Collection Type

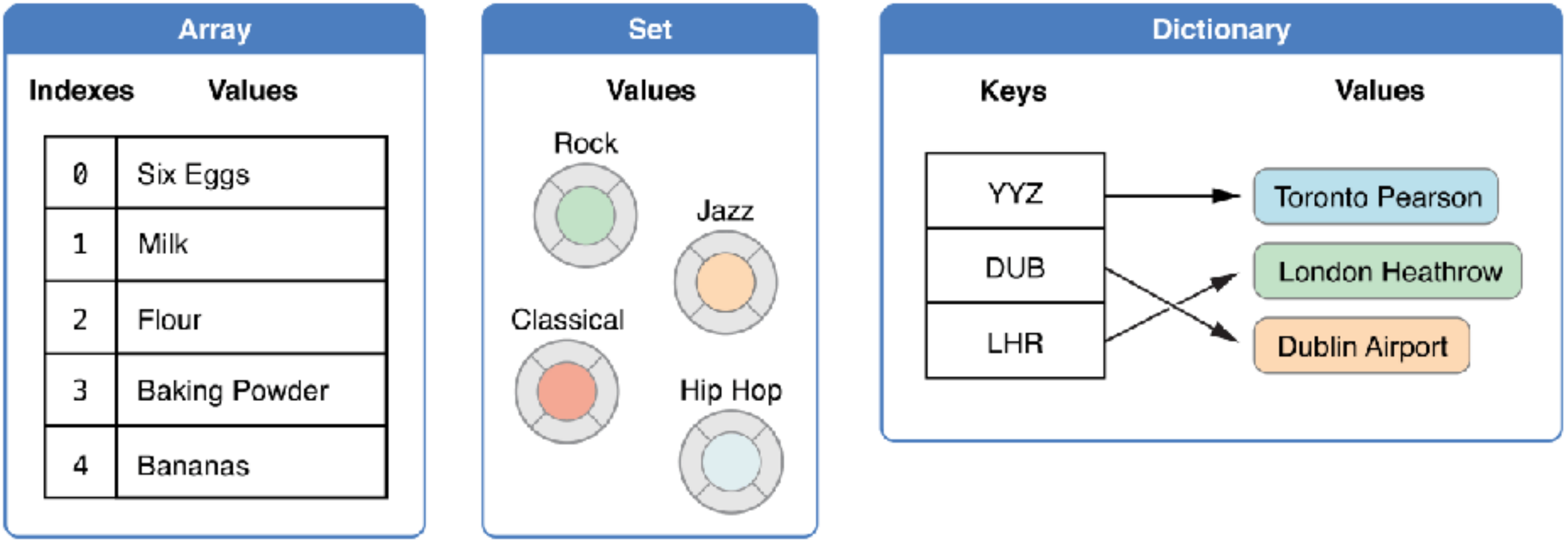
collection : 자료구조 -> 정리 방식은 어떤지

- Array : 5
- Dictionary : 4
- Set : 1 -> 활용도가 좋으나, set이 없다고 불가능하지 않다.

Array를 많이 사용 (Program에서 사용하는 알고리즘에서 많이 사용됨)
Server와 연결시 dictionary를 많이 사용

Collection Type

- Swift는 값의 모음을 저장하기위한 배열, 집합 및 사전이라는 세 가지 기본 형식을 제공 합니다. 배열은 정렬 된 값 모음입니다. 집합은 고유 한 값의 정렬되지 않은 모음입니다. 사전은 키 - 값 연관의 정렬되지 않은 모음입니다.



Mutability of Collections

어떤 값을 변경할 경우 : var
let을 사용할 경우 변경 불가능

```
var p: Person = Person()

let list: Array<Int> = Array()
list.***
==> let 으로 선언하면 바꿀 수 없음
```

- 변수(var) 에 할당하면 Collection를 변경 가능하다.
  Collection에 추가, 제거, 수정할수 있다.
- 하지만 상수(let)에 할당하면 Collection를 변경 불가능 하다.

Array

순차적
인덱스를 갖고 있다
꼼수 : Any를 사용해서 지정하고, 배열경우는 casting을 하고 사용해야 함 -> 추천하지 않음, 비효율적

- 배열(영어: array)은 번호(인덱스)와 번호에 대응하는 데이터들로 이루어진 자료 구조를 나타낸다. 일반적으로 **배열에는 같은 종류의 데이터들이 순차적으로 저장**되어, 값의 번호가 곧 배열의 시작점으로부터 값이 저장되어 있는 상대적인 위치가 된다.

Array 문법

- 기본 표현은 Array<Element>로 Array Type을 나타낸다.

⬆type 작성 "<>" -> generic

- 여기에서 Element는 배열에 저장할수 있는 타입이다.
- 또 다른 축약 문법으로 [Element] 로 표현할 수 있다.

```
var someInts:[Int] = [Int]()
var someInts:Array<Int> = Array<Int>()
```

배열 리터럴

값을 바로 대입

- 배열 리터럴 문법은 대괄호 [ ] 를 사용한다.

```
[ 값 1 , 값 2 , 값 3 ]
```

```
var someInts:[Int] = [1,2,3,4]
someInts = []
```

배열 Element 가져오기

- index를 통해 배열의 값을 가져올수 있다.
- index는 0부터 시작된다.

```
var someInts:[Int] = [1,2,3,4]
print("\(someInts[0])")
print("\(someInts[3])")
```

인덱스

배열 추가 기능

- 배열의 Element 갯수 count
- 빈 배열 확인 isEmpty
- Element 추가 append
- Element 삽입 insert
- Element 삭제 remove

Quick Help

- command + shift + O

배열 예제

```
func arrayTest()
{
    var list:[String] = ["my", "name", "is", "a", "joo",
"youngmin" ]
    list.append("입니다.")
    print("list배열의 총 갯수는", list.count)
    print(list[3])
    list.remove(at: 3)
    list.insert("my introduce it my self.", at: 0)

    for text in list
    {
        print(text)
    }
}
```

Set

- Set은 같은 타입의 데이터가 순서없이 모여있는 자료구조, 각 항목의 순서가 중요치 않거나 한번만 표시해야하는 경우 배열대신 사용된다.

Set 문법

- 기본 표현은 Set<Element>로 Set Type을 나타낸다.
- 여기에서 Element는 배열에 저장할수 있는 타입이다.
- Set은 Array와 다르게 축약 문법이 없다.

```
var someInts:Set<Int> = Set<Int>()
```

Set 리터럴 사용

- Set Type으로 설정된 변수는 배열 리터럴을 이용해서 값을 설정 할수 있다.

```
[ 값 1 , 값 2 , 값 3 ]
```

type 을 지정하지 않으면 array임

```
var someInts:Set<Int> = [1,2,3,4]
someInts = []
var someStrings:Set = [“joo”,”young”]
```

Set Element 가져오기

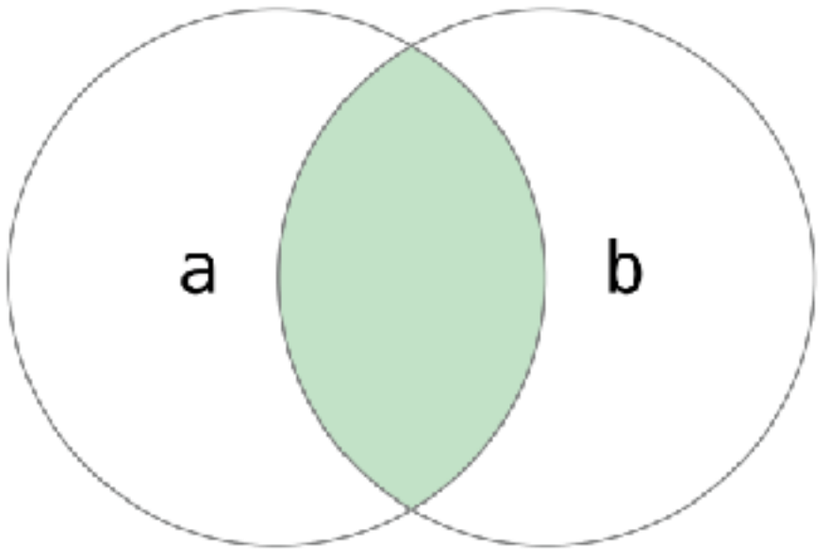
활용도 : 모든 data 들이 모두 다를 때 사용

- Set은 순서가 정해져 있지 않기 때문에 for-in 구문을 통해서 데이터를 가져와야 한다.

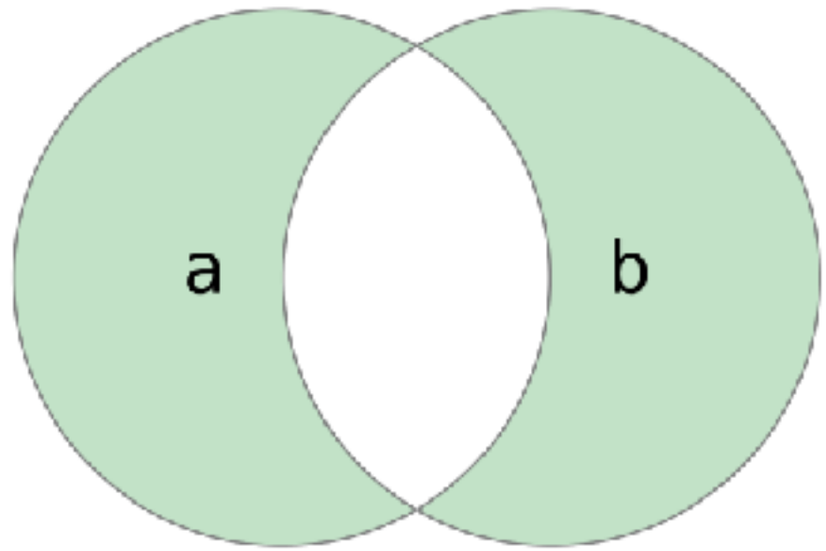
- 순서는 정해져 있지 않지만 정렬을 통해 데이터를 원하는 순서대로 가져올수 있다. Set을 정렬하면 return값이 array임

Set 기능

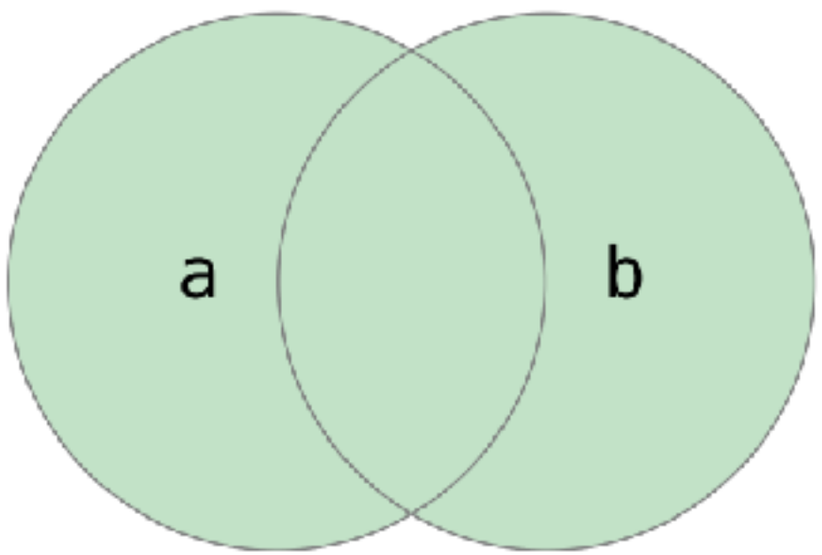
a.intersection(b)



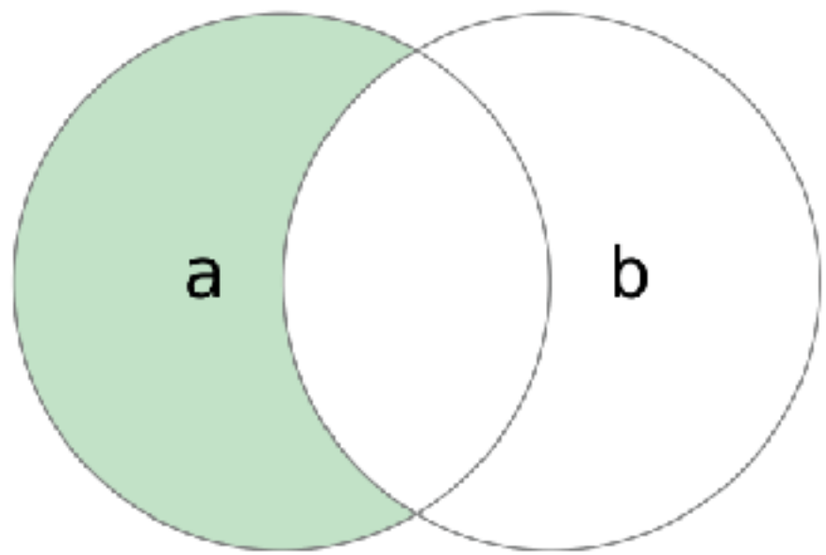
a.symmetricDifference(b)



a.union(b)



a.subtracting(b)



Set 예제

```
func setTest()
{
    let oddDigits : Set = [ 1, 3, 5, 7, 9 ]
    let evenDigits : Set = [2, 4, 6, 8]
    let primeDigits : Set = [2, 3, 5, 7]

    print("test=======")
    //교집합
    let intersectList = oddDigits.intersection(evenDigits)
    print(intersectList)
    //교집합의 여집합
    let differenceList = oddDigits.symmetricDifference(primeDigits)
    print(differenceList)
    //합집합 & 정렬
    let unionList = oddDigits.union(evenDigits).sorted()
    print(unionList)
    //차집합 & 정렬
    let subtractList = oddDigits.subtracting(primeDigits).sorted()
    print(subtractList)
}
```

Dictionary

hash table
hash 라는 알고리즘을 사용
중간역할을 함

key -> 검색 색인

- Dictionary는 순서가 정해져 있지 않은 데이터에 키값을 통해 구분할수 있는 자료구조. 항목의 순서가 중요치 않고 key값을 통해서 데이터를 접근할때 사용합니다.

Dictionary 문법

- 기본 표현은 Dictionary<key, value>로 Dictionary Type을 나타낸다.

- Key값 은 Dictionary에서 value를 가져오는데 사용되는 값이다.

- 또 다른 축약 문법으로 [key:value] 로 표현할 수 있다.

key는 일반적으로 대부분 String
value는 Any를 많이 사용함

```
var someInts:[String:Int] = [String:Int]()
var someInts:Dictionary<String,Int> = [:]
```

딕셔너리 리터럴

- 딕셔너리의 리터럴 문법은 [:] 를 사용한다.

```
[ 키 1 : 값 1 , 키 2 : 값 2 , 키 3 : 값 3 ]
```

```
var airports: [String:String] = ["ICH": "인천공항", "CJU": "제주공항"]
```

딕셔너리 Value 가져오기

- key값을 통해 Value값을 가져올수 있다.

```
var airports: [String:String] = ["ICH": "인천공항", "CJU": "제주공항"]
print("\(airports["ICH"])")
print("\(airports["CJU"])")
```

Dictionary 기능

- 딕셔너리의 Element 갯수
- 빈 배열 확인
- Element 추가
- Element 삽입
- Element 삭제

Down casting “as”

- 일반적으로 key의 타입은 String으로 지정합니다. 하지만 value에 타입은 다양하게 지정해야 되는 경우가 많은데요, 이 경우 어쩔수 없이 Any Type을 사용합니다.

- Any에서 내가 원하는 타입으로 캐스팅하는 경우를 다운 캐스팅이라고 하고, as 키워드를 사용해서 캐스팅 합니다.

Down casting “as”

```
var person:[String:Any] = ["name":"joo", "age":20, "isSingle":true]
let name1 = person["name"]  //type은 Any
let name2 = person["name"] as! String //type은 String
```

- as로 다운 캐스팅을 할때 캐스팅이 실패할 확률이 있기때문에 옵셔널로 지정이 됩니다. 때문에 ? or !를 붙여야 하지만, 우린 아직 옵셔널 수업 전 이기에 패스!!

Dictionary 예제

```
func dicTest()
{
    //기본 딕셔너리
    var dic:[String:Any] = ["name":"joo", "age":20, "job":"Developer",
"isSingle":true]

    //딕셔너리 추가
    dic.updateValue("address", forKey: "Seoul")
    //딕셔너리 수정
    dic.updateValue("name", forKey: "winman")
    //삭제
    dic.removeValue(forKey: "isSingle")

    //값 불러오기
    let introduce: String = "제 이름은" + (dic["name"] as! String) + "입니다."

    let doubleAge = (dic["age"] as! Int) * 2

}
```