

# 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 `Element`는 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

---

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

- 배열 리터럴 문법은 대괄호 [ ] 를 사용한다.

[ 값 1 , 값 2 , 값 3 ]

# 배열 Element 가져오기

---

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

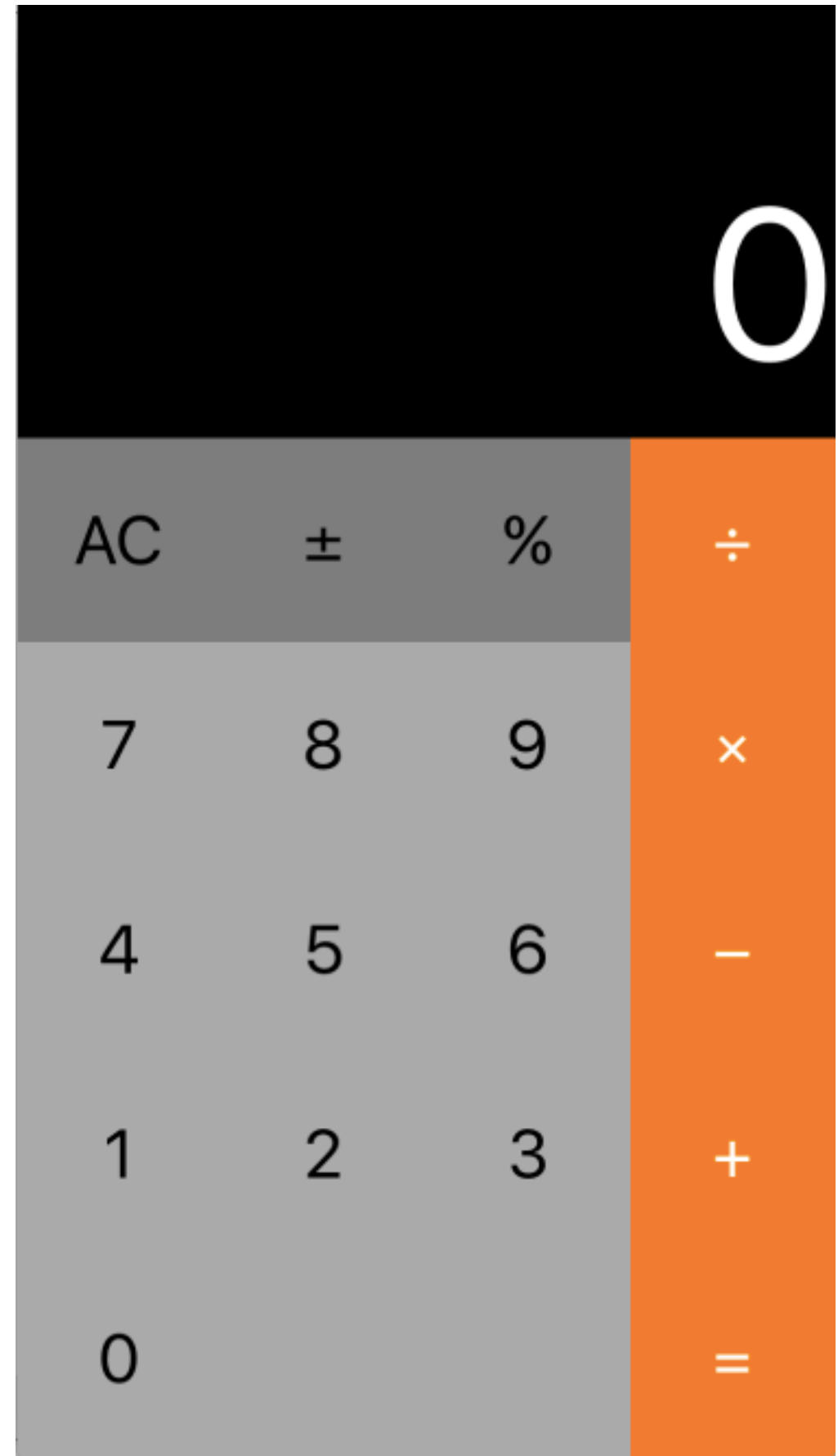
```
var someInts:[Int] = [1,2,3,4]
print("\ (someInts[0])")
print("\ (someInts[3])")
```

# 연습문제 - 배열

---

- 요일 구하기: 2017년도 1월 1일은 일요일 입니다. 그렇다면 내가 원하는 월/일을 받아서 해당 날의 요일을 구하는 함수를 만들어 보세요.
  1. 함수 입력 : `getWeekDay(atMonth:3, day: 13)`
  2. 리턴결과 : "Mon"
- 중복 숫자 줄이기: 연속으로 중복된 숫자를 없애는 함수
  1. 함수 입력 : `shoter(num:1002233422)`
  2. 리턴결과 : 102342
- 소수찾기 : 2부터 입력된 숫자까지의 모든 소수를 찾아서 반환
  1. 함수 입력 : `allPrimeNumber(endNum:10)`
  2. 결과 : [2,3,5,7]
- 시저 암호: 어떤 문장의 각 알파벳을 일정한 거리만큼 밀어서 다른 알파벳으로 바꾸는 암호화 방식을 시저 암호라고 합니다. A를 3만큼 밀면 D가 되고 z를 1만큼 밀면 a가 됩니다
  1. 함수 입력 : `ceasar(data:["a", "B", "C", "d"], keyNum:4)`
  2. 결과 : ["e", "F", "G", "h"]

## Step 2. 객체 만들기



---

# 클래스 & 객체

---

# Swift Class Architecture

```
class ClassName : superClass
{
    var vName1 = "1"
    var vName2 = 4

    func fName1() - > Any
    {

    }

    func fName2(_ ani:Bool)
    {

    }
}
```

<ClassName.swift>



## 구조

```
class Subject
{
}

```

# 프로퍼티

---

```
class Subject
{
    var name:String = “소프트웨어 입문”
    var score:Int = 95
    var gradeNumber:Int = 3
    var grade:String = “A+”
}
```

프로퍼티 : 객체가 가지고 있는 속성

# 인스턴스 생성(객체만들기)

---

```
class Subject
{
    var name:String = “소프트웨어 입문”
    var score:Int = 95
    var gradeNumber:Int = 3
    var grade:String = “A+”
}
```

```
var introductionSoftwaer:Subject = Subject()
```

# 초기화(initialization)

---

```
init() {  
    // perform some initialization here  
}
```

---

```
class Subject  
{  
    var name:String = ""  
    var score:Int = 0  
    var gradeNumber:Int = 3  
    var grade:String = "F"  
  
    init() {  
        name = "소프트공학 입문"  
        score = 95  
    }  
}
```

# Custom init

---

- init에 parameter를 추가해서 custom하게 만들수 있다.

```
class Subject
{
    var name:String = ""
    var score:Int = 0
    var gradeNumber:Int = 3
    var grade:String = "F"

    init(name:String, socre:Int) {
        self.name = name
        self.score = socre
    }
}
```

# 학점계산기 만들기

---

- 객체지향형으로 학점 계산기를 만들어 봅시다.
- 어떤 객체가 필요 할까요?

# 학점계산기 만들기

---

- 과목 클래스(Subject Class)
- 학생 클래스(Student Class)
- 연산 클래스(Calculator Class)

# Subject Class

---

- 과목이름 변수
- 과목점수 변수



# Student Class만들기

---

- 이름
- 과목들(과목 객체 배열)
- 평균점수

# Calculator Class만들기

---

- 학생 객체를 받아서 평균을 알려주는 클래스

# 실습 : 포켓몬 클래스 만들기

---



# 실습 : 포켓몬 클래스 만들기

---



# 클래스의 상속

---

- Subclassing
- 기존에 구현되어있는 클래스를 확장, 변형
- 부모 클래스(super class, parent class)와 자식 클래스(sub class, child class)로 관계를 표현
- 상속 할 수록 더 확장되는 구조
  - 즉, 자식이 기능이 더 많을 가능성이 크다

# 클래스의 상속

---

```
class UniversityStudent: Student {  
  
}
```

# UniversityStudent

**Student**

**Person**

name  
age  
eat()

grade  
study()

major  
goMT()

# 클래스의 상속

---

- UniversityStudent클래스를 만들어 Student를 상속받게 만듭니다.
- grade와 point프로퍼티 추가
- 그에따른 calculator에 추가 함수 만들기



---

# 재정의

---

# 재정의

---

- 영어로 Override

# 재정의

---

- 영어로 override(오버라이드)
- 부모 클래스에게서 물려받은 성질을 그대로 사용하지 않고 자식 클래스에게 맞는 형태 또는 행위로 변경하여 사용할 수 있는 기능

# 재정의

---

- Person 클래스의 eat 메서드는 집밥을 먹도록 하고,  
Person 클래스를 상속받은 Student 클래스의 eat 메서드는  
급식을 먹게 하고,  
Student 클래스를 상속받은 UniversityStudent 클래스의 eat  
메서드는 학식을 먹게 만들어 봅시다

# 다형성

---

- 재정의(Override)와 중복정의(Overload)는 OOP의 **다형성**의 또다른 모습
- Objective-C는 **중복정의(Overload)**를 지원하지 않습니다

# 예제

---

- 학점계산기 업그레이드

---

# 변수 & 함수

---

# Swift Class Architecture

```
class ClassName : superClass
{
    var vName1 = "1"
    var vName2 = 4

    func fName1() - > Any
    {

    }

    func fName2(_ ani:Bool)
    {

    }
}
```

<CalssName.swift>



# 변수 & 함수

---

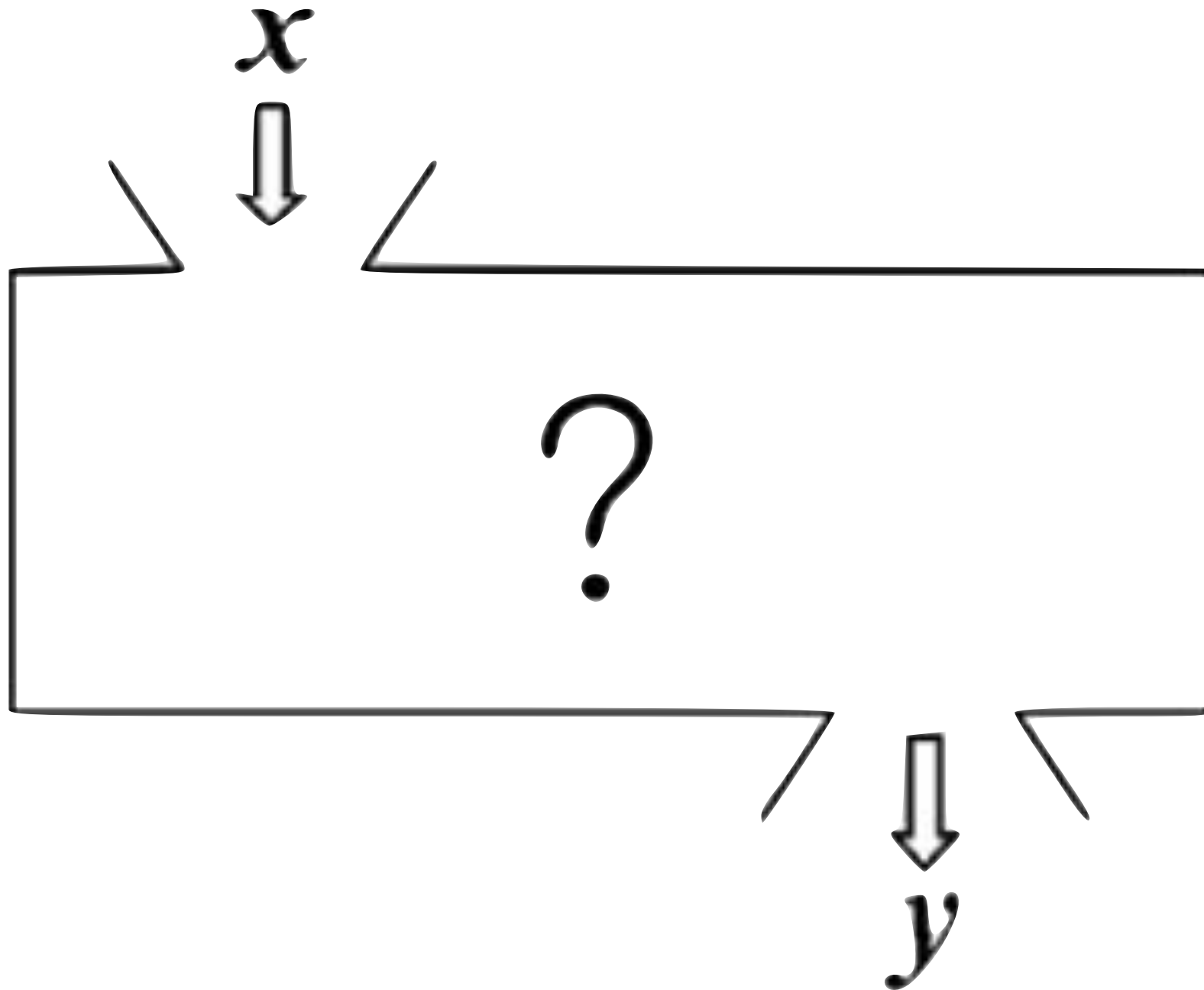
- 변수 : 프로그램에서 데이터의 저장공간을 담당
- 함수 : 프로그램이 실행되는 행동을 담당

- 변수를 만드는데 있어 필요한 것은?

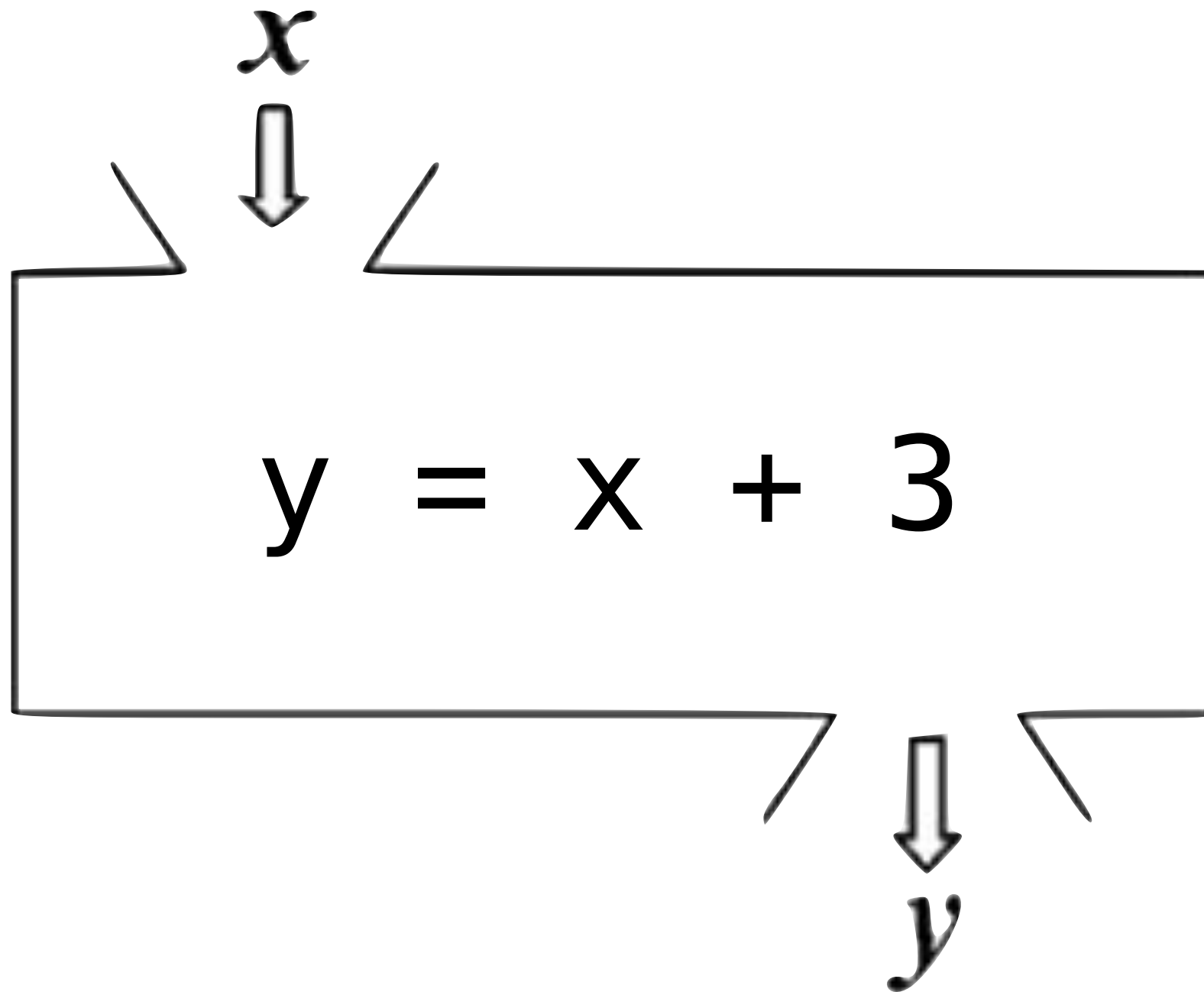
키워드 + 변수 명(Name) + 변수 타입(Type)

문법 : `var vName:Any`

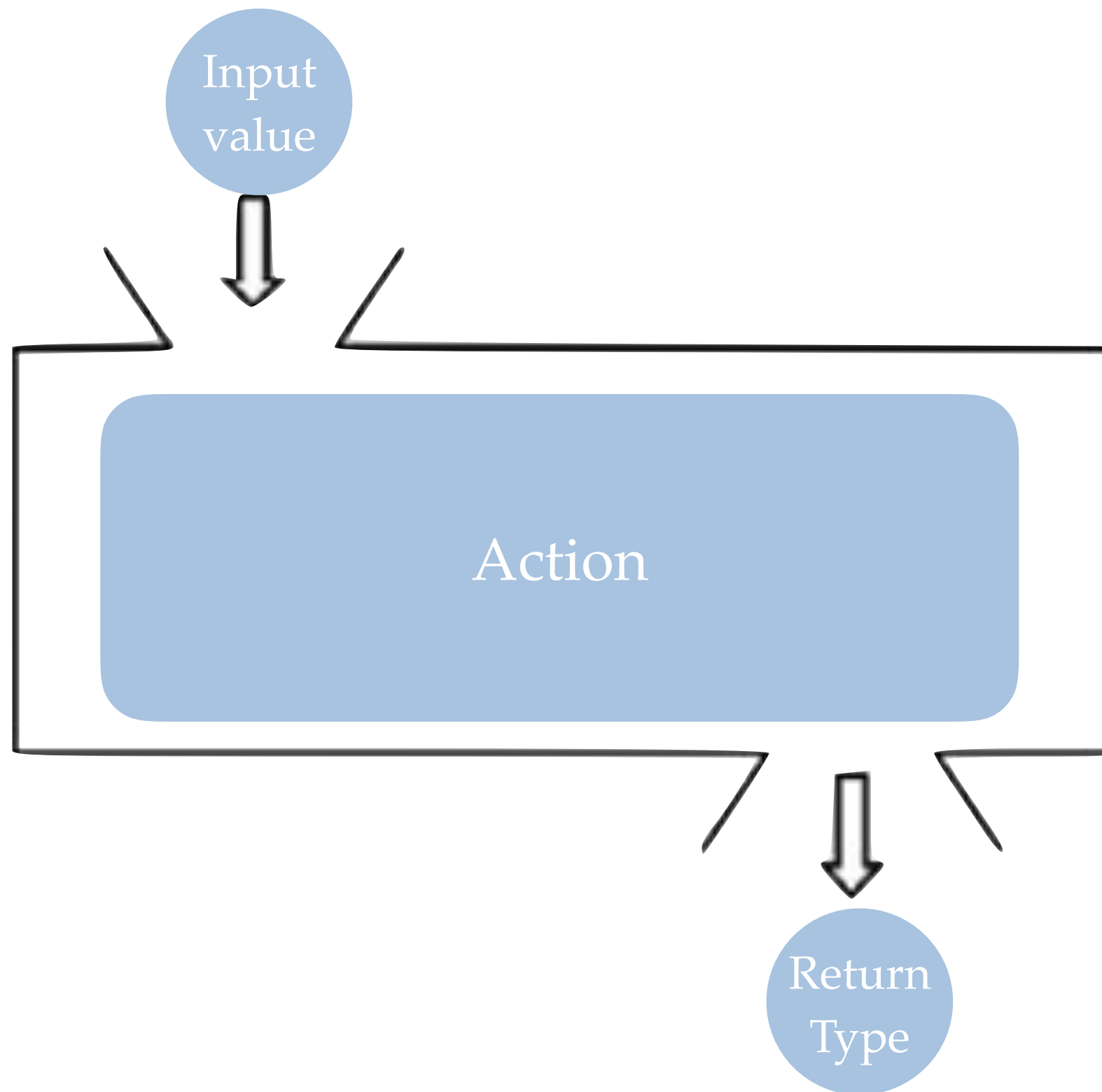
# 함수



# 함수



# 함수



- 함수 만들기 위해 필요한것?

키워드 + 함수명(Name) + 입력값(Input Value) +  
함수 내용(Action) + 결과타입(Return Type)

문법 : `func vName(_ parameter: Any) -> Any`  
    {  
        //함수 내용  
    }

# Swift 문법 - 변수

---

키워드                      변수타입

`var` `name` `:` `Type` `=` `value`

변수명                      값

# 키워드

---

- 변수 : 변할수 있는 값

```
var name:String = "joo"
```

- 상수 : 변할수 없는 고정 값

```
let name:String = "joo"
```



# 키워드

---

- 변수 : 변할수 있는 값

```
var name:String = "joo"  
name = "iOS개발 스쿨" ———— O
```

- 상수 : 변할수 없는 고정값

```
let name:String = "joo"  
name = "iOS개발 스쿨" ———— X
```

# 변수명

---

- 명명규칙에 따라 작성
- 유니 코드 문자를 포함한 거의 모든 문자가 포함될 수 있다.(한글 가능)
- 변수안에 들어있는 데이터를 표현해 주는 이름으로 작성
- 중복작성 불가 ( 한 클래스, 함수, 구문 안에서)

# 명명규칙

---

- 시스템 예약어는 사용할 수 없다.
- 숫자는 이름으로 시작될 수는 없지만 이름에 포함될 수 있다.
- 공백을 포함 할 수 없다.
- 변수 & 함수명을 lowerCamelCase,  
클래스 명은 UpperCamelCase로 작성한다.

# 변수 타입

---

## 기본형

타입이름	타입	설명	Swift 문법 예제
정수	Int	1, 2, 3, 10, 100	var intName: Int
실수	Double	1.1, 2.35, 3.2	var doubleName: Double
문자열	String	"this is string"	var stringName: String
불리언	Bool	true or false	var boolName: Bool

## 참조형

타입이름	타입	설명	Swift 문법 예제
Custom Type	ClassName	클래스 객체를 다른곳에서 사용할 경우	let customView: UIView
			let timer: Timer

# Int & Uint

---

- 정수형 타입 (Integer)
- Int : +/- 부호를 포함한 정수이다.
- Uint : - 부호를 포함하지 않은(0은 포함) 정수
- 최대값과 최소값은 max, min프로퍼티를 통해 알아볼수 있다.
- Int8, Int16, Int32, Int64, UInt8, UInt16, UInt32, UInt64의 타입으로 나뉘져 있는데 시스템 아키텍처에 따라서 달라진다.
- 접두어에 따라 진수를 표현할수 있다. (2진법 0b, 8진법0o, 16진법 0x)

# Bool

---

- 불리언 타입 (true, false)

# Float & Double

---

- 부동 소수점을 사용하는 실수형 타입
- 64비트의 부동소수점은 Double, 32비트 부동 소수점은 Float으로 표현한다.
- Double은 15자리,Float은 6자리의 숫자를 표현가능
- 상황에 맞는 타입을 사용하는것이 좋으나 불확실할때는 Double을 사용하는 것을 권장.

# Character

---

- 단어나 문장이 아닌 문자 하나!
- 스위프트는 유니코드 문자를 사용함으로, 영어는 물론, 유니코드 지원 언어, 특수기호등을 모두 사용 할 수 있다.
- 문자를 표현하기 위해서는 앞뒤에 쌍 따옴표(“ ”)를 붙여야 한다.



# String

---

- 문자의 나열, 문자열이라고 한다.
- Character와 마찬가지로 유니코드로 이뤄져 있다.
- 문자열을 다루기 위한 다양한 기능이 제공된다.  
(hasPrefix, uppercased, isEmpty등)

# String 조합

---

1. string 병합: + 기호를 사용

```
var name:String  
name = "주" + "영민"
```

2. interpolation(삽입): \ (참조값)

```
var name:String = "주영민"  
print("my name is \ (name) ")
```

\ ()가 interpolation

# 튜플

---

- 정해지지 않은 데이터 타입의 묶음
- 소괄호 ( ) 안에 타입을 묶음으로 새로운 튜플타입을 만들수 있다. ex) (Int, Int) // (String, Int, String)
- 각 타입마다 이름을 지정해 줄수도 있다.  
ex) (name:String, age:Int)

# 튜플 예시

---

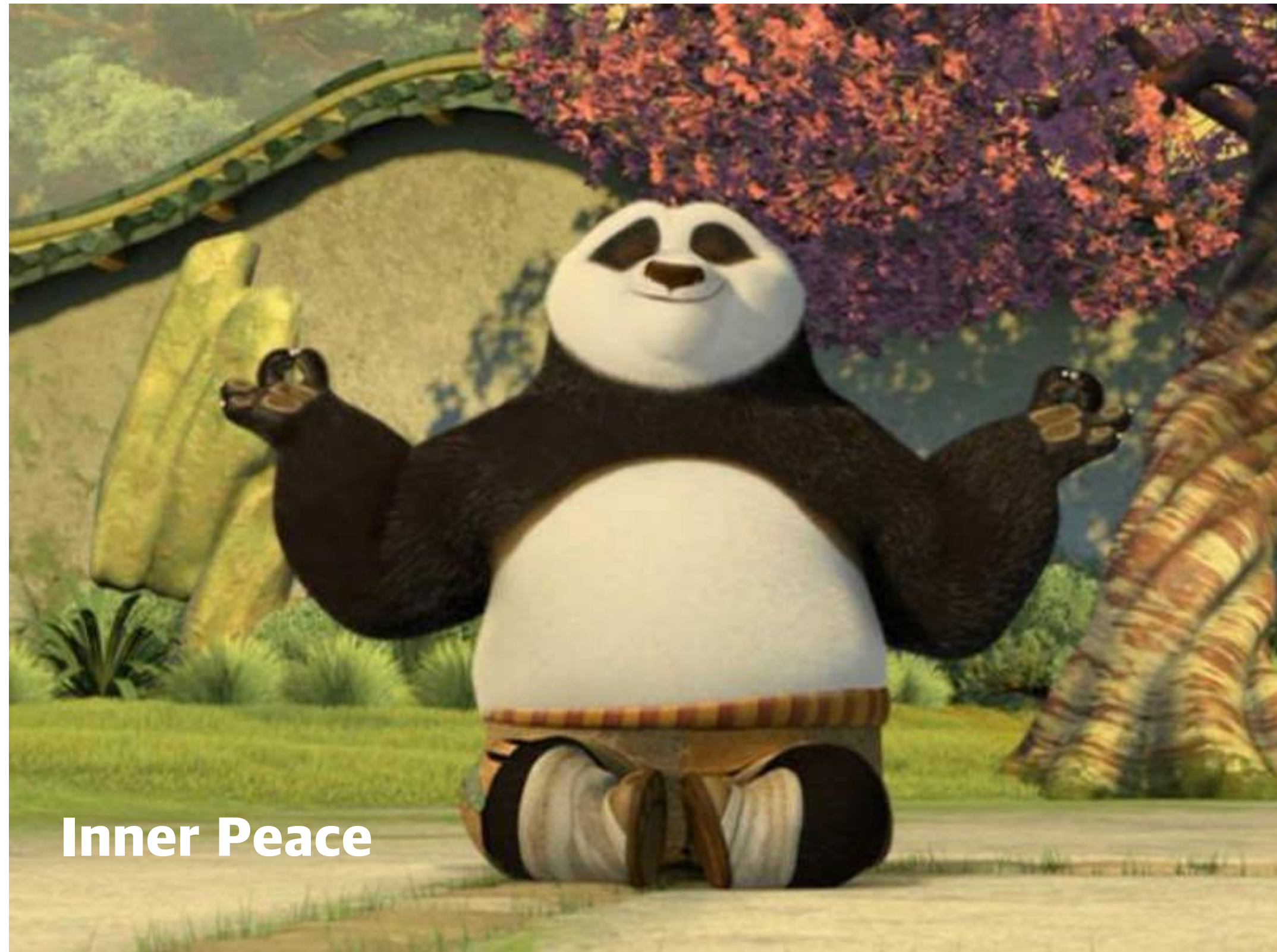
```
var coin:(Int,Int,Int,Int) = (3,1,5,3)
print("10원짜리 : \" + coin.0)
print("50원짜리 : \" + coin.1)
print("100원짜리 : \" + coin.2)
print("500원짜리 : \" + coin.3)
```

```
var person:(name:String, age:Int, weight:Double)
           = ("joo", 30, 180.2)
print("이름 : \" + person.name)
print("나이 : \" + person.age)
print("몸무게 : \" + person.age)
```

# Any, AnyObject, nil

---

- Any : 스위프트 내의 모든 타입을 나타냄
- AnyObject : 스위프트 내의 모든 객체 타입을 나타낸다.(클래스)
- nil : 데이터가 없음 을 나타내는 키워드



**Inner Peace**

# 캐스팅(형변환)

---

```
var total:Int = 107
```

```
var average:Double
```

```
average = total/5
```

← type Error

# 캐스팅을 해야하는 이유

---

실수 : 107.0

1	1	0	0	1	0	0	0	1	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

정수 : 107

1	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



# 캐스팅(형변환)

---

```
var total:Int = 107
```

```
var average:Double
```

```
average = total/5 ← type Error
```

```
average = Double(total)/5 ← casting
```

# 캐스팅(형변환)

---

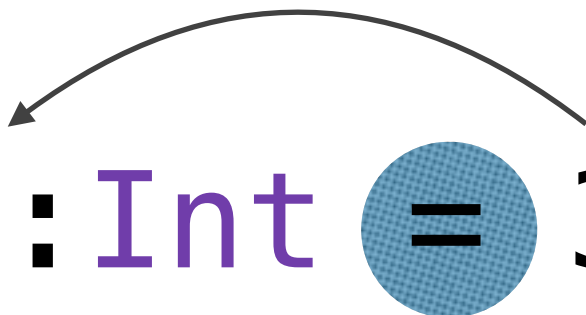
```
var stringNum:String  
var doubleNum:Double  
let intNum:Int = 3
```

```
stringNum = String(intNum) ← int to string  
doubleNum = Double(intNum) ← int to double
```

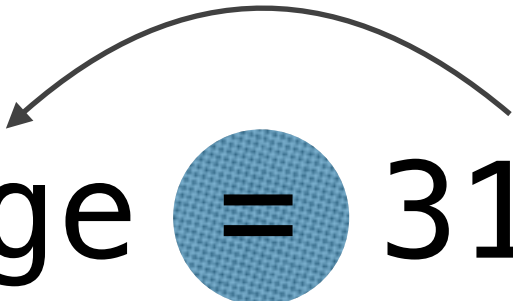
# 변수 값 지정

---

`var number: Int = 3`



`var age = 31` (타입 추론)



대입연산자	예제	설명
=	number = 4	number변수에 숫자 4를 넣는다.

# Swift 문법 - 함수

---

```
func fName(agumentName paramName:Int) -> Int
{
    return paramName + 3
}
```

# Swift 문법 - 함수

---

키워드                      인수명                      매개변수명                      반환타입

함수 이름                      매개변수타입

```
func fName(argumentName paramName: Int) -> Int  
{  
    return paramName + 3  
}
```

함수 내용

The diagram illustrates the Swift function syntax with the following components and annotations:

- func**: Keyword (키워드), circled in red.
- fName**: Function name (함수 이름), circled in red.
- argumentName**: Parameter name (인수명), circled in red.
- paramName**: Parameter name (매개변수명), circled in red.
- Int**: Parameter type (매개변수타입), circled in red.
- >**: Return type arrow, circled in red.
- Int**: Return type (반환타입), circled in red.
- { ... }**: Function body (함수 내용), enclosed in a blue box.
- return paramName + 3**: Return statement inside the function body.

# Argument Labels and Parameter Names

---

인수레이블 명

매개변수명

매개변수타입

```
func fName(argumentName paramName: Int) -> Int  
{  
    return paramName + 3  
}
```

- 인수레이블은 함수 호출시 사용 되는 이름표.
- 매개변수는 함수 내부에서 사용 되는 변수명
- 인수레이블은 생략가능하며 없을때는 매개변수명이 인수레이블로 사용된다.

# Default Parameter Values

---

```
func number(num1:Int, num2:Int = 10) -> Int {  
    return num1 + num2  
}
```

```
number(num1: 10)           ← 20  
number(num1: 10, num2: 5) ← 15
```

- 매개변수에는 기본값을 설정할수 있다.
- 기본값은 인자로 값이 들어오지 않을때 사용된다.

# In-Out Parameter Keyword

---

inout Keyword

```
func swapTwoInts(_ a: inout Int, _ b: inout Int) {  
    let temporaryA = a  
    a = b  
    b = temporaryA  
}
```

- 매개변수는 기본 상수값이다.
- 만약 매개변수의 값을 변경해야 한다면 inout 키워드를 사용하여 inout 변수로 지정해야만 한다.
- inout 변수 지정은 타입 앞에 inout keyword를 작성해준다.
- inout 변수가 지정된 함수의 인수앞에서 & 가 붙어야 한다.



# In-Out Parameter Keyword

---

```
func swapTwoInts(_ a: inout Int, _ b: inout Int) {  
    let temporaryA = a  
    a = b  
    b = temporaryA  
}
```

```
var someInt = 3  
var anotherInt = 107  
swapTwoInts(&someInt, &anotherInt)
```



---

```
swapTwoInts(3, 107)  
swapTwoInts(&3, &107)
```



# 여러가지 함수 - 매개변수

---

```
func getNumber(firstNum num1:Int) -> Int {  
    return num1  
}
```

```
func getNumber(num1:Int) -> Int {  
    return num1  
}
```

```
func getNumber() -> Int {  
    var num1:Int = 22  
    return num1  
}
```

```
func getNumber(firstNum num1:Int, secondNum num2:Int) -> Int {  
    return num1 + num2  
}
```

```
func sumNumber(num1:Int, num2:Int = 5) -> Int {  
    return num1 + num2  
}
```

# 반환타입

---

반환타입

```
func fName(agumentName paramName: Int) -> Int
{
    return paramName + 3
}
```

- 함수 실행 결과의 타입을 명시 해준다. (Return Type)
- **return** 키워드를 사용하여 함수 결과 반환.  
반환 타입과 같은 타입의 데이터를 반환 해야 한다.
- 한개의 값만 반환 할수 있다.
- 반환값이 없는 경우는 Return Type을 작성하지 않고( -> 제거)  
**return** 키워드를 사용할 필요가 없다.(반환값이 없기때문)

# 반환타입 예제

---

```
func printName() -> String{  
    return "my name is youngmin"  
}
```

```
func printName(){  
    print("my name is youngmin")  
}
```

```
func printName(name:String = "youngmin"){  
    print("my name is \(name)")  
}
```

```
func printName(explain str:String, name str2:String) -> String{  
    return str + str2  
}
```

```
func printName(explain str: inout String) -> String{  
    str += "joo"  
    return str  
}
```