파이썬 기초

파이썬의 자료형 : 숫자형

숫자형:

- 정수(integer): 123, 33, 0, -123, 등.
- 실수(floating point): 123.456, -45.123, 9.12e15, 등.
- 8진수: 0o123, 0o456, 등.
- 16진수: 0xA0, 0xFFAA, 등.

파이썬의 자료형 : 숫자형

사칙연산:

더하기, 빼기, 곱하기, 나누기는 계산기 사용하듯 기호만 넣어주면 된다:

```
In[1] : 1.5 + 1
Out[1]: 2.5
```

```
In[2] : 3.0 / 1.5
Out[2]: 2.0
```

```
In[3] : 4 * 5
Out[3]: 20
```

파이썬의 자료형 : 숫자형

기타 연산자:

제곱 (**), 나머지 (%), 나눗기 후 소수점 이하 버리기 (//):

```
In[1] : 2 ** 3
Out[1]: 8
```

```
In[2] : 8 % 5
Out[2]: 3
```

```
In[3] : 8.0 // 5.0
Out[3]: 1.0
```

파이썬의 자료형 : 숫자형

변수 사용:

변수에 숫자를 대입하여 사용할 수 있다:

```
In[1] : x = 2.1
In[2] : y = 3.5
In[3] : x + y
Out[3]: 5.6
```

파이썬의 자료형 : 문자열

문자열(string) 만들기:

큰 따옴표나 작은 따옴표 사용.

```
"Hello World"
 'Hello World'
```

~~"He said, "Python is very easy" "~~
~~'He said, 'Python is very easy' '~~

```
"He said, 'Python is very easy' "
'He said, "Python is very easy" '
```

파이썬의 자료형 : 문자열

문자열(string) 만들기:

Multi-line 문자열:

```
In[1]  : my_string = """Hello !!!
    ...: my
    ...: name
    ...: is
    ...: Python"""
In[2]  : print(my_string)
Hello !!!
my
name
is
Python
```

파이썬의 자료형 : 문자열

문자열(string) 만들기:

백슬래시(\) 사용하기:

```
"He said,□" Python is very easy □" "
'He said,□' Python is very easy □' '
```

파이썬의 자료형 : 문자열

이스케이프 코드 (escapement code):

문자열 안에서 사용되는 일종의 “특수문자”:

코드	역할
\n	줄바꿈.
\t	탭.
\\	문자 ‘\’
\'	작은 따옴표.
\"	큰 따옴표.
\r	캐리지 리턴
\a	벨소리.
\b	백 스페이스.

파이썬의 자료형 : 문자열

문자열 연산하기:

문자열 연결하기 (+), 문자열 반복하기 (*):

```
In[1] : str1 = 'First string. '
In[2] : str2 = 'Second string.'
In[3] : str1 + str2
Out[3]: 'First string. Second string.'
```

```
In[1] : str = 'Python. '
In[2] : str * 3
Out[2]: 'Python. Python. Python. '
```

파이썬의 자료형 : 문자열

문자열 인덱싱과 슬라이싱:

문자열의 일부분을 가져오기:

```
In[1] : str = 'Life is too short, You need Python!'
In[2] : str[0]
Out[2]: 'L'
```

```
In[3] : str[3]
Out[3]: 'e'
```

```
In[4] : str[-1]
Out[4]: '!'
```

파이썬의 자료형 : 문자열

문자열 인덱싱과 슬라이싱:

문자열의 일부분을 가져오기:

```
In[5] : str[0:4]                    # 0~3 번째 문자.
Out[5]: 'Life'

In[6] : str[:17]                    # 0~16 번째 문자.
Out[6]: 'Life is too short'

In[7] : str[19:]                    # 19~끝.
Out[7]: 'You need Python!'
```

파이썬의 자료형 : 문자열

문자열 관련 함수:

```
In[1] : x = 'Python'
In[2] : len(x)                    # 문자열의 길이.
Out[2]: 6
```

```
In[3] : x.count('h')              # 문자 개수 세기.
Out[3]: 1
```

```
In[4] : x.upper()                 # 대문자 변환.
Out[4]: 'PYTHON'
```

```
In[5] : x.lower()                 # 소문자 변환.
Out[5]: 'python'
```

파이썬의 자료형 : 문자열

문자열 관련 함수:

```
In[1] : x = 'Python'
In[2] : x.find('o')                    # 위치 찾기.
Out[2]: 4
```

```
In[3] : x.find('w')                    # 위치 찾기.
Out[3]: -1                             # 존재하지 않음.
```

```
In[4] : x.index('o')                   # 위치 찾기.
Out[4]: 4
```

```
In[5] : x.index('w')                   # 위치 찾기.
ValueErrorTraceback (most recent call last)
<ipython-input-46-47c892f4e96d> in <module>()
----> 1 x.index('w')
ValueError: substring not found        # 존재하지 않음.
```

파이썬의 자료형 : 문자열

문자열 관련 함수:

```
In[1] : x = 'Life is too short, You need Python!'
In[2] : x.split(' ')
Out[2]: ['Life', 'is', 'too', 'short,', 'You', 'need', 'Python!']
```

```
In[3] : y = x.split(' ')
In[4] : a = ' '
In[5] : a.join(y)
Out[5]: 'Life is too short, You need Python!'
```

파이썬의 자료형 : 문자열

문자열 관련 함수:

함수	역할
x.lstrip()	왼쪽 공백 지우기.
x.rstrip()	오른쪽 공백 지우기.
x.strip()	양쪽 공백 지우기.
x.replace(str1, str2)	문자열 바꾸기 (str1 → str2).
x.count(str)	문자 (문자열) 개수 세기.
x.find(str)	위치 알려주기. (-1)
x.index(str)	위치 알려주기. (오류)
a.join(str_list)	a 삽입 문자열 연결.
x.split(a)	문자열을 a로 토막냄.
x.upper()	대문자로 변환.
x.lower()	소문자로 변환.
len(x)	문자열의 길이.

파이썬의 자료형 : 리스트

리스트 (list):

대괄호 [ ] 로 감싸고 쉼표로 구분해 준다:

```
In[1] : a = [ ]                          # 빈 리스트 (empty list).
In[2] : b = [1, 2, 3]                    # 숫자형 원소.
In[3] : c = ['Life', 'is', 'too', 'short'] # 문자열 원소.
In[4] : d = [1, 2, 'Life', 'is']         # 숫자형과 문자열 혼재.
In[5] : e = [1, 2, ['Life', 'is']]       # 리스트안에 또다른 리스트.
```

```
In[6] : e[1]                             # 리스트의 인덱싱.
Out[6]: 2
In[7] : e[2]
Out[7]: ['Life', 'is']
In[8] : e[2][1]
Out[8]: 'is'
```

파이썬의 자료형 : 리스트

리스트의 슬라이싱 (slicing):

```
In[1] : a = [1, 2, 3, 4, 5]
In[2] : a[:2]
Out[2]: [1, 2]
In[3] : a[2:]
Out[3]:  [3, 4, 5]
In[4] : a[:]                                    # 전체!
Out[4]:  [1, 2, 3, 4, 5]
```

```
In[1] : a = [1, 2, 3, ['a', 'b', 'c'], 4, 5]
In[2] : a[2:5]
Out[2]: [3, ['a', 'b', 'c'], 4]
In[3] : a[3][:2]
Out[3]: ['a', 'b']
```

파이썬의 자료형 : 리스트

리스트 연산하기:

리스트 연결하기 (+), 리스트 반복하기 (*):

```
In[1] : a = [1, 2, 3]
In[2] : b = [4, 5, 6]
In[3] : a + b
Out[3]: [1, 2, 3, 4, 5, 6]
```

```
In[1] : a = [1, 2, 3]
In[2] : a * 3
Out[2]: [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

파이썬의 자료형 : 리스트

리스트의 변경과 삭제:

리스트의 일부 변경:

```
In[1] : a = [1, 2, 3]
In[2] : a[1] = -1
In[3] : a
Out[3]: [1, -1, 3]
```

```
In[4] : a[1:3]
Out[4]: [-1,3]
In[5] : a[1:3] = [3, 5, 7, 9, 11]
In[6] : a
Out[6]: [1, 3, 5, 7, 9, 11]
```

파이썬의 자료형 : 리스트

리스트의 변경과 삭제:

리스트의 일부 변경 (주의):

```
In[1] : a = [1, 2, 3]
In[2] : a[1:2] = [-1, -2, -3]          # 리스트의 일부 변경.
In[3] : a
Out[3]: [1, -1 , -2, -3, 3]
```

```
In[1] : a = [1, 2, 3]
In[2] : a[1] = [-1, -2, -3]            # 리스트의 특정 원소 변경.
In[3] : a
Out[3]: [1, [-1 , -2, -3], 3]
```

파이썬의 자료형 : 리스트

리스트의 변경과 삭제:

리스트의 일부 삭제:

```
In[1] : a = [1, 2, 3, 4, 5]
In[2] : a[1:3] = []                    # 리스트의 일부 삭제. 주의!
In[3] : a
Out[3]: [1, 4, 5]
```

```
In[1] : a = [1, 2, 3, 4, 5]
In[2] : del a[2]                       # 리스트의 원소 삭제.
In[3] : a
Out[3]: [1, 2, 4, 5]
```

파이썬의 자료형 : 리스트

리스트 관련 함수:

```
In[1] : a = [3, 1, 5, 2, 4]
In[2] : a.sort()                    # (항구적) 정렬.
In[3] : a
Out[3]: [1, 2, 3, 4, 5]
```

```
In[1] : a = [1, 2, 3]
In[2] : a.append(4)                 # 숫자형 값을 원소로 추가.
In[3] : a.append([5,6])             # 리스트를 원소로 추가.
In[4] : a
Out[4]: [1, 2, 3, 4, [5, 6]]
```

파이썬의 자료형 : 리스트

리스트 관련 함수:

```
In[1] : a = [3, 1, 5, 2, 4]
In[2] : a.sort(reverse=True)                    # 역정렬.
In[3] : a
Out[3]: [5, 4, 3, 2, 1]
```

```
In[1] : a = [1, 2, 3, 4, 5]
In[2] : a.reverse()                             # 뒤집기.
In[3] : a
Out[3]: [5, 4, 3, 2, 1]
```

파이썬의 자료형 : 리스트

리스트 관련 함수:

```
In[1] : a = [3, 1, 5, 2, 4]
In[2] : sorted(a)                    # 정렬하여 보여준다.
Out[2]: [1, 2, 3, 4, 5]
In[3] : a
Out[3]: [3, 1, 5, 2, 4]              # 정렬 효과는 항구적이지 않음.
```

파이썬의 자료형 : 리스트

리스트 관련 함수:

함수	역할
x.insert(pos, val)	pos 위치에 val 삽입.
x.remove(val)	val 값을 리스트에서 제거.
x.pop()	마지막 원소를 끄집어 냄.
x.count(val)	val 의 횟수.
x.extend(list_y)	리스트 확장: x = x + list_y
x.append(val)	리스트에 원소 추가.
x.sort()	리스트 정렬.
x.reverse()	리스트 뒤집기.
x.index(val)	val의 위치. (오류)
len(x)	리스트의 원소 갯수.

파이썬의 자료형 : 튜플

튜플 (tuple):

- 리스트와 튜플은 비슷한 면이 많다.
- 리스트는 [ ] 로 둘러싸는데 튜플은 ( )로 둘러싼다.
- 리스트와는 다르게 튜플은 그 값을 바꿀 수 없다.

파이썬의 자료형 : 튜플

튜플 (tuple):

괄호 ( ) 로 감싸고 쉼표로 구분해 준다:

```
In[1] : a = ( )                          # 빈 튜플 (empty tuple).
In[2] : b = (1, 2, 3)                    # 숫자형 원소.
In[3] : c = ('Life', 'is', 'too', 'short')   # 문자열 원소.
In[4] : d = (1, 2, 'Life', 'is')         # 숫자형과 문자열 혼재.
In[5] : e = (1, 2, ('Life', 'is'))       # 튜플의 원소가 또다른 튜플.
```

```
In[6] : e[1]                             # 튜플의 인덱싱.
Out[6]: 2
In[7] : e[2]
Out[7]: ('Life', 'is')
In[8] : e[2][1]
Out[8]: 'is'
```

파이썬의 자료형 : 튜플

튜플의 슬라이싱 (slicing):

```
In[1] : a = (1, 2, 3, 4, 5)
In[2] : a[:2]
Out[2]: (1, 2)
In[3] : a[2:]
Out[3]:  (3, 4, 5)
```

```
In[1] : a = (1, 2, 3, ('a', 'b', 'c'), 4, 5)
In[2] : a[2:5]
Out[2]: (3, ('a', 'b', 'c'), 4)
In[3] : a[3][:2]
Out[3]: ('a', 'b')
```

파이썬의 자료형 : 튜플

튜플 연산하기:

튜플 연결하기 (+), 튜플 반복하기 (*):

```
In[1] : a = (1, 2, 3)
In[2] : b = (4, 5, 6)
In[3] : a + b
Out[3]: (1, 2, 3, 4, 5, 6)
```

```
In[1] : a = (1, 2, 3)
In[2] : a * 3
Out[2]: (1, 2, 3, 1, 2, 3, 1, 2, 3)
```

파이썬의 자료형 : 튜플

튜플 변경과 삭제 불가능:

다음과 같은 경우 오류 발생:

```
In[1] : a = (1, 2, 3)
In[2] : a[1] = -1

TypeErrorTraceback (most recent call last)
<ipython-input-134-c684cd0a8714> in <module>()
----> 1 a[1] = -1
TypeError: 'tuple' object does not support item assignment          # 오류 발생.
```

```
In[1] : a = (1, 2, 3)
In[2] : del a[1]

TypeErrorTraceback (most recent call last)
<ipython-input-135-dbf82171d8ac> in <module>()
----> 1 del a[1]
TypeError: 'tuple' object doesn't support item deletion             # 오류 발생.

In[3] : del a                                                       # 객체 a 전체를 삭제. OK!
```

파이썬의 자료형 : 딕셔너리

딕셔너리 (dictionary):

- 연관 배열 (associative array) 라고도 불리운다.
- 딕셔너리의 원소는 key 와 value로 이루어진 짝이다.

  {key1: value1, key2:value2, key3:value3,…}
- Key는 배열의 “인덱스” 역할을 한다.
- 리스트, 튜플, 문자열 등과는 다르게 +, * 연산은 불가능하다.

파이썬의 자료형 : 딕셔너리

딕셔너리 (dictionary):

중괄호 { } 로 감싸고 key-value 짝은 ':' 로 구분된다:

```
In[1] : a = { }                                  # 빈 딕셔너리 (empty dictionary).
In[2] : b = {'이름': '홍길동', '성별': '남', '나이': 35}   # Value로 문자열, 숫자형 혼재.
In[3] : c = {1: 'Life', 2: 'is', 3: 'short' }
In[4] : d = {'x':[1,2,3], 'y':[4,5,6]}           # 리스트가 value로.
In[5] : e = {1:b, 2:c}                           # 딕셔너리 안에 다른 딕셔너리가 value로.
```

```
In[6] : b['이름']                                 # 딕셔너리의 인덱싱.
Out[6]: '홍길동'
In[7] : e[2]
Out[7]: {1: 'Life', 2: 'is', 3: 'short' }
In[8] : e[2][3]
Out[8]: 'short'
```

파이썬의 자료형 : 딕셔너리

딕셔너리 짝 추가, 삭제, 변경:

```
In[1] : a = {}                                  # 빈 딕셔너리 (empty dictionary).
In[2] : a['이름'] = '홍길동'                      # 짝 추가.
In[3] : a['성별'] = '남'                          # 짝 추가.
In[4] : a['나이'] = 35                            # 짝 추가.
In[5] : a
Out[5]: {'이름': '홍길동', '성별': '남', '나이': 35}
```

```
In[6] : del a['나이']                           # 짝 삭제.
In[7] : a
Out[7]: {'이름': '홍길동', '성별': '남'}
In[8] : a['이름'] = '임꺽정'                      # Value 수정.
In[9] : a
Out[9]: {'이름': '임꺽정', '성별': '남'}
```

파이썬의 자료형 : 딕셔너리

딕셔너리 관련 함수:

```
In[1] : a = {'이름': '홍길동', '성별': '남', '나이': 35}
In[2] : a.keys()                                    # key 리스트 (Python 2.x).
Out[2]: ['이름', '성별', '나이']
```

```
In[3] : a.values()                                  # value 리스트 (Python 2.x).
Out[3]: ['홍길동', '남', 35]
```

```
In[4] : a.items()                                   # key-value 짝 (터플)의 리스트 (Python 2.x).
Out[4]: [('이름', '홍길동'), ('성별', '남'), ('나이', 35)]
```

```
In[5] : a.clear()                                   # 모든 항목을 지움.
In[6] : a
Out[6]: {}
```

파이썬의 자료형 : 딕셔너리

딕셔너리 관련 함수:

```
In[1] : a = {'name': 'JONE', 'gender': 'MALE', 'age': 35}
In[2] : a['wage']
KeyErrorTraceback (most recent call last)
<ipython-input-189-127109090150> in <module>()
----> 1 a['wage']

KeyError: 'wage'                                    # 오류 발생
```

```
In[3] : a.get('name')                               # a['name']와 같음.
Out[3]: 'JONE'
In[4] : a.get('wage')                               # a['wage']와 같음.
                                                    # None을 리턴하며 오류는 발생하지 않음.
```

```
In[5] : a.get('wage', 0)                            # 디폴트값 (0) 설정.
Out[5]: 0                                           # Key가 딕셔너리에 없으니까 디폴트값 리턴.
```

파이썬의 자료형 : 딕셔너리

딕셔너리 관련 함수:

해당 key가 딕셔너리에 포함되어 있는지 확인 (in):

```
In[1] : a = {'name': 'JOHN', 'gender': 'MALE', 'age': 35}
In[2] : 'name' in a                                    # key 포함 확인.
Out[2]: True
In[3] : 'wage' in a
Out[3]: False
```

파이썬의 자료형 : 집합

집합 (set):

- 원소의 중복을 허용하지 않는다.
- 집합에는 정렬 (순서)의 개념이 없다.
- 그러므로 인덱싱의 개념도 없다.

파이썬의 자료형 : 집합

집합 (set):

중괄호 { } 로 감싸서 표기되지만 set() 함수를 사용하여 생성한다:

```
In[1] : a = set()                          # 빈 집합 (empty set).
In[2] : b = set([1,2,3,3,3,3,4,5])         # 리스트를 입력하여 초기화 할 수 있다.
In[3] : b
Out[3]: {1,2,3,4,5}                        # 중복이 없음!
In[4] : type(b)
Out[3]: set
```

파이썬의 자료형 : 집합

집합 연산하기:

교집합, 합집합, 차집합 구하기:

```
In[1] : s1 = set([1, 2, 3, 4, 5])
In[2] : s2 = set([4, 5, 6, 7, 8])
In[3] : s1 & s2
Out[3]: {4, 5}                          # 교집합.
In[4] : s1.intersection(s2)
Out[4]: {4, 5}                          # 교집합.
```

```
In[5] : s1 | s2
Out[5]: {1, 2, 3, 4, 5, 6, 7, 8}        # 합집합.
In[6] : s1.union(s2)
Out[7]: {1, 2, 3, 4, 5, 6, 7, 8}        # 합집합.
```

파이썬의 자료형 : 집합

집합 연산하기:

교집합, 합집합, 차집합 구하기:

```
In[8] : s1 - s2
Out[8]: {1, 2, 3}                    # 차집합.
In[9] : s1.difference(s2)
Out[9]: {1, 2, 3}                    # 차집합.
In[10] : s2 – s1
Out[10]: {6, 7, 8}                   # 차집합
```

파이썬의 자료형 : 집합

집합 관련 함수:

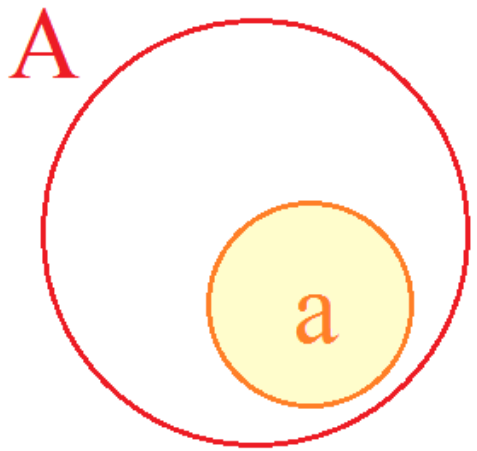
```
In[1] : a = set([1, 2, 3, 4, 5])
In[2] : a.add(6)                          # 하나의 값 추가.
In[3] : a
Out[3]: {1, 2, 3, 4, 5, 6}
```

```
In[4] : a.update([7, 8, 9])               # 여러개의 값 동시에 추가.
In[5] : a
Out[5]: {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
In[6] : a.remove(9)                       # 특정값 삭제.
In[7] : a
Out[7]: {1, 2, 3, 4, 5, 6, 7, 8}
```

수학 기초지식 : 집합

집합 (벤 다이어그램): 부분집합



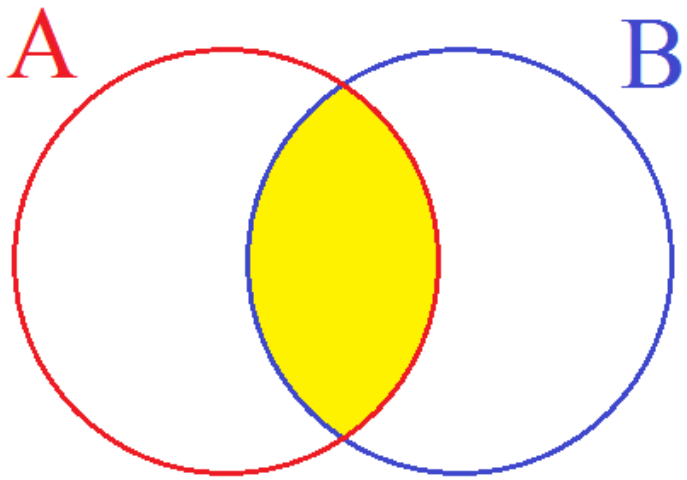
예). $A = \{1, 2, 3, 4, 5, 6\}$

$a = \{3,4\}$

$a \subset A$

수학 기초지식 : 집합

집합 (벤 다이어그램): 교집합



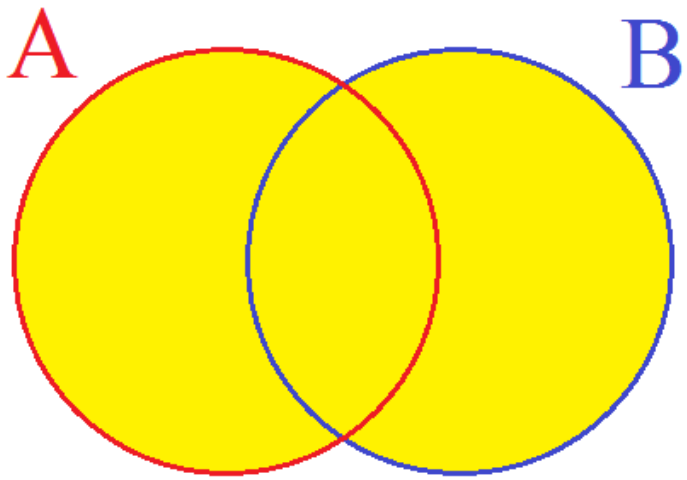
예). $A = \{1, 2, 3, 4\}$

$B = \{3, 4, 5, 6\}$

$A \cap B = \{3, 4\}$

수학 기초지식 : 집합

집합 (벤 다이어그램): 합집합



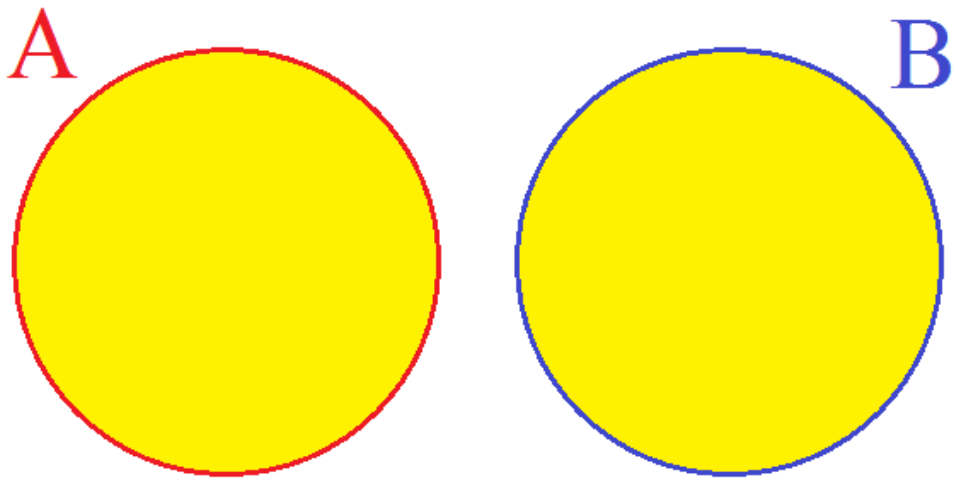
예). $A = \{1, 2, 3, 4\}$

$B = \{3, 4, 5, 6\}$

$A \cup B = \{1, 2, 3, 4, 5, 6\}$

수학 기초지식 : 집합

집합 (벤 다이어그램): 합집합



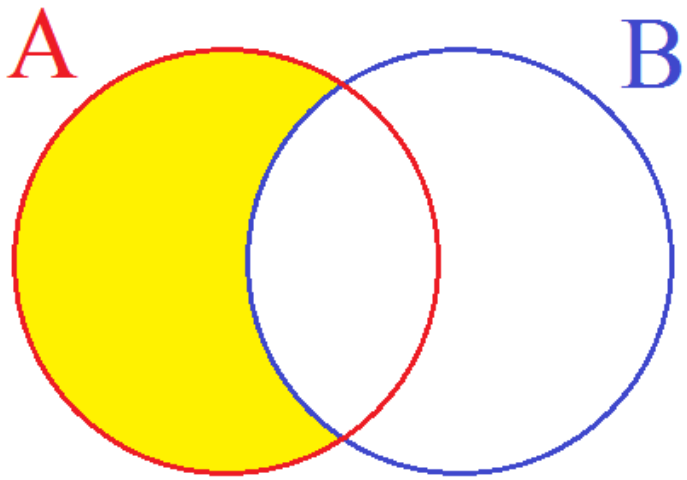
예). $A = \{1, 2, 3\}$

$B = \{4, 5, 6\}$

$A \cup B = \{1, 2, 3, 4, 5, 6\}$, $A \cap B = \phi$

수학 기초지식 : 집합

집합 (벤 다이어그램): 차집합



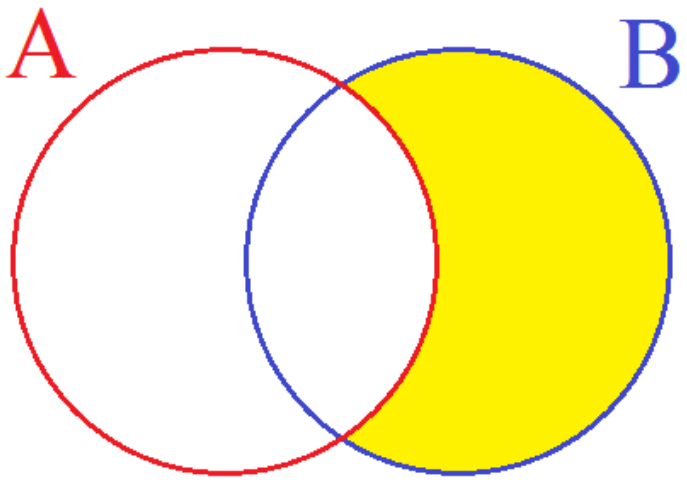
예). $A = \{1, 2, 3, 4\}$

$B = \{3, 4, 5, 6\}$

$A - B = \{1, 2\}$

수학 기초지식 : 집합

집합 (벤 다이어그램): 차집합



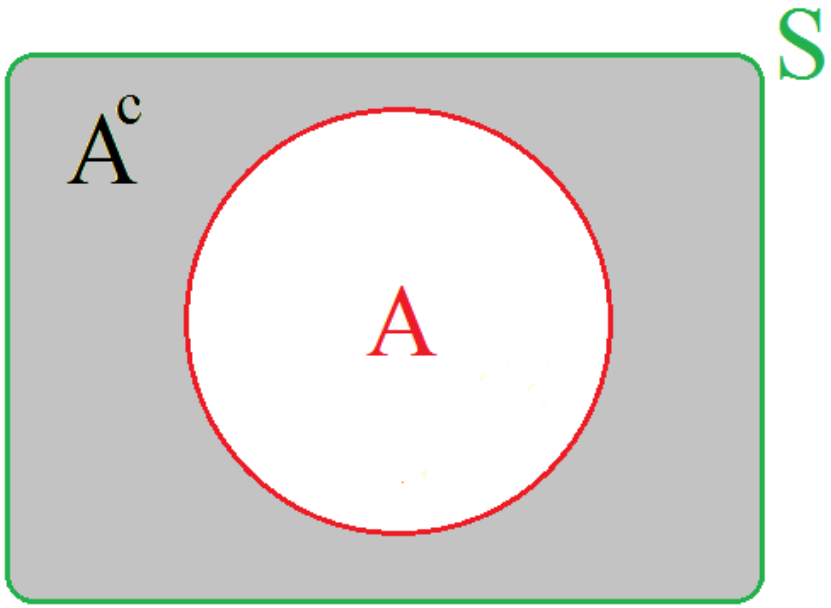
예). $A = \{1, 2, 3, 4\}$

$B = \{3, 4, 5, 6\}$

$B - A = \{5, 6\}$

수학 기초지식 : 집합

집합 (벤 다이어그램): 여집합



예). $S = \{1, 2, 3, 4, 5, 6\}$

$A = \{3, 4\}$

$A^c = S - A = \{1, 2, 5, 6\}$

파이썬의 자료형 : 불 자료형

불 자료형 (boolean):

- 참 (True)과 거짓 (False) 두 가지 값을 나타내는 자료형.
- 조건문의 리턴값으로 사용되기도 한다:

  2 == 3 False

  1 < 2 True

파이썬의 자료형 : 불 자료형

불 자료형의 연산:

AND, OR, NOT 연산:

```
In[1] : True and False                    # AND.
Out[1]: False
In[2] : True and True
Out[2]: True
In[3] : False or False
Out[3]: False                             # OR.
In[4] : False or True
Out[4]: True
In[5] : not False                         # NOT.
Out[5]: True
```

파이썬의 자료형 : 불 자료형

불 자료형의 연산:

AND의 로직 테이블:

X	Y	X and Y
False	False	False
False	True	False
True	False	False
True	True	True

파이썬의 자료형 : 불 자료형

불 자료형의 연산:

OR의 로직 테이블:

X	Y	X or Y
False	False	False
False	True	True
True	False	True
True	True	True

파이썬의 자료형 : 불 자료형

불 자료형의 연산:

NOT의 로직 테이블:

X	not X
False	True
True	False

파이썬의 자료형 : 불 자료형

자료형의 참과 거짓:

자료	True/False
"Python"	**True**
""	False
[ ]	False
( )	False
{ }	False
[1, 2, 3]	**True**
0	False
1	**True**
None	False

파이썬의 자료형 : 변경 가능성

변경 불가능한 자료형 (immutable):

- 숫자형, 불, 문자열, 튜플, 등.
- 동일 변수에 새로운 값을 대입하면 새로운 “객체” 생성됨. 주소 (address) 변경됨.

파이썬의 자료형 : 변경 가능성

변경 불가능한 자료형 (immutable):

```
In[1] : a = 999                    # 숫자형.
In[2] : b = a
In[3] : id(a)
Out[3]: 60560760L                  # 주소 (address).
In[4] : id(b)
Out[4]: 60560760L                  # 같은 주소 확인.
```

```
In[5] : b = 0                      # 변경은 불가능하고 완전히 새로운 객체 생성.
In[6] : b
Out[6]: 0
In[7] : a
Out[7]: 999
In[8] : id(b)
Out[8]: 32333952L                  # 주소 변경!
```

파이썬의 자료형 : 변경 가능성

변경 불가능한 자료형 (immutable):

```
In[1] : a = 'abcd'                    # 문자열.
In[2] : b = a
In[3] : id(a)
Out[3]: 87535088L                     # ID.
In[4] : id(b)
Out[4]: 87535088L                     # 같은 ID 확인.
```

```
In[5] : b = b + 'e'                   # 변경은 불가능하고 완전히 새로운 객체 생성.
In[6] : b
Out[6]: 'abcde'
In[7] : a
Out[7]: 'abcd'
In[8] : id(b)
Out[8]: 87535368L                     # ID 변경!
```

파이썬의 자료형 : 변경 가능성

변경 가능한 자료형 (mutable):

- 리스트, 딕셔너리, 집합 등.
- 내용 변경 이후에도 동일한 객체와 주소 (address) 유지.
- 객체를 새롭게 생성하는 경우에만 새로운 주소가 주어짐.

파이썬의 자료형 : 변경 가능성

변경 가능한 자료형 (mutable):

```
In[1] : a = [1, 2, 3, 4, 5]                    # 리스트 객체.
In[2] : b = a
In[3] : id(a)
Out[3]: 87729352L                              # ID.
In[4] : id(b)
Out[4]: 87729352L                              # 같은 ID 확인.
```

```
In[5] : b[0] = -999                            # 리스트 b의 원소값 변경.
In[6] : b
Out[6]: [-999, 2, 3, 4, 5]
In[7] : a
Out[7]: [-999, 2, 3, 4, 5]                     # 리스트 a의 원소값도 같이 변경됨!
In[8] : id(b)
Out[8]: 87729352L                              # 주소 그대로 유지됨!
```

파이썬의 자료형 : 변경 가능성

Immutable 안의 mutable 자료:

```
In[1] : a = (1, 2, [3, 4] )          # 튜플안에 리스트 원소가 있는 복합객체.
In[2] : b = a
In[3] : id(a)
Out[3]: 86385864L                    # ID.
In[4] : id(b)
Out[4]: 86385864L                    # 같은 ID 확인.
```

```
In[5] : b[2][0] = -999               # 변경 가능!
In[6] : b
Out[6]: (1, 2, [-999,4])
In[7] : a
Out[7]: (1, 2, [-999,4])             # 리스트 a의 원소값도 같이 변경됨!
In[8] : id(b)
Out[8]: 86385864L                    # ID는 그대로 유지됨!
```

파이썬의 자료형 : 변경 가능성

Immutable 안의 mutable 자료:

```
In[9]  : b[2] = [4, 5]

TypeErrorTraceback (most recent call last)
<ipython-input-359-b52acb363bbe> in <module>()
---> 1 b[2] = [4,5]
TypeError: 'tuple' object does not support item assignment
```

튜플의 원소 자체는 변경 불가.

파이썬의 자료형 : 변수

변수 (variable):

```
In[1] : a = 'abcd'                    # 문자열.
In[2] : b = 3
In[3] : type(a)                       # 자료형 확인.
Out[3]: str
In[4] : id(a)
Out[4]: 87535088L                     # 주소 (address).
In[5] : id(b)
Out[5]: 60535011L                     # 다른 주소 (객체).
In[6] : c = b
In[7] : b is c
Out[7]: True
In[8] : a is b
Out[8]: False
In[9] : del a                         # 객체 a 삭제.
```

파이썬의 자료형 : 변수

변수 (variable):

```
In[1] : a, b, c = (111, True, 'aaa')          # 동시에 여러 번수 정의.
In[2] : a, b, c = [111, True, 'aaa']          # 동시에 여러 번수 정의.
In[3] : a = b = c = 777                       # 동일값의 여러 변수 정의.
In[4] : x, y = 666, 777
In[5] : x, y = y, x                           # 변수값 서로 swapping 하기.
In[6] : x, y
Out[6]: (777, 666)
```

파이썬의 자료형 : 객체의 복사 방법

객체를 복사 (복제)하는 방법:

1). 객체의 단순 복제: 이름만 다를 뿐 동일한 주소 (address) 공유.

2). 얕은 복사 (shallow copy): 새로운 복합객체를 생성하되 원소는 기존의 객체임.

3). 깊은 복사 (deep copy): 새로운 복합객체를 생성하며 원소또한 새로운 객체임.

파이썬의 자료형 : 객체의 복사 방법

객체를 복사 (복제)하는 방법:

서로 다른 두개의 객체:

```
In[1] : a = {'name': 'JOHN', 'gender': 'MALE', 'age': 35}          # 딕셔너리 객체.
In[2] : b = {'name': 'JOHN', 'gender': 'MALE', 'age': 35 }         # 전혀 다른 객체.
In[3] : a['name'] = 'JACK'
In[4] : a
Out[4]: {'age': 35, 'gender': 'MALE', 'name': 'JACK'}
In[5] : b
Out[5]: {'age': 35, 'gender': 'MALE', 'name': 'JOHN'}
In[6] : id(a)
Out[5]: 86505672L
In[7] : id(b)
Out[5]: 86254184L
```

파이썬의 자료형 : 객체의 복사 방법

객체를 복사 (복제)하는 방법:

1). 객체의 단순 복제:

```
In[1] : a = {'name': 'JOHN', 'gender': 'MALE', 'age': 35}        # 딕셔너리 객체.
In[2] : b = a                                                    # 객체의 단순 복제.
In[3] : a['name'] = 'JACK'
In[4] : a
Out[4]: {'age': 35, 'gender': 'MALE', 'name': 'JACK'}
In[5] : b
Out[5]: {'age': 35, 'gender': 'MALE', 'name': 'JACK'}            # 이름만 다른 사실상 동일 객체.
In[6] : id(a)
Out[5]: 86349208L
In[7] : id(b)
Out[5]: 86349208L
```

파이썬의 자료형 : 객체의 복사 방법

객체를 복사 (복제)하는 방법:

2). 얕은 복사 (shallow copy):

```
In[1] : import copy                          # 모듈 불러옴.
In[1] : a = [1, 2, [3, 4, 5]]                # 리스트 복합 객체.
In[2] : b = copy.copy(a)                     # 얕은 복사.
In[3] : a[0] = 666
In[4] : a
Out[4]: [666, 2, [3, 4, 5]]                  # immutable 원소인 1이 새로운 객체 666으로.
In[5] : b
Out[5]: [1, 2, [3, 4, 5]]                    # OK!
```

파이썬의 자료형 : 객체의 복사 방법

객체를 복사 (복제)하는 방법:

2). 얕은 복사 (shallow copy):

```
In[1] : import copy                          # 모듈 불러옴.
In[1] : a = [1, 2, [3, 4, 5]]                # 리스트 복합 객체.
In[2] : b = copy.copy(a)                     # 얕은 복사.
In[3] : a[2][0] = 0
In[4] : a
Out[4]: [1, 2, [0, 4, 5]]
In[5] : b
Out[5]: [1, 2, [0, 4, 5]]                    # 복합 객체의 mutable 원소는 그대로 공유!!!
In[6] : id(a[2])
Out[6]: 86206856L
In[7] : id(b[2])
Out[7]: 86206856L
```

파이썬의 자료형 : 객체의 복사 방법

객체를 복사 (복제)하는 방법:

2). 리스트의 얕은 복사 (shallow copy):

```
In[1] : a = [1, 2, [3, 4, 5]]          # 리스트 복합 객체.
In[2] : b = a[:]                       # 얕은 복사.
In[3] : a[0] = 0
In[4] : a[2][0] = -999
In[5] : a
Out[5]: [0, 2, [-999, 4, 5]]
In[6] : b
Out[6]: [1, 2, [-999, 4, 5]]           # 복합 객체의 원소는 그대로 공유!!!
```

파이썬의 자료형 : 객체의 복사 방법

객체를 복사 (복제)하는 방법:

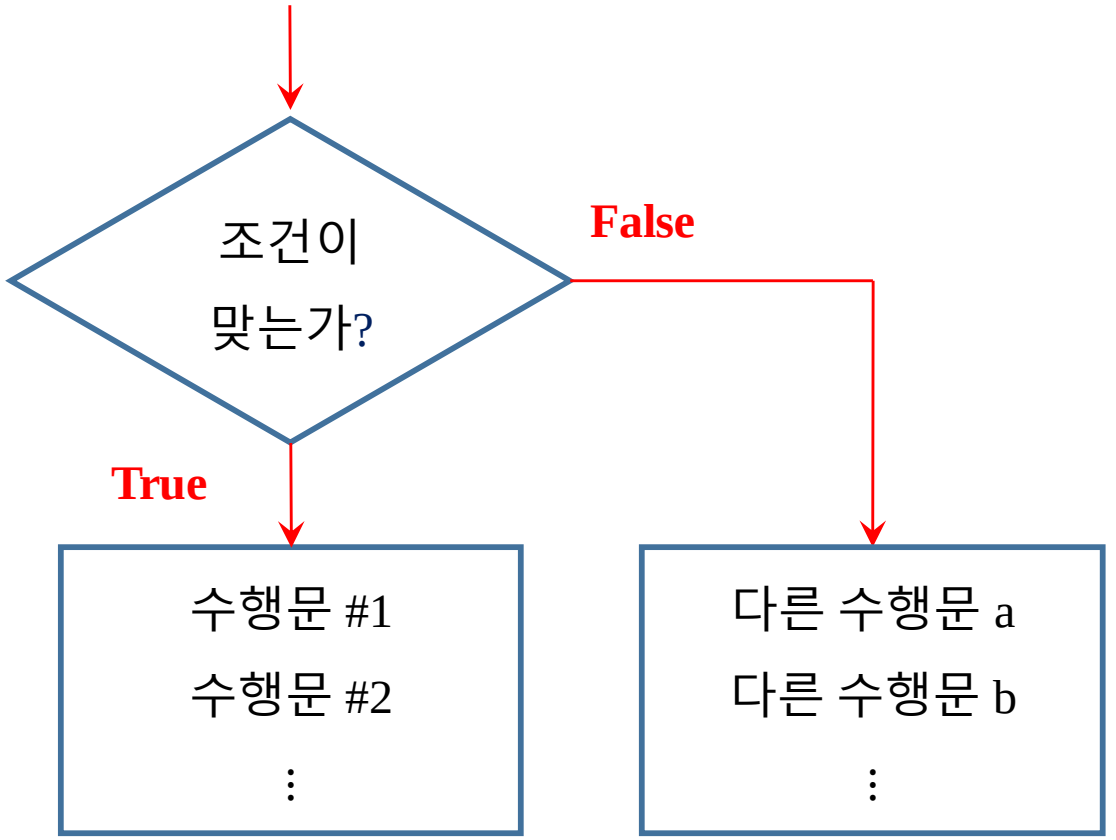
3). 깊은 복사 (deep copy):

```
In[1] : import copy                          # 모듈 불러옴.
In[1] : a = [1, 2, [3, 4, 5]]                # 리스트 복합 객체.
In[2] : b = copy.deepcopy(a)                 # 깊은 복사.
In[3] : a[2][0] = 0
In[4] : a
Out[4]: [1, 2, [0, 4, 5]]
In[5] : b
Out[5]: [1, 2, [3, 4, 5]]                    # mutable 원소도 완전히 다른 객체.
In[6] : id(a[2])
Out[6]: 87731848L
In[7] : id(b[2])
Out[7]: 86205960L
```

파이썬의 제어구조 : 조건문

if-else 조건문:

if –else 조건문의 플로우는 다음과 같다:



파이썬의 제어구조 : 조건문

if-else 조건문:

if-else 조건문의 기본 구조는 다음과 같다:

```
if <조건>:
    <수행할 문장 1>                    # 들여쓰기 적용 (indentation).
    <수행할 문장 2>
        ...
else:                                  # <조건>이 False인 경우 아래 실행.
    <수행할 문장 3>                    # 들여쓰기 적용 (indentation).
    <수행할 문장 4>
        ...
```

파이썬의 제어구조 : 조건문

if-elif-else 조건문:

if-elif-else 조건문의 기본 구조는 다음과 같다:

```
if <조건 A>:
    <수행할 문장 1>                          # 들여쓰기 적용 (indentation).
    <수행할 문장 2>
        ...
elif <조건 B>:                               # <조건 A>가 False인 경우 새로운 <조건 B> 평가.
    <수행할 문장 3>                          # 들여쓰기 적용 (indentation).
    <수행할 문장 4>
        ...
else:                                        # <조건 A>와 <조건 B> 둘 모두 False인 경우.
    <수행할 문장 5>                          # 들여쓰기 적용 (indentation).
    <수행할 문장 6>
        ...
```

파이썬의 제어구조 : 조건문

조건문:

조건을 만드는데 사용되는 비교 연산자:

비교 연산자	설명
x < y	x가 y 보다 작다.
x > y	x가 y 보다 크다.
x == y	x와 y가 같다.
x != y	x와 y가 다르다.
x >= y	x가 y 보다 크거나 같다.
x <= y	x가 y 보다 작거나 같다.

파이썬의 제어구조 : 조건문

조건문:

조건을 만드는데 사용되는 불 (bool) 연산자:

연산자	결과가 참인 경우
x **and** y	x와 y 둘 모두 참이어야 한다.
x **or** y	x와 y 둘 중 하나라도 참이면 된다.
not x	x가 거짓이면 된다.

파이썬의 제어구조 : 조건문

조건문:

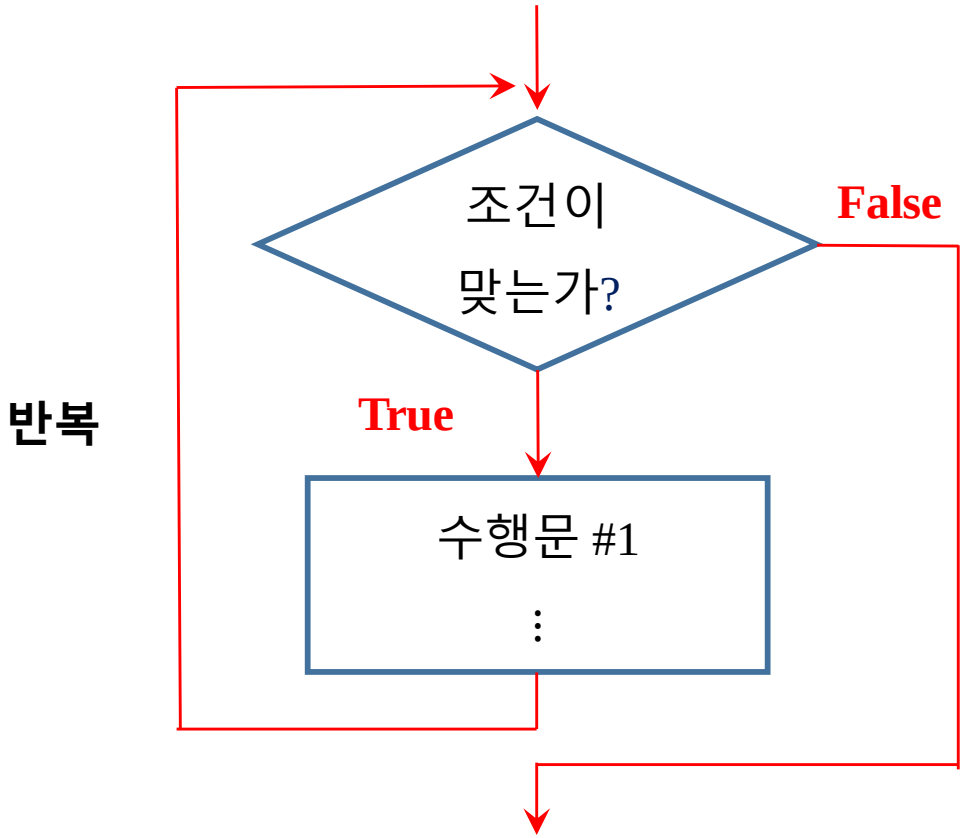
소속되어 있음으로 조건을 만들어주는 in과 not in:

```
In[1] : 1 in [1, 2, 3]                    # 리스트에 소속여부 확인.
Out[1]: True
In[2] : 1 not in (1, 2, 3)                # 튜플에 소속여부 확인.
Out[2]: False
In[3] : 'o' in 'Python'                   # 문자열에 포함됨을 확인. 대소문자 구분 주의.
Out[3]: True
```

파이썬의 제어구조 : 반복문

while 반복문:

while 반복문의 플로우는 다음과 같다:



파이썬의 제어구조 : 반복문

while 반복문:

while 반복문의 기본 구조는 다음과 같다:

```
while <조건>:
    <수행할 문장 1>                    # 들여쓰기 적용 (indentation).
    <수행할 문장 2>
        ...
```

파이썬의 제어구조 : 반복문

while 반복문:

반복문에는 출구 조건이 있어야 한다:

```
In[1] : i = 0
In[2] : while i < 4:                    # i 가 4 이상이면 루프를 벗어난다.
    ... :     print(i)
    ... :     i += 1                    # 출구 조건은 블록 안에서 만들어 진다.
0
1
2
3
```

파이썬의 제어구조 : 반복문

while 반복문:

다음과 같이 반복문의 흐름을 바꿀 수 있다:

```
while <조건 A>:
    <수행할 문장 1>
    <수행할 문장 2>
        ...
    if <조건 B>:
        break                                  # <조건 B> 가 충족되면 루프를 나간다.
    <수행할 문장 3>
        ...
```

파이썬의 제어구조 : 반복문

while 반복문:

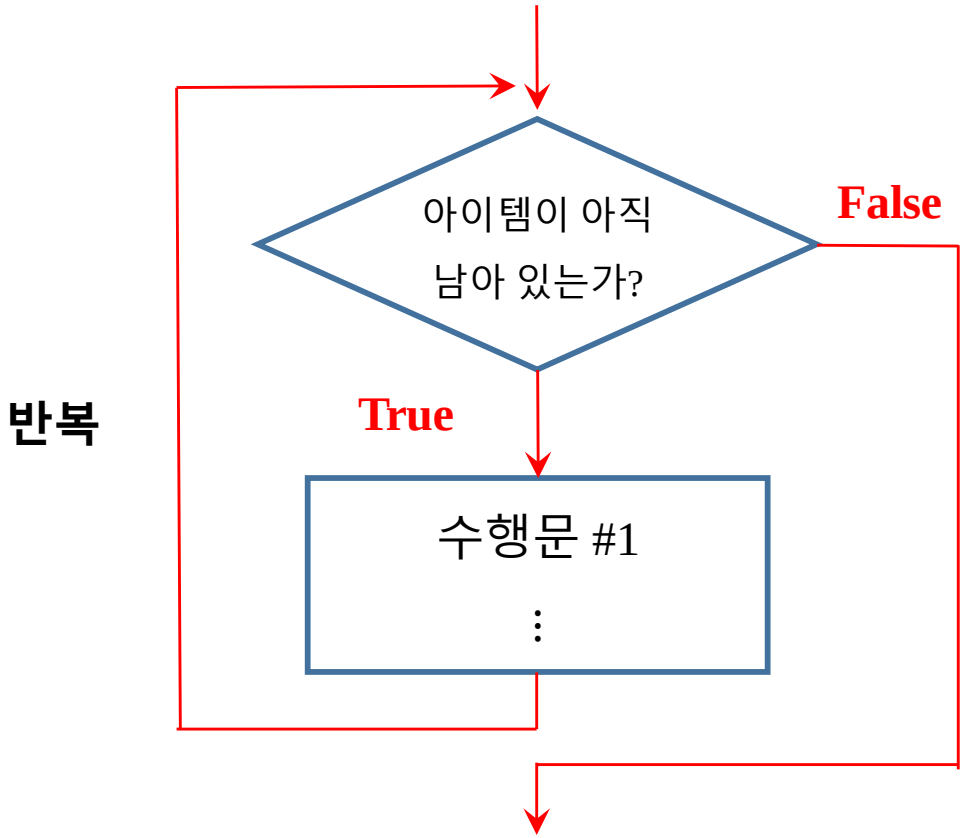
다음과 같이 반복문의 흐름을 바꿀 수 있다:

```
while <조건 A>:
    <수행할 문장 1>
    <수행할 문장 2>
        ...
    if <조건 B>:
        continue                    # <조건 B> 가 충족되면 루프의 처음으로 간다.
    <수행할 문장 3>
        ...
```

파이썬의 제어구조 : 반복문

for 반복문:

for 반복문의 플로우는 다음과 같다:



파이썬의 제어구조 : 반복문

for 반복문:

for 반복문의 기본 구조는 다음과 같다:

```
for 변수 in 리스트(또는 튜플, 문자열):
    <수행할 문장 1>                                    # 들여쓰기 적용 (indentation).
    <수행할 문장 2>
        ...
```

파이썬의 제어구조 : 반복문

for 반복문:

```
In[1] : X = ['You', 'need', 'Python!']
In[2] : for x in X:
    ... :     print(x)
You
need
Python!
```

파이썬의 제어구조 : 반복문

for 반복문:

```
In[1] : sum = 0
In[2] : for x in range(1, 11):                    # 1에서 시작 10까지 반복문 실행.
    ... :     sum += x
    ... :
In[3] : print(sum)
55
```

파이썬의 함수

함수를 사용하는 이유:

- 반복된 작업을 하나의 유닛으로 묶어 놓은 것. (재사용 가능)
- 프로그램의 흐름을 일목요연하고 깔끔하게 정리해 놓을 수 있다.
- 이미 작성된 함수를 새로운 프로그램에 쉽게 가져다 사용할 수 있다.

파이썬의 함수 : 사용자 정의 함수

사용자 정의 함수:

사용자 정의 함수의 구조는 다음과 같다:

```
def  함수명(인수 1, 인수2, …):
    <실행할 문장 1>                                    # 들여쓰기 적용 (indentation).
    <실행할 문장 2>
        …
     return 반환값
```

파이썬의 함수 : 사용자 정의 함수

사용자 정의 함수:

하나의 입력값을 받아서 결과를 리턴하는 함수의 예:

```
In[1] : def times2(a):
    ... :     x = 2*a
    ... :     return x
    ... :
In[2] : times2(11)
Out[2]: 22
In[3] : times2(7)
Out[3]: 14
```

파이썬의 함수 : 사용자 정의 함수

입력값과 반환값에 따른 함수의 형태:

1). 입력값과 반환값이 있는 함수 (일반적인 형태):

```
In[1] : def prod(a, b):
    ... :     x = a * b
    ... :     return x
    ... :
In[2] : prod(3, 4)
Out[2]: 12
```

파이썬의 함수 : 사용자 정의 함수

입력값과 반환값에 따른 함수의 형태:

2). 입력값은 없고 반환값은 있는 함수:

```
In[1]  : def output():
    ... :      return  'Hello World!'
    ... :
In[2]  : print(output())
Hello World!
```

파이썬의 함수 : 사용자 정의 함수

입력값과 반환값에 따른 함수의 형태:

3). 입력값은 있는데 반환값이 없는 함수:

```
In[1] : def times3(a):
    ... :     print(3*a)
    ... :     return                                    # return은 옵션. 반환값은 없음.
    ... :
In[2] : times3(7)
21
```

파이썬의 함수 : 사용자 정의 함수

입력값과 반환값에 따른 함수의 형태:

4). 입력값과 반환값이 없는 함수:

```
In[1] : sum = 0
In[2] : def increase():
    ... :     global sum                                    # 전역 변수.
    ... :     for i in range(11):
    ... :         sum += i
    ... :
In[3] : increase()
In[4] : sum
Out[4]: 55
```

파이썬의 함수 : 사용자 정의 함수

인수의 갯수가 미정인 경우:

인수 앞에 *를 붙여주면 튜플로 입력된다:

```
In[1] : def sum(*vals):
    ... :     total = 0
    ... :     for x in vals:
    ... :         total += x
    ... :     return total
    ... :
In[2] : sum(1,2,3)
Out[2]: 6
In[3] : sum(1, 2, 3, 4, 5)
Out[3]: 15
```

파이썬의 함수 : 사용자 정의 함수

반환값이 한개 이상인 경우:

튜플의 형태로 반환된다:

```
In[1]  : def sachik(a,b):
    ... :      return a + b, a – b, a * b, a/b
    ... :
In[2]  : type(sachik(4,2))
Out[2]: tuple                                  # 튜플은 변경이 불가능한 자료형.
In[3]  : sachik(4,2)
Out[3]: (6, 2, 8, 2)                           # 반환값은 하나의 "객체"이다.
```

파이썬의 함수 : 사용자 정의 함수

인수의 초기값 (default) 설정:

인수의 초기값을 다음과 같이 설정할 수 있다:

```
In[1] : def sachik(a=2,b=1):
    ... :     return a + b, a – b, a * b, a/b
    ... :
In[2] : sachik()
Out[2]: (3, 1, 2, 2)
In[3] : sachik(3)                    # 첫번 째 인수의 값 입력.
Out[3]: (4, 2, 3, 3)
In[3] : sachik(b=3)                  # 인수의 이름으로 매칭.
Out[3]: (5, -1, 6, 0)
```

파이썬의 함수 : 사용자 정의 함수

변수의 효력 범위:

지역변수 (local variable)는 함수 안에서 정의 된 변수이며 효력이 국한된다:

```
In[1] : result = 333
In[2] : def average(*a):
    ... :     result = 0                          # result는 지역변수 (local variable).
    ... :     for x in a:
    ... :         result += x
    ... :     result /= len(a)
    ... :     return result
    ... :
In[3] : average(1,2,3,4,5)
Out[3]: 3
In[4] : result
Out[4]: 333
```

파이썬의 함수 : 사용자 정의 함수

변수의 효력 범위:

전역 변수 (global variable):

```
In[1] : result = 0
In[2] : def average(*a):
    ... :     global result                    # result는 전역변수 (global variable).
    ... :     for x in a:
    ... :         result += x
    ... :     result /= len(a)
    ... :     return                           # 반환값은 없다.
    ... :
In[3] : average(1,2,3,4,5)
In[4] : result
Out[4]: 3
```

파이썬의 함수 : 사용자 정의 함수

람다 함수:

람다 함수 (lambda function)은 보통 def 예약어를 사용할 수 없는 곳에 유용하다:

```
In[1] : def makeMyFunc(a):
    ... :      return lambda x: a*x
    ... :
In[2] : myFunc = makeMyFunc(3)
In[3] : type(myFunc)
Out[3]: function                                    # 함수를 반환값으로 받음.
In[4] : myFunc(4)
Out[4]: 12
```

파이썬의 함수 : 사용자 정의 함수

람다 함수:

Map, Filter, Reduce:

```
In[1] : a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
In[2] : list(map(lambda x: x * 2, a))          # 개개 원소에 람다함수 적용. 1회 사용후 폐기.
Out[2]: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
In[3] : list(filter(lambda x: x % 3 ==0, a))   # 3으로 나누면 나머지가 0인 조건으로 필터링.
Out[3]: [3, 6, 9]
In[4] : from functools import reduce           # Python 3인 경우 필요!
In[5] : reduce(lambda x, y: x + y, a)          # lambda 함수는 두개의 인수 필요.
Out[5]: 55                                     # 1~10의 합.
In[6] : reduce(lambda x, y: x * y, a)
Out[6]: 3628800                                # 1~10의 곱.
```

파이썬의 함수 : 내장 함수

내장 함수:

- 파이썬이 기본적으로 제공하는 함수.
- 외부 모듈을 import 하지 않아도 사용할 수 있는 함수.

파이썬의 함수 : 내장 함수

내장 함수:

abs	enumerate	int	max	range
all	eval	isinstance	min	sorted
any	filter	lambda	oct	str
chr	hex	len	open	tuple
dir	id	list	ord	type
divmod	input	map	pow	zip

파이썬의 함수 : 외장 함수

외장 함수:

- 외부 모듈에 저장된 함수로서 모듈을 import한 후 사용할 수 있다.
- os, sys, numpy, pandas, scipy, matplotlib, 등의 모듈이 있다.

파이썬의 함수 : 외장 함수

외장 함수:

```
In[1] : import os                                                    # os 모듈 불러옴.
In[2] : os.getcwd()                                                  # 현 폴더.
Out[2]: ' C:□□Users□□sycha_000□□Documents□□Python Scripts '
In[3] : os.chdir( ' .. ' )                                           # 폴더 (WD) 변경.
In[4] : os.getcwd()                                                  # 현 폴더.
Out[4]: ' C:□□Users□□sycha_000□□Documents '
In[5] : os.listdir( ' . ' )                                          # 현 폴더 내용 리스팅.
['.ipynb_checkpoints',
'Thumbs.db',
 'Untitled.ipynb',
 'Untitled1.ipynb',
 'Untitled2.ipynb',
 'Untitled3.ipynb']
```

파이썬의 입력과 출력 : 사용자 입출력

사용자 입출력:

- 입력은 input 또는 raw_input 함수를 통해서.
- 출력은 print 함수를 통해서.

파이썬의 입력과 출력 : 사용자 입출력

사용자 입출력:

```
In[1] : x = input('Enter a value : ')                    # 숫자 입력 (Python 2.x).
Enter a value : 123
In[2] : print('The value you entered is : ' + str(x))    # 숫자를 문자열로 변환하는 함수 str 사용.
The value you entered is : 123
```

```
In[1] : x = raw_input('Enter anything : ')               # 입력 (Python 2.x).
Enter anything : Hello Python!
In[2] : print(x)                                         # 출력.
Hello Python!
```

파이썬의 입력과 출력 : 파일 입출력

파일 생성/열기:

```
In[1] : f = open('new_file.txt', 'w')          # 파일 열기.
In[2] : f.close()                              # 파일 닫기.
```

파일 열기 모드	설명
r	읽기모드.
w	쓰기모드.
a	추가모드.

파이썬의 입력과 출력 : 파일 입출력

파일 읽기:

```
In[1] : f = open('my_file.txt', 'r')          # 읽기 모드로 파일 열기.
In[2] : all = f.read()                        # 모든 내용을 한번에 읽어들임.
In[3] : print(all)                            # 한번에 모두 출력.
Life
is
short,
You
need
Python!
In[4] : f.close()                             # 파일 닫음.
```

파이썬의 입력과 출력 : 파일 입출력

파일 읽기:

```
In[1] : f = open('my_file.txt', 'r')                    # 읽기 모드로 파일 열기.
In[2] : all = f.read()                                  # 모든 내용을 한번에 읽어들임.
In[3] : for line in all:
    ... :       print(line)                             # 라인 하나씩 출력.
    ... :
Life
is
short,
You
need
Python!
In[4] : f.close()                                       # 파일 닫음.
```

파이썬의 입력과 출력 : 파일 입출력

파일 읽기:

```
In[1] : f = open('my_file.txt', 'r')                    # 읽기 모드로 파일 열기.
In[2] : while True:
    ... :     line = f.readline()                       # 라인 한줄씩 읽어 들이기.
    ... :     if not line: break                        # 더이상 읽을 라인이 없으면 나가기.
    ... :     print(line)
    ... :
Life
is
short,
You
need
Python!
In[3] : f.close()                                       # 파일 닫음.
```

파이썬의 입력과 출력 : 파일 입출력

새로운 파일 생성 후 내용 쓰기:

```
In[1] : f = open('new_file.txt', 'w')          # 쓰기 모드로 파일 열기.
In[2] : f.write('This is a new file.')         # 쓰기.
In[3] : f.close()                              # 파일 닫음.
```

```
In[1] : f = open('new_file.txt', 'w')          # 같은 이름의 파일 쓰기 모드로 열기. (덮어쓰기)
In[2] : f.write('This is another file!')       # 쓰기.
In[3] : f.close()                              # 파일 닫음.
```

파이썬의 입력과 출력 : 파일 입출력

파일에 새로운 내용 추가:

```
In[1] : f = open('my_file.txt', 'a')                    # 추가 모드로 파일 열기.
In[2] : f.write('This is the end!')                     # 추가.
In[3] : f.close()                                       # 파일 닫음.
```

```
In[1] : with open('my_file.txt', 'a') as f:             # 추가 모드로 파일 열기. with 문 사용.
    ... :     f.write('This is the added line #1 □n')   # 추가 #1.
    ... :     f.write('This is the added line #2 □n')   # 추가 #2.
    ... :     f.write('This is the added line #3 □n')   # 추가 #3.
In[2] : f.close()                                       # f.close() 불필요.
```

파이썬의 클래스 : 클래스와 객체

객체지향 프로그래밍:

- 절차지향 프로그래밍은 순차적으로 처리되도록 작성된다: C 언어.
- 절차지향 프로그래밍은 확장성과 재사용성에 한계가 있다.
- 객체지향 프로그래밍은 재사용과 확장성의 장점이 있는 방식이다: C++, Java, Python, 등.

파이썬의 클래스 : 클래스와 객체

객체지향 프로그래밍:

- 클래스 (class)는 객체를 만들기 위한 “틀”의 개념을 갖는다 → 쿠키 커터.
- 객체는 클래스에 의해서 만들어진 “실체”의 개념을 갖는다 → 쿠키.

파이썬의 클래스 : 클래스와 객체

객체지향 프로그래밍:

- 같은 클래스로 여러 다른 객체를 만들 수 있다.
- 객체가 실제 메모리 공간에 활당된 것을 **인스턴스** (instance)라고 부른다.
- 클래스에는 **멤버** 변수, **멤버** 함수 또는 **멤버** 메서드 (method)의 개념이 있다.
- 또한 클래스에는 **상속**의 개념이 있다.

파이썬의 클래스 : 생성과 소멸

클래스의 생성자 메서드와 소멸자 메서드:

생성자는 클래스 생성 시 초기화를 위해서 사용되는 메서드이다:

```
In[1] : class Dog:                                  # 클래스 Dog 선언.
    ... :     def __init__(self, name, age):        # 생성자 메서드.
    ... :         self.name = name                  # 멤버 변수 정의.
    ... :         self.age = age                    # 멤버 변수 정의.
    ... :         print('A Dog object is created!')
In[2] : baduk = Dog('Baduk', 2)
A Dog object is created!
```

파이썬의 클래스 : 생성과 소멸

클래스의 생성자 메서드와 소멸자 메서드:

소멸자는 클래스 소멸 시 실행되는 메서드이다:

```
In[1] : class Dog:                                  # 클래스 Dog 선언.
    ... :     def __init__(self, name, age):        # 생성자 메서드.
    ... :         self.name = name                  # 멤버 변수 정의.
    ... :         self.age = age                    # 멤버 변수 정의.
    ... :         print('A Dog object is created!')
    ... :     def __del__(self):                    # 소멸자 메서드.
    ... :         print('A Dog object is deleted!')
In[2] : baduk = Dog('Baduk', 2)
A Dog object is created!
In[3] : del baduk
A Dog object is deleted!
```

파이썬의 클래스 : 멤버 변수

멤버 변수:

멤버 변수는 객체 개개의 특성이다:

```
In[1] : class Dog:                          # 클래스 Dog 선언.
    ... :     def __init__(self, name, age):    # 생성자 메서드.
    ... :         self.name = name              # 멤버 변수 정의.
    ... :         self.age = age                # 멤버 변수 정의.
In[2] : dog1 = Dog('Baduk', 2)
In[3] : dog2 = Dog('Sundol', 3)
In[4] : dog1.name
Out[4]: Baduk
In[5] : dog2.age
Out[5]: 3
```

파이썬의 클래스 : 클래스 변수

클래스 변수:

클래스 변수는 클래스의 특성이다:

```
In[1] : class Dog:                                  # 클래스 Dog 선언.
    ... :       counter = 0                         # 클래스 변수 정의.
    ... :       def __init__(self, name):           # 생성자 메서드.
    ... :           self.name = name                # 멤버 변수 정의.
    ... :           Dog.counter += 1                # 클래스 변수 증가.
    ... :       def __del__(self):                  # 소멸자 메서드.
    ... :           Dog.counter -= 1                # 클래스 변수 감소.
In[2] : dog1 = Dog('Baduk')
In[3] : dog2 = Dog('Sundol')
In[4] : Dog.counter                                 # 클래스 변수.
Out[4]: 2
```

파이썬의 클래스 : 멤버 메서드

클래스 변수:

클래스 변수는 클래스의 특성이다:

```
In[5] : del dog2
In[6] : Dog.counter                    # 클래스 변수.
Out[6]: 1
In[7] : del dog1
In[8] : Dog.counter                    # 클래스 변수.
Out[8]: 0
```

파이썬의 클래스 : 멤버 메서드

멤버 메서드:

멤버 메서드를 호출시 객체의 변수를 사용한 연산이 가능하다:

```
In[1] : class Dog:                                  # 클래스 Dog 선언.
    ... :     def __init__(self, name, age):        # 생성자 메서드.
    ... :         self.name = name                  # 멤버 변수 정의.
    ... :         self.age = age                    # 멤버 변수 정의.
    ... :     def bark(self):                       # 멤버 메서드.
    ... :         print(self.name + ' is barking.... mung... mung...')
In[2] : baduk = Dog('Baduk', 2)
In[3] : baduk.bark()
Baduk is barking... mung... mung...
```

파이썬의 클래스 : 상속

상속 (inheritance):

"부모" 클래스를 상속한 "자식" 클래스를 만들 수 있다:

```
In[1] : class Pet:                                  # 부모 클래스 Pet 선언.
   ... :     def __init__(self, name):              # 생성자 메서드.
   ... :         self.name = name
In[2] : class Cat(Pet):                             # Pet의 자식 클래스 Cat 선언.
   ... :     def meow(self):
   ... :         print(self.name + ' is meowing...')
In[3] : class Dog(Pet):                             # Pet의 자식 클래스 Dog 선언.
   ... :     def bark(self):
   ... :         print(self.name + ' is barking...')
```

파이썬의 클래스 : 상속

상속 (inheritance):

"부모" 클래스를 상속한 "자식" 클래스를 만들 수 있다:

```
In[4] : cat1 = Cat('Nabi')
In[5] : dog1 = Dog('Sundol')
In[6] : cat1.meow()
Nabi is meowing…
In[7] : dog1.bark()
Sundol is barking…
```

파이썬의 모듈 : 모듈

모듈 (module):

다른 프로그램에서 불러서 쓸수 있도록 함수, 변수 또는 클래스를 모아놓은 파일이다:

```
# 다음은 모듈 module1.py의 내용.
def sum(a, b):
    return a + b
def subtract(a, b):
    return a – b
def product(a, b):
    return a * b
def divide(a, b):
    return a / b
```

파이썬의 모듈 : 모듈

모듈 (module):

다른 프로그램에서 불러서 쓸수 있도록 함수, 변수 또는 클래스를 모아놓은 파일이다:

```
In[1] : import module1 as md1                    # 모듈 module1.py 불러 오기.
In[2] : md1.sum(3,4)
Out[2]: 7
In[3] : md1.product(4, 5)
Out[3]: 20
```

파이썬의 모듈 : 모듈

모듈 (module):

if 조건문은 프롬프트에서 python module1.py를 실행할 경우에만 True가 된다 :

```
# 다음은 모듈 module1.py의 내용.
def sum(a, b):
    return a + b
def subtract(a, b):
    return a – b
def product(a, b):
    return a * b
def divide(a, b):
    return a / b
if __name__ == "__main__":
    print(sum(3, 4))
    print(product(4, 5))
```

파이썬의 예외 처리

예외 처리 (exception handling):

- 프로그램의 오류 상황을 처리할 필요가 생긴다.
- 문법에 맞게 작성된 프로그램도 실행 중 에러가 발생할 수 있다.
- 실행중 오류 발생 시 프로그램의 흐름을 조정하는 것을 예외 처리라고 한다.
- 파이썬은 에러 코드를 알려준다. 대표적인 에러의 예는 다음과 같다.

에러	설명
ZeroDivisionError	0으로 나누려 할 때 발생.
IndexError	리스트의 인덱스 오류.
ValueError	리스트의 index(value)의 서치 실패.
KeyError	딕셔너리에서 없는 key로 인덱싱 한 경우.

파이썬의 예외 처리

예외 처리 (exception handling):

모든 에러 처리:

```
try:
    <예외 발생이 가능한 코드 블럭>
except:
    <예외 발생시 수행되는 코드 블럭>
```

파이썬의 예외 처리

예외 처리 (exception handling):

특정 에러 처리:

```
try:
    <예외 발생이 가능한 코드 블럭>
except <발생 에러> :
    <해당 에러 발생시 수행되는 코드 블럭>
```

```
try:
    <예외 발생이 가능한 코드 블럭>
except <발생 에러> as <에러 변수> :                    # 변수로 에러 메세지를 받음.
    <해당 에러 발생시 수행되는 코드 블럭 >
```

파이썬의 예외 처리

예외 처리 (exception handling):

else와 finally:

```
try:
    <예외 발생이 가능한 코드 블럭>
except:
    <예외 발생시 수행되는 코드 블럭>
else:
    <예외가 발생하지 않은경우 실행되는 코드 블럭>
finally:
    <예외 발생 여부와 상관없이 수행되는 코드 블럭>
```

파이썬의 예외 처리

예외 처리 (exception handling):

else와 finally:

```
In[1] : try:
    ... :     result = 123/x
    ... : except ZeroDivisionError as err:
    ... :     print(err)
    ... : else:
    ... :     print(result)
    ... : finally:
    ... :     print('끝')
```

파이썬의 예외 처리

예외 처리 (exception handling):

else와 finally:

```
In[1] : try:
    ... :     result = x.find(1234)
    ... : except ValueError as err:
    ... :     print(err)
    ... : else:
    ... :     print(result)
    ... : finally:
    ... :     print('끝')
```