

# 딕셔너리와 반복문

# 목차

- 시작하기 전에
- 딕셔너리 선언하기
- 딕셔너리의 요소에 접근하기
- 딕셔너리에 값 추가하기/제거하기
- 딕셔너리 내부에 키가 있는지 확인하기
- for 반복문 : 딕셔너리와 함께 사용하기
- 키워드로 정리하는 핵심 포인트
- 확인문제

**[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 만들

```
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	틀린 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', '치자황산']  
origin: 필리핀  
  
name: 8D 건조 망고
```

- 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

```
>>> 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 추가하기/제거하기

- 딕셔너리에 값 추가할 때는 키를 기반으로 값 입력

딕셔너리[새로운 키] = 새로운 값

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

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

새로운 값으로 대체되었습니다.

- 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: {}
```



- **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'
```

# 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 Enter  
70 건조 망고

> 접근하고자 하는 키: ㅇ ㅂ ㅇ Enter  
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

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
```

- **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>

- 다음 빈칸을 채워서 numbers 내부에 들어있는 숫자가 몇 번 등장하는지를 출력하는 코드를 작성해보세요.

# 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:
```

```
    # 빈칸 채우기
```

# 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
```