핵심 정리

불

불은 `True`, `False`로 표현합니다. 특히 비교 연산자, 논리 연산자의 판단 결과로 `True`, `False`를 사용합니다.

비교 연산자

▼ **표 12-1** 파이썬 비교 연산자

연산자	문법	설명
==	`a == b`	같음
!=	`a != b`	같지 않음, 다름
>	`a > b`	큼, 초과
<	`a < b`	작음, 미만
>=	`a >= b`	크거나 같음, 이상
<=	`a <= b`	작거나 같음, 이하
is	`a is b`	같음(객체 비교)
is not	`a is not b`	같지 않음, 다름(객체 비교)

논리 연산자

▼ **표 12-2** 파이썬 논리 연산자

연산자	문법	설명
and	`a and b`	AND(논리곱), 양쪽 모두 참일 때 참
or	`a or b`	OR(논리합), 양쪽 중 한쪽만 참이라도 참
not	`not x`	NOT(논리 부정), 참과 거짓을 뒤집음

문자열

문자열은 `' '`(작은따옴표) 또는 `" "`(큰따옴표)로 묶어서 표현합니다.

```
'Hello, world!'
"Hello, world!"
```

여러 줄로 된 문자열

여러 줄로 된 문자열은 '''(작은따옴표 3개)로 시작해서 '''로 닫거나 """(큰따옴표 3개)로 시작해서 """로 닫아서 표현합니다.

```
'''Hello, world!
안녕하세요.
Python입니다.'''

"""Hello, world!
안녕하세요.
Python입니다."""
```

리스트

리스트는 여러 개의 값(요소)을 일렬로 늘어놓은 형태입니다. 변수에 값을 저장할 때 [ ](대괄호)로 묶어주면 리스트가 되며 각 값은 , (콤마)로 구분합니다. 리스트에 저장된 요소에 접근할 때는 [ ] 안에 인덱스를 지정해줍니다. 특히 리스트의 인덱스는 0부터 시작합니다.

```
리스트 = [값, 값, 값]        # 리스트 만들기
리스트 = []                  # 빈 리스트 만들기
리스트 = list()              # 빈 리스트 만들기
리스트 = list(range(횟수))   # range로 리스트 만들기

리스트[인덱스]               # 리스트의 요소에 접근
리스트[0]                    # 리스트의 인덱스는 0부터 시작하므로 첫 번째 요소
리스트[인덱스] = 값          # 리스트의 요소에 값 저장
```

range

range는 연속된 숫자를 생성합니다. 이때 지정한 횟수는 생성되는 숫자에 포함되지 않습니다. 그리고 시작하는 숫자와 끝나는 숫자를 지정했을 때 끝나는 숫자는 생성되는 숫자에 포함되지 않습니다.

```
range(횟수)
range(시작, 끝)
range(시작, 끝, 증가폭)
```

튜플

튜플은 여러 개의 값(요소)을 일렬로 늘어놓은 형태입니다. 단, 요소의 값을 변경하거나 추가할 수 없습니다(읽기 전용). 변수에 값을 저장할 때 ( ) (괄호)로 묶어주면 튜플이 되며 각 값은 콤마로 구분합니다. 또는, 괄호로 묶지 않고 값만 콤마로 구분해도 튜플이 됩니다. 튜플에 저장된 요소에 접근할 때는 [ ]안에 인덱스를 지정해줍니다. 그리고 리스트와 마찬가지로 튜플의 인덱스도 0부터 시작합니다.

```
튜플 = (값, 값, 값)          # 튜플 만들기
튜플 = 값, 값, 값            # 괄호 없이 튜플 만들기
튜플 = ()                    # 빈 튜플 만들기
튜플 = tuple()               # 빈 튜플 만들기
튜플 = tuple(list())         # tuple에 list()를 넣어서 빈 튜플 만들기
튜플 = tuple(리스트)         # tuple에 리스트를 넣어서 튜플 만들기
튜플 = tuple(range(횟수))    # range로 튜플 만들기

튜플[인덱스]                 # 튜플의 요소에 접근
튜플[0]                      # 튜플의 인덱스는 0부터 시작하므로 첫 번째 요소

튜플 = (값, )                # 요소가 한 개인 튜플 만들기
튜플 = 값,                   # 요소가 한 개인 튜플 만들기
```

시퀀스 자료형

파이썬에서 리스트(list), 튜플(tuple), range, 문자열(str)과 같이 값이 연속적으로 이어진 자료형을 시퀀스 자료형(sequence types)이라고 합니다. 그리고 시퀀스 자료형으로 만든 객체를 시퀀스 객체라고 하며, 시퀀스 객체에 들어있는 각 값을 요소(element)라고 부릅니다.

시퀀스 자료형의 공통 기능

파이썬의 시퀀스 자료형은 공통된 동작과 기능을 제공합니다. 따라서 리스트, 튜플, range, 문자열 등의 시퀀스 자료형은 같은 문법을 사용합니다.

```
값 in 시퀀스객체              # 시퀀스 객체에 특정 값이 있는지 확인
값 not in 시퀀스객체          # 시퀀스 객체에 특정 값이 없는지 확인

시퀀스객체1 + 시퀀스객체2    # 시퀀스 객체를 서로 연결하여 새 시퀀스 객체를 만듦
시퀀스객체 * 정수             # 시퀀스 객체를 특정 횟수만큼 반복하여 새 시퀀스 객체를 만듦
정수 * 시퀀스객체             # 시퀀스 객체를 특정 횟수만큼 반복하여 새 시퀀스 객체를 만듦

len(시퀀스객체)               # 시퀀스 객체의 요소 개수(길이) 구하기

시퀀스객체[인덱스]            # 시퀀스 객체의 요소에 접근
시퀀스객체[0]                 # 시퀀스 객체의 인덱스는 0부터 시작하므로 첫 번째 요소
시퀀스객체[-음수]             # 인덱스를 음수로 지정하면 뒤에서부터 요소에 접근, -1은 뒤에서 첫 번째
시퀀스객체[인덱스] = 값       # 시퀀스 객체의 요소에 값 저장

del 시퀀스객체[인덱스]        # 시퀀스 객체의 요소를 삭제
```

시퀀스 자료형의 슬라이스

시퀀스 자료형은 시퀀스 객체의 일부를 잘라내서 가져오는 슬라이스(slice)를 사용할 수 있습니다. [ ](대괄호) 안에 시작 인덱스와 끝 인덱스를 지정하면 해당 범위의 요소를 잘라서 새 시퀀스 객체를 만듭니다. 단, 끝 인덱스는 가져오려는 범위에 포함되지 않습니다.

```
시퀀스객체[시작인덱스:끝인덱스]                # 지정된 범위의 요소를 잘라서 새 시퀀스 객체를 만듦
시퀀스객체[시작인덱스:끝인덱스:인덱스증가폭]    # 인덱스 증가폭을 지정하면 해당 값만큼
                                              # 인덱스를 증가시키면서 요소를 가져옴

시퀀스객체[:끝인덱스]      # 시작 인덱스를 생략하여 객체의 처음부터 끝 인덱스 - 1까지 가져옴
시퀀스객체[시작인덱스:]    # 끝 인덱스를 생략하여 시작 인덱스부터 마지막 요소까지 가져옴
시퀀스객체[:]              # 시작 인덱스와 끝 인덱스를 생략하여 객체 전체를 가져옴

시퀀스객체[0:len(시퀀스객체)]     # len을 응용하여 객체 전체를 가져옴
시퀀스객체[:len(시퀀스객체)]      # 시작 인덱스 생략, len을 응용하여 객체 전체를 가져옴

시퀀스객체[:끝인덱스:증가폭]      # 객체의 처음부터 증가폭만큼 인덱스를 증가시키면서
                                 # 끝 인덱스 - 1까지 요소를 가져옴
시퀀스객체[시작인덱스::증가폭]    # 시작 인덱스부터 증가폭만큼 인덱스를 증가시키면서
                                 # 마지막 요소까지 가져옴
시퀀스객체[::증가폭]              # 객체 전체에서 증가폭만큼 인덱스를 증가시키면서 요소를 가져옴

시퀀스객체[::]                   # 객체 전체를 가져옴, 시퀀스객체[:]와 같음

시퀀스객체[시작인덱스:끝인덱스] = 시퀀스객체                # 범위를 지정하여 여러 요소에 값 할당
시퀀스객체[시작인덱스:끝인덱스:인덱스증가폭] = 시퀀스객체    # 증가폭만큼 인덱스를 건너뛰면서 할당

del 시퀀스객체[시작인덱스:끝인덱스]    # 특정 범위의 요소를 삭제(원본 객체가 변경됨)
```

딕셔너리

딕셔너리는 연관된 값을 묶어서 저장하는 자료형입니다. { }(중괄호) 안에 **키: 값** 형식으로 저장하며 각 키와 값은 , (콤마)로 구분합니다. 딕셔너리에 저장된 값에 접근할 때는 [ ](대괄호) 안에 키를 지정해줍니다.

```
딕셔너리 = {키1: 값1, 키2: 값2}    # 딕셔너리 만들기
딕셔너리 = {}                      # 빈 딕셔너리 만들기
딕셔너리 = dict()                  # 빈 딕셔너리 만들기

딕셔너리[키]                       # 딕셔너리에서 키로 값에 접근
딕셔너리[키] = 값                  # 딕셔너리에서 키에 값 할당

키 in 딕셔너리                     # 딕셔너리에 특정 키가 있는지 확인
키 not in 딕셔너리                 # 딕셔너리에 특정 키가 없는지 확인

len(딕셔너리)                      # 딕셔너리의 키 개수(길이) 구하기
```