

# 2026.01.20.화

🕒 생성일	@2026년 1월 20일 오전 9:18
🏷 태그	Python

## 목차

1. 튜플
2. 딕셔너리
3. 집합

### ▼ 1) 튜플

- 순서가 있음
  - 요소들이 특정 순서로 저장
- 변경 불가능(Immutable)
  - 생성 된 후에는 요소를 변경, 추가, 삭제할 수 없음
- 다양한 자료형 저장
  - 문자열, 숫자, 리스트 등 다양한 타입 저장 가능
- 인덱싱과 슬라이싱
  - 리스트와 유사하게 위치로 접근 가능
- 중복 허용
  - 동일한 값을 여러 번 저장할 수 있음

```
# 1. 소괄호로 생성
coordinates = (10, 20)

# 2. tuple() 함수 사용
numbers = tuple([1, 2, 3, 4, 5]) # (1, 2, 3, 4, 5)

# 3. 소괄호 없이도 생성 가능
colors = "red", "green", "blue" # ('red', 'green', 'blue')

# 4. 단일 요소 튜플 (주의: 콤마 필요)
single_item = (42,) # (42,)
not_tuple = (42) # 42 (튜플이 아닌 일반 정수)

# 5. 빈 튜플 생성
empty_tuple = ()
empty_tuple2 = tuple()

# 6. 다양한 데이터 타입 혼합
mixed_tuple = (1, "안녕", 3.14, True, (1, 2, 3))

# 튜플 접근과 슬라이싱
fruits = ("사과", "바나나", "체리", "딸기", "오렌지")

# 인덱싱 (리스트와 동일)
print(fruits) # '사과'
print(fruits[-1]) # '오렌지'

# 슬라이싱 (리스트와 동일)
print(fruits[1:4]) # ('바나나', '체리', '딸기')
```

```

print(fruits[:3]) # ('사과', '바나나', '체리')
print(fruits[::-2]) # ('사과', '체리', '오렌지')

# 중첩 튜플 접근
nested = (1, 2, (3, 4, 5))
print(nested[2][1]) # 4

# 튜플 길이
print(len(fruits)) # 5

# 요소 존재 여부 확인
print("바나나" in fruits) # True

# 튜플은 변경 불가능
coordinates = (10, 20, 30)

# 아래 코드는 오류 발생
# coordinates = 100 # TypeError: 'tuple' object does not support item assignment

# 튜플 자체를 새로운 튜플로 재할당은 가능
coordinates = (100, 200, 300) # 새로운 튜플 객체 생성

# 튜플 메서드와 연산
numbers = (1, 2, 3, 2, 4, 2)

# count(): 특정 요소 개수 반환
print(numbers.count(2)) # 3 (2가 3개 있음)

# index(): 특정 요소의 첫 번째 위치 반환
print(numbers.index(3)) # 2 (3은 인덱스 2에 위치)

# 튜플 연결
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
combined = tuple1 + tuple2 # (1, 2, 3, 4, 5, 6)

# 튜플 반복
repeated = tuple1 * 3 # (1, 2, 3, 1, 2, 3, 1, 2, 3)

# 튜플 비교 (요소별 비교)
print((1, 2, 3) < (1, 3, 0)) # True (첫 번째 요소가 같아서 두 번째 요소 비교)

# 튜플 언패킹(Unpacking)
# 기본 언패킹
rgb = (255, 100, 50)
red, green, blue = rgb
print(f"Red: {red}, Green: {green}, Blue: {blue}") # Red: 255, Green: 100, Blue: 50

# 일부 요소만 언패킹
numbers = (1, 2, 3, 4, 5)
first, *middle, last = numbers
print(first) # 1
print(middle) # [2, 3, 4]
print(last) # 5

# 함수에서 여러 값 반환 시 튜플 활용
def get_user_info():
    return "홍길동", 30, "서울"

name, age, city = get_user_info()

```

```

print(f"{name}은 {age}세이고 {city}에 살고 있습니다.")

# 좌표 데이터는 쉽게 변경되면 안됨
critical_coordinates = (37.5665, 126.9780)

def process_location(coord):
    # coord가 튜플이면 함수 내에서 변경될 위험 없음
    latitude, longitude = coord
    # 처리 로직...
    return

def get_dimensions():
    width = 800
    height = 600
    depth = 3
    return width, height, depth # 암시적 튜플 반환

# 반환값을 튜플로 받기
dimensions = get_dimensions()
print(f"튜플: {dimensions}") # 튜플: (800, 600, 3)

# 언패킹으로 개별 값 받기
w, h, d = get_dimensions()
print(f"너비: {w}, 높이: {h}, 깊이: {d}") # 너비: 800, 높이: 600, 깊이: 3

# 1. 소괄호로 생성
coordinates = (10, 20)

# 2. tuple() 함수 사용
numbers = tuple([1, 2, 3, 4, 5]) # (1, 2, 3, 4, 5)

# 3. 소괄호 없이도 생성 가능
colors = "red", "green", "blue" # ('red', 'green', 'blue')

# 4. 단일 요소 튜플 (주의: 콤마 필요)
single_item = (42,) # (42,)
not_tuple = (42) # 42 (튜플이 아닌 일반 정수)

# 5. 빈 튜플 생성
empty_tuple = ()
empty_tuple2 = tuple()

# 6. 다양한 데이터 타입 혼합
mixed_tuple = (1, "안녕", 3.14, True, (1, 2, 3))

from collections import namedtuple

# 네임드 튜플 정의
Person = namedtuple('Person', ['name', 'age', 'city'])

# 생성 및 사용
person1 = Person('홍길동', 30, '서울')

# 인덱스로 접근
print(person1) # 홍길동

# 필드명으로 접근 (가독성 높음)
print(person1.name) # 홍길동
print(person1.age) # 30
print(person1.city) # 서울

```

```
def add_to_each(data, value) :
    # 원본 데이터를 변경하지 않고 새 튜플 반환
    return tuple(item + value for item in data)

nubers = (1, 2, 3)
new_numbers = add_to_each(numbers, 10)

print(numbers)    # (1, 2, 3) - 원본 그대로 유지
print(new_numbers) # (11, 12, 13) - 새 튜플 생성
```

## ▼ 2) 딕셔너리

- 키-값 쌍
  - 모든 항목은 키(key)와 값(value)의 쌍으로 구성
- 순서가 없음(Python 3.7 부터는 삽입 순서 유지)
  - 인덱스가 아닌 키로 값에 접근.
- 중복된 키를 가질 수 없음.
- 키는 불변 객체만 가능.
  - 문자열, 숫자, 튜플 등(리스트는 불가능)
- 값은 모든 타입 가능
  - 숫자, 문자열, 리스트 딕셔너리 등
- 변경 가능(Mutable)
  - 생성 후 내용을 수정, 추가, 삭제 가능
- 해시 테이블
  - 딕셔너리는 내부적으로 해시 테이블로 구현되어 빠른 검색 속도를 제공
  - 해시 테이블은 내부적으로 크기가 정해진 배열을 사용
  - 키-값 쌍을 저장할 때 키를 해시 함수에 통과시켜 배열의 특정 위치(인덱스)를 계산
  - 예를 들어, `apple` 이라는 키의 해시값이 42라면 배열의 42번 위치에 해당 키와 연관된 값을 저장
  - 나중에 `my_dict[apple]` 처럼 값을 조회할 때 딕셔너리는 다시 `apple` 이라는 키를 해시 함수에 통과시켜 42라는 동일한 인덱스를 얻고 배열의 42번 위치에서 값을 바로 찾아옴
  - 이것이 딕셔너리가  $O(1)$  시간 복잡도로 빠르게 검색할 수 있는 이유
  - 일반 리스트에서 특정 값을 찾으려면 처음부터 끝까지 모든 요소를 확인해야 하지만 ( $O(n)$  시간 복잡도), 딕셔너리는 해시 함수를 통해 키가 저장된 정확한 위치를 바로 계산하므로 데이터 크기와 상관없이 거의 일정한 시간에 검색이 가능

```
import time

# 데이터 준비
data_size = 1000000 # 백만 개의 데이터
search_key = f"key_{data_size-1}" # 마지막 요소 검색 (최악의 경우)

# 딕셔너리 생성
dict_data = {f"key_{i}": i for i in range(data_size)}

# 리스트 생성
list_data = [(f"key_{i}", i) for i in range(data_size)]

# 딕셔너리 검색 ( $O(1)$ ) 시간 측정
start_time = time.time()
result_dict = dict_data[search_key]
dict_time = time.time() - start_time

# 1. 중괄호와 콜론으로 생성
```

```

student = {"name": "홍길동", "age": 20, "grade": "A"}

# 2. dict() 함수 사용
student2 = dict(name="김철수", age=22, grade="B")

# 3. 키-값 튜플의 리스트로 생성
items = [("name", "이영희"), ("age", 25), ("grade", "C")]
student3 = dict(items)

# 4. 빈 딕셔너리 생성
empty_dict = {}
empty_dict2 = dict()

# 5. dict.fromkeys() 사용 (여러 키에 동일한 값 할당)
keys = ["name", "age", "grade"]
default_student = dict.fromkeys(keys, "미정") # {'name': '미정', 'age': '미정', 'grade': '미정'}

# 딕셔너리 접근과 수정
person = {
    "name": "홍길동",
    "age": 30,
    "city": "서울",
    "skills": ["Python", "Java", "SQL"]
}

# 값 접근
print(person["name"]) # '홍길동'
print(person.get("age")) # 30

# get() 메서드의 장점: 키가 없을 때 에러 대신 기본값 반환
print(person.get("email", "이메일 없음")) # '이메일 없음'

# 값 수정
person["age"] = 31

# 새 항목 추가
person["email"] = "hong@example.com"

# 항목 삭제
del person["city"] # 키로 삭제

# 다른 삭제 방법
skill = person.pop("skills") # 키를 지정하여 값 반환 후 삭제
print(f"제거된 skills: {skill}")

# 딕셔너리 메서드
student = {
    "name": "홍길동",
    "age": 20,
    "courses": ["수학", "영어", "과학"]
}

# keys(): 모든 키 반환
print(list(student.keys())) # ['name', 'age', 'courses']

# values(): 모든 값 반환
print(list(student.values())) # ['홍길동', 20, ['수학', '영어', '과학']]

# items(): 키-값 쌍 반환
print(list(student.items())) # [('name', '홍길동'), ('age', 20), ('courses', ['수학', '영어', '과학'])]

```

```

# update(): 딕셔너리 병합/갱신
new_info = {"grade": "A", "age": 21}
student.update(new_info)
print(student) # {'name': '홍길동', 'age': 21, 'courses': ['수학', '영어', '과학'], 'grade': 'A'}

# clear(): 모든 항목 삭제
temp_dict = {"temp": 1}
temp_dict.clear()
print(temp_dict) # {}

# copy(): 딕셔너리 얇은 복사
student_copy = student.copy()

# 딕셔너리 순회
student = {
    "name": "홍길동",
    "age": 20,
    "grade": "A",
    "courses": ["수학", "영어", "과학"]
}

# 키 순회
print("*** 키 목록 ***")
for key in student: # student_key() 와 동일
    print(key)

# 값 순회
print("*** 값 목록 ***")
for value in student.values():
    print(value)

# 키-값 쌍 순회
print("*** 키-값 쌍 ***")
for key, value in student.items():
    print(f"(key): {value}")

# 중첩 딕셔너리
users = {
    "user1": {
        "name": "홍길동",
        "age": 30,
        "email": "hong@example.com"
    },
    "user2": {
        "name": "김철수",
        "age": 25,
        "email": "kim@example.com"
    }
}

# 중첩 값 접근
print(users["user1"]["name"]) # '홍길동'

# 중첩 딕셔너리 순회
for user_id, user_info in users.items():
    print(f"\n사용자 ID: {user_id}")
    for key, value in user_info.items():
        print(f" {key}: {value}")

```

```
# 딕셔너리 컴프리헨션
# 1. 기본 형태
squares = {x: x**2 for x in range(1, 6)}
print(squares) # {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# 2. 조건부 딕셔너리 컴프리헨션
even_squares = {x: x**2 for x in range(1, 11) if x % 2 == 0}
print(even_squares) # {2: 4, 4: 16, 6: 36, 8: 64, 10: 100}

# 3. 값 변환 예제
fruits = ['apple', 'banana', 'cherry']
fruit_lengths = {fruit: len(fruit) for fruit in fruits}
print(fruit_lengths) # {'apple': 5, 'banana': 6, 'cherry': 6}
```

### ▼ 3) 집합

```
# 1. 중괄호로 생성
fruits = {"사과", "바나나", "체리"}

# 2. set() 함수 사용
numbers = set([1-3]) # {1, 2, 3} (중복 제거됨)

# 3. 문자열로 생성 (각 문자가 요소가 됨)
chars = set("hello") # {'h', 'e', 'l', 'o'} (중복 'l' 제거됨)

# 4. 빈 집합 생성 (주의: 빈 중괄호 {}는 딕셔너리)
empty_set = set() # 빈 집합
not_set = {} # 빈 딕셔너리, 집합 아님

# 5. 집합 컴프리헨션
squares = {x**2 for x in range(1, 6)} # {1, 4, 9, 16, 25}

# 기본 집합 생성
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# 합집합 (Union): A와 B의 모든 요소
print(A | B) # {1, 2, 3, 4, 5, 6, 7, 8}
print(A.union(B)) # 위와 동일

# 교집합 (Intersection): A와 B 모두에 있는 요소
print(A & B) # {4, 5}
print(A.intersection(B)) # 위와 동일

# 차집합 (Difference): A에는 있지만 B에 없는 요소
print(A - B) # {1, 2, 3}
print(A.difference(B)) # 위와 동일

# 대칭 차집합 (Symmetric Difference): A 또는 B에 있지만 양쪽에 모두 있지는 않은 요소
print(A ^ B) # {1, 2, 3, 6, 7, 8}
print(A.symmetric_difference(B)) # 위와 동일

# 부분집합 확인 (Subset)
C = {1, 2}
print(C.issubset(A)) # True (C는 A의 부분집합)
print(C <= A) # 위와 동일

# 진부분집합 확인 (Proper Subset)
print(C < A) # True (C는 A의 진부분집합)
```

```

# 상위집합 확인 (Superset)
print(A.issuperset(C)) # True (A는 C의 상위집합)
print(A >= C) # 위와 동일

# 진상위집합 확인 (Proper Superset)
print(A > C) # True (A는 C의 진상위집합)

# 서로소 확인 (Disjoint)
D = {10, 11, 12}
print(A.isdisjoint(D)) # True (A와 D는 공통 요소가 없음)

# 기본 집합
fruits = {"사과", "바나나", "체리"}

# 요소 추가
fruits.add("딸기") # {'사과', '바나나', '체리', '딸기'}

# 여러 요소 추가
fruits.update(["망고", "블루베리"]) # {'사과', '바나나', '체리', '딸기', '망고', '블루베리'}

# 요소 제거 (존재하지 않으면 오류 발생)
fruits.remove("바나나")

# 요소 제거 (존재하지 않아도 오류 없음)
fruits.discard("키위") # 없어도 오류 없음

# 임의의 요소 제거 및 반환
popped = fruits.pop() # 집합에서 임의의 요소 제거 및 반환
print(f"제거된 요소: {popped}")

# 모든 요소 제거
fruits.clear() # 빈 집합 {}

# 집합 순회
# 숫자 집합
numbers = {10, 20, 30, 40, 50}

# 기본 순회 (순서 보장되지 않음)
print("집합 요소:")
for num in numbers:
    print(num)

# 정렬된 순회가 필요하면 정렬 필요
print("\n정렬된 집합 요소:")
for num in sorted(numbers):
    print(num)

# 집합 내포 사용
squared = {x**2 for x in numbers}
print("\n제곱값 집합:", squared)

# 두 반 학생들의 취미 분석
class_a_hobbies = {"축구", "농구", "독서", "게임", "요리"}
class_b_hobbies = {"야구", "농구", "독서", "그림", "요리", "음악"}

# 양쪽 반에 모두 있는 취미 (교집합)
common_hobbies = class_a_hobbies & class_b_hobbies
print(f"공통 취미: {common_hobbies}")

```

```

# A반에만 있는 취미 (차집합)
only_a_hobbies = class_a_hobbies - class_b_hobbies
print(f"A반 전용 취미: {only_a_hobbies}")

# B반에만 있는 취미 (차집합)
only_b_hobbies = class_b_hobbies - class_a_hobbies
print(f"B반 전용 취미: {only_b_hobbies}")

# 모든 취미 목록 (합집합)
all_hobbies = class_a_hobbies | class_b_hobbies
print(f"모든 취미 목록: {all_hobbies}")

# 고유한 취미 (대칭 차집합)
unique_hobbies = class_a_hobbies ^ class_b_hobbies
print(f"한쪽 반에만 있는 취미: {unique_hobbies}")

# 취미의 종류 수
print(f"전체 취미 종류 수: {len(all_hobbies)}")

numbers = [1, 2, 3, 2, 1, 4, 5, 4, 3, 2]

# 방법 1: 집합 변환 후 다시 리스트로 변환
unique_numbers = list(set(numbers))
print(f"중복 제거된 숫자: {unique_numbers}") # 순서 보장 안 됨

# 방법 2: 순서 유지하며 중복 제거
def remove_duplicates(items):
    seen = set()
    result = []
    for item in items:
        if item not in seen:
            seen.add(item)
            result.append(item)
    return result

unique_ordered = remove_duplicates(numbers)
print(f"순서 유지하며 중복 제거: {unique_ordered}")

```

#### 실습

```

# 공통 관심사를 갖는 친구 응답
# 공통 관심사가 없는 친구 응답

test_data = {
    "Alice": ["음악", "영화", "독서"],
    "Bob": ["스포츠", "여행", "음악"],
    "Charlie": ["프로그래밍", "게임", "영화"],
    "David": ["요리", "여행", "사진"],
    "Eve": ["프로그래밍", "독서", "음악"],
    "Frank": ["스포츠", "게임", "요리"],
    "Grace": ["영화", "여행", "독서"]
}

def have_common_interest(myName, data):
    for name, favorite in data.items():
        if (name != myName):
            common_interest = set(favorite) & set(test_data[myName])
            if (len(common_interest) >= 1):
                print(f"{myName}과 {name}은 공통 관심사를 갖습니다.")

```

```
def no_common_interest(myName, data):  
    for name, favorite in data.items():  
        if (set(favorite).isdisjoint(set(test_data[myName]))):  
            print(f"{myName}과 {name}은 공통 관심사가 없습니다.")
```

---