변수의 선언과 사용

변수: 값을 저장하는 메모리 공간
변수의 종류:
 정수형(int), 실수형(float), 문자열(string) 자료형, 불형(Boolean, True or False 저장)
변수명 지정:
 대소문자 구분
 문자, 숫자, _ 포함 but 첫글자≠숫자
 예약어는 사용금지 - True, False , or, and, import,...
type() 함수사용 ~ 변수형 확인
주석(comment) ~ # 사용
print() 함수 ~ print(변수1, 변수1)
 ~ print('문장')
 ~ print('문장1', 변수1, '문장2', 변수2)

eg) 편집창 or 콘솔창 에서

```
x1=5; x2=2 # 한 줄에서 여러 문장 쓸 때 " ; " 로 구분
# 위와 같은 표현:  x1, x2=5, 2
x3=5; x4=5
# 위와 같은 표현:  x3=x4=5
type(x1) # 결과는 int ~ 정수형
x1+x2 # 결과는 7 ~ 정수형
x1-x2 # 결과는 3 ~ 정수형
x1*x2 # 결과는 10 ~ 정수형
x12=x1/x2 # 결과는 2.5 ~ 실수형
print(x12)
print('x12=',x12)

x3=1.0
type(x3) # 결과는 float ~ 실수형
x4='hello'
type(x4) # 결과는 str ~ 문자열 자료형
x5=True; x6=False
type(x4) # 결과는 bool ~ 불형
# 41.2, 0.0412 대신 다음과 같이 표현도 가능
y=4.12e1; z=4.12e-2
print('y=',y, 'z=',z)
```

숫자형 연산:
사칙연산(+,-,*,/), 제곱(**), 나눗셈의 몫과 나머지(//,%)

```
y**2
x1//x2
x1%x2 # x1이 x2의 배수이면 x1%x2==0
```

```
x=1; y=1.2; z=3
print(x+z, x+y)
#x+z는 5자리 정수형으로, x+y는 전체자리수(소수점포함)=5 & 소수점이하 자리수=3로 표현
print('%5d %5.3f'%(x+z, x+y))
#x+z는 2자리 정수형으로, x+y는 전체자리수(소수점포함)=5 & 소수점이하 자리수=2로 표현
print('x+z=%2d , x+y=%5.2f'%(x+z, x+y))
```

문자열 연산:
문자열 더하기, 문자열 곱하기, 문자열 길이구하기

```
s1="Python "; s2="is fun!"
s12=s1+s2
print(s12)
s111=s1*3;
print(s111); print('s111 길이는',len(s111))
```

[진수변환]
bit-byte
2진수(0,1), 8진수(0~7), 16진수(0~9, a~f)
10진수 2 = 2진수 10
2진수: 0b로 시작
eg) 십진수 30 = 이진수 11110

```
 print(bin(30)) # 결과: 0b11110
 print(int('11110',2))
```

연산자

산술연산자: 기본적인 계산, =+-*/, //,%,**
대입연산자: =,+=,-=,*=,/=,%=,**=
eg) a+=3 ≡ a=a+3, a-=3 ≡ a=a-3
a*=3 ≡ a=a*3, a/=3 ≡ a=a/3
a//=3 ≡ a=a//3, a%=3 ≡ a=a%3, a**=3 ≡ a=a**3

관계(비교)연산자: 비교할 때 사용. 결과는 참(True) or 거짓(False)
==, !=, >,<,>=, <=
eg) a,b=100,200 ≡ a=100; b=200

```
 print(a==b, a!=b, a>b, a<b, a>=b, a<=b)
```

→ 결과: False True False True False True

논리연산자: 여러조건을 복합해서 사용. and, or, not
eg) a=99

```
 print((a>100) and (a<200), (a>100) or(a<200), not(a==100))
```

→ 결과: False True True

자료형:
프로그래밍을 할 때 쓰이는 숫자, 문자열 등 자료 형태로 사용하는 것
숫자형, 문자열, 리스트, 튜플, 딕셔너리, 집합 자료형 ~ type() 이용

1.리스트(list)
여러 원소(요소)를 묶어 하나의 변수로 활용할 때 사용(다른 자료형태 사용가능),
eg)
a1=[3, 2, 1, 4]; a2=['I', 'am', 'fine']; a3=[2, 'three', ['I', 'am', 'Sam']] # list in list
0~(k-1)번째 원소(요소)의 위치=0~(k-1)
type(a1); type(a2); type(a3)
(1) 인덱싱(indexing)

a1[0] a1[1] a1[2] a1[-1]	a2[0] a2[1] a2[2] a2[-1]	a3[0] a3[1] a3[2] a3[-1]	a3[2][0] a3[2][1] a3[2][1]

(2) 슬라이싱(slicing)

a1[0:2] a1[1:3] a1[1:] a1[:3]	a2[0:2] a2[1:3] a2[1:] a2[:3]	a3[0:2] a3[1:4] a3[1:] a3[:3]	a3[2][0:2]

(3) 연산
a1[1]=3
a1 # [3, 3, 1, 4]
a1[0]+a1[2] #4
a2[0]+a2[2] # 'Ifine'
a1[0]+a2[0] # error
a1*2; a1+a1 # 결과: [3, 2, 1, 4, 3, 2, 1, 4]

(4) 리스트 관련함수
len(), sort(), append(), index(), insert(), remove(), del
len(a1) # 4
a1.sort()
a1 # [1, 3, 3, 4]
a1.reverse()
a1 #[4, 3, 3, 1]

a1.append(8)
a1 # [4, 3, 3, 1, 8]
a1.index(4) # a1내 4의 위치, 결과:0
a1.insert(0,5) #a의 0번째 원소에 5 추가
a1 # [5, 4, 3, 3, 1, 8]
a1.pop() #a1의 마지막 원소 제거, 결과: 8
a1 # [5, 4, 3, 3, 1]
a1.count(3) # a1안에 3이 몇 개? 결과: 2
c1=a1.copy() # a1과 내용이 같은 리스트 c1 생성, c1=a1[:]

a=[4,2,5,4,3,7]
a.remove(4) # a에서 첫번째로 나오는 4 제거
a # 결과: [2,5,4,3,7]
a[1:3]=[] # a의 1~(3-1)번째 원소 제거
a # 결과: [2,3,7]
del a # a 제거
a # 결과: NameError: name 'a' is not defined

2.튜플(tuple)
여러 원소(요소)를 묶어 하나의 변수로 활용할 때 사용~ 리스트와 비슷
but 소괄호() 이용, 튜플은 값바꾸기 불가능.
eg)
t2=(10,20,30)
type(t2) # tuple
t2[0]=11 # 결과: error
t2+t2; 2*t2 # 결과: (10, 20, 30, 10, 20, 30)
len(t2)
del t2
t2 # 결과: NameError: name 't2' is not defined

3.딕셔너리(dictionary)
쌍(key & 값(value))으로 이루어진 구조의 자료형,
eg) d1={'name':'Kim', 'ht':178, 'wt': 72};
d1
d1['name'] # 'Kim'
쌍추가:
d1['phone']='010-321-4567';
d1 #{'name': 'Kim', 'ht': 178, 'wt': 72, 'phone': '010-321-4567'}
쌍제거:
del d1['wt'];
d1 #{'name': 'Kim', 'ht': 178, 'phone': '010-321-4567'}

4.집합(set)

집합을 쉽게 처리하기 위해 만든 자료형.

eg) s1=set([1,2,3]); s1
    type(s1)
    s2=set([3,1,2]); s2
    s3=set([3,1,2,2,3]); s3 # 중복되는 원소는 자동제거, 결과: {1, 2, 3}

교집합, 합집합, 차집합 구하기:

eg) s1=set(['a','b','c','d']); s2=set(['c','d','e','f','g'])

s1|s2 # 합집합 {'a', 'b', 'c', 'd', 'e', 'f', 'g'}

s1&s2 # 공통집합 {'c', 'd'}

s1-s2 # 차집합 {'a', 'b'}

s2-s1 # 차집합 {'e', 'f', 'g'}

조건문:

프로그래밍에서 조건의 참(True)-거짓(False)을 판단하여 그에 따라 프로그래밍을 실행하는 데 사용.

1. 기본if문

```
if 조건문:
  수행할 문장1 # 여러개 가능
```

조건문; 참과 거짓을 판단하는 문장

조건의 결과가 참이면 문장1 실행, 아니면 실행없음

```
eg) x=99
    if x<100:
      print('x가 100보다 작군요.')
    if x<90:
      print('x가 90보다 작군요.')
```

* 비교연산자: >, <. >=, <=, ==, !=

```
eg) x=90; y=100
    if x<y:
      print('x가 y보다 작군요.')
    if x!=y:
      print('x와 y는 다르군요.')
    if x==y:
      print('x와 y는 같군요.')
```

* 논리연산자: and, or, not

```
eg) x=90; y=100
    if x>=90 and y>=100:
      print('x는 90이상이고 y는 100이상 이군요.')
```

```
    if x>=90 or y<100:
      print('x가 90이상이거나 y가 100미만 이군요.')
    if x<90 or y<100:
      print('x가 90미만이거나 y가 100미만 이군요.')
```

2. if~else 조건문:

```
if 조건문:
  수행할 문장1
else:
  수행할 문장2
```

조건의 결과가 참이면 문장1 실행, 아니면 문장2 실행

eg1) x=90; y=100

```
    if x<y:
      print('x<y')
    else:
     print('x>=y')

    if x==y: print('x=y')
    else: print('x !=y')
```

eg2) x=90; y=100

```
    if x>=90 and y>=100:
      print('x>=90 and y>=100')
    else:
      print('x<90 or y<100')

    if x>=90 or y<100:print('x>=90 or y<100')
    else: print('x<90 and y>=100')
```

3. if 항목 in(not in) 리스트(문자열, 튜플):

```
  수행할 문장
```

~리스트(문자열, 튜플)안에 항목이 있으면(없으면), 문장 실행.

eg4)

```
fruit=['peach', 'apple', 'grape', 'orange'] # list
if 'apple' in fruit:
 print('사과가 있네요')
else:
 print('사과가 없네요')
```

eg5) #문자열
greet ='Hello, I am Sam'
if 'k' in greet: print('k 포함')
else: print('k 포함안함')

4. 조건문이 참인 경우, 아무런 실행 안하기: pass
eg6)
wallet=['photo','card','money']
if 'money' in wallet: pass
else: print('카드 사용하시오')

반복문:
실행할 문장을 반복해서 만드는 문장: for문, while문
1. 기본for문
(1) for 변수 in 리스트(또는 튜플, 문자열):
    수행할 문장 # 여러개 가능
*리스트(튜플, 문자열)의 첫 번째 원소부터 마지막 원소까지 차례로 변수에 대입되어 "수행할 문장"이 수행된다.
eg1)
x=['abc','def','ghi'] # 튜플은, x=('abc','def','ghi')
for i in x:
 print(i)

x='abcdef' #문자열
for j in x:
 print(j)

(2) for 변수 in range(시작값, 끝값+1, 증가값):
    반복 수행할 문장 # 여러개 가능
*숫자 리스트를 자동으로 만들어 주는 range 함수 이용:
eg2)
a=range(5) # 0,1,2,3,4
a=range(1,10) # 1,2,3,~,9
a1=range(1,17,2) # 1,3,5,~15 = [1~16]내의 홀수
a1[-1]
a2=range(2,17,2) # 2,4,6,~16 = [2~16]내의 짝수
a2[-1]

eg3) [0,9] 구간내 있는 수들
for i in range(10):
 print(i)

```
#[0,9] 구간내 있는 수들의 합, 평균
total=0
for i in range(10):
 total=total+i
print('합=',total)
```

2. for문 중간에 끝내기

```
eg)
for i in range(10):
  if i==5: break
  print(i)
```

3. 오류 예외 처리

```
try :
 ...
except:
 ...
eg)
x=[2, 4, 1, 5]
for k in x:
  print('2 나누기',x[i],'=',2/k)

x=[2, 4, 0, 5]
for k in x:
  print('2 나누기',x[i],'=',2/k)

for k in x:
 try: print('2 나누기',x[i],'=',2/k)
 except: print('2 나누기 0은 불가능')
```

함수(function)

(1) 함수의 결과값(반환값)이 있는 함수 ~ 일반적인 형태

```
def 함수명(매개변수):
  수행할 문장1
  수행할 문장2
      ⋮
  return 결과값(반환값)
```

* 매개변수: 함수의 입력으로 전달된 값을 받는 변수

```
#두 입력값의 합을 구하는 함수
def add(a,b):
```

```
  return a+b
add(7,9) #함수호출,  7,9=인수: 함수를 호출시 입력값
out=add(7,9) # 함수의 결과값을 out이라는 변수에 저장
print('결과=',out)
```

(2) 함수의 입력값이 없는 함수

```
def Hi():
  return 'Hi!'
z1=Hi()
print('결과=',z1)
```

(3) 함수의 입력값& 결과값(반환값)이 없는 함수

```
def Hi():
  print('Hello!')
Hi()
z2=Hi()
print('결과=',z2) #None
```

(4) 함수의 복수 결과값

```
def add_mul(a,b):
  return a+b, a*b
y=add_mul(7,3)
print(y) #=print(y[0], y[1])
type(y) # 자료형 =tuple
```