

불 자료형과 if 조건문

- Boolean

- 불린 / 불리언 / 불
- True와 False 값만 가질 수 있음

```
>>> print(True)
True
>>> print(False)
False
```

- 비교 연산자를 통해 만들 수 있음

연산자	설명	연산자	설명
==	같다	>	크다
!=	다르다	<=	작거나 같다
<	작다	>=	크거나 같다

불 만들기 : 비교 연산자

- 숫자 또는 문자열에 적용

```
>>> print(10 == 100)
False
>>> print(10 != 100)
True
>>> print(10 < 100)
True
>>> print(10 > 100)
False
>>> print(10 <= 100)
True
>>> print(10 >= 100)
False
```

조건식	의미	결과
10 == 100	10과 100은 같다	거짓
10 != 100	10과 100은 다르다	참
10 < 100	10은 100보다 작다	참
10 > 100	10은 100보다 크다	거짓
10 <= 100	10은 100보다 작거나 같다	참
10 >= 100	10은 100보다 크거나 같다	거짓

불 만들기 : 비교 연산자

- 문자열에도 비교 연산자 적용 가능

```
>>> print("가방" == "가방")
True
>>> print("가방" != "하마")
True
>>> print("가방" < "하마")
True
>>> print("가방" > "하마")
False
```

불 연산하기 : 논리 연산자

- 불끼리 논리 연산자 사용 가능

연산자	의미	설명
not	아니다	불을 반대로 전환합니다.
and	그리고	피연산자 두 개가 모두 참일 때 True를 출력하며, 그 외는 모두 False를 출력합니다.
or	또는	피연산자 두 개 중에 하나만 참이라도 True를 출력하며, 두 개가 모두 거짓일 때만 False를 출력합니다.

- not 연산자

- 단항 연산자
- 참과 거짓 반대로 바꿈

```
>>> print(not True)
False
>>> print(not False)
True
```

불 연산하기 : 논리 연산자

- and 연산자와 or 연산자

- and 연산자는 양쪽 변의 값이 모두 참일 때만 True를 결과로 냄

- and 연산자

좌변	우변	결과
True	True	True
True	False	False
False	True	False
False	False	False

- or 연산자

좌변	우변	결과
True	True	True
True	False	True
False	True	True
False	False	False

논리 연산자의 활용

- and 연산자



- or 연산자



if 조건문이란

- if 조건문

- 조건에 따라 코드 실행하거나 실행하지 않게 할 때 사용하는 구문
- 조건 분기

if 불 값이 나오는 표현식: → if의 조건문 뒤에는 반드시 콜론(:)을 붙여줘야 합니다.

□□□ 불 값이 참일 때 실행할 문장

□□□ 불 값이 참일 때 실행할 문장

□□□□는 들여쓰기 4칸

↓
if문 다음 문장은 4칸 들여쓰기 후 입력합니다.


if 조건문이란

- 예시 - 조건문의 기본 사용

```
01  # 입력을 받습니다.  
02  number = input("정수 입력> ")  
03  number = int(number)  
04  
05  # 양수 조건  
06  if number > 0:  
07      print("양수입니다")  
08  
09  # 음수 조건  
10  if number < 0:  
11      print("음수입니다")  
12  
13  # 0 조건  
14  if number == 0:  
15      print("0입니다")
```

 실행결과 1 ×

정수 입력> 273
양수입니다

 실행결과 2 ×

정수 입력> -52
음수입니다

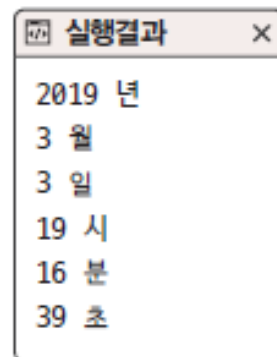
 실행결과 3 ×

정수 입력> 0
0입니다

날짜/시간 활용하기

- 예시 – 날짜/시간 출력하기 <https://docs.python.org/ko/3.8/library/datetime.html>
- datetime.datetime.now() 함수

```
01  # 날짜/시간과 관련된 기능을 가져옵니다.
02  import datetime
03
04  # 현재 날짜/시간을 구합니다.
05  now = datetime.datetime.now()
06
07  # 출력합니다.
08  print(now.year, "년")
09  print(now.month, "월")
10  print(now.day, "일")
11  print(now.hour, "시")
12  print(now.minute, "분")
13  print(now.second, "초")
```



실행결과

2019 년
3 월
3 일
19 시
16 분
39 초


날짜/시간 활용하기

- 예시 - 계절을 구분하는 프로그램

```
01  # 날짜/시간과 관련된 기능을 가져옵니다.
02  import datetime
03
04  # 현재 날짜/시간을 구합니다.
05  now = datetime.datetime.now()
06
07  # 봄 구분
08  if 3 <= now.month <= 5:
09      print("이번 달은 {}월로 봄입니다!".format(now.month))
10
11  # 여름 구분
12  if 6 <= now.month <= 8:
```

날짜/시간 활용하기

```
13     print("이번 달은 {}월로 여름입니다!".format(now.month))
14
15     # 가을 구분
16     if 9 <= now.month <= 11:
17         print("이번 달은 {}월로 가을입니다!".format(now.month))
18
19     # 겨울 구분
20     if now.month == 12 or 1 <= now.month <= 2:
21         print("이번 달은 {}월로 겨울입니다!".format(now.month))
```

 실행결과

×

이번 달은 3월로 봄입니다!

컴퓨터의 조건

- if 조건문의 형식

if 불 값이 나오는 표현식:

 **** 불 값이 참일 때 실행할 문장

****는 들여쓰기 4칸

- 예시 – 끝자리로 짝수와 홀수 구분

```
01  # 입력을 받습니다.
02  number = input("정수 입력> ")
03
04  # 마지막 자리 숫자를 추출
05  last_character = number[-1]
06
07  # 숫자로 변환하기
08  last_number = int(last_character)
09
```

컴퓨터의 조건

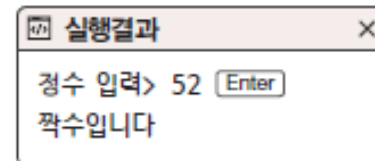
```
10  # 짝수 확인
11  if last_number == 0 \
12      or last_number == 2 \
13      or last_number == 4 \
14      or last_number == 6 \
15      or last_number == 8:
16      print("짝수입니다")
17
18  # 홀수 확인
19  if last_number == 1 \
20      or last_number == 3 \
21      or last_number == 5 \
22      or last_number == 7 \
23      or last_number == 9:
24      print("홀수입니다")
```



컴퓨터의 조건

- 예시 - in 연산자를 활용한 수정

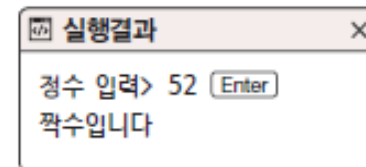
```
01  # 입력을 받습니다.  
02  number = input("정수 입력> ")  
03  last_character = number[-1]  
04  
05  # 짝수 조건  
06  if last_character in "02468":  
07      print("짝수입니다")  
08  
09  # 홀수 조건  
10  if last_character in "13579":  
11      print("홀수입니다")
```



컴퓨터의 조건

- 예시 - 나머지 연산자를 활용한 짝수와 홀수 구분

```
01  # 입력을 받습니다.  
02  number = input("정수 입력> ")  
03  number = int(number)  
04  
05  # 짝수 조건  
06  if number % 2 == 0:  
07      print("짝수입니다")  
08  
09  # 홀수 조건  
10  if number % 2 == 1:  
11      print("홀수입니다")
```



- 정반대되는 상황에서 두 번이나 if 조건문을 사용해 조건을 비교하는 것은 낭비일 수 있다.

```
01  # 입력을 받습니다.  
02  number = input("정수 입력> ")  
03  number = int(number)  
04  
05  # 짝수 조건  
06  if number % 2 == 0:  
07      print("짝수입니다")  
08  
09  # 홀수 조건  
10  if number % 2 == 1:  
11      print("홀수입니다")
```

else 조건문의 활용

- else 구문

- if 조건문 뒤에 사용하며, if 조건문의 조건이 거짓일 때 실행되는 부분

```
if 조건:  
    조건이 참일 때 실행할 문장  
else:  
    조건이 거짓일 때 실행할 문장
```

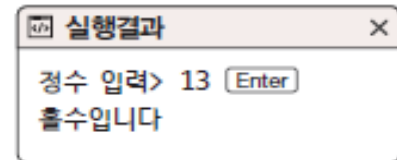
□□□□는 들여쓰기 4칸

- 조건문이 오로지 두 가지로만 구분될 때 if else 구문을 사용하면 조건 비교를 단 한 번만 하므로 이전의 코드보다 효율적

else 조건문의 활용

- 예시 - if 조건문에 else 구문 추가해서 짝수와 홀수 구분

```
01  # 입력을 받습니다.  
02  number = input("정수 입력> ")  
03  number = int(number)  
04  
05  # 조건문을 사용합니다.  
06  if number % 2 == 0:  
07      # 조건이 참일 때, 즉 짝수 조건  
08      print("짝수입니다")  
09  else:  
10      # 조건이 거짓일 때, 즉 홀수 조건  
11      print("홀수입니다")
```



- elif 구문

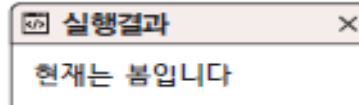
- 세 개 이상의 조건을 연결해서 사용
- if 조건문과 else 구문 사이에 입력

```
if 조건A:
    조건A가 참일 때 실행할 문장
elif 조건B:
    조건B가 참일 때 실행할 문장
elif 조건C:
    조건C가 참일 때 실행할 문장
...
else:
    모든 조건이 거짓일 때 문장
```

□□□□는 들여쓰기 4칸

- 예시 - 계절 구하기

```
01  # 날짜/시간과 관련된 기능을 가져옵니다.
02  import datetime
03
04  # 현재 날짜/시간을 구하고
05  # 쉽게 사용할 수 있게 월을 변수에 저장합니다.
06  now = datetime.datetime.now()
07  month = now.month
08
09  # 조건문으로 계절을 확인합니다.
10  if 3 <= month <= 5:
11      print("현재는 봄입니다.")
12  elif 6 <= month <= 8:
13      print("현재는 여름입니다.")
14  elif 9 <= month <= 11:
15      print("현재는 가을입니다.")
16  else:
17      print("현재는 겨울입니다.")
```



실행결과

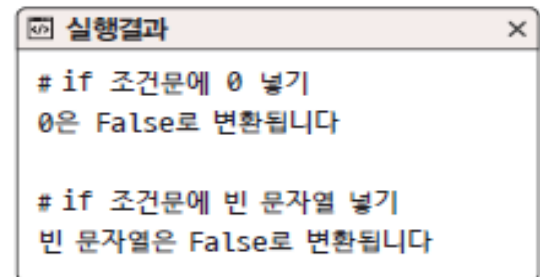
현재는 봄입니다

False로 변환되는 값

- 빈 컨테이너

- if 조건문의 매개변수에 볼 아닌 다른 값이 올 때 자동으로 볼로 변환
- 이 때 **False로 변환되는 값**: None, 0.0, 빈 문자열, 빈 바이트열, 빈 리스트

```
01 print("# if 조건문에 0 넣기")
02 if 0:
03     print("0은 True로 변환됩니다")
04 else:
05     print("0은 False로 변환됩니다")
06 print()
07
08 print("# if 조건문에 빈 문자열 넣기")
09 if "":
10     print("빈 문자열은 True로 변환됩니다")
11 else:
12     print("빈 문자열은 False로 변환됩니다")
```



실행결과

```
# if 조건문에 0 넣기
0은 False로 변환됩니다

# if 조건문에 빈 문자열 넣기
빈 문자열은 False로 변환됩니다
```

- 나중에 구현하고자 구문을 비워 두는 경우

```
if zero == 0
```

```
    빈 줄 삽입
```

```
else:
```

```
    빈 줄 삽입
```

```
01  # 입력을 받습니다.
```

```
02  number = input("정수 입력> ")
```

```
03  number = int(number)
```

```
04
```

```
05  # 조건문 사용
```

```
06  if number > 0:
```

```
07      # 양수일 때: 아직 미구현 상태입니다.
```

```
08  else:
```

```
09      # 음수일 때: 아직 미구현 상태입니다.
```

pass 키워드

- IndentationError

- if 조건문 사이에는 무조건 들여쓰기 4칸 넣고 코드 작성해야 함

- pass 키워드

- 아무것도 작성하지 않고 임시적으로 비워 둠

```
01  # 입력을 받습니다.
02  number = input("정수 입력> ")
03  number = int(number)
04
05  # 조건문 사용
06  if number > 0:
07      # 양수일 때: 아직 미구현 상태입니다.
08      pass
09  else:
10      # 음수일 때: 아직 미구현 상태입니다.
11      pass
```

정리하기

- 불 만들기 : 비교 연산자

- 불 연산하기 : 논리 연산자
- 논리 연산자의 활용

- if 조건문이란

- 날짜/시간 활용하기
- 컴퓨터의 조건
- else 조건문의 활용
- elif 구문
- if 조건문을 효율적으로 사용하기
- False로 변환되는 값

- pass 키워드

리스트와 반복문

- 리스트 (list)

- 여러 가지 자료를 저장할 수 있는 자료
- 자료들을 모아서 사용할 수 있게 해 줌
- 대괄호 내부에 자료들 넣어 선언

```
>>> array = [273, 32, 103, "문자열", True, False]
>>> print(array)
[273, 32, 103, '문자열', True, False]
```

리스트 선언하고 요소에 접근하기

- **요소** (element)
 - 리스트의 대괄호 내부에 넣는 자료

```
[요소, 요소, 요소...]
```

```
>>> [1, 2, 3, 4]                                # 숫자만으로 구성된 리스트
[1, 2, 3, 4]
>>> ["안", "녕", "하", "세", "요"]              # 문자열만으로 구성된 리스트
['안', '녕', '하', '세', '요']
>>> [273, 32, 103, "문자열", True, False]        # 여러 자료형으로 구성된 리스트
[273, 32, 103, '문자열', True, False]
```

리스트 선언하고 요소에 접근하기

- 리스트 내부의 요소 각각 사용하려면 리스트 이름 바로 뒤에 대괄호 입력 후 자료의 위치 나타내는 숫자 입력

```
list_a = [273, 32, 103, "문자열", True, False]
```

list_a	273	32	103	문자열	True	False
	[0]	[1]	[2]	[3]	[4]	[5]

- 인덱스 (index)
 - 대괄호 안에 들어간 숫자

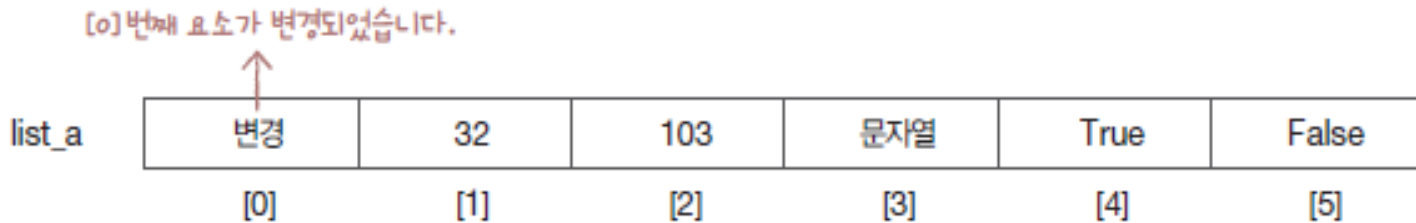
리스트 선언하고 요소에 접근하기

```
>>> list_a = [273, 32, 103, "문자열", True, False]
>>> list_a[0]
273
>>> list_a[1]
32
>>> list_a[2]
103
>>> list_a[1:3]
[32, 103]
```

리스트 선언하고 요소에 접근하기

- 리스트 특정 요소를 변경할 수 있음

```
>>> list_a = [273, 32, 103, "문자열", True, False]
>>> list_a[0] = "변경"
>>> list_a
['변경', 32, 103, '문자열', True, False]
```



리스트 선언하고 요소에 접근하기

- 대괄호 안에 음수 넣어 뒤에서부터 요소 선택하기

```
>>> list_a = [273, 32, 103, "문자열", True, False]
>>> list_a[-1]
False
>>> list_a[-2]
True
>>> list_a[-3]
'문자열'
```

273	32	103	문자열	True	False
[-6]	[-5]	[-4]	[-3]	[-2]	[-1]

리스트 선언하고 요소에 접근하기

- 리스트 접근 연산자를 이중으로 사용할 수 있음

```
>>> list_a = [273, 32, 103, "문자열", True, False]
>>> list_a[3]
'문자열'
>>> list_a[3][0]
'문'
```

- 리스트 여러 개를 가지는 리스트

```
>>> list_a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> list_a[1]
[4, 5, 6]
>>> list_a[1][1]
5
```

리스트 선언하고 요소에 접근하기

- 리스트에서의 `IndexError` 예외
 - 리스트의 길이 넘는 인덱스로 요소에 접근하려는 경우 발생

```
>>> list_a = [273, 32, 103]
>>> list_a[3]
```

오류

```
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    IndexError: list index out of range
```


리스트 연산자: 연결(+), 반복(*), len()

- 예시 – 리스트 연산자

```
01  # 리스트를 선언합니다.  
02  list_a = [1, 2, 3]  
03  list_b = [4, 5, 6]  
04  
05  # 출력합니다.  
06  print("# 리스트")  
07  print("list_a =", list_a)
```

리스트 연산자: 연결(+), 반복(*), len()

```
08 print("list_b =", list_b)
09 print()
10
11 # 기본 연산자
12 print("# 리스트 기본 연산자")
13 print("list_a + list_b =", list_a + list_b)
14 print("list_a * 3 =", list_a * 3)
15 print()
16
17 # 함수
18 print("# 길이 구하기")
19 print("len(list_a) =", len(list_a))
```

 실행결과 ×

```
# 리스트
list_a = [1, 2, 3]
list_b = [4, 5, 6]

# 리스트 기본 연산자
list_a + list_b = [1, 2, 3, 4, 5, 6]
list_a * 3 = [1, 2, 3, 1, 2, 3, 1, 2, 3]

# 길이 구하기
len(list_a) = 3
```

리스트에 요소 추가하기: append, insert

- append() 함수

- 리스트 뒤에 요소를 추가

```
리스트명.append(요소)
```

- insert() 함수

- 리스트 중간에 요소를 추가

```
리스트명.insert(위치, 요소)
```

리스트에 요소 추가하기: append, insert

- 예시

```
01  # 리스트를 선언합니다.
02  list_a = [1, 2, 3]
03
04  # 리스트 뒤에 요소 추가하기
05  print("# 리스트 뒤에 요소 추가하기")
06  list_a.append(4)
07  list_a.append(5)
08  print(list_a)
09  print()
10
11  # 리스트 중간에 요소 추가하기
12  print("# 리스트 중간에 요소 추가하기")
13  list_a.insert(0, 10)
14  print(list_a)
```

실행결과

리스트 뒤에 요소 추가하기
[1, 2, 3, 4, 5]

리스트 중간에 요소 추가하기
[10, 1, 2, 3, 4, 5]

리스트에 요소 추가하기: append, insert

- extend() 함수

- 원래 리스트 뒤에 새로운 리스트의 요소 모두 추가
- 매개변수로 리스트 입력

```
>>> list_a = [1, 2, 3]
>>> list_a.extend([4, 5, 6])
>>> print(list_a)
[1, 2, 3, 4, 5, 6]
```

리스트에 요소 추가하기: append, insert

- 리스트 연결 연산자와 요소 추가의 차이
 - 리스트 연결 연산자 사용하면 결과상 원본에 변화는 없음

```
>>> list_a = [1, 2, 3]
>>> list_b = [4, 5, 6]
>>> list_a + list_b → 리스트 연결 연산자로 연결하니,
[1, 2, 3, 4, 5, 6] → 실행결과로 [1, 2, 3, 4, 5, 6]이 나왔습니다.
>>> list_a → list_a와 list_b에는 어떠한 변화도 없습니다(비파괴적 처리).
[1, 2, 3]
>>> list_b
[4, 5, 6]
```


리스트에 요소 추가하기: append, insert

- extend() 함수 사용할 경우

```
>>> list_a = [1, 2, 3]
>>> list_b = [4, 5, 6]
>>> list_a.extend(list_b) → 실행결과로 아무 것도 출력하지 않았습니다.
>>> list_a → 앞에 입력했던 list_a 자체에 직접적인 변화가 있습니다(파괴적 처리).
[1, 2, 3, 4, 5, 6]
>>> list_b
[4, 5, 6]
```

리스트에 요소 제거하기

- 인덱스로 제거하기: `del` 키워드, `pop()` 함수

```
del 리스트명[인덱스]
```

```
리스트명.pop(인덱스)
```

```
01 list_a = [0, 1, 2, 3, 4, 5]
02 print("# 리스트의 요소 하나 제거하기")
03
04 # 제거 방법[1] - del
05 del list_a[1]
06 print("del list_a[1]:", list_a)
07
08 # 제거 방법[2] - pop()
09 list_a.pop(2)
10 print("pop(2):", list_a)
```

실행결과

```
# 리스트의 요소 하나 제거하기
del list_a[1]: [0, 2, 3, 4, 5]
pop(2): [0, 2, 4, 5]
```

리스트에 요소 제거하기

- **del 키워드** 사용할 경우 범위 지정해 리스트 요소를 한꺼번에 제거 가능

```
>>> list_b = [0, 1, 2, 3, 4, 5, 6]
>>> del list_b[3:6]
>>> list_b
[0, 1, 2, 6]
```

- 범위 한 쪽을 입력하지 않으면 지정 위치 기준으로 한쪽을 전부 제거

```
>>> list_c = [0, 1, 2, 3, 4, 5, 6]
>>> del list_c[:3]
>>> list_c
[3, 4, 5, 6]
```

리스트에 요소 제거하기

- 값으로 제거하기: `remove()` 함수
 - 특정 값을 지정하여 제거

```
리스트.remove(값)
```

```
>>> list_c = [1, 2, 1, 2]    # 리스트 선언하기
>>> list_c.remove(2)         # 리스트의 요소를 값으로 제거하기
>>> list_c
[1, 1, 2]
```

리스트에 요소 제거하기

- 모두 제거하기 : `clear()` 함수
 - 리스트 내부의 요소를 모두 제거

리스트.clear()

```
>>> list_d = [0, 1, 2, 3, 4, 5]
>>> list_d.clear()
>>> list_d
[] → 요소가 모두 제거되었습니다.
```

리스트 내부에 있는지 확인하기 : in/not in 연산자

- in 연산자

- 특정 값이 리스트 내부에 있는지 확인

값 in 리스트

```
>>> list_a = [273, 32, 103, 57, 52]
>>> 273 in list_a
True
>>> 99 in list_a
False
>>> 100 in list_a
False
>>> 52 in list_a
True
```

리스트 내부에 있는지 확인하기 : in/not in 연산자

- not in 연산자
 - 리스트 내부에 해당 값이 없는지 확인

```
>>> list_a = [273, 32, 103, 57, 52]
>>> 273 not in list_a
False
>>> 99 not in list_a
True
>>> 100 not in list_a
True
>>> 52 not in list_a
False
>>> not 273 in list_a
False
```

for 반복문

- 반복문
 - 컴퓨터에 반복 작업을 지시

```
print("출력")  
print("출력")  
print("출력")  
print("출력")  
print("출력")
```

```
for i in range(100):  
    print("출력")
```

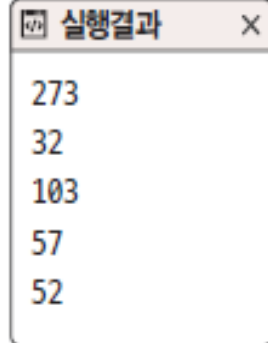
→ 반복에 사용할 수 있는 자료

for 반복문 : 리스트와 함께 사용하기

- 문자열, 리스트, 딕셔너리 등과 조합하여 for 반복문을 사용

```
for 반복자 in 반복할 수 있는 것:  
    코드
```

```
01  # 리스트를 선언합니다.  
02  array = [273, 32, 103, 57, 52]  
03  
04  # 리스트에 반복문을 적용합니다.  
05  for element in array:  
06      # 출력합니다.  
07      print(element)
```



실행결과

```
273  
32  
103  
57  
52
```

정리하기

- **리스트 선언하고 요소에 접근하기**

- 리스트 연산자: 연결(+), 반복(*), len()
- 리스트에 요소 추가하기: append, insert
- 리스트에 요소 제거하기
- 리스트 내부에 있는지 확인하기 : in/not in 연산자

- **for 반복문**

- for 반복문 : 리스트와 함께 사용하기

딕셔너리와 반복문

- **딕셔너리** (dictionary)

- 키를 기반으로 값을 저장하는 것

```
{  
  키  값  
  ↓  ↓  
  "키A": 10,      # 문자열을 키로 사용하기  
  "키B": 20,  
  "키C": 30,  
  1:    40,      # 숫자를 키로 사용하기  
  False: 50      # 불을 키로 사용하기  
}
```

자료형	의미	가리키는 위치	선언 형식
리스트	인덱스를 기반으로 값을 저장	인덱스	변수 = []
딕셔너리	키를 기반으로 값을 저장	키	변수 = {}

딕셔너리 선언하기

- 딕셔너리 선언
 - 중괄호로 선언하며 '키: 값' 형태를 쉼표로 연결해서 만들

```
변수 = {  
    키: 값,  
    키: 값,  
    ...  
    키: 값  
}
```

```
>>> dict_a = {  
    "name": "어벤저스 엔드게임",  
    "type": "히어로 무비"  
}
```

딕셔너리의 요소에 접근하기

- 특정 키 값만 따로 출력하기
 - 딕셔너리 뒤에 대괄호 입력하고 그 내부에 키 입력

```
>>> dict_a  
{'name': '어벤저스 엔드게임', 'type': '히어로 무비'}
```

```
>>> dict_a["name"]  
'어벤저스 엔드게임'  
>>> dict_a["type"]  
'히어로 무비'
```

딕셔너리의 요소에 접근하기

- 딕셔너리 내부 값에 문자열, 숫자, 불 등 다양한 자료 넣기

```
>>> dict_b = {  
    "director": ["안소니 루소", "조 루소"],  
    "cast": ["아이언맨", "타노스", "토르", "닥터스트레인지", "헐크"]  
}
```

```
>>> dict_b  
{'director': ['안소니 루소', '조 루소'], 'cast': ['아이언맨', '타노스', '토르', '닥터스트레인지', '헐크']}  
>>> dict_b["director"]  
['안소니 루소', '조 루소']
```

딕셔너리의 요소에 접근하기

- 리스트 안의 특정 값 출력하려는 경우


```
>>> dictionary["ingredient"]  
['망고', '설탕', '메타중아황산나트륨', '치자황색소']  
>>> dictionary["ingredient"][1]  
'설탕'
```


딕셔너리의 요소에 접근하기

- 딕셔너리의 문자열 키와 관련된 실수
- **NameError 오류**
 - name이라는 이름이 정의되지 않음

<https://docs.python.org/ko/3.8/library/functions.html?highlight=type#type>

```
>>> dict_key = {  
    name: "7D 건조 망고",  
    type: "당절임"  
}
```

 오류

```
Traceback (most recent call last):  
  File "<pyshell#5>", line 2, in <module>  
    name: "7D 건조 망고",  
NameError: name 'name' is not defined
```

딕셔너리의 요소에 접근하기

- name 이름을 변수로 만들어 해결

```
>>> name = "이름"
>>> dict_key = {
    name: "7D 건조 망고",
    type: "당절임"
}
>>> dict_key
{'이름': '7D 건조 망고', <class 'type'>: '당절임'}
```

딕셔너리에 값 추가하기/제거하기

- 딕셔너리에 값 추가할 때는 키를 기반으로 값 입력

딕셔너리[새로운 키] = 새로운 값

- dictionary에 새로운 자료 추가

```
>>> dictionary["price"] = 5000
>>> dictionary
{'name': '8D 건조 망고', 'type': '당절임', 'ingredient': ['망고', '설탕', '메타중아황산나트륨', '치자황색소'], 'origin': '필리핀', 'price': 5000} → "price" 키가 추가되었습니다.
```

- 딕셔너리에 이미 존재하는 키 지정하고 값 넣으면 기존 값을 대체

```
>>> dictionary["name"] = "8D 건조 파인애플"
>>> dictionary
{'name': '8D 건조 파인애플', 'type': '당절임', 'ingredient': ['망고', '설탕', '메타중아황산나트륨', '치자황색소'], 'origin': '필리핀', 'price': 5000}
```

새로운 값으로 대체되었습니다.

딕셔너리에 값 추가하기/제거하기

- 딕셔너리 요소의 제거 : del 키워드

```
>>> del dictionary["ingredient"]  
>>> dictionary  
{'name': '8D 건조 파인애플', 'type': '당절임', 'origin': '필리핀', 'price': 5000}
```

딕셔너리에 값 추가하기/제거하기

- 예시 – 딕셔너리에 요소 추가하기

```
01  # 딕셔너리를 선언합니다.
02  dictionary = {}
03
04  # 요소 추가 전에 내용을 출력해 봅니다.
05  print("요소 추가 이전:", dictionary)
06
07  # 딕셔너리에 요소를 추가합니다.
08  dictionary["name"] = "새로운 이름"
09  dictionary["head"] = "새로운 정신"
10  dictionary["body"] = "새로운 몸"
11
12  # 출력합니다.
13  print("요소 추가 이후:", dictionary)
```

실행결과


요소 추가 이전: {}

요소 추가 이후: {'name': '새로운 이름', 'head': '새로운 정신', 'body': '새로운 몸'}

딕셔너리에 값 추가하기/제거하기

- 예시 – 딕셔너리에 요소 제거하기

```
01  # 딕셔너리를 선언합니다.
02  dictionary = {
03      "name": "7D 건조 망고",
04      "type": "당절임"
05  }
06
07  # 요소 제거 전에 내용을 출력해 봅니다.
08  print("요소 제거 이전:", dictionary)
09
10  # 딕셔너리의 요소를 제거합니다.
11  del dictionary["name"]
12  del dictionary["type"]
13
14  # 요소 제거 후에 내용을 출력해 봅니다.
15  print("요소 제거 이후:", dictionary)
```


 실행결과

요소 제거 이전: {'name': '7D 건조 망고', 'type': '당절임'}
요소 제거 이후: {}

딕셔너리에 값 추가하기/제거하기

- **KeyError 예외**
 - 딕셔너리에서 존재하지 않는 키에 접근할 경우

```
>>> dictionary = {}  
>>> dictionary["Key"]
```

 오류

```
Traceback (most recent call last):  
  File "<pyshell#7>", line 1, in <module>  
    dictionary["Key"]  
KeyError: 'Key'
```

딕셔너리에 값 추가하기/제거하기

- 값 제거할 경우도 같은 원리

```
>>> del dictionary["Key"]
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    del dictionary["Key"]
KeyError: 'Key'
```


딕셔너리 내부에 키가 있는지 확인하기


- in 키워드

- 사용자로부터 접근하고자 하는 키 입력 받은 후 존재하는 경우에만 접근하여 값을 출력

```
01  # 딕셔너리를 선언합니다.
02  dictionary = {
03      "name": "7D 건조 망고",
04      "type": "당절임",
05      "ingredient": ["망고", "설탕", "메타중아황산나트륨", "치자황색소"],
06      "origin": "필리핀"
07  }
08
09  # 사용자로부터 입력을 받습니다.
10  key = input("> 접근하고자 하는 키: ")
```

딕셔너리 내부에 키가 있는지 확인하기

```
11
12 # 출력합니다.
13 if key in dictionary:
14     print(dictionary[key])
15 else:
16     print("존재하지 않는 키에 접근하고 있습니다.")
```

 실행결과 ✕

> 접근하고자 하는 키: name

7D 건조 망고

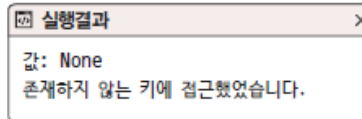
> 접근하고자 하는 키: ㅇ ㅁ ㅇ

존재하지 않는 키에 접근하고 있습니다.

딕셔너리 내부에 키가 있는지 확인하기

- `get()` 함수
 - 딕셔너리의 키로 값을 추출
 - 존재하지 않는 키에 접근할 경우 `None` 출력

```
01 # 딕셔너리를 선언합니다.
02 dictionary = {
03     "name": "7D 건조 망고",
04     "type": "당절임",
05     "ingredient": ["망고", "설탕", "메타중아황산나트륨", "치자황색소"],
06     "origin": "필리핀"
07 }
08
09 # 존재하지 않는 키에 접근해 봅니다.
10 value = dictionary.get("존재하지 않는 키")
11 print("값:", value)
12
13 # None 확인 방법
14 if value == None: → None과 같은지 확인만 하면 됩니다.
15     print("존재하지 않는 키에 접근했었습니다.")
```

실행결과 창은 제목 '실행결과'와 닫기 버튼 'X'을 가지고 있습니다. 창 내부에는 두 줄의 텍스트가 표시되어 있습니다: '값: None'과 '존재하지 않는 키에 접근했었습니다.'

실행결과

값: None
존재하지 않는 키에 접근했었습니다.

for 반복문 : 딕셔너리와 함께 사용하기

- for 반복문과 딕셔너리의 조합

```
for 키 변수 in 딕셔너리:  
    코드
```

for 반복문 : 딕셔너리와 함께 사용하기

```
01 # 딕셔너리를 선언합니다.
02 dictionary = {
03     "name": "7D 건조 망고",
04     "type": "당절임",
05     "ingredient": ["망고", "설탕", "메타중아황산나트륨", "치자황색소"],
06     "origin": "필리핀"
07 }
08
09 # for 반복문을 사용합니다.
10 for key in dictionary:
11     # 출력합니다.
12     print(key, ":", dictionary[key])
```

실행결과

```
name : 7D 건조 망고
type : 당절임
ingredient : ['망고', '설탕', '메타중아황산나트륨', '치자황색소']
origin : 필리핀
```

- 범위 (range)

- 특정 횟수만큼 반복해서 돌리고 싶을 때 for 반복문과 조합하여 사용

범위

- 매개변수에 숫자를 한 개 넣는 방법
 - 0부터 A-1까지의 정수로 범위 만들기

`range(A)` → A는 숫자

- 매개변수에 숫자를 두 개 넣는 방법
 - A부터 B-1까지의 정수로 범위 만들기

`range(A, B)` → A와 B는 숫자

- 매개변수에 숫자를 세 개 넣는 방법
 - A부터 B-1까지의 정수로 범위 만들되 앞뒤의 숫자가 c만큼의 차이 가짐

`range(A, B, C)` → A, B, C는 숫자

범위

- 예시
 - 매개변수에 숫자 한 개 넣은 범위

```
>>> a = range(5)
```

```
>>> a  
range(0, 5)
```

```
>>> list(range(10))  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```


범위

- 매개변수에 숫자 두 개 넣은 범위

```
>>> list(range(0, 5)) → 0부터 (5-1)까지의 정수로 범위를 만듭니다.
```

```
[0, 1, 2, 3, 4]
```

```
>>> list(range(5, 10)) → 5부터 (10-1)까지의 정수로 범위를 만듭니다.
```

```
[5, 6, 7, 8, 9]
```

- 매개변수에 숫자 세 개 넣은 범위

```
>>> list(range(0, 10, 2)) → 0부터 2씩 증가하면서 (10-1)까지의 정수로 범위를 만듭니다.
```

```
[0, 2, 4, 6, 8]
```

```
>>> list(range(0, 10, 3)) → 0부터 3씩 증가하면서 (10-1)까지의 정수로 범위를 만듭니다.
```

```
[0, 3, 6, 9]
```

범위

- 범위 만들 때 매개변수 내부에 수식 사용하는 경우
 - 코드 특정 부분의 강조

```
>>> a = range(0, 10 + 1)
>>> list(a)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

- 예시 - 나누기 연산자 사용

```
>>> n = 10
>>> a = range(0, n / 2) → 매개변수로 나눗셈을 사용한 경우 오류가 발생합니다.
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
TypeError: 'float' object cannot be interpreted as an integer
```

- TypeError 발생

- 정수 나누기 연산자

```
>>> a = range(0, int(n / 2)) → 실수를 정수로 바꾸는 방법보다
```

```
>>> list(a)
```

```
[0, 1, 2, 3, 4]
```

```
>>> a = range(0, n // 2) → 정수 나누기 연산자를 많이 사용합니다!
```

```
>>> list(a)
```

```
[0, 1, 2, 3, 4]
```

for 반복문: 범위와 함께 사용하기

- for 반복문과 범위의 조합

for 숫자 변수 in 범위:

코드

```
01  # for 반복문과 범위를 함께 조합해서 사용합니다.
02  for i in range(5):
03      print(str(i) + "= 반복 변수")
04  print()
05
06  for i in range(5, 10):
07      print(str(i) + "= 반복 변수")
08  print()
09
10  for i in range(0, 10, 3):
11      print(str(i) + "= 반복 변수")
12  print()
```

실행결과

```
0 = 반복 변수
1 = 반복 변수
2 = 반복 변수
3 = 반복 변수
4 = 반복 변수

5 = 반복 변수
6 = 반복 변수
7 = 반복 변수
8 = 반복 변수
9 = 반복 변수

0 = 반복 변수
3 = 반복 변수
6 = 반복 변수
9 = 반복 변수
```

for 반복문 : 리스트와 범위 조합하기

- 몇 번 반복인지를 알아야 하는 경우

```
# 리스트를 선언합니다.  
array = [273, 32, 103, 57, 52]  
  
# 리스트에 반복문을 적용합니다.  
for element in array:  
    # 출력합니다.  
    print(element)
```

현재 무엇을 출력하고 있는지 보다, 몇 번째 출력인지를 알아야 하는 경우가 있습니다.

```
01 # 리스트를 선언합니다.  
02 array = [273, 32, 103, 57, 52]  
03  
04 # 리스트에 반복문을 적용합니다.  
05 for i in range(len(array)):  
06     # 출력합니다.  
07     print("{}번째 반복: {}".format(i, array[i]))
```

실행결과	
0번째 반복:	273
1번째 반복:	32
2번째 반복:	103
3번째 반복:	57
4번째 반복:	52

for 반복문: 반대로 반복하기

- 역반복문

- 큰 숫자에서 작은 숫자로 반복문 적용
- `range()` 함수의 매개변수 세 개 사용하는 방법

```
01  # 역반복문
02  for i in range(4, 0 - 1, -1):
03      # 출력합니다.
04      print("현재 반복 변수: {}".format(i))
```

실행결과

현재 반복 변수: 4
현재 반복 변수: 3
현재 반복 변수: 2
현재 반복 변수: 1
현재 반복 변수: 0

for 반복문: 반대로 반복하기

- `reversed()` 함수 사용하는 방법

```
01  # 역반복문
02  for i in reversed(range(5)):
03      # 출력합니다.
04      print("현재 반복 변수: {}".format(i))
```

실행결과

현재 반복 변수: 4
현재 반복 변수: 3
현재 반복 변수: 2
현재 반복 변수: 1
현재 반복 변수: 0

while 반복문

- while 반복문

- 리스트 또는 딕셔너리 내부의 요소를 특정 횟수만큼 반복

```
while 불 표현식:  
    문장
```


```
01  # while 반복문을 사용합니다.  
02  while True:  
03      # "."을 출력합니다.  
04      # 기본적으로 end가 "\n"이라 줄바꿈이 일어나는데  
05      # 빈 문자열 ""로 바꿔서 줄바꿈이 일어나지 않게 합니다.  
06      print(".", end="")
```

실행결과

```
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....
```


while 반복문 : for 반복문처럼 사용하기

```
01  # 반복 변수를 기반으로 반복하기
02  i = 0
03  while i < 10:
04      print("{}번째 반복입니다.".format(i))
05      i += 1
```

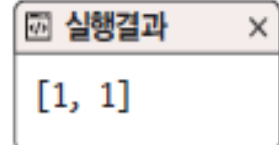
 실행결과 ×

0번째 반복입니다.
1번째 반복입니다.
2번째 반복입니다.
3번째 반복입니다.
4번째 반복입니다.
5번째 반복입니다.
6번째 반복입니다.
7번째 반복입니다.
8번째 반복입니다.
9번째 반복입니다.

while 반복문: 상태를 기반으로 반복하기

- 리스트 내부에서 해당하는 값을 여러 개 제거
 - while 반복문의 조건을 '리스트 내부에 요소가 있는 동안'으로 지정

```
01  # 변수를 선언합니다.  
02  list_test = [1, 2, 1, 2]  
03  value = 2  
04  
05  # list_test 내부에 value가 있다면 반복  
06  while value in list_test:  
07      list_test.remove(value)  
08  
09  # 출력합니다.  
10  print(list_test)
```



실행결과

[1, 1]

while 반복문 : 시간을 기반으로 반복하기

- 예시 - 유닉스 타임 구하기
 - 시간 관련된 기능 가져오기

```
>>> import time
```

- 유닉스 타임


```
>>> time.time()  
1557241486.6654928
```

<https://docs.python.org/ko/3.8/library/time.html?highlight=time#module-time>

while 반복문 : 시간을 기반으로 반복하기

- 유닉스 타임과 while 반복문을 조합
 - 5초 동안 반복하기

```
01  # 시간과 관련된 기능을 가져옵니다.  
02  import time  
03  
04  # 변수를 선언합니다.  
05  number = 0  
06  
07  # 5초 동안 반복합니다.  
08  target_tick = time.time() + 5  
09  while time.time() < target_tick:  
10      number += 1  
11  
12  # 출력합니다.  
13  print("5초 동안 {}번 반복했습니다.".format(number))
```

 실행결과

5초 동안 14223967번 반복했습니다.

while 반복문: break 키워드/continue 키워드

- break 키워드
 - 반복문 벗어날 때 사용하는 키워드

```
01  # 변수를 선언합니다.
02  i = 0
03
04  # 무한 반복합니다.
05  while True:
06      # 몇 번째 반복인지 출력합니다.
07      print("{}번째 반복문입니다.".format(i))
08      i = i + 1
09      # 반복을 종료합니다.
10      input_text = input("> 종료하시겠습니까?(y): ")
11      if input_text in ["y", "Y"]:
12          print("반복을 종료합니다.")
13          break
```

실행결과

0번째 반복문입니다
> 종료하시겠습니까?(y/n): n

1번째 반복문입니다
> 종료하시겠습니까?(y/n): n

2번째 반복문입니다
> 종료하시겠습니까?(y/n): n

3번째 반복문입니다
> 종료하시겠습니까?(y/n): n

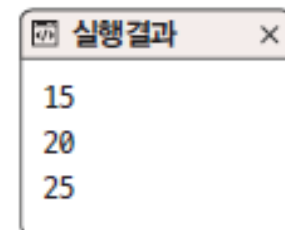
4번째 반복문입니다
> 종료하시겠습니까?(y/n): y

반복을 종료합니다.

while 반복문: break 키워드/continue 키워드

- continue 키워드
 - 현재 반복을 생략하고 다음 반복으로 넘어감

```
01  # 변수를 선언합니다.  
02  numbers = [5, 15, 6, 20, 7, 25]  
03  
04  # 반복을 돌립니다.  
05  for number in numbers:  
06      # number가 10보다 작으면 다음 반복으로 넘어갑니다.  
07      if number < 10:  
08          continue  
09      # 출력합니다.  
10      print(number)
```



while 반복문: break 키워드/continue 키워드

- if else 구문 사용도 가능한 경우이나, continue 키워드 사용하면 이후 처리의 들여쓰기를 하나 줄일 수 있음

continue 키워드를 사용하지 않은 경우

```
# 반복을 돌립니다.  
for number in numbers:  
    # 반복 대상을 한정합니다.  
    if number >= 10:  
        # 문장  
        # 문장  
        # 문장  
        # 문장  
        # 문장
```

continue 키워드를 사용한 경우

```
# 반복을 돌립니다.  
for number in numbers:  
    # 반복 대상에서 제외해버립니다.  
    if number < 10:  
        continue  
    # 문장  
    # 문장  
    # 문장  
    # 문장  
    # 문장
```

- **딕셔너리 선언하기**

- 딕셔너리의 요소에 접근하기
- 딕셔너리에 값 추가하기/제거하기
- 딕셔너리 내부에 키가 있는지 확인하기
- for 반복문 : 딕셔너리와 함께 사용하기

정리하기

- **범위(range)**

- for 반복문: 범위와 함께 사용하기
- for 반복문 : 리스트와 범위 조합하기
- for 반복문: 반대로 반복하기

- **while 반복문**

- while 반복문 : for 반복문처럼 사용하기
- while 반복문: 상태를 기반으로 반복하기
- while 반복문 : 시간을 기반으로 반복하기
- while 반복문: break 키워드/continue 키워드