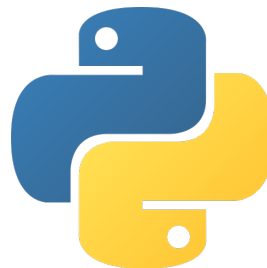




Python 프로그래밍 [4]

[리스트 / 반복문 / 딕셔너리]



SW융합학부

강희국

※ 수강생을 위한 참고자료로 제3자에 대한 배포를 금지합니다. 법적인 문제 발생 시 배포자에게 책임이 있습니다.

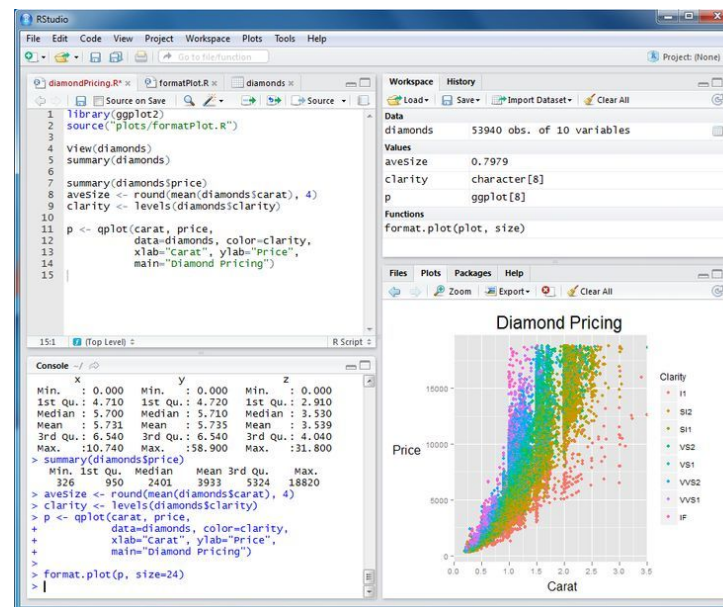
1 리스트와 반복문

2 딕셔너리와 반복문

3 반복문과 while 반복문

4 관련 기본 함수

5 -



1. 리스트와 반복문



[참고] 윤인성, "혼자 공부하는 파이썬", 한빛미디어

[핵심 키워드] 리스트, 요소, 인덱스, for 반복문

[핵심 포인트] 여러 개의 값을 나타낼 수 있게 해주는 리스트, 딕셔너리 등의 자료형도 존재한다. 이번 절에서는 리스트에 대해 알아보고, 이러한 자료가 반복문에 의해 어떻게 활용되는지 살펴본다.

● 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]
```

- 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 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

```
>>> 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]
```

- 결과로 [32, 103] 출력

- 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]
```

[0]번째 요소가 변경되었습니다.

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

- 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 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
```

● 리스트에서의 IndexError 예외

- 리스트의 길이 넘는 인덱스로 요소에 접근하려는 경우 발생

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

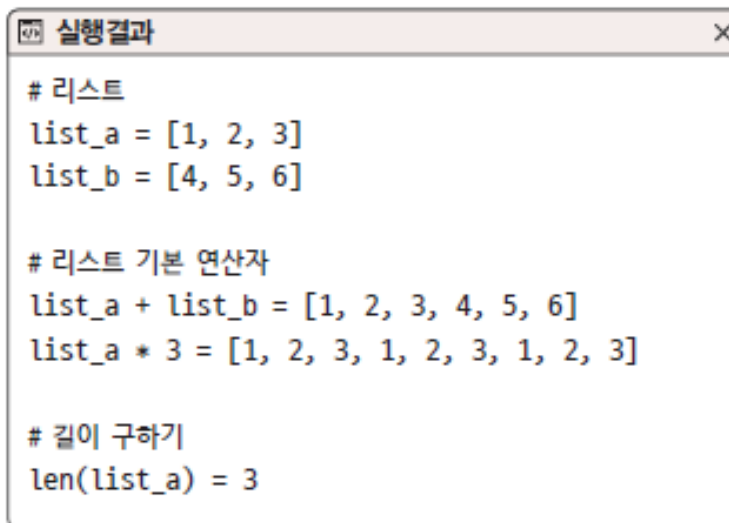
오류

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

● 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)
```

```
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
```

- 13행에서 문자열 연결 연산자 사용해 2, 3행과 7, 8행에서 선언 및 출력된 list_a와 list_b의 자료 연결
- 14행에서 문자열 반복 연산자 사용해 list_a의 자료 3번 반복
- 19행에서 len() 함수로 list_a에 들어있는 요소의 개수 구함

- append() 함수

- 리스트 뒤에 요소를 추가

```
리스트명.append(요소)
```

- insert() 함수

- 리스트 중간에 요소를 추가

```
리스트명.insert(위치, 요소)
```


● 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]
```

➤ 6 및 7행 실행 결과

```
list_a.append(4)
list_a.append(5)
```

1	2	3	4	5
[0]	[1]	[2]	[3]	[4]

➤ 13행 실행 결과

```
list_a.insert(0, 10)
```

삽입할 위치
삽입할 값

10	1	2	3	4	5
[0]	[1]	[2]	[3]	[4]	[5]

0번째 위치에 10을 추가합니다.

● 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]
```

- 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]
```

➤ 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]
```

➤ 파괴적 / 비파괴적

- 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]
```

- 5행 실행하면 자료에서 1 제거

```
del list_a[1]
```

0	2	3	4	5
[0]	[1]	[2]	[3]	[4]

- 9행에서 2번째 요소인 3 제거

```
list_a.pop(2)
```

0	2	4	5
[0]	[1]	[2]	[3]

- **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]
```


- 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]
```

● 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.
```

● 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
```

● 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
```

● 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

- 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

- **리스트** : 여러 가지 자료를 저장할 수 있는 자료형
- **요소** : 리스트 내부에 있는 각각의 내용을 의미
- **인덱스** : 리스트 내부에서 값의 위치를 의미
- **for 반복문** : 특정 코드를 반복해서 실행할 때 사용하는 기본 구문

- list_a = [0, 1, 2, 3, 4, 5, 6, 7] 입니다. 다음 표의 함수들을 실행했을 때 list_a의 결과가 어떻게 나오는지 적어보세요

함수	list_a의 값
list_a.extend(list_a)	
list_a.append(10)	
list_a.insert(3, 0)	
list_a.remove(3)	
list_a.pop(3)	
list_a.clear()	

- 다음 반복문 내부에 if 조건문의 조건식을 채워서 100 이상의 숫자만 출력하게 만들어보세요.

```
numbers = [273, 103, 5, 32, 65, 9, 72, 800, 99]

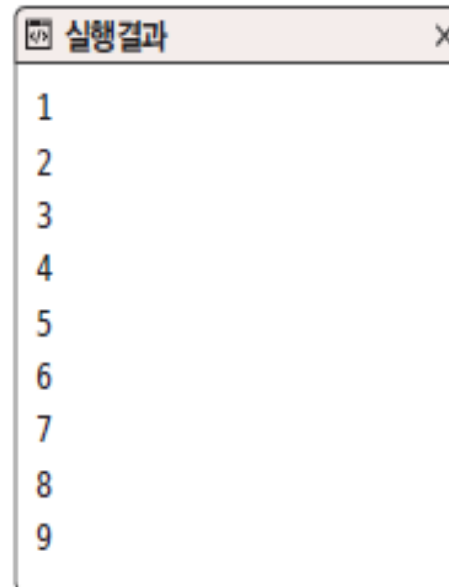
for number in numbers:
    if 
        print("- 100 이상의 수:", number)
```

실행결과

- 100 이상의 수: 273
- 100 이상의 수: 103
- 100 이상의 수: 800

- 다음 빈칸을 채워서 실행결과처럼 숫자를 하나하나 모두 출력해보세요

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



```
실행결과 X  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

2. 딕셔너리와 반복문



[참고] 윤인성, "혼자 공부하는 파이썬", 한빛미디어

[핵심 키워드] : 딕셔너리, 키, 값

[핵심 포인트]

여러 개의 값을 나타낼 수 있게 해주는 자료형 중
딕셔너리에 대해 알아봅니다.

● 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 선언

- 중괄호로 선언하며 '키: 값' 형태를 쉼표로 연결해서 만듦

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

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

- 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'
```

- 딕셔너리 내부 값에 문자열, 숫자, 불 등 다양한 자료 넣기

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

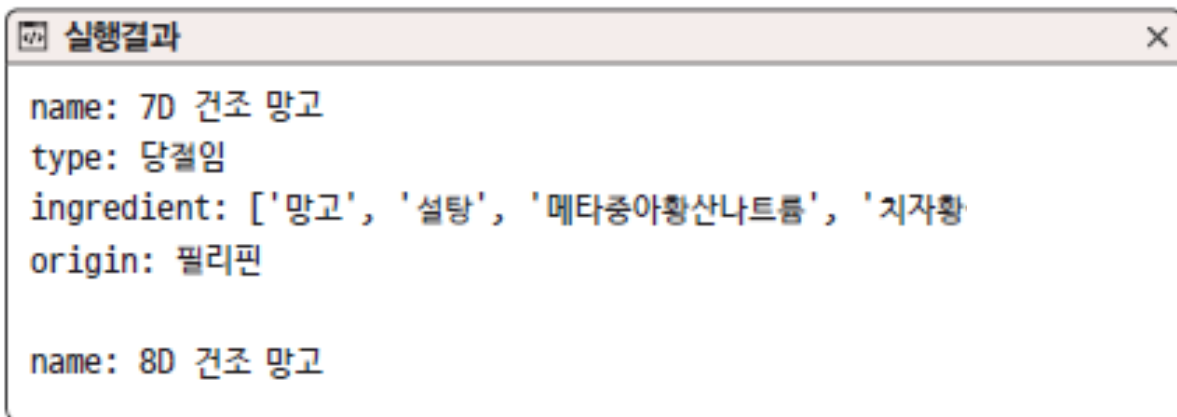
```
>>> dict_b  
{'director': ['안소니 루소', '조 루소'], 'cast': ['아이언맨', '타노스', '토르', '닥터스트레인지', '헐크']}  
>>> dict_b["director"]  
['안소니 루소', '조 루소']
```


구분	선언 형식	사용 예	불린 예
리스트	<code>list_a = []</code>	<code>list_a[1]</code>	
딕셔너리	<code>dict_a = {}</code>	<code>dict_a["name"]</code>	<code>dict_a{"name"}</code>

➤ 예시 - 딕셔너리 요소에 접근하기

```
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  # 출력합니다.  
10  print("name:", dictionary["name"])  
11  print("type:", dictionary["type"])  
12  print("ingredient:", dictionary["ingredient"])  
13  print("origin:", dictionary["origin"])  
14  print()  
15  
16  # 값을 변경합니다.  
17  dictionary["name"] = "8D 건조 망고"  
18  print("name:", dictionary["name"])
```



```
실행결과  
name: 7D 건조 망고  
type: 당절임  
ingredient: ['망고', '설탕', '메타중아황산나트륨', '치자황'  
origin: 필리핀  
  
name: 8D 건조 망고
```

- 리스트 안의 특정 값 출력하려는 경우

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

- 딕셔너리의 문자열 키와 관련된 실수
- **NameError 오류**

➤ name이라는 이름이 정의되지 않음

```
>>> 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
```

- 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 추가할 때는 키를 기반으로 값 입력

딕셔너리[새로운 키] = 새로운 값

- 슬라이드 #8, 9에서 만든 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 대치

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

새로운 값으로 대치되었습니다.

- 딕셔너리 요소의 제거 : del 키워드

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

➤ 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'}

➤ 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: {}

● 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'
```

- 값 제거할 경우도 같은 원리

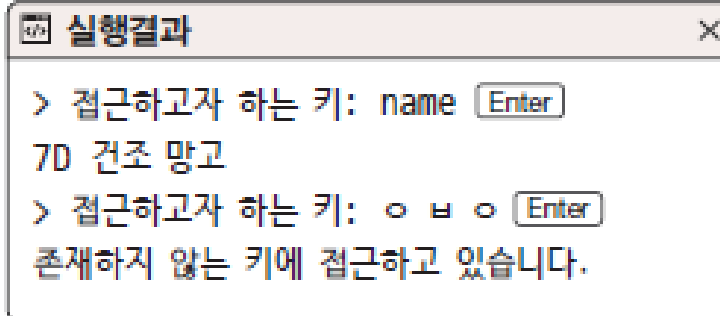
```
>>> del dictionary["Key"]  
Traceback (most recent call last):  
  File "<pyshell#8>", line 1, in <module>  
    del dictionary["Key"]  
KeyError: 'Key'
```

● 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: ")
```

```
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 Enter

7D 건조 망고

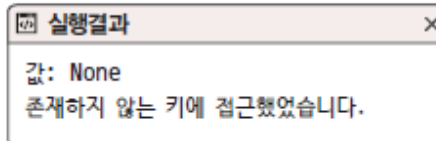
> 접근하고자 하는 키: ㅇ ㅁ ㅇ Enter

존재하지 않는 키에 접근하고 있습니다.

● 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

값: None
존재하지 않는 키에 접근했었습니다.

- for 반복문과 딕셔너리의 조합

```
for 키 변수 in 딕셔너리:  
    코드
```

```
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
```


- **딕셔너리** : 키를 기반으로 여러 자료 저장하는 자료형
- **키** : 딕셔너리 내부에서 값에 접근할 때 사용하는 것
- **값** : 딕셔너리 내부에 있는 각각의 내용

- 다음 표에서 dict_a의 결과가 나오도록 빈칸을 채워보세요.

dict_a의 값	dict_a에 적용할 코드	dict_a의 결과
<code>{}</code>	<input type="text"/>	<code>{ "name": "구름" }</code>
<code>{ "name": "구름" }</code>	<input type="text"/>	<code>{}</code>

- ```
숫자는 무작위로 입력해도 상관 없습니다.
numbers = [1,2,6,8,4,3,2,1,9,5,4,9,7,2,1,3,5,4,8,9,7,2,3]
counter = {}
```

```
for number in numbers:
```

```
최종 출력
print(counter)
```

## ☐ 실행결과

{1: 3, 2: 4, 6: 1, 8: 2, 4: 3, 3: 3, 9: 3, 5: 2, 7: 2}

- 아래 예시를 참조해 다음 빈칸을 채워 실행결과와 같이 출력되게 만들어보세요.

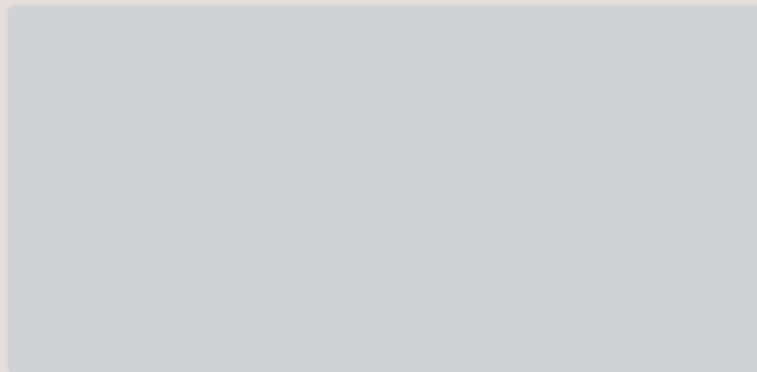
```
type("문자열") is str # 문자열인지 확인
type([]) is list # 리스트인지 확인
type({}) is dict # 딕셔너리인지 확인
```

# 딕셔너리를 선언합니다.

```
character = {
 "name": "기사",
 "level": 12,
 "items": {
 "sword": "불꽃의 검",
 "armor": "풀플레이트"
 },
 "skill": ["베기", "세게 베기", "아주 세게 베기"]
}
```

# for 반복문을 사용합니다.

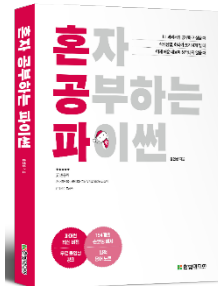
```
for key in character:
```



실행결과

```
name : 기사
level : 12
sword : 불꽃의 검
armor : 풀플레이트
skill : 베기
skill : 세게 베기
skill : 아주 세게 베기
```

### 3. 반복문과 while 반복문



[참고] 윤인성, "혼자 공부하는 파이썬", 한빛미디어

**[핵심 키워드]** : 범위, while 반복문, break 키워드, continue 키워드

**[핵심 포인트]**

특정 횟수 / 특정 시간만큼, 그리고 어떤 조건이 될 때까지 반복하는 등의 경우에 대해 알아본다.

- 범위 (range)

- 특정 횟수만큼 반복해서 돌리고 싶을 때 for 반복문과 조합하여 사용



- 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 만들기 앞뒤의 숫자가 C만큼의 차이 가짐

`range(A, B, C)` → A, B, C는 숫자

## ➤ 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]
```

- 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 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 연산사 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 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

- 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  |

## ● 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 |



## ➤ 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

- 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

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

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

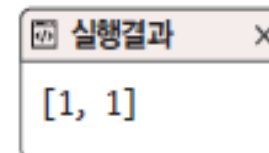
실행결과

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

- 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]

- 예시 - 유닉스 타임 구하기

- 시간 관련된 기능 가져오기

```
>>> import time
```

- 유닉스 타임

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

## ● 유닉스 타임과 while 반복문을 조합

### ➤ 5초 동안 바보하기

```
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.

## ● 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
```

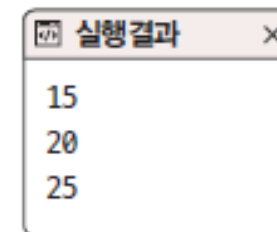
실행결과

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.

## ● 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)
```



실행결과

|    |
|----|
| 15 |
| 20 |
| 25 |



- 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
```

- **범위** : 정수의 범위 나타내는 값으로, range() 함수로 생성
- **while 반복문** : 조건식을 기반으로 특정 코드를 반복해서 실행할 때 사용하는 구문
- **break 키워드** : 반복문을 벗어날 때 사용하는 구문
- **continue 키워드** : 반복문의 현재 반복을 생략할 때 사용하는 구문

- 다음 표를 채워 보세요.

코드가 여러 개 나올 수 있는 경우 가장 간단한 형태를 넣어 주세요.

| 코드                           | 나타내는 값                       |
|------------------------------|------------------------------|
| <code>range(5)</code>        | <code>[0, 1, 2, 3, 4]</code> |
| <code>range(4, 6)</code>     |                              |
| <code>range(7, 0, -1)</code> |                              |
| <code>range(3, 8)</code>     | <code>[3, 4, 5, 6, 7]</code> |
|                              | <code>[3, 6, 9]</code>       |

- 빈칸을 채워 키와 값으로 이루어진 각 리스트를 조합해 하나의 딕셔너리를 만들어 보세요.

```
숫자는 무작위로 입력해도 상관없습니다.
key_list = ["name", "hp", "mp", "level"]
value_list = ["기사", 200, 30, 5]
character = {}
```

```
최종 출력
print(character)
```

실행결과

```
{'name': '기사', 'hp': 200, 'mp': 30, 'level': 5}
```

- 1부터 숫자를 하나씩 증가시키면서 더하는 경우를 생각해 봅시다. 몇을 더할 때 1000을 넘는지 구해 보세요. 그리고 그때의 값도 출력해보세요. 다음은 1000이 넘는 경우를 구한 예입니다.

1,  $1 + 2 = 3$ ,  $1 + 2 + 3 = 6$ ,  $1 + 2 + 3 + 4 = 10...$

```
limit = 10000
i = 1
sum은 파이썬 내부에서 사용하는 식별자이므로 sum_value라는 변수 이름을 사용합니다.

print("{}를 더할 때 {}을 넘으며 그때의 값은 {}입니다.".format(i, limit, sum_value))
```

실행결과

142를 더할 때 10000을 넘으며 그때의 값은 10011입니다.