반복문

프로그램에서 반복과 선택이 있으면 거의 다 할 수 있음

반복문

- 반복적으로 실행되는 코드를 만드는 구문
- 대표적인 반복문으로 while문과 for-in문이 있다.

while문

조건에 따라 반복하는 구문
종료조건을 만족하지 않으면 무조건 돌아감

조건이 참일경우 구문 반복 실행

```
while 조건
{
    //구문  실행

}
```

while문 사용예제

반복문을 할 경우 실행 순서를 잘 따져야 한다.

실행 순서를 잘 따져야 함!

```
var index = 0;

while index < 10
{
    print("현재 횟수는 \(index)입니다.")
    index += 1;
}
```

*구문안에 조건을 변화시키는 내용이 없으면 무한 반복이 될수 있다.

예제 : 구구단 구현

단을 받아서 내용을 출력해주는 함수

```
func timesTable(_ times:Int)
{
    print("\(times)단")
    var count: Int = 1
    while count < 10
    {
        print("\(times) * \(count) = \(times * count)")
        count += 1
    }
}
```

for-in 문

for -> for - each -> for - in

- 배열의 항목, 숫자의 범위 또는 문자열의 문자와 같은 시퀀스를 반복 하려면 for- in 반복문를 사용합니다.

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

for number in numbers
{
    print("\(number)")

}
```

숫자 범위의 연산

```
for index in 1...5 {
    print("\(index) times 5 is \(index * 5)")
}
```

범위 연산자	예제	설명
a...b	3...10	a~b까지의 숫자
a..<b	0..<10	a~b까지 숫자중 b는 포함안함

와일드 카드 (_)

- 문법상 필요하지만 실제 동작에서 사용하지 않는 변수 이름에 밑줄을 그어 표현한다.

```
let base = 3
let power = 10
var answer = 1

for _ in 1...power {
    answer *= base
}
```

예제 : 구구단 - for문으로 변경

단을 받아서 출력해주는 계산후 출력해주는 함수

```
func timesTable(_ times:Int)
{
    print("\(times)단")
    for count in 1...9
    {
        print("\(times) * \(count) = \(times * count)")
    }
}
```

문제 : 팩토리얼

정수 하나를 받아서 그 수의 팩토리얼을 구하시오

3 = 3 * 2 * 1

5 = 5 * 4 * 3 * 2 * 1

문제

1. 삼각수 구하기 : 삼각수란 1부터 모든 수의 합이다. ex)삼각수 10 = 55

   - 메소드 실행:triangular(num:10) // triangular(num:-1)
   - 결과 : 55 // 0

2. 각 자리수 더하는 메소드

   - 메소드 실행 :addAll(num:123) // addAll(num:5792)
   - 결과 : 6 // 23

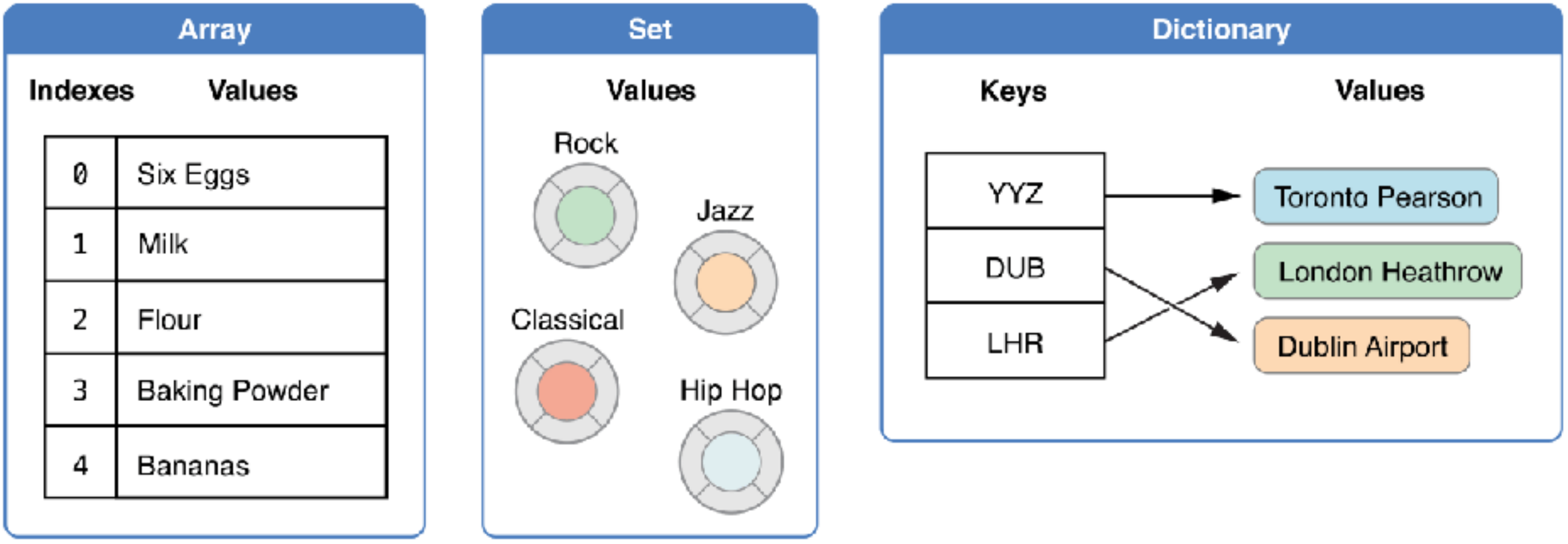
3. 숫자 리버스 함수

   - 메소드 실행 : revers(num:123) // revers(num:341)
   - 결과 : 321 // 143

Collection Type

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

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

Array

- 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.
- 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

- 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])")
```

배열 추가 기능

- 현재 배열의 element count
- 빈 배열 확인
- element 추가
- element 삽입
- element 삭제

Quick Help

- command + shift + O

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 ]
```

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

Set Element 가져오기

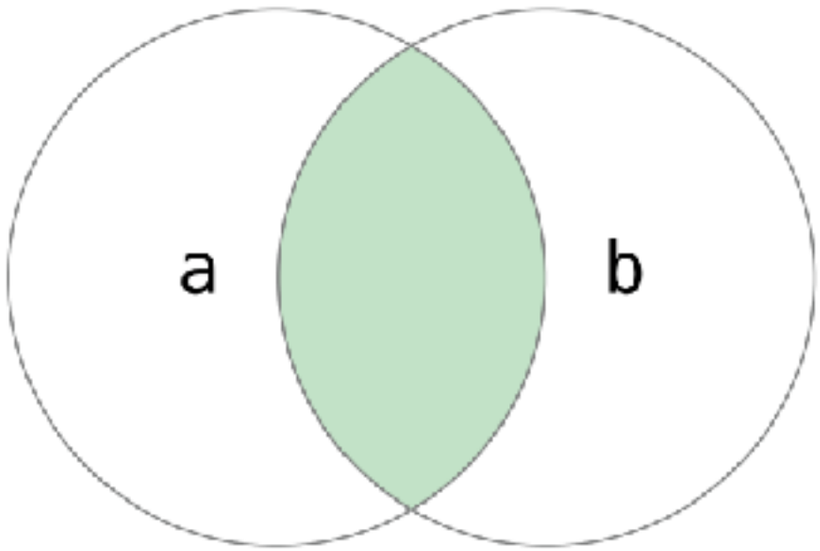
- Set은 순서가 정해져 있지 않기 때문에 for-in 구문을 통해서 데이터를 가져와야 한다.
- 순서는 정해져 있지 않지만 정렬을 통해 데이터를 원하는 순서대로 가져올수 있다.

Set 기능

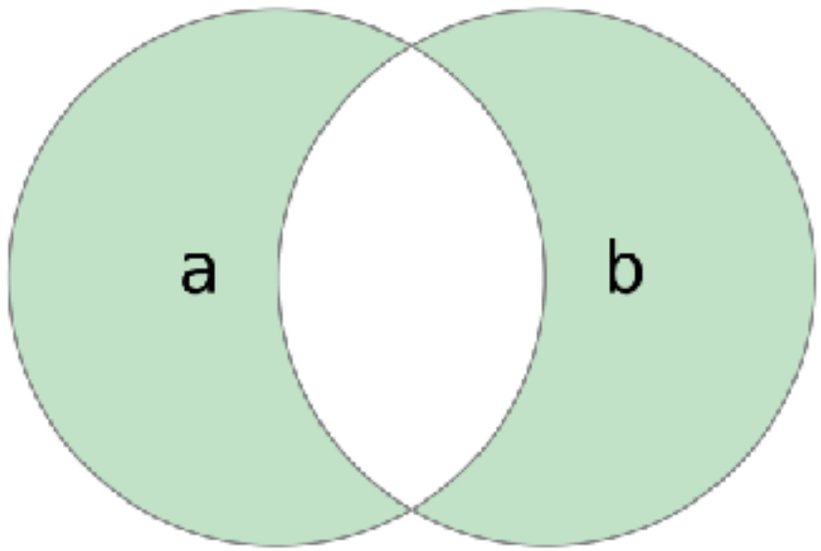
- Set의 기능에 대해 알아 봅시다.

Set 집합 연산

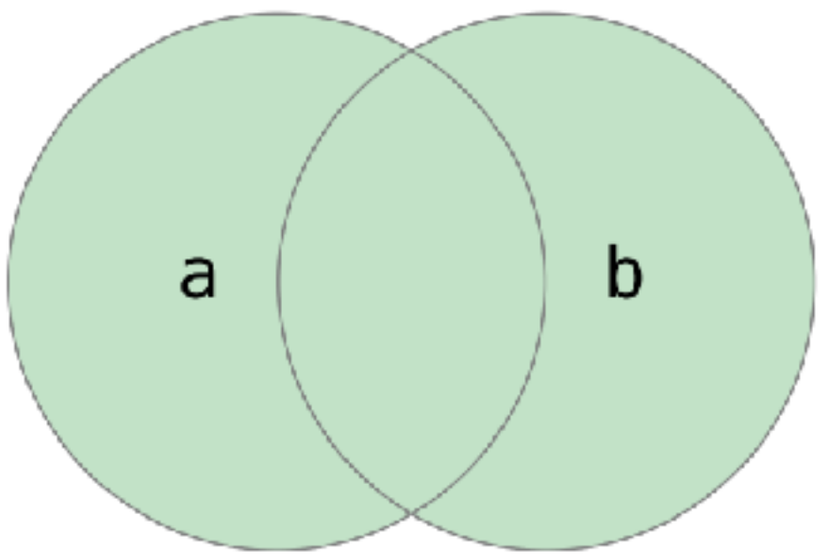
a.intersection(b)



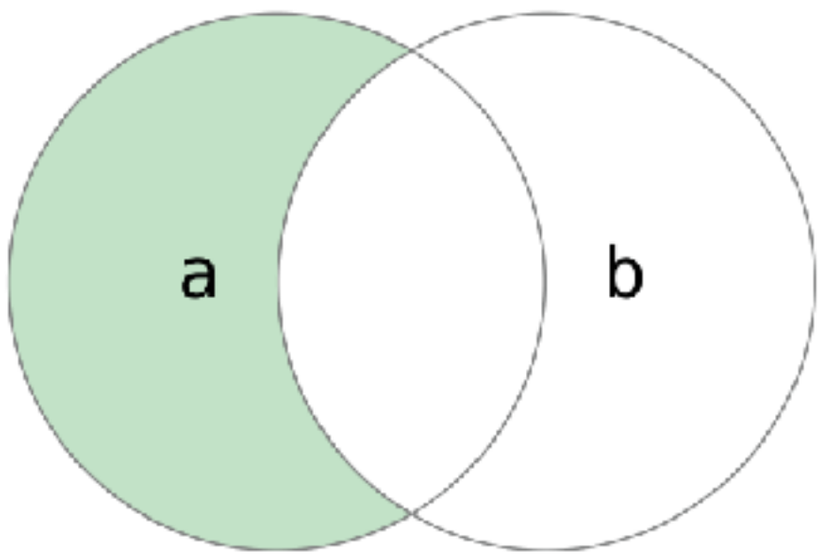
a.symmetricDifference(b)



a.union(b)



a.subtracting(b)



Set 집합 연산

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

oddDigits.intersection(evenDigits)

oddDigits.symmetricDifference(primeDigits)

oddDigits.union(evenDigits).sort()

oddDigits.subtract(primeDigits).sort()
```

Dictionary

- 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.

```
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

- 이젠 말 안해도 아시죠?