ✓ -= 왼쪽 변수에서 오른쪽 값을 빼고 결과를 왼쪽변수에 할당 c -= a → c = c - a
  - ✓ *= 왼쪽 변수에 오른쪽 값을 곱하고 결과를 왼쪽변수에 할당 c *= a → c = c * a
  - ✓ /= 왼쪽 변수에서 오른쪽 값을 나누고 결과를 왼쪽변수에 할당 c /= a → c = c / a
  - ✓ %= 왼쪽 변수에서 오른쪽 값을 나눈 나머지의 결과를 왼쪽변수에 할당 c %= a → c = c % a
  - ✓ **= 왼쪽 변수에 오른쪽 값만큼 제곱을 하고 결과를 왼쪽변수에 할당 c **= a → c = c ** a
  - ✓ //= 왼쪽 변수에서 오른쪽 값을 나눈 몫의 결과를 왼쪽변수에 할당 c //= a → c = c // a

- type(데이터): 데이터의 자료형을 문자열 리턴

- id(데이터): 저장위치를 구분하기 위한 코드 리턴

4. 내장함수(Built-in Function)

- 별도의 모듈의 추가없이 기본적으로 제공되는 함수
- 내장 함수: https://docs.python.org/3.9/library/functions.html

Python » English 3.7.7 Documentation » The Python Standard Library »

Previous topic
Introduction

Next topic
Built-in Constants

This Page
Report a Bug
Show Source

Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

4. 내장함수(Built-in Function)

❖max(s) : 시퀀스 자료형(문자열, 리스트, 튜플)을 입력받아 그 자료가 지닌 원소 중 최대값을 반환하는 함수

❖range([start,]stop[,step]): 수치 자료형으로 start, stop, step 등을 입력받아 해당 범위에 해당하는 정수를 range 타입으로 반환하는 함수

- ✓ 인수가 하나(stop)인 경우: 0부터 stop-1까지의 정수 리스트를 반환한다.
- ✓ 인수가 두 개(start, stop)인 경우: start부터 stop-1까지의 정수 리스트를 반환한다.
- ✓ 인수가 세 개(start, stop, step)인 경우: start부터 stop-1까지의 정수를 반환하되 각 정수 사이의 거리가 step인 것들만 반환한다.

4. 내장함수(Built-in Function)

```
print(max(10,30, 50))
print(max([100,300,200]))
print(max('Hello World'))
print(range(10))
print(range(3, 10))
print(range(3, 10, 2))
```

```
50
300
r
range(0, 10)
range(3, 10)
range(3, 10, 2)
```

5. 콘솔 입출력

- 콘솔 (Console): Windows에서는 Command창, 리눅스/맥에서는 Terminal창 각 IDE (예, 이클립스)에서는 별도의 콘솔 창이 제공됨

- print(데이터 나열): 화면으로 데이터를 출력할 때 가장 보편적으로 사용되는 함수로 데이터를 출력하고 줄 바꿈을 수행
- 여러 개의 데이터를 ,로 구분해서 출력할 수 있습니다.

```
print (1,2); print (3,4)
1 2
3 4
```

- end라는 매개변수를 이용해서 줄 바꿈을 대신할 문자열을 추가할 수 있음

```
print(1,2, end=' '); print(3,4)
1 2 3 4
```

- sep 매개변수를 이용해서 데이터를 구분하는 문자열을 추가할 수 있음

```
print(1,2,3,4,5, sep='□t')
1  2  3  4  5
```

5. 콘솔 입출력

- format(데이터, format_spec): format에 맞추어서 데이터를 문자열로 리턴

```
print(format(4, '10d'))
print(format(43.3, '10.3f'))
print(format('안녕하세요!', '10s'), format('반갑습니다.', '10s'))
```

- {숫자}와 format()을 이용한 데이터 매핑

```
print('{0} is {1}'. format('Python', 'fun'))
```

5. 콘솔 입출력

- input('메시지')
  - ✓ 문자열을 입력받는 내장함수
  - ✓ 매개변수로 하나의 문자열을 받는데 매개변수로 입력된 내용은 키보드의 입력을 알려주는 프롬프트
  - ✓ Enter를 누를 때까지 입력을 받아서 하나의 문자열로 리턴합니다.
  - ✓ 정수나 실수형으로 사용할 때는 문자열을 입력받고 난 후 수치형으로 변환해야 합니다.

```
name = input('이름:')
print(name)
age = int(input('나이:'))
print(age)
Print(type(name)
print(type(age))
```

이름:안화수
안화수
나이:35
35
<class 'str'>
<class 'int'>

6. 제어문

- 파이썬의 제어문

✓ 조건문
if 문 (if, if - else, if - elif - else)

✓ 반복문
while 문, for문

✓ 보조 제어문
break 문, continue 문

6. 제어문

- 프로그램은 일반적으로 위에서 아래로 한 문장씩 수행되는데 이것을 순차적인 제어 흐름이라고 합니다.

- 때로는 이러한 제어 흐름에서 벗어날 때도 있습니다.

- 건너뛰기도 해야하고 반복하기도 해야 합니다.

- 파이썬의 제어문은 if, for, while 만 존재합니다.

- 파이썬의 코드 블록 작성 규칙
  - ✓가장 바깥쪽에 있는 블록의 코드는 반드시 1열부터 시작
  - ✓내부 블록은 같은 거리만큼 들여쓰기
  - ✓블록은 { }, begin, end 등으로 구분하지 않고 들여쓰기만으로 구분
  - ✓탭과 공백을 함께 사용하는 것은 권장하지 않음
  - ✓들여쓰기 간격은 일정하기만 하면 됩니다.

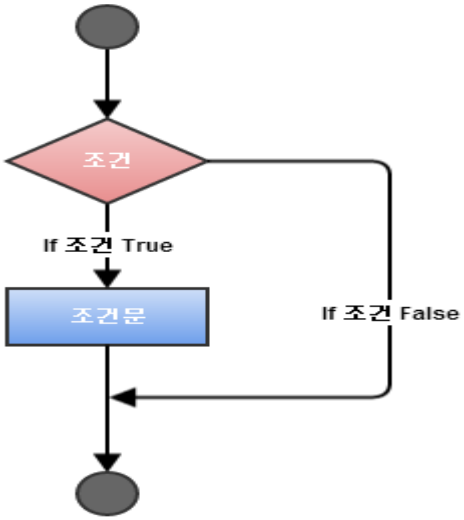
6. 제어문

❖단순if

```
if expression :
    code block;
```

❖expression

- ✓boolean 경우 : 표현식(expression)이 사실이면 true를 리턴해 if문 안쪽 문(statement)이 실행되고, 거짓이면 false를 리턴해 다음으로 진행 한다.
- ✓value 경우: non-zero, non-null → true, zero, null → false 로 리턴



6. 제어문

❖예제

```
var1 = 100
if var1:
    print("1 - true 일 때 수행될 문장")
    print(var1)

var2 = 0
if var2:
    print("2 - true 일 때 수행될 문장")
    print(var2)
print("Good bye!")
```

❖결과

```
1 - true 일 때 수행될 문장
100
Good bye!
```

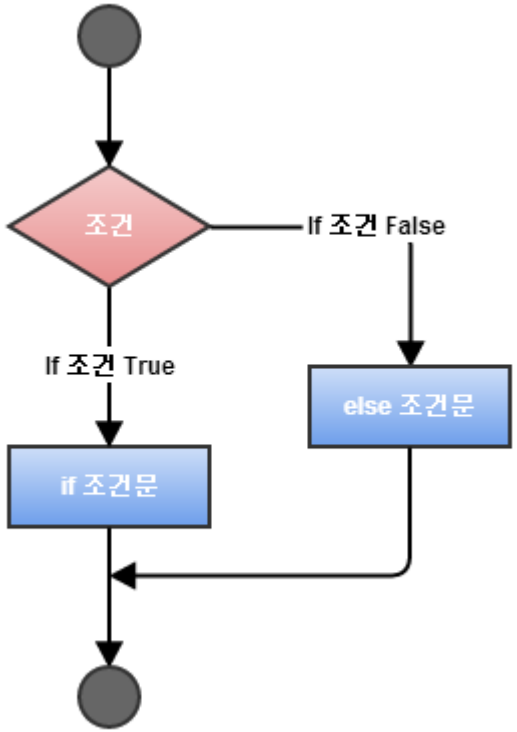
6. 제어문

❖if-else

```
if expression :
    code block;
else:
    code block;
```

❖삼항 연산자

statement if expression else statement



6. 제어문

- if-elif-else

  if와 함께 elif 절 사용

```
if 조건1:
    코드블록
    …
elif 조건2:
    코드블록
    …
elif 조건3:
    코드블록
    …
elif 조건4:
    코드블록
    …
else:
    코드블록
    …
```

첫 번째 조건은 항상 if로 시작합니다.

두 번째 조건부터는 elif를 이용합니다.

마지막의 else는 생략할 수 있습니다.

6. 제어문

❖연습
월-일 까지의 문자열을 입력하면 영문으로 요일을 표시하는 프로그램을 작성

```
dow = input('요일(월~일)을 입력하세요 : ')

if dow == '월':
    print('Monday')
elif dow == '화':
    print('Tuesday')
elif dow == '수':
    print('Wednesday')
elif dow == '목':
    print('Thursday')
elif dow == '금':
    print('Friday')
elif dow == '토':
    print('Saturday')
elif dow == '일':
    print('Sunday')
else:
    print('잘못 입력된 요일입니다.')
```

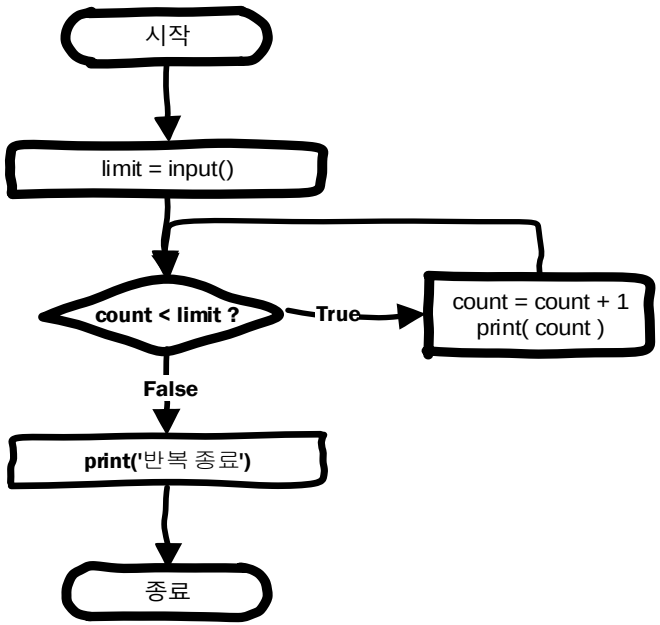
6. 제어문

❖while – 반복문

while 조건:

코드블록

=>조건이 참인 동안 코드 블록을 수행




```
print('몇 번 반복할까요? : ')
limit = int(input())

count = 0
while count < limit:
    count = count + 1
    print('{0}회 반복.'.format(count))

print('반복이 종료되었습니다.')
```

6. 제어문

- 구구단 2단 출력

```
i = 0;
while i < 9:
    i = i + 1
    print('2 * {0} = {1}'. format(i, 2*i))
```

6. 제어문

❖while – 반복문

```
while 조건:
    코드블록
else:
    코드블록
```

⇒조건이 참일 때 까지 코드 블럭 1을 수행하고 반복문 수행을 종료하거나 조건에 맞지 않으면 코드 블록 2를 수행

❖무한반복

while문의 조건이 항상 참이 되는 루프

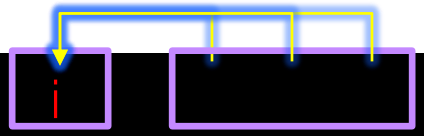
```
while True:
    코드블록
```

6. 제어문

❖for

for문은 조건을 평가하는 대신 순서열을 순회하다가 순서열의 끝에 도달하면 반복을 종료함.

```
for 반복변수 in 순서열:
    코드블록
```



변수 i에는 매 반복마다 튜플 (1, 2, 3)의 요소들이 차례대로 복사됩니다. 튜플의 길이는 3이므로 이 for문은 3번 반복을 수행합니다.

```
>for .py
1
2
3
```

6. 제어문

❖for

```
# for  변수  in  range():
#     반복 실행 문장

# range(초기값, 최종값, 증감값) : 초기값 ~ 최종값-1까지 2씩 증가
for i in range(1, 10, 2):
    print(i, end=' ')                          # 1 3 5 7 9
print()

# range(초기값, 최종값) : 초기값 ~ 최종값-1까지 1씩 증가
for i in range(1, 10):                         # 1 ~ 9
    print(i, end=' ')                          # 1 2 3 4 5 6 7 8 9
print()

# range(최종값) : 0 ~ 최종값-1까지 1씩 증가
for i in range(10):                            # 0 ~ 9
    print(i, end=' ')                          # 0 1 2 3 4 5 6 7 8 9
print()

# 10부터 2까지 1씩 감소
for i in range(10,1,-1):
    print(i, end=' ')                          # 10 9 8 7 6 5 4 3 2
```

6. 제어문

- continue
  - 반복문이 실행하는 코드블록의 나머지 부분을 실행하지 않고 다음 반복으로 건너가도록 흐름을 조정

```
for i in range(10):
    if i % 2 == 1:
        continue

    print(i)
```

i가 홀수일 때 코드블록의 나머지 부분을 실행하지 않고 다음 반복으로 바로 건너갑니다.

continue가 실행되는 경우에는 이 코드는 실행되지 않습니다.

```
>continue.py
0
2
4
6
8
```

6. 제어문

- break
  - 루프를 중단시키는 기능 수행

```
i = 0
while(True):
    i = i+1
    if i == 1000:
        print('i가 {0}이 됐습니다. 반복문을 중단합니다.'.format(i))
        break
    print(i)
```

```
>break.py
0
1
2
…
997
998
999
i가 1000이 됐습니다. 반복문을 중단합니다.
```

6. 제어문

❖제어문 안에 다른 제어문을 사용할 수 있음

❖구구단 전체 출력

```
print("★ 구구단을 출력★□n")
for x in range(1, 10):
    for y in range(2, 9):
        print(y,"X",x,"=",format(x*y,'2d'),end=" ")
    print('')
```

6. 제어문

- *을 5개씩 5개 출력

```
for x in range(0, 5):
    for y in range(0, 5):
        print('{0}'.format('*'), end='')
    print('')
```

```
*****
*****
*****
*****
*****
```

6. 제어문

❖ *을 1-5개씩 출력

```
for x in range(0, 5):
    for y in range(0, x+1):
        print('{0}'.format('*'), end='')
    print('')
```

```
*
**
***
****
*****
```

DataType

1. 자료형

int, float, complex	정수, 실수, 복소수
bool	True 또는 False를 저장
str	문자열
list	순서가 있는 데이터의 집합 : 내용 변경 가능
tuple	순서가 있는 데이터의 집합 : 내용 변경 불가능
set	순서가 없는 데이터의 집합
dict	키와 값을 쌍으로 만들어진 데이터의 집합

2. bool

- True 와 False를 저장할 수 있는 자료형
- 0이 아닌 값이나 데이터가 존재하는 시퀀스 자료형의 경우는 True로 간주하고 0이나 데이터가 존재하지 않는 시퀀스 자료형의 경우는 False
- bool 자료형끼리 산술연산 가능
- 식이나 연산의 결과를 bool 타입으로 변경하고자 하는 경우에는 bool(값이나 연산식)

```
print(bool([]))
print(bool(3))
print(bool(3+4))
```

False
True
True

3. 수치 데이터

- 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, 진법)를 이용
  - ✓ 문자열 대신에 실수를 대입하면 소수를 버립니다.
  - ✓ 실수로 된 문자열이나 복소수는 정수로 변환이 되지 않습니다.

3. 수치 데이터

❖ 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
```

3. 수치 데이터

- 실수형
  - ✓ 파이썬에서는 실수를 지원하기 위해 부동 소수형을 제공
  - ✓ 부동 소수형은 소수점을 움직여서 소수를 표현하는 자료형
  - ✓ 부동 소수형은 8바이트를 이용해서 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 메서드를 통해서 정수로 변환시의 오차 문제 확인 가능
  - ✓ 실수를 정수로 변경할 때 사용할 때 사용할 수 있는 함수

    int, round, math.ceil, math.floor

3. 수치 데이터

❖ 예제

```
import sys
print("실수 정보:", sys.float_info)
a = 1.2
b = 3e3
c=-0.2e-4
print(a, b, c)
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=1.797693134862315
    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
a: 1
a: 2
a: 2
a: 1
```

3. 수치 데이터

- 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
```

3. 수치 데이터

- 복소수
  - ✓ 실수부와 허수부로 숫자를 표현
  - ✓ 허수부는 j를 추가해서 표현
  - ✓ 복소수는 출력될 때 ( ) 안에 출력됩니다.
  - ✓ 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)
```

3. 수치 데이터

- fractions 모듈을 이용한 분수 표현
  - ✓ fractions 모듈을 이용해서 분수 표현 가능
  - ✓ 5/7을 표현하고자 할 때 Fraction('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)
```

3. 수치 데이터

- 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 Decimal
hap = Decimal(0.0)
delta = Decimal('0.1')
for i in range(0,1000):
    hap += delta
print("hap:" , hap)
```

3. 수치 데이터

- 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

3. 수치 데이터

- 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
```

4. Sequential Type

- Sequential Type
  - ✓ 객체를 순서대로 저장하는 자료형
  - ✓ 문자열(str), list(값을 변경할 수 있음), tuple(값을 변경할 수 없음)
  - ✓ 문자열은 ‘’ 또는 “” 로 묶인 문자들의 모임
  - ✓ 리스트는 [ ] 안에 묶인 데이터의 모임
  - ✓ 튜플은 ( )안에 묶인 데이터의 모임
  - ✓ Sequential Type의 공통 연산
    - 인덱싱(특정 번째에 해당하는 데이터를 가져오는 것): 변수명[인덱스] – 인덱스 번째 데이터 리턴, 음수는 뒤에서부터 진행
    - 슬라이싱(특정 범위에 해당하는 데이터를 가져오는 것): [s:t:p] – s번째부터 t번째 바로 앞 까지 p 간격으로 가져오는 기능을 수행하게 되는데 t가 생략되면 s이후 전체, p를 생략하면 순서대로 하나씩
    - 연결하기: +연산자
    - 반복하기: * 연산자
    - 멤버 검사: in 연산자
    - 데이터 개수: len(SequentialType의 데이터)

4. 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 5번째 글자 앞까지 2개씩 건너뛰면서
print(str[:-1])#-1 번째까지 가져오기
print(str[::-1])#반대로 출력
str = "Hello" + "World"
print(str)
str = str * 4
print(str)
```

```
K
e
or
Kra
Kore
aeroK
HelloWorld
HelloWorldHelloWorldHelloWorldHell
oWorld
```

5. str

- str
  - ✓ 파이썬의 문자열 자료형
  - ✓ 한 줄 문자열: ' ' 또는 " " 안에 기재하는데 단일 인용부호 안에 이중 인용부호를 이용해서 문자열을 대입할 수 있고 이중 인용부호 안에 단일 인용부호를 사용할 수 있습니다.
  - ✓ 여러 줄 문자열:''' ''' 또는 """ """ 안에 기재
- 문자열은 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)
```

5. str

- ESCAPE 코드: 이스케이프 코드란 프로그래밍할 때 사용할 수 있도록 미리 정의해 둔 "문자 조합"으로 출력물을 보기 좋게 정렬하는 용도로 이용

코드	설명
□n	개행 (줄바꿈)
□t	수평 탭
□□	문자 "□"
□'	단일 인용부호(')
□"	이중 인용부호(")
□r	캐리지 리턴
□f	폼 피드
□a	벨 소리
□b	백 스페이스
□000	널문자

이중에서 활용빈도가 높은 것은 □n, □t, □□, □', □"

5. str

- 문자열 포맷 코드 : 데이터 매핑

코드	설명
%s	모든 데이터 가능
%c	문자 1개(character)
%d	정수 (Integer)
%f	부동소수 (floating-point)
%o	8진수
%x	16진수
%%	Literal % (문자 % 자체)

- 숫자 1개를 같이 사용하면 자릿수를 확보해서 생성하는데 양수를 사용하면 오른쪽 맞춤이고 음수를 대입하면 왼쪽 맞춤을 수행하며 앞에 0을 대입하면 남는 자리는 0으로 채워줍니다.

  %10d : 10칸을 확보한 후 정수 데이터 매핑

- 실수는 숫자1.숫자2 형식으로 만들 수 있는데 이 경우 뒤의 숫자는 소수 자릿수 입니다.

5. str

- 문자열의 서식 지정
  - ✓ print() 함수를 사용하면 출력 문자열의 서식을 지정해서 출력 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))
```

5. str

- 문자열의 서식 지정
  - ✓ 문자열의 format 메서드를 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

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

  - ✓ 서식에 사용할 수 있는 보조 서식 문자
    - <: 왼쪽 맞춤
    - >: 오른쪽 맞춤
    - 공백: 양수 일 때 앞에 공백 출력
    - +: 양수 일 때 + 출력

5. str

- 문자열의 대소문자 변환 메서드
  - upper(): 대문자로 변환
  - lower(): 소문자로 변환
  - swapcase(): 대소문자 스위치
  - capitalize(): 첫 글자만 대문자로 변환
  - title: 각 단어의 첫 글자를 대문자로 변환

- 문자열의 검색 관련 메서드
  - 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

5. str

- 문자열의 편집과 치환 메서드
  - 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 메서드
  - split(): 공백으로 문자열을 분리해서 리스트로 반환
  - split('문자열'):문자열로 분리해서 리스트로 반환
  - split('문자열', 숫자):문자열로 분리하는데 숫자만큼만 분리해서 리스트로 반환
  - '문자열'.join(문자열 리스트): 문자열 리스트를 문자열을 각 요소 별로 문자열을 추가해서 결합
  - splitlines(): 줄 단위로 분리
- 맞춤 관련 메서드
  - 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

5. str

- 예제

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

```
-1
확장자: txt
c:\python\python.txt
```

5. str

- 탭 문자 변환 메서드
  - ✓ expandtabs(숫자): 숫자를 생략하면 탭을 8자 공백으로 변경하고 숫자를 대입하면 숫자만큼의 공백으로 변경

- 문자열 확인 메서드
  - ✓ isdigit()
  - ✓ isnumeric()
  - ✓ isdecimal()
  - ✓ isalpha()
  - ✓ isalnum()
  - ✓ islower()
  - ✓ isupper()
  - ✓ isspace()
  - ✓ istitle()
  - ✓ isidentifier()
  - ✓ isprintable()

5. str

- 문자열 매핑 메서드
  - ✓ 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))
```

5. 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') 메서드를 이용합니다.
  - ✓ 바이트 코드는 문자열 함수를 거의 사용 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)
```

5. 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")
```

5. 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))
```

6. 리스트

- list
  - ✓ 순서를 갖는 객체의 집합
  - ✓ 내부 데이터는 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))

6. 리스트

- 예제

```
L = [1,2,3]
print ("자료형:", type(L))
print ("길이:" , len(L))
print ("두번째 데이터:", L[1])
print ("뒤에서 첫번째:", L[-1])
print ("두번째 부터 3번째  까지:",L[1:3])
print ("리스트 결합:", L + L)
print ("리스트 반복:",L * 3)
```

```
자료형: <class 'list'>
길이: 3
두번째 데이터: 2
뒤에서 첫번째: 3
두번째 부터 3번째 까지: [2, 3]
리스트 결합: [1, 2, 3, 1, 2, 3]
리스트 반복: [1, 2, 3, 1, 2, 3, 1, 2,3]
```

6. 리스트

- 예제

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

```
[0, 1, 2, 3]
[1, 3]
[1, 2, 3]
[0, 1, 2, 3]
0
1
2
3
```

6. 리스트

- 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)
```

```
['begin', [0, 1, 2, 3], 'end']
['begin', [200, 1, 2, 3], 'end']
```

6. 리스트

- list 메서드
  - ✓ 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

6. 리스트

- 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() 메서드를 이용하면 됩니다.

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

6. 리스트

- 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) 메서드를 이용하면 됩니다.

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

6. 리스트

- 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)
```

6. 리스트

- 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)
```

6. 리스트

- 정렬
  - ✓ 사용자 정의 함수를 이용한 정렬

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

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

6. 리스트

- 정렬
  - ✓ 리스트를 정렬하지 않고 정렬한 결과를 리스트로 리턴하는 함수는 sorted()
  - ✓ key와 reverse 키워드를 동일하게 지원
  - ✓ 이 메서드는 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)
```

7. 튜플

- tuple
  - ✓ 임의 객체들이 순서를 가지는 모임으로 리스트와 유사
  - ✓ 차이점은 변경이 불가능하다는 것
  - ✓ 튜플은 시퀀스 자료형이므로 인덱싱과 슬라이싱, 연결하기, 반복하기, 길이 정보 등의 연산 가능
  - ✓ 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))
```

7. 튜플

- 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)
```

8. 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)
```

8. 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 메서드가 제공되는데 결과를 리턴합니다.
  - ✓ update(), intersection_update(), difference_update(), symmetric_difference_update()는 호출하는 객체의 데이터를 변경

8. 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))
```

9. 사전

- dict
  - ✓ 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'])
```

9. 사전

- dict
  - ✓ 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])
```

9. 사전

- dict
  - ✓ keys(): 키에 대한 모든 데이터를 리턴
  - ✓ values(): 값에 대한 모든 데이터를 리턴
  - ✓ items(): 모든 데이터 리턴
  - ✓ list()의 매개변수로 위 3개의 메서드 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

9. 사전

- OrderedDict
  - ✓ 키의 순서를 유지하는 디셔너리를 만들고자 할 때는 collections 모듈의 OrderedDict를 사용
  - ✓ dict와 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])
```