컬렉션 타입

튜플
지정된 데이터의 묶음
인덱스로 값 접근

```
var person : (String, Int, Double) = ("Kim", 23, 184.2)

print(person.0)//Kim
print(person.1)//23
print(person.2)//184.2
```

각 요소가 어떤 의미인지 유추하기 힘듦 -> 타입 별칭으로 해결
같은 모양의 튜플을 사용하고 싶을때마다 긴 튜플 타입을 써야함 -> 타입 별칭으로 해결

```
typealias person = (name : String, age : Int, height : Double)

let Kim : person = ("Kim", 23, 184.3)

print(Kim.name)
print(Kim.age)
print(Kim.height)
```

배열
같은 타입의 데이터를 이렬로 나열한 후 순서대로 저장하는 형태
자바의 배열과는 다르게 참조타입인 아닌 값 타입이다.

```
1  var names : Array<String> = ["Kim","Min","Lee"]
2  var etcName : [String] = ["Kim","Min","Lee"]
3
4  var emptyArray : [Int] = [Int]() // Int타입의 빈 배열 생성
5  var emptyArr : [Int] = Array<Int>()
6
7  var typeAnnotationArr : [Int] = []
8
```

배열의 값 접근은 인덱스를 통해서 접근할 수 있다.

배열의 다양한 메소드

firstIndex(of:)
앞에서부터 조회하며 중복된 요소가 있다면 제일 먼저 발견된 요소의 인덱스를 반환

lastIndex(of:)

firstIndex(of:)의 조회 순서는 다르지만 기능은 똑같다.

append()
배열의 맨 뒤에 요소를 추가해주는 메소드

insert(_:at:)
해당값을 원하는 인덱스에 삽입

remove(at:)
해당하는 인덱스의 값을 삭제

딕셔너리
키와 값의 쌍으로 이루어짐

키
같은 이름을 중복해서 사용할 수 없음

var person : Dictionary<String, Int> = Dictionary<String, Int>() // 빈 딕셔너리 선언
Var person : [String:Int] = [String:Int]() // 빈 딕셔너리 선언
Var person : [String:Int] = [:] // 빈 딕셔너리 선언

Var person : [String:Int] = [“Kim” : 23 , “Lee” : 25] // 초깃값을 넣어 생성 가능

세트
같은 타입의 데이터를 순서 없이 하나의 묶음으로 저장하는 컬렉션 타입
값은 모두 유일한 값

Var names : Set<String> = Set<String>() // 빈 세트 생성
Var names : Set<String> = [] // 빈 세트 생성
Var names : Set<String> = [“Kim”, “Lee”] // 타입 추론 사용시 Array로 타입이 지정되므로 타입 선언 중요

컬렉션의 다양한 메소드
https://www.boostcourse.org/mo122/lecture/11202?isDesc=false