Python

담당 : 이도연

교육기간 : 2019.5.18~19.6.8

>> Python 특징

01 가독성

- 간결하고 가독성이 좋음
- 코드블럭을 들여쓰기(indentation)로 구분

02 풍부한 라이브러리

- 매우 광범위한 라이브러리가 기본으로 포함되어 있음
- 외부 라이브러리가 풍부하며 확장이 쉬움

>> Python 특징

03 접착성

- 쉽게 라이브러리를 추가할 수가 있음
- 파이썬에서 C로 구현된 부분을 사용할 수도 있으며, C에서 파이썬을 사용할 수도 있음

04 무료

- 파이썬은 파이썬 소프트웨어 재단 (Python Software Foundation)에서 관리
- 무료와 다름없는 Python Software Foundation License

>> Python 특징

05 유니코드

- 파이썬에서의 문자열들은 모두 유니코드
- 우리가 한글, 한자 등을 표현하려고 특별한 노력을 할 필요가 없음

06 동적타이핑

- 런타임 시에 타입 체크를 하는 동적타이핑을 지원
- 메모리 관리 자동으로 함

>> 2.x와 3.x의 차이

◉ print가 함수로 변경

2.x style

```
>>> print “welcome to”, “python3k”
welcome to python3k
```

3.x style

```
>>> print("welcome to", "python3k")
welcome to python3k
```

>> 2.x와 3.x의 차이

◉ long 자료형이 없어지고 int로 통일

2.x style

```
>>> type(2**31)
<type 'long'>
>>> sys.maxint
2147483647
```

3.x style

```
>>> type(2**31)
<class 'int'>
>>> type(2**40)
<class 'int'>
```

>> 2.x와 3.x의 차이

◉ 'int / int'의 결과가 float으로 처리

2.x style

```
>>> 3/2
1
```

3.x style

```
>>> 3/2
1.5
>>> type(2/2)
<class 'float'>
```

>> 2.x와 3.x의 차이

◉ String, Unicode 체계 변경

2.x style

```
>>> type('가')
<type 'str'>
>>> type(u'가')
<type 'unicode'>
```

3.x style

```
>>> type('가')
<class 'str'>
>>> type('가'.encode('cp949'))
<class 'bytes'>
```

PEP 8 의 중요한 부분에 대한 요약(1)

→ https://www.python.org/dev/peps/pep-0008/

카테고리	코딩 스타일	예제
코드 레이아웃	들여쓰기를 할 때 Tab 대신 공백(Space)을 사용한다. 특히 Python 3는 Tab과 공백을 혼용해서 사용하는 것을 허용하지 않는다.	
	문법적으로 들여쓰기를 할 때는 4개의 공백을 사용한다	
	각 라인은 79자 이하로 한다. 라인이 길어서 다음 라인으로 넘어갈 때는 원래 들여쓰기 자리에서 4개 공백을 더 들여쓴다	
	함수나 클래스는 2개의 공백 라인을 추가하여 구분한다. 메서드는 한 개의 공백 라인으로 구분한다	
	import는 (여러 모듈을 콤마로 연결하지 말고) 한 라인에 하나의 모듈을 import한다	No: import os, sys Yes: import os import sys
	컬렉션 인덱스나 함수 호출, 함수 파라미터 등에서 불필요한 공백을 넣지 않는다	No: spam(ham[1], { eggs: 2 }) bar = (0,) spam (1) Yes: spam(ham[1], {eggs: 2}) bar = (0,) spam(1)
	변수 할당시 할당자 앞뒤로 하나의 공백만 넣는다	No: i=i+1 Yes: i = i + 1

PEP 8 의 중요한 부분에 대한 요약(2)

명명규칙	함수, 변수, Attribute는 소문자로 단어 간은 밑줄(_)을 사용하여 연결한다	예: total_numbers
	클래스는 단어 첫 문자마다 대문자를 써서 연결하는 CapWords 포맷으로 명명한다	예: CoreClass
	모듈명은 짧게 소문자로 사용하며 밑줄을 쓸 수 있다. 패키지명 역시 짧게 소문자를 사용하지만 밑줄은 사용하지 않는다.	예: serial_reader
	모듈 상수는 모두 대문자를 사용하고 단어마다 밑줄로 연결하는 ALL_CAPS 포맷으로 명명한다	예: MAX_COUNT = 100
	클래스의 public attribute는 밑줄로 시작하지 말아야 한다	예: name
	클래스의 protected instance attribute는 하나의 밑줄로 시작한다	예: _initialized
	클래스의 private instance attribute는 2개의 밑줄로 시작한다	예: __private_var
	인스턴스 메서드는 (객체 자신을 가리키기 위해) self 를 사용한다	예: def copy(self, other):
	클래스 메서드는 (클래스 자신을 가리키기 위해) cls 를 사용한다	예: def clone(cls, other):

PEP 8 의 중요한 부분에 대한 요약(3)

문장과 표현식	if, for, while 블럭 문장을 한 라인으로 작성하지 말 것. 여러 라인에 걸쳐 사용하는 것이 더 명료함	No: if a < 0: a = 0 Yes: if a < 0: a = 0
	a는 b가 아니다를 표현할 때 a is not b 를 사용한다. not a is b 를 사용하지 말 것	No: if not a is b Yes: if a is not b
	값이 비어있는지 아닌지를 검사하기 위해 길이를 체크하는 방식을 사용하지 말 것. 대신 if mylist 와 같이 표현함	No: if len(mylist) == 0 Yes: if not mylist No: if len(mylist) > 0 Yes: if mylist
	import 문은 항상 파일의 상단에 위치하며, 표준 라이브러리 모듈, 3rd Party 모듈, 그리고 자신의 모듈 순으로 import 한다	예: import os import numpy import mypkg
	모듈 import시 절대 경로를 사용할 것을 권장한다. 예를 들어, sibling 모듈이 현재 모듈과 같은 폴더에 있더라도 패키지명부터 절대 경로를 사용함. 단, 복잡한 패키지 경로를 갖는 경우 상대경로(.)를 사용할 수 있다.	No: import sibling Yes: import mypkg.sibling from mypkg import sibling from . import sibling # 상대경로 from .sibling import example

예약어(Reserved Words)

- 예약어는 모두 **30**개이다.
- 예약어는 상수 또는 변수나 다른 식별자의 이름으로 사용 할 수 없다.
- 예약어는 모두 **소문자**이다.

Keyword	Keyword	Keyword
and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

식별자 (Identifiers)

- 파이썬 식별자는 변수, 함수, 클래스, 모듈 또는 다른 개체를 식별하는데 사용되는 이름이다.
- 식별자는 문자 A~Z로 시작하고 밑줄(_), 숫자(0~9) 를 사용할 수 있다.
- 파이썬은 문자 @, $, %는 식별자로 사용할 수 없다.
- 파이썬은 대소문자를 구분한다.

줄 들여쓰기 (Lines and Indentation)

- 파이썬 프로그램 시 첫번째 주의 사항.
- 클래스와 함수 정의시 코드 블럭을 표시하거나 흐름 제어 할 중괄호가 없다.
- 코드블럭은 줄 들여쓰기에 의해 엄격히 적용 된다.
- 들여쓰기는 공백(Space), 탭(Tab) 모두 가능.
- 단, 들여쓰기 공백 수는 블럭내에 동일 수로 해야 함.

소스(정상)

```
if True:
    print("True")
else:
    print("False")
```

소스(에러)

```
if True:
    print("Answer")
    print("True")
else:
    print("Answer")
  print("False")
```

멀티 라인 문 (Multi-Line Statements)

- **줄 연속 문자(₩)**의 사용으로 줄을 계속 사용할 수 있다.

소스

```
total = item_one + \
        item_two + \
        item_three
```

[], {}에 포함 된 문이나 () 괄호는 줄 연속 문자를 사용할 필요가 없다.

소스

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

인용 (Quotation)

- single('), double("), triple(''' or """) 를 사용하여 문자열을 감싸서 사용한다.
- triple 따음표는 여러 줄에 걸쳐 문자열을 사용할 수 있다.

소스

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

주석 (Comments)

- 기호 (#)를 사용해 주석을 시작한다.

소스

```
# First comment
print("Hello, Python!")  # second comment
# last comment
```

결과

```
Hello, Python!
```

```
1  # 여러 줄 주석문
2  """abcde
3  12345
4  한글"""
5  print()
6  '''abcde22
7  1234533
8  한글45'''
```

변수에 값 할당, 다중 할당 (Multiple Assignment)

- 파이썬 변수는 명시 적으로 메모리 공간을 예약 선언 할 필요가 없다.
- 변수에 값을 할당 할 때 선언은 자동으로 이루진다.
- 등호 (=)는 변수에 값을 할당하는 데 사용됩니다.
- 파이썬은 동시에 여러 변수에 하나의 값을 할당 할 수 있다.

소스

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

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

결과

```
100
1000.0
홍길동
```

소스

```
a = b = c = 1
```

- 여러 변수에 여러 개체를 할당 할 수 있다.

소스

```
a, b, c = 1, 2, "홍길동"
```

Python 기본 데이타 타입, 복소수

타입	설명	표현 예
int	정수형 데이타	100, 0xFF (16진수), 0o56 (8진수)
float	소숫점을 포함한 실수	a = 10.25
bool	참, 거짓을 표현하는 부울린	a = True
None	Null과 같은 표현	a = None

```
1  int(3.5)       # 3
2  2e3            # 2000.0
3  float("1.6")   # 1.6
4  float("inf")   # 무한대
5  float("-inf")  # -무한대
6  bool(0)        # False. 숫자에서 0만 False임,
7  bool(-1)       # True
8  bool("False")  # True
9  a = None       # a는 None
10 a is None      # a가 None 이므로 True
```

복소수는 a+bj 와 같이 표현
실수부의 값을 얻기 위해서는 복소수 변수명.real을,
허수부의 값을 얻기 위해선 변수명.imag 를 사용한다.

```
1  v = 2 + 3j
2  v.real  # 2
3  v.imag  # 3
```

복소수 – real, imag, conjugate(), abs()

```
1   # 복소수
2   a = 1+2j
3   print(a.real)   # 실수 부분
4   print(3-4j.real)
5   print(a.imag)   # 허수 부분
6   print(3-4j.imag)
7   print(a.conjugate())    # 켤레복소수
8   print(3-4j.conjugate())
9   print(abs(a))   # 복소수의 절대값
10  print(abs(3-4j))
```

결과

```
3.0
2.0
-1.0
(1-2j)
(3+4j)
2.23606797749979
5.0
```

표준 데이터 형식 (Standard Data Types)

- Numbers
- String
- List
- Tuple
- Dictionary

Numbers

- Number 타입은 숫자 값을 저장한다.
- long 타입에 소문자 l을 사용할 수도 있지만 숫자 1과의 혼동을 피하기 위해 대문자 L을 사용한다.
- int (부호있는 정수)
- long (int 보다 긴 정수, 8, 16 진수 표현가능)
- float (부동소수점이 있는 실수)
- complex (복소수)

소스

```
int_var1 = 1
int_var2 = -10

long_var1 = 0122L
long_var2 = -0x19323L

float_var1 = 15.20
float_var2 = 70.2-1E12

complex_var1 = 3.14j
complex_var2 = 4.53e1-7j
```

String

- 문자열은 인용 부호 사이에 있는 문자의 연속된 집합으로 식별 된다.
- 인용부호로 단일(') 또는 이중(") 따움표를 사용한다.
- [] [:] 을 사용하여 문자열의 하위 집합을 사용할 수 있다.
- index는 0 부터 전체 길이에서 -1 한 것 까지 이다.
- '+' 기호는 문자를 더하는 것이고 '*' 기호는 문자를 반복하는 것 이다.

소스

```
str = 'Hello World!'

print(str)
print(str[0])
print(str[2:5])
print(str[2:])
print(str * 2)
print(str + "TEST")
```

결과

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

List

- 리스트는 복합 데이터 유형에 가장 유용하다.
- 리스트 내의 항목은 쉼표(,)로 구분하고 ([]) 사각 괄호 안에 포함되어 있다.
- C의 배열과 비슷하며 리스트에 속하는 모든 항목이 서로 다른 데이터 유형이 될 수 있다.
- [] [:] 을 사용하여 리스트를 액세스 할 수 있다.
- 리스트의 index는 0 부터 전체 항목수에서 -1 한 것 까지 이다.
- '+' 기호는 연결 연산자, '*' 기호는 반복 연산자 이다.

소스

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print (list)
print (list[0])
print (list[1:3])
print (list[2:])
print (tinylist * 2)
print (list + tinylist)
```

결과

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

Tuple

- 튜플은 리스트와 유사한 또 다른 시퀀스 데이터 형식이다.
- 튜플 내의 항목은 쉼표(,)로 구분하고 (()) 괄호 안에 포함되어 있다.
- C의 배열과 비슷하며 튜플에 속하는 모든 항목이 서로 다른 데이터 유형이 될 수 있다.
- [] [:] 을 사용하여 리스트를 액세스 할 수 있다.
- 튜플의 index는 0 부터 전체 항목수에서 -1 한 것 까지 이다.
- '+' 기호는 연결 연산자, '*' 기호는 반복 연산자 이다.
- 리스트와의 차이점은 **읽기 전용**이다.

소스

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')

print (tuple)
print (tuple[0])
print (tuple[1:3])
print (tuple[2:])
print (tinytuple * 2)
print (tuple + tinytuple)
```

결과

```
('abcd', 786, 2.23, 'john', 70.200000000000003)
abcd
(786, 2.23)
(2.23, 'john', 70.200000000000003)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john')
```

Dictionary

- 사전은 해시 테이블 형식의 일종이다.
- Perl에 있는 연관 배열이나 해시처럼 작동 및 키-값(key-value) 쌍으로 구성 되어 있다.
- key는 문자열, 숫자를 많이 사용.
- value는 모든 유형과 임의의 파이썬 개체 일 수 있다.

소스

```
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

print (dict['one'])
print (dict[2])
print (tinydict)
print (tinydict.keys())
print (tinydict.values())
```

결과

```
This is one
This is two
{'dept': 'sales', 'code': 6734, 'name': 'john'}
['dept', 'code', 'name']
['sales', 6734, 'john']
```

집합 자료형 – set

- 중복을 허용하지 않는다.
- 순서가 없다
- 순서가 없기 때문에 인덱싱을 통해 자료형의 값을 얻을 수 없다

```
>>> s = set([1,2,3,4,5])
>>> s
{1, 2, 3, 4, 5}
>>> s2 = set("Happy DoYeon")
>>> s2
{'H', ' ', 'y', 'Y', 'p', 'e', 'n', 'D', 'a', 'o'}
```

```
1  myset = {1, 3, 5}
2
3  # 하나만 추가
4  myset.add(7)
5  print(myset)
6
7  # 여러 개 추가
8  myset.update({4,2,10})
9  print(myset)
10
11 # 하나만 삭제
12 myset.remove(1)
13 print(myset)
14
15 # 모두 삭제
16 myset.clear()
17 print(myset)
```

집합 자료형 활용 – 교집합, 합집합, 차집합

- & , intersection()
- | , union()
- - , difference()

```
>>> s1 = set([1,2,3,4,5,6])
>>> s2 = set([4,5,6,7,8,9])
>>> s1
{1, 2, 3, 4, 5, 6}
>>> s2
{4, 5, 6, 7, 8, 9}
>>> s1 & s2
{4, 5, 6}
>>> s1.intersection(s2)
{4, 5, 6}
>>> s1 | s2
{1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> s1.union(s2)
{1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> s1 - s2
{1, 2, 3}
>>> s2 - s1
{8, 9, 7}
>>> s1.difference(s2)
{1, 2, 3}
>>> s2.difference(s1)
{8, 9, 7}
```

참, 거짓

타입	설명	표현 예
int	정수형 데이타	100, 0xFF (16진수), 0o56 (8진수)
float	소숫점을 포함한 실수	a = 10.25
bool	참, 거짓을 표현하는 부울린	a = True
None	Null과 같은 표현	a = None

```
int(3.5)        # 3
2e3             # 2000.0
float("1.6")    # 1.6
float("inf")    # 무한대
float("-inf")   # -무한대
bool(0)         # False. 숫자에서 0만 False임,
bool(-1)        # True
bool("False")   # True
a = None        # a는 None
a is None       # a가 None 이므로 True
```

포맷팅 연산자 (Formatting Operator)

Conversion Specifier	의미
%s	문자열 (파이썬 객체를 str()을 사용하여 변환)
%r	문자열 (파이썬 객체를 repr()을 사용하여 변환)
%c	문자(char)
%d 또는 %i	정수 (int)
%f 또는 %F	부동소수 (float) (%f 소문자 / %F 대문자)
%e 또는 %E	지수형 부동소수 (소문자 / 대문자)
%g 또는 %G	일반형: 값에 따라 %e 혹은 %f 사용 (소문자 / 대문자)
%o 또는 %O	8진수 (소문자 / 대문자)
%x 또는 %X	16진수 (소문자 / 대문자)
%%	% 퍼센트 리터럴

>> 2.x와 3.x의 차이

```python
sub1 = "python string!"
sub2 = "an arg"

a = "i am a %s" % sub1
# b = "i am a {0}".format(sub1)
b = "i am a {}".format(sub1)

c = "with %(kwarg)s!" % {'kwarg':sub2}
d = "with {kwarg}!".format(kwarg=sub2)

print(a)    # "i am a python string!"
print(b)    # "i am a python string!"
print(c)    # "with an arg!"
print(d)    # "with an arg!"
```

```python
# default(implicit) order
default_order = "{}, {} and {}".format('John','Bill','Sean')
print('\n--- Default Order ---')
print(default_order)

# order using positional argument
positional_order = "{1}, {0} and {2}".format('John','Bill','Sean')
print('\n--- Positional Order ---')
print(positional_order)

# order using keyword argument
keyword_order = "{s}, {b} and {j}".format(j='John',b='Bill',s='Sean')
print('\n--- Keyword Order ---')
print(keyword_order)
```

데이터 형식 변환(Data Type Conversion)

Function	Description
int(x [,base])	Converts x to an integer. base specifies the base if x is a string.
long(x [,base])	Converts x to a long integer. base specifies the base if x is a string.
float(x)	Converts x to a floating-point number.
complex(real [,imag])	Creates a complex number.
str(x)	Converts object x to a string representation.
repr(x)	Converts object x to an expression string.
eval(str)	Evaluates a string and returns an object.
tuple(s)	Converts s to a tuple.
list(s)	Converts s to a list.
set(s)	Converts s to a set.
dict(d)	Creates a dictionary. d must be a sequence of (key,value) tuples.
frozenset(s)	Converts s to a frozen set.
chr(x)	Converts an integer to a character.
unichr(x)	Converts an integer to a Unicode character.
ord(x)	Converts a single character to its integer value.
hex(x)	Converts an integer to a hexadecimal string.
oct(x)	Converts an integer to an octal string.

변환 함수 예제

```
>>> # 정수가 나타내는 문자를 반환
>>> chr(4)
'\x04'
>>> chr(10)
'\n'

>>> # 정수를 부동소수점수로 반환
>>> float(8)
8.0

>>> # 문자를 나타내는 정수 값으로 반환
>>> ord('A')
65
>>> ord('C')
67
>>> ord('z')
122

>>> # 객체의 표현에 대하여 repr()를 사용
>>> repr(3.14)
'3.14'
>>> x = 40 * 5
>>> y = 2**8
>>> repr((x, y, ('one','two','three')))
"(200, 256, ('one', 'two', 'three'))"
```

기본 연산자 (Basic Operators)

- **산술 연산자** (Arithmetic Operators)
- **비교 (즉, 관계형) 연산자** (Comparison (i.e., Relational) Operators)
- **할당 연산자** (Assignment Operators)
- **논리 연산자** (Logical Operators)
- **비트 연산자** (Bitwise Operators)
- **구성원 연산자** (Membership Operators)
- **식별 연산자** (Identity Operators)

산술 연산자 (Arithmetic Operators)

a = 10, b = 20, c = 3 이라 가정한다.

Operator	Description	Example
+	더하기	a + b = 30
-	빼기	a - b = -10
*	곱하기	a * b = 200
/	나누기	b / a = 2.0
%	나머지	b % a = 0
**	제곱	a ** c = 1000
//	몫	a // c = 3

비교 연산자 (Comparison Operators)

a = 10, b = 20 이라 가정한다.

Operator	Description	Example
==	값이 동일하다	(a == b) → false
!=	값이 동일하지 않다	(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

할당 연산자 (Assignment Operators)

a = 10, b = 20 이라 가정한다.

Operator	Description	Example
=	왼쪽 변수에 오른쪽 값을 할당한다	c = a + b → c = a + b
+=	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

비트 연산자 (Bitwise Operators)

a = 60, b = 13 이라 가정한다.

a = 0011 1100

b = 0000 1101

Operator	Description	Example
&	AND 연산. 둘다 참일때만 만족	(a & b) = 12 → 0000 1100
|	OR 연산. 둘 중 하나만 참이여도 만족	(a | b) = 61 → 0011 1101
^	XOR 연산. 둘 중 하나만 참일 때 만족	(a ^ b) = 49 → 0011 0001
~	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

논리 연산자 (Logical Operators)

a = True, b = False 이라 가정한다.

Operator	Description	Example
and	논리 AND 연산. 둘다 참일때만 참	(a and b) = False
or	논리 OR 연산. 둘 중 하나만 참이여도 참	(a or b) = True
not	논리 NOT 연산. 논리 상태를 반전	not(a and b) = True

맴버 연산자 (Membership Operators)

a = 10, b = 10, list = [1, 2, 3, 4, 5] 라 가정한다.

Operator	Description	Example
in	list 내에 포함되어 있으면 참	(a in list) = False
not in	list 내에 포함되어 있지 않으면 참	(b not in list) = True

식별 연산자 (Identity Operators)

두 개체의 메모리 위치를 비교한다.

a = 20, b = 20 이라 가정한다.

Operator	Description	Example
is	개체메모리 위치나 값이 같다면 참	(a is b) = True
is not	개체메모리 위치나 값이 같지 않다면 참	(a is not b) = False

연산자 우선순위 (Operators Precedence)

가장 높은 우선 순위에서 가장 낮은 모든 연산자를 보여 준다.

Operator	Description
**	지수 (전원으로 인상)
~ + -	Ccomplement, 단항 플러스와 마이너스 (마지막 두의 메서드 이름은 + @이며, - @)
* / % //	곱하기, 나누기, 나머지, 몫
+ -	덧셈과 뺄셈
>> <<	좌우 비트 시프트
&	비트 'AND'
^ |	비트 전용 'OR'와 정기적 인 'OR'
<= < > >=	비교 연산자
<> == !=	평등 연산자
= %= /= //= -= += *= **=	할당 연산자
is is not	식별 연산자
in not in	맴버 연산자
not or and	논리 연산자

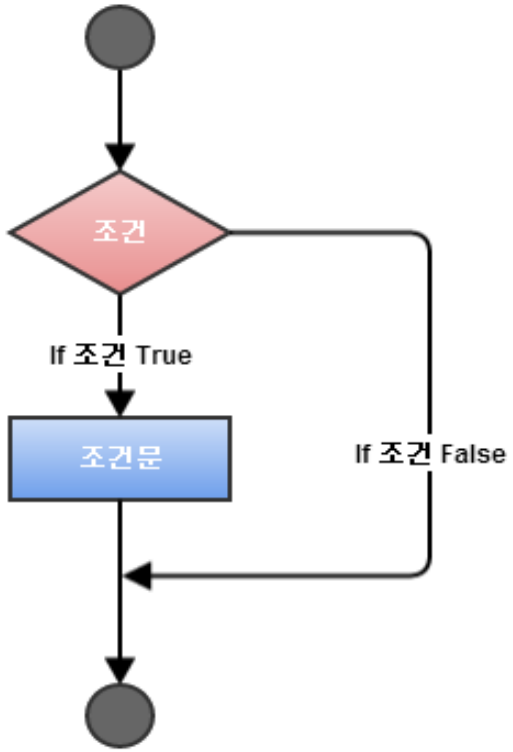
연산 순서 예제

```
>>> # 몇 가지 변수를 정의
>>> x = 10
>>> y = 12
>>> z = 14

>>> # (y*z)를 먼저 평가한 다음, x를 더함
>>> x + y * z
178

>>> # (x * y)를 먼저 평가한 다음, 그 결과에서 z를 뺌
>>> x * y - z
106

>>> # 연이은 비교는, 논리적인 'and'를 암시함

>>> # 이 경우, x < y and y <= z and z > x
>>> x < y <= z > x
True

>>> # (2 * 0)이 먼저 평가되며 그 결과는 False 또는 0이므로, 그것이 반환됨
>>> 2 * 0 and 5 + 1
0

>>> # (2 * 1)이 먼저 평가되며 그 결과는 True이거나 0 아닌 값이므로, (5 + 1)이 평가되어 반환됨
>>> 2 * 1 and 5 + 1
6

>>> # 만약 x가 참인 경우 x를 반환하며, 그렇지 않은 경우 y가 거짓이면 y를 반환.
>>> # 둘 중 어느 것도 일어나지 않는 경우에는 z를 반환.
>>> x or (y and z)
10

>>> # 이 예제에서는 'and' 'or' 논리로 인하여 (7 - 2)를 평가하여 반환
>>> 2 * 0 or ((6 + 8) and (7 - 2))
5

>>> # 이 경우는, 제곱 연산이 먼저 평가되고, 그 다음이 덧셈임
>>> 2 ** 2 + 8
12
```

제어문 - if 문

```
if expression :
    statement(s)
```

- **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



소스

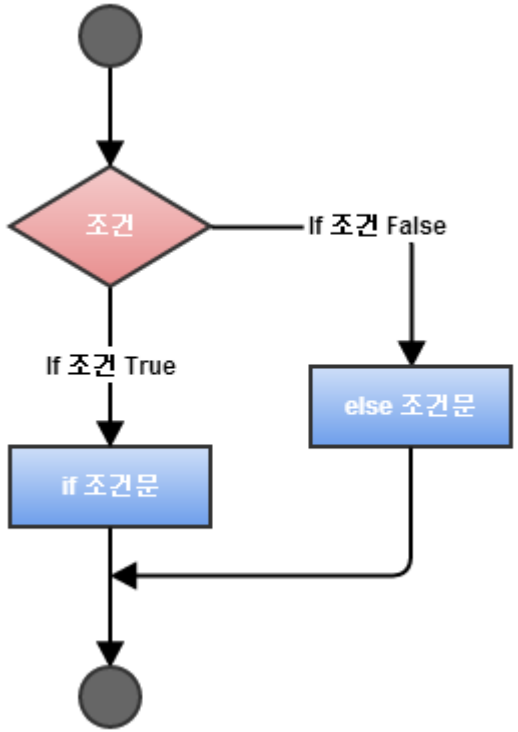
```
var1 = 100
if var1:
   print("1 - Got a true expression value")
   print(var1)

var2 = 0
if var2:
   print("2 - Got a true expression value")
   print(var2)
print("Good bye!")
```

결과

```
1 - Got a true expression value
100
Good bye!
```

if..else 문, elif 구문

```
if expression :
    statement(s)
else :
    statement(s)
```



소스

```
var1 = 100
if var1:
   print("1 - Got a true expression value")
   print(var1)
else:
   print("1 - Got a false expression value")
   print(var1)

var2 = 0
if var2:
   print("2 - Got a true expression value")
   print(var2)
else:
   print("2 - Got a false expression value")
   print(var2)

print("Good bye!")
```

결과

```
1 - Got a true expression value
100
2 - Got a false expression value
0
Good bye!
```

```
if expression1 :
    statement(s)
elif expression2 :
    statement(s)
elif expression3 :
    statement(s)
else :
    statement(s)
```

소스

```
var = 100
if var == 200:
   print("1 - Got a true expression value")
   print(var)
elif var == 150:
   print("2 - Got a true expression value")
   print(var)
elif var == 100:
   print("3 - Got a true expression value")
   print(var)
else:
   print("4 - Got a false expression value")
   print(var)

print("Good bye!")
```

결과

```
3 - Got a true expression value
100
Good bye!
```

중첩된 if 문

```
if expression1 :
   statement(s)
   if expression2 :
      statement(s)
   elif expression3 :
      statement(s)
   else
      statement(s)
elif expression4 :
   statement(s)
else:
   statement(s)
```

소스

```
var = 100
if var < 200:
   print("Expression value is less than 200")
   if var == 150:
      print("Which is 150")
   elif var == 100:
      print("Which is 100")
   elif var == 50:
      print("Which is 50")
elif var < 50:
   print("Expression value is less than 50")
else:
   print("Could not find true expression")

print("Good bye!")
```

결과

```
Expression value is less than 200
Which is 100
Good bye!
```

for문

```
for 변수 in 리스트(또는 튜플, 문자열):
    수행할 문장1
    수행할 문장2
    ...
```

```
1  sum = 0
2  for i in range(11):
3      sum += i
4  print(sum)
```

```
1  list = ["This", "is", "a", "book"]
2  for s in list:
3      print(s)
```

```
>>> test_list = ['one', 'two', 'three']
>>> for i in test_list:
...     print(i)
...
one
two
three
```

첫번째, index를 이용할 경우

```
1  fruits = ['apple', 'banana', 'pineapple', 'watermelon', 'pitch']
2
3  for i in range(0, len(fruits)):
4      print(fruits[i])
```

두번째, 배열 자체를 이용할 경우

```
1  fruits = ['apple', 'banana', 'pineapple', 'watermelon', 'pitch']
2
3  for fruit in fruits:
4      print(fruit)
```

```
scores=[90, 35, 67, 45, 88]
number =0
for check in range(len(scores)) :
    if scores[check] < 60 : continue
    print("%d번 학생은 합격." % (check + 1))

print("----------#####------------")
scores=[90, 35, 67, 45, 88]
number =0
for check in scores :
    number += 1
    if check < 60 : continue
    print("%d번 학생은 합격입니다." % number)
print("----------&&---------------")
scores=[90, 35, 67, 45, 88]
number =0
for check in scores :
    number += 1
    if check >= 60 :
        print("%d번 학생은 합격입니다." % number)
    else :
        print("%d번 학생은 불합격입니다." % number)
```

range() 함수

예제	파라미터 의미	리턴값
range(3)	Stop	0, 1, 2
range(3,6)	Start, Stop	3, 4, 5
range(2,11,2)	Start, Stop, Step	2, 4, 6, 8, 10

```
numbers = range(2, 11, 2)

for x in numbers:
    print(x)

# 출력: 각 라인에 2 4 6 8 10 출력
```

```
for i in range(10):
    print("Hello")
```

while문

```
while <조건문>:
    <수행할 문장1>
    <수행할 문장2>
    <수행할 문장3>
    ...
```

```
1  i=1
2  while i <= 10:
3      print(i)
4      i = i + 1
```

```
1  fruits = ['apple', 'banana', 'pineapple', 'watermelon', 'pitch']
2  i = 0
3  while i < len(fruits):
4      print(fruits[i])
5      i = i + 1
```

```
>>> treeHit = 0
>>> while treeHit < 10:
...     treeHit = treeHit +1
...     print("나무를 %d번 찍었습니다." % treeHit)
...     if treeHit == 10:
...         print("나무 넘어갑니다.")
...
나무를 1번 찍었습니다.
나무를 2번 찍었습니다.
나무를 3번 찍었습니다.
나무를 4번 찍었습니다.
나무를 5번 찍었습니다.
나무를 6번 찍었습니다.
나무를 7번 찍었습니다.
나무를 8번 찍었습니다.
나무를 9번 찍었습니다.
나무를 10번 찍었습니다.
나무 넘어갑니다.
```

```
>>> coffee = 10
>>> money = 300
>>> while money:
...     print("돈을 받았으니 커피를 줍니다.")
...     coffee = coffee -1
...     print("남은 커피의 양은 %d개입니다." % coffee)
...     if not coffee:
...         print("커피가 다 떨어졌습니다. 판매를 중지합니다.")
...         break
```

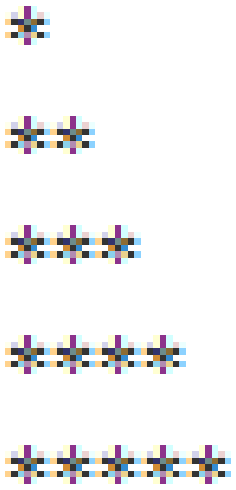
```python
# coffee.py

coffee = 10
while True:
    money = int(input("돈을 넣어 주세요: "))
    if money == 300:
        print("커피를 줍니다.")
        coffee = coffee -1
    elif money > 300:
        print("거스름돈 %d를 주고 커피를 줍니다." % (money -300))
        coffee = coffee -1
    else:
        print("돈을 다시 돌려주고 커피를 주지 않습니다.")
        print("남은 커피의 양은 %d개 입니다." % coffee)
    if not coffee:
        print("커피가 다 떨어졌습니다. 판매를 중지 합니다.")
        break
```

제어문 관련 문제

```
13  # 문제1
14  str="Life is too short, you need python"
15
16  if "wife" in str: print("whife")
17  elif "python" in str and "you" not in str: print("python")
18  elif "shirt" not in str: print("shirt")
19  elif "need" in str: print("need")
20  else : print("none")
```

결과는?

문제2] while문 이용하여 아래 결과 출력하는 프로그램 작성.



문제3] for문 이용할것

A학급에 총 10명의학생의 중간고사 점수는 아래와 같다

A=[70, 60, 55, 75, 95, 90, 80, 80, 85, 100]

학급의 평균 점수 구하는 프로그램 작성

결과 : 79.0

문제2]

```
11  # 문제2
12  su = 0
13  while True :
14      su += 1
15      if su > 5 : break
16      print('*' * su)
```

문제3]

```
1  # 문제3
2  A=[70, 60, 55, 75, 95, 90, 80, 80, 85, 100]
3  total = 0
4  for score in A :
5      total += score
6  average = total / len(A)
7  print(average)
```

>> 구구단 전체 구하는 프로그램(while,for문 2가지 방법)

```
1       # 구구단 출력하는 프로그램
2       for i in range(1,10,1):
3           for j in range(1,10,1):
4               print("{} X {} = {}" .format(i, j, (i*j)))
```

```
1       #구구단 출력하기 (while문 사용)
2       i = 1
3       while i < 10:
4           j = 1
5           while j < 10 :
6               print("{} X {} = {} " .format(i,j,(i*j)))
7               j += 1
8           i += 1
```

>> 윤년 / 평년 판정 프로그램

```
1  year = int(input("input year: "))
2
3  if year % 4 == 0 and (year % 100 != 0 or year % 400 == 0):
4      print("윤년")
5  else:
6      print("평년")
```

```
1  year = int(input('년도를 입력하세요 '))
2
3  if (year % 4 == 0) & (year % 100 ==0) & (year % 400 == 0):
4      print("윤년!")
5  elif (year % 4 == 0) & (year % 100 == 0):
6      print("평년..")
7  elif year % 4 == 0:
8      print("윤년!!!")
9  else:
10     print("아무것도 아님..")
```