Collection Type

- 컬렉션은 구조체로 구현된 값 형식이기 때문에 값이 사용될 때마다 복사되어야 하는데, 반드시 복사가 필요한 경우에만 실제 복사를 수행한다. 이를 copy on write 최적화라고 한다.
- 컬렉션이 변경되지 않는 한 항상 동일한 데이터를 사용하다가, 특정 시점에 컬렉션을 변경하면 그때 복사본을 생성하고 변경사항을 저장한다.

Array

생성

```
var someInt = [Int]()
```

기본 값으로 생성

```
var threeDouble = Array(repeating: 0.0, count: 3)
// [0.0, 0.0, 0.0]
```

다른 배열을 추가한 배열의 생성

- + 연산자를 이용해 배열을 합칠 수 있습니다.

```
var anotherThreeDoubles = Array(repeating: 2.5, count: 3)

var sixDoubles = threeDoubles + anotherThreeDoubles
// sixDoubles : [0.0, 0.0, 0.0, 2.5, 2.5, 2.5]
```

리터럴을 이용한 배열의 생성

```
var shoppingList = ["Eggs", "Milk"]
```

배열의 접근 및 변환

- array.count : 배열의 원소 개수 확인.
- array.isEmpty : 배열이 비었는지 확인.
- array.append(원소) : 배열에 원소 추가.

```
shoppingList += ["Baking Powder"]
// shoppingList.count = 4
shoppingList += [Chocolate Spread", "Cheese", "Butter"]
// shoppingList.count = 7
```

- array[0], array[4..6] : 배열의 특정 위치의 원소에 접근.

```
shoppingList[4..6] = ["Bananas", "Apples"]
// 4, 5, 6번째 인덱스 아이템을 Banana, Apples로 변환
// 즉, 아이템 3개가 2개로 줄었다.
```

- array.insert(원소, at: 인덱스) : 특정 위치에 원소 추가

```
shoppingList.insert("Maple Syrup", at:0)
```

- array.remove(at: 인덱스) : 특정 위치에 원소 삭제. 삭제값을 반환함.

```
let mapleSyrup = shoppingList.remove(at: 0)
let apples = shoppingList.removeLast()
```

배열의 순회

- 배열의 값과 인덱스가 필요할 때는 enumerated() 메소드를 사용.

```
for (index, value) in shoppingList.enumerated() {
    print("Item \(index + 1): \(value)")
}
// Item 1: Six eggs
// Item 2: Milk
// Item 3: Flour
// Item 4: Baking Powder
// Item 5: Bananas
```

Set

- 수학에서의 '집합' 개념. 집합의 특징 이용할 수 있음.
- 중복을 허용하지 않음.
- 정렬을 하면 배열이 반환.
- Set 형태로 저장되기 위해서는 반드시 타입이 hashable이어야만 합니다. Swift에서 String, Int, Double, Bool 같은 기본 타입은 기본적으로 hashable입니다. Swift에서 Set 타입은 Set으로 선언합니다.
- 정렬 순서보다 검색 속도가 중요할 때 사용.
  ⇒ 검색 속도가 빠른 이유는? Hashing(요소의 유효성을 판단하고 빠른 검색을 위해 사용하는 값) 때문.
- Set를 타입으로 명시해주지 않으면 배열로 지정됨.
- 배열과 Set 차이
  1. 배열은 순서가 있지만, Set은 순서가 없다.
  2. 배열은 중복 가능하지만, Set은 중복이 불가능하다.
  3. Set은 배열처럼 인덱스를 사용하지 않음.

생성

```
var letters = Set<Character>()
```

배열 리터럴을 이용한 Set 생성

```
var favoriteGenres: Set = ["Rock", "Classical", "Hip hop"]
```

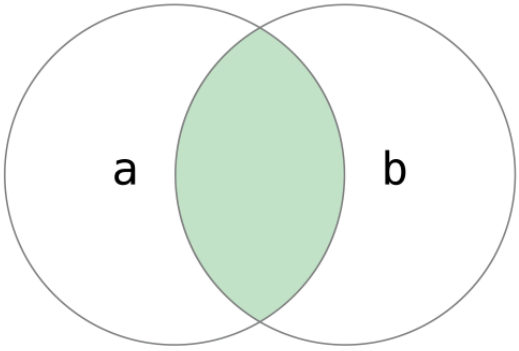
Set의 접근과 변경

- set.count : set의 원소 개수 확인.
- set.isEmpty : set가 비었는지 확인.
- set.insert(원소) : set에 원소 추가.
- set.contain(원소) : set가 원소를 갖고 있는지 확인.

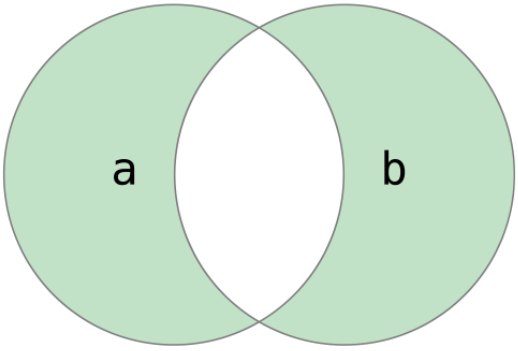
- set.remove(원소) : set에 원소 삭제.

Set의 명령

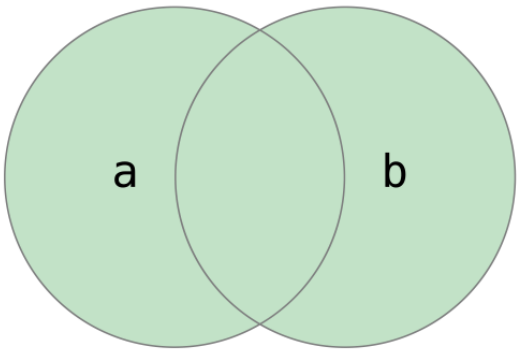
a.intersection(b)



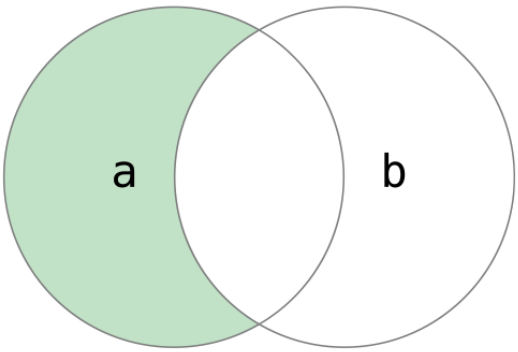
a.symmetricDifference(b)



a.union(b)



a.subtracting(b)



- a.intersection(b) : 교집합.
- a.symmetricDifference(b) : 합집합 - 교집합,
- a.union(b) : 합집합
- a.subtracting(b) : 차집합.

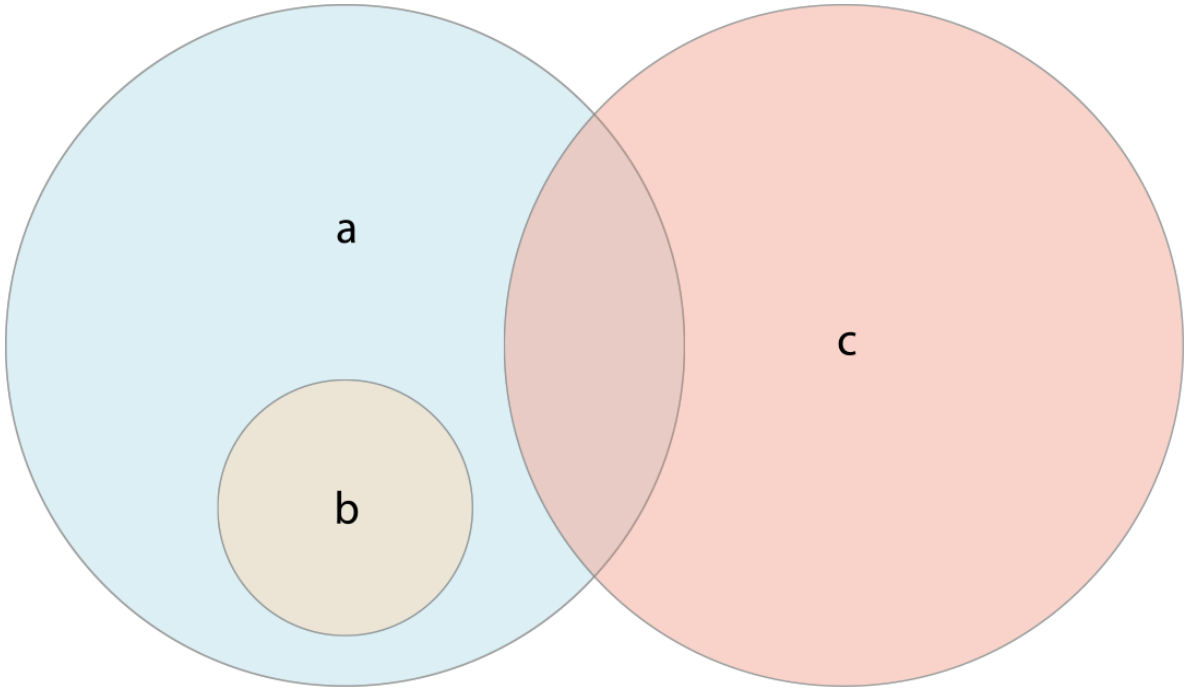
```
let oddDigits: Set = [1, 3, 5, 7, 9]
let evenDigits: Set = [0, 2, 4, 6, 8]
let singleDigitPrimeNumbers: Set = [2, 3, 5, 7]

oddDigits.union(evenDigits).sorted()
// [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
oddDigits.intersection(evenDigits).sorted()
// []
oddDigits.subtracting(singleDigitPrimeNumbers).sorted()
// [1, 9]
```

```
oddDigits.symmetricDifference(singleDigitPrimeNumbers).sorted()
// [1, 2, 9]
```

Set의 맴버십과 동등 비교

- Set a는 Set b의 SuperSet.
- Set b는 Set a의 SubSet.
- Set b와 Set a는 disjoint.



- == : 두 Set의 요소가 전부 같은지.
- b.isSubset(of: a) : b는 a의 subset인지.
- a.isSuperset(of: b) : a는 b의 superset인지.
- b.isDisjoint(of: c) : b와 c는 disjoint인지. 둘 사이의 공통값이 없을 경우 true를 반환.

```
let houseAnimals: Set = ["🐶", "🐱"]
let farmAnimals: Set = ["🐮", "🐔", "🐑", "🐶", "🐱"]
let cityAnimals: Set = ["🐦", "🐭"]

houseAnimals.isSubset(of: farmAnimals)
// 참
farmAnimals.isSuperset(of: houseAnimals)
// 참
```

```
farmAnimals.isDisjoint(with: cityAnimals)
// 참
```

Dictionary

- 'key'로는 검색할 수 있지만, '값'으로는 검색할 수 없음 → nil 반환.
- key가 존재하지 않으면 새로운 요소를 추가(insert), key가 존재하면 기존값을 교체(update)
  ⇒ update + insert = upsert라고 함.

빈 Dictionary의 생성

```
var namesOfIntegers = [Int: String]()
```

리터럴을 이용한 Dictionary의 생성

```
var airports: [String: String] = = ["YYZ": "Toronto Pearson", "DUB": "Dublin"]
```

Dictionary의 접근과 변경

- dictionary.count : 딕셔너리의 key와 값 개수 확인.
- dictionary.isEmpty : 딕셔너리가 비었는지 확인.
- 값 할당 : key와 값을 함께.

```
airports["LHR"] = "London"
```

- 혹시 key에 없는 값을 검색할 때 nil을 반환하지 않게 default 값 지정해줄 수 있음.

```
let a = words["E", default: "Empty"]
//어떤 경우에도 nil이 아닌 문자열을 리턴
```