열거형(enum)

- https://docs.swift.org/swift-book/documentation/the-swift-programming-language/enumerations/
- 관련있는 데이터들이 멤버로 구성되어 있는 자료형 객체
  - 원치 않는 값이 잘못 입력되는 것 방지
  - 입력 받을 값이 한정되어 있을 때
  - 특정 값 중 하나만 선택하게 할 때
- 색깔
  - 빨강, 녹색, 파랑
- 성별
  - 남, 여

열거형 정의

- enum 열거형명{
  열거형 정의
  }

```
enum Planet {
   case Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune
}   //하나의 case문에 멤버들 나열하는 것도 가능
```

```
enum Compass {
    case North
    case South
    case East
    case West
  }
print(Compass.North) //North
var x : Compass  //Compass형 인스턴스 x
x = Compass.West
print(x, type(of:x)) // West Compass
x = .East
print(x) //East
```

- 문맥에서 타입의 추론이 가능한 시점 (등호 좌변의 변수 타입이 확정적일 때)에는 열거형명 생략 가능

열거형 멤버별 기능 정의

```
enum Compass {
    case North
    case South
    case East
    case West
}
var direction : Compass
direction = .South
switch direction { //switch의 비교값이 열거형 Compass
case .North:    //direction이 .North이면 "북" 출력
    print("북")
case .South:
    print("남")
case .East:
    print("동")
case .West:
    print("서")  //모든 열거형 case를 포함하면 default 없어도 됨
}
```

열거형 멤버에는 메서드도 가능

```
enum Week: String {
    case Mon,Tue,Wed,Thur,Fri,Sat,Sun
    func printWeek() {  //메서드도 가능
        switch self {
        case .Mon, .Tue, .Wed, .Thur, .Fri:
            print("주중")
        case .Sat, .Sun:
            print("주말")
        }
    }
}
Week.Sun.printWeek()  //레포트
```

열거형의 rawValue

```
enum Color: Int {
    case red
    case green = 2
    case blue
}
print(Color.red)  //red
print(Color.blue)
print(Color.red.rawValue) //0
print(Color.blue.rawValue)
```

String형 값을 갖는 열거형의 rawValue

```
enum Week: String {
    case Monday = "월"
    case Tuesday = "화"
    case Wednesday = "수"
    case Thursday = "목"
    case Friday = "금"
    case Saturday //값이 지정되지 않으면 case 이름이 할당됨
    case Sunday   // = "Sunday"
}
print(Week.Monday) //Monday
print(Week.Monday.rawValue) //월
print(Week.Sunday)
print(Week.Sunday.rawValue)
```

연관 값(associated value)을 갖는 Enum

```
enum Date {
    case intDate(Int,Int,Int)  //(int,Int,Int)형 연관값을 갖는 intDate
    case stringDate(String)    //String형 연관값을 값는 stringDate
}
var todayDate = Date.intDate(2023,4,30)
todayDate = Date.stringDate("2023년 5월 20일") //주석처리하면?
switch todayDate {
  case .intDate(let year, let month, let day):
     print("\(year)년 \(month)월 \(day)일")
  case .stringDate(let date):
     print(date)
}
```

Generic

<>

Swift Generic

- https://en.wikipedia.org/wiki/Generic_programming
- https://docs.swift.org/swift-book/documentation/the-swift-programming-language/generics/
- Generics are one of the most powerful features of Swift, and much of the Swift standard library is built with generic code.
- var a : [Int] = [1,2,3,4]
- var b : Array<Int> = [1,2,3,4]
  - https://developer.apple.com/documentation/swift/array
  - Array
  - Generic Structure
  - @frozen struct Array<Element>
- https://developer.apple.com/documentation/uikit/uiresponder/1621142-touchesbegan
  - func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?)
- func decode<T>(_ type: T.Type, from data: Data) throws -> T where T : Decodable

오류가 발생하지 않도록 함수를 수정하시오.

```
func myPrint(a: Int, b: Int) {
    print(b,a)
}
myPrint(a:1,b:2)
myPrint(a:2.5,b:3.5)
//error: cannot convert value of type 'Double' to expected argument type 'Int'
```

기능은 같고 매개변수형만 다른 함수는 제네릭 함수로 구현

```
func myPrint<T>(a: T, b: T) {
    print(b,a)
}
myPrint(a:1,b:2)
myPrint(a:2.5,b:3.5)
//myPrint(a:"Hi",b:"Hello")  //가능?
```

기능은 같고 매개변수형만 다른 함수

```
func swapInt(_ a: inout Int, _ b: inout Int) {
    let temp = a
    a = b
    b = temp
}
var x = 10
var y = 20
swapInt(&x, &y)
print(x,y)
func swapDouble(_ a: inout Double, _ b: inout Double) {
    let temp = a
    a = b
    b = temp
}
var xd = 10.3
var yd = 20.7
swapDouble(&xd, &yd)
print(xd,yd)
```

```
func swapString(_ a: inout String, _ b: inout String)
{
    let temp = a
    a = b
    b = temp
}
var xs = "Hi"
var ys = "Hello"
swapString(&xs, &ys)
print(xs,ys)
```

기능은 같고 매개변수형만 다른 함수

```
func swapInt(_ a: inout Int, _ b: inout Int) {
    let temp = a
    a = b
    b = temp
}
func swapDouble(_ a: inout Double, _ b: inout Double) {
    let temp = a
    a = b
    b = temp
}
func swapString(_ a: inout String, _ b: inout String) {
    let temp = a
    a = b
    b = temp
}
```

```
func swapAny<T>(_ a: inout T, _ b: inout T) {
    let temp = a
    a = b
    b = temp
} //T는 type이름
```

주의 : swift에는 이미 swap함수가 있음으로 다른 이름 사용해야 함

```
public func swap<T>(_ a : inout, _ b : inout T)
```

기능은 같고 매개변수형만 다른 함수 : generic 함수

```
func swapAny<T>(_ a: inout T, _ b: inout T) {
    let temp = a
    a = b
    b = temp
}
var x = 10
var y = 20
swapAny(&x, &y)
print(x,y)
var xd = 10.3
var yd = 20.7
swapAny(&xd, &yd)
print(xd,yd)
var xs = "Hi"
var ys = "Hello"
swapAny(&xs, &ys)
print(xs,ys)
```

Int형 스택 구조체

```
struct IntStack {
    var items = [Int]()
    mutating func push(_ item: Int) {
        items.append(item)
    }
    mutating func pop() -> Int {
        return items.removeLast()
    }
}

//구조체는 value타입이라 메서드 안에서
//프로퍼티 값 변경불가
//mutating 키워드를 쓰면 프로퍼티 값 변경 가능
```

```
var stackOfInt = IntStack()
print(stackOfInt.items) //[]
stackOfInt.push(1)
print(stackOfInt.items) //[1]
stackOfInt.push(2)
print(stackOfInt.items) //[1,2]
stackOfInt.push(3)
print(stackOfInt.items) //[1,2,3]
print(stackOfInt.pop()) //3
print(stackOfInt.items) //[1,2]
print(stackOfInt.pop()) //2
print(stackOfInt.items) //[1]
print(stackOfInt.pop()) //1
print(stackOfInt.items) //[]
```

일반 구조체 vs. generic 구조체

```
struct IntStack {
    var items = [Int]()
    mutating func push(_ item: Int) {
        items.append(item)
    }
    mutating func pop() -> Int {
        return items.removeLast()
    }
}

//구조체는 value타입이라 메서드 안에서
//프로퍼티 값 변경불가
//mutating 키워드를 쓰면 가능
```

```
struct Stack <T> {
    var items = [T]()
    mutating func push(_ item: T) {
        items.append(item)
    }
    mutating func pop() -> T {
        return items.removeLast()
    }
}
```

Generic 스택 구조체에서 Int형 사용

```
struct Stack <T> {
    var items = [T]()
    mutating func push(_ item: T) {
        items.append(item)
    }
    mutating func pop() -> T {
        return items.removeLast()
    }
}
```

```
var stackOfInt = Stack<Int>()
//var stackOfInt = IntStack()
print(stackOfInt.items)  //[]
stackOfInt.push(1)
print(stackOfInt.items) //[1]
stackOfInt.push(2)
print(stackOfInt.items)
stackOfInt.push(3)
print(stackOfInt.items)
print(stackOfInt.pop())  //3
print(stackOfInt.items)
print(stackOfInt.pop())
print(stackOfInt.items)
print(stackOfInt.pop())
print(stackOfInt.items) //[]
```

Generic 스택 구조체에서 Int형, String형 사용

```
struct Stack <T> {
    var items = [T]()
    mutating func push(_ item: T) {
        items.append(item)
    }
    mutating func pop() -> T {
        return items.removeLast()
    }
}

//다양한 자료형에 대해 같은 알고리즘 적용
```

```
var stackOfInt = Stack<Int>()
stackOfInt.push(1)
print(stackOfInt.items)
stackOfInt.push(2)
print(stackOfInt.items)
print(stackOfInt.pop())
print(stackOfInt.items)
print(stackOfInt.pop())

var stackOfString = Stack<String>()
stackOfString.push("일")
print(stackOfString.items)
stackOfString.push("이")
print(stackOfString.items)
print(stackOfString.pop())
print(stackOfString.items)
print(stackOfString.pop())
```

swift의 Array도 generic 구조체

- `var x : [Int] = []  //빈 배열`
- `var y = [Int]()`
- `var z : Array<Int> = []`

- `var a : [Int] = [1,2,3,4]`
- `var b : Array<Int>  = [1,2,3,4]`
- `var c : Array<Double>  = [1.2,2.3,3.5,4.1]`

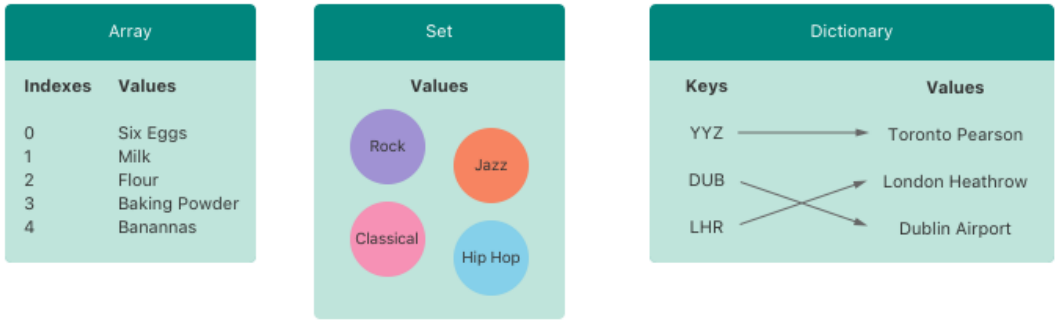
- `@frozen struct Array<Element>`
  - @frozen은 저장프로퍼티 추가, 삭제 불가

Collection Type

Collection Type

- https://en.wikipedia.org/wiki/Collection_(abstract_data_type)
- https://docs.swift.org/swift-book/LanguageGuide/CollectionTypes.html

Swift provides three primary *collection types*, known as arrays, sets, and dictionaries, for storing collections of values. Arrays are ordered collections of values. Sets are unordered collections of unique values. Dictionaries are unordered collections of key-value associations.



Arrays, sets, and dictionaries in Swift are always clear about the types of values and keys that they can store. This means that you can't insert a value of the wrong type into a collection by mistake. It also means you can be confident about the type of values you will retrieve from a collection.

> NOTE
>
> Swift's array, set, and dictionary types are implemented as *generic collections*. For more about generic types and collections, see Generics.

Array

Array	
Indexes	**Values**
0	Six Eggs
1	Milk
2	Flour
3	Baking Powder
4	Banannas

Array

- 연결리스트(linked list)
- https://developer.apple.com/documentation/swift/array

Generic Structure

Array

An ordered, random-access collection.

Declaration

```
@frozen struct Array<Element>
```

Overview

Arrays are one of the most commonly used data types in an app. You use arrays to organize your app's data. Specifically, you use the `Array` type to hold elements of a single type, the array's `Element` type. An array can store any kind of elements—from integers to strings to classes.

swift의 배열은 generic 구조체

- `var x : [Int] = []  //빈 배열`
- `var y = [Int]()`
- `var z : Array<Int> = []`

- `var a : [Int] = [1,2,3,4]`
- `var b : Array<Int>  = [1,2,3,4]`
- `var c : Array<Double>  = [1.2,2.3,3.5,4.1]`

- `@frozen struct Array<Element>`
  - @frozen은 저장프로퍼티 추가, 삭제 불가

Array의 자료형

```
let number = [1, 2, 3, 4]   //타입 추론
let odd : [Int] = [1, 3, 5]
let even : Array<Int> = [2, 4, 6]

print(type(of:number)) //Array<Int>
print(number)//[1, 2, 3, 4]

print(type(of:odd))
print(odd)

print(type(of:even))
print(even)

let animal = ["dog", "cat","cow"]
print(type(of:animal))//Array<String>
print(animal)
```

빈 배열(empty array)

```
var number : [Int] = []
var odd = [Int]()
var even : Array<Int> = Array()
print(number) //[]
```

빈 배열(empty array) 주의 사항

```
let number : [Int] = []
//빈 배열을 let으로 만들 수는 있지만 초기값에서 변경 불가이니 배열의 의미 없음
var odd = [Int]()
var even : Array<Int> = Array()
print(number)
number.append(100)  //let으로 선언한 불변형 배열이라 추가 불가능
//error: cannot use mutating member on immutable value: 'number' is a 'let' constant
odd.append(1)
even.append(2)
```

- 가변형(mutable)
  - var animal = ["dog", "cat","cow"]

- 불변형 (immutable)
  - 초기화 후 변경 불가
  - let animal1 = ["dog", "cat","cow"]