

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 syntax of a Swift function. The code is: `func fName(argumentName paramName: Int) -> Int { return paramName + 3 }`. Annotations above the code identify parts: '키워드' (keyword) points to 'func'; '인수명' (argument name) points to 'argumentName'; '매개변수명' (parameter name) points to 'paramName'; '매개변수타입' (parameter type) points to 'Int' in the parameter list; '반환타입' (return type) points to 'Int' after the arrow. A bracket below the opening curly brace is labeled '함수 이름' (function name), pointing to 'fName'. A bracket below the closing curly brace is labeled '함수 내용' (function body), pointing to the code inside the braces. Red circles highlight the keywords 'func', 'return', and the types 'Int' and 'Int'.

Argument Labels and Parameter Names

인수레이블 명

매개변수명

매개변수타입

```
func fName(agumentName paramName: Int) -> Int  
{
```

```
    return paramName + 3
```

```
}fName(agumentName: 10) ← 함수호출
```

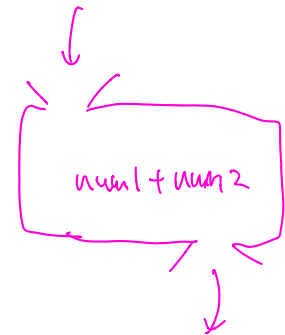
- 인수레이블은 함수 호출시 사용 되는 이름표.
- 매개변수는 함수 내부에서 사용 되는 변수명
- 인수레이블은 생략가능하며 없을때는 매개변수명이 인수레이블로 사용된다.

argumentName은 안써도 되지만 매개변수를 설명 할 수 있기 때문에
사용하는게 좋다.

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  
}
```

parameter를 통해서 들어온 값은 상수.

- 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

wildCard. : 이름을 쓰지않겠다.

```
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)
```



swap 함수. 값을 바꾸.

여러가지 함수 - 매개변수

```
func getNumber(argument NamefirstNum parameter Namenum1:parameter TypeInt) -> Int { 기반형.  
    return num1 Return type.  
}
```

```
func getNumber(num1: Int) -> Int { argument X  
    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 { Default X.  
    return num1 + num2  
}
```

공수는 class 내에서 behavior 를 정의하는 것.



Inner Peace

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. *→ return 하면 값은 반환한
종료.*
반환 타입과 같은 타입의 데이터를 반환 해야 한다.
- 한개의 값만 반환 할수 있다.
- 반환값이 없는 경우는 Retrun Type을 작성하지 않고(-> 제거)
retrun키워드를 사용할 필요가 없다.(반환값이 없기때문)

반환타입 예제

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

반환값이 있으면
변수에 넣어야 함.

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

반환값 X

프린트

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

자

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

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

Step 2. Class



함수 꾸미기

- 단순 명령어의 순서가 아닌 복잡한 명령을 내리기 위해선 프로그램 명령어 컨트롤 방법(Controll Flow)이 필요하다.
- 조건문(while, for-in) & 선택문(if, guard, switch)을 통해 함수를 컨트롤 할수 있다.

조건문

강사 주영민

조건문 개론

- 함수 내부에서 실행되는 선택문
- 특정 조건에 따라 선택적으로 코드를 실행시킨다.
- 대표적인 조건문으로 if-else문과 switch-case문이 있다.

if-else문

조건이 참일경우 if문 대괄호 안의 코드가 실행된다.

만약 조건이 거짓인 경우 else문 대괄호 안의 코드가 실행된다.

```
if 조건 {  
    //조건이 만족되면 실행  
}else{  
    //조건이 만족되지 않을때 실행  
}
```

***조건값은 참,거짓의 나타나는 Bool값으로 표현.**

else if문

추가 조건 방법으로 반복해서 추가 할수 있다.

```
if 조건1 {  
    //조건1이 만족되면 실행  
}else if 조건2{  
    //조건1이 만족되지 않을때 실행  
}else{  
    //조건들 모두 만족되지 않을때 실행  
}
```

***조건2는 조건1이 거짓일때 실행된다.**

조건 만들기

- 비교연산자를 통해 조건의 결과가 bool값으로 나와야 한다.
- 논리 연산자로 다양한 조건의 조합이 가능하다.

연산자

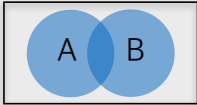
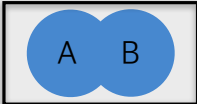

산술 연산자

기호	예제	설명
+	$1 + 2 = 3$	더하기
-	$2 - 1 = 1$	빼기
*	$2 * 1 = 2$	곱하기
/	$10 / 3 = 3$	나누기
%	$10 \% 3 = 1$	나머지

비교 연산자

기호	예제	설명
==	$A == B$	A와 B가 같다.
>=	$A \geq B$	A가 B보다 크거나 같다
<=	$A \leq B$	A가 B보다 작거나 같다.
>	$A > B$	A가 B보다 크다
<	$A < B$	A가 B보다 작다

논리 연산자

기호	예제	집합	설명
&&	A조건 && B조건		A조건이 참이고, B조건이 참이면 참이다.
	A조건 B조건		A조건이나, B조건 둘중에 하나가 참이면 참이다.
!	!(A조건 B조건)		A B조건을 반대

추가연산자

복합연산자	예제	설명
<code>+=</code>	<code>a += 1</code>	a에 값을 더하기
<code>-=</code>	<code>b -= 2</code>	b에 값을 빼기

범위 연산자	예제	설명
<code>a...b</code>	<code>3...10</code>	a~b까지의 숫자
<code>a..<b</code>	<code>0..<10</code>	a~b까지 숫자중 b는 포함 안함

Identity 연산자	예제	설명
<code>===</code>	<code>person2 === person1</code>	person1과 person2는 같은 인스턴스를 참조하고 있다.
<code>!==</code>	<code>person2 !== person1</code>	person1과 person2는 다른 인스턴스를 참조하고 있다.

삼항연산자

삼항연산자	설명
question ? answer1 : answer2	question이 참이면 answer1값을 거짓이면 answer2값을 지정한다.

```
let age = 20
var result:String = age > 19 ? "성년" : "미성년"
```

조건의 예

```
func printGeneration(age: Int)
{
    if (age >= 30) {
        print("30대 이상")
    } else if (age >= 20)
    {
        print("20대")
    } else
    {
        print("미성년자")
    }
}
```

다음 실행 결과는?

1. printGeneration(10);
 >> 미성년자
2. printGeneration(40);
 >> 30대 이상
3. printGeneration(20);
 >> 20대

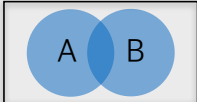
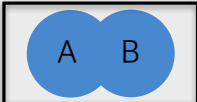
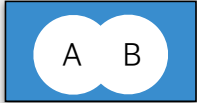
조건문 예시

```
func check(name compareName:String )
{
    if compareName == "주영민"
    {
        print("주영민 입니다.")
    }else
    {
        print("주영민이 아닙니다")
    }
}
```

조건문 예시

```
func isEven(number num:Int) -> Bool
{
    if(num%2 == 0)
    {
        //짝수
        return true
    }else
    {
        //홀수
        return false
    }
}
```

논리연산자

기호	예제	집합	설명
&&	A조건 && B조건		(A조건이 참이고, B조건이 참이면) 참이다.
	A조건 B조건		(A조건이나, B조건 둘중에 하나가 참이면) 참이다.
!	!(A조건 B조건)		조건의 반대(참 -> 거짓, 거짓 -> 참)

조건의 조합

1. or 조합(||) : 조건들 중 한개만 참이어도 전체 조건이 참이다.
2. and 조합(&&) : 조건들 모두 참이어만 전체 조건이 참이다.
3. not (!) : 조건 결과의 반대값

조건의 조합

<시험 점수를 구하는 함수>

시험점수 95점 이상 100점 이하는 A+,
90점 이상 95점 미만은 A,
85점 이상 90점 미만은 B+
80점 이상 85점 미만은 B...

* 다음과 조건들은 어떻게 처리해야 될까요?

조건의 조합

시험점수 95점 이상 100점 이하는 A+

```
if( 100 >= score && 95 <= score)
```

90점 이상 95점 미만은 A

조건을 적어보세요

85점 이상 90점 미만은 B+

조건을 적어보세요

80점 이상 85점 미만은 B...

조건을 적어보세요

문제

1. 시험 점수를 받아서 해당 점수의 등급(**Grade**)을 반환해주는 함수

***Grade** = A+, A, B+, B, C+...

2. **Grade** 받아서 **Point**로 변환해주는 함수

Point = 4.5, 4.0, 3.5, 3.0...

Switch문

- 패턴 비교문
- 가장 첫번째 매칭되는 패턴의 구문이 실행된다.

switch 문법

```
switch some value to consider {  
  case value 1:  
    respond to value 1  
  case value 2,  
    value 3:  
    respond to value 2 or 3  
  default:  
    otherwise, do something else  
}
```

switch 문법

- 각의 상태는 case 키워드를 통해 나타낼수 있다.
- 각 case 상태 끝에는 콜론(:)을 작성해서 매칭 패턴을 종료한다.
- 여러개의 case가 필요하면 콤마(,)를 통해 상태를 추가 할수 있다.
- 각 case는 분기로 실행되며 매칭된 해당 case문이 종료되면 switch 문을 종료시킨다.
- 각 case의 상태는 switch 문의 value값에 매칭된 지점을 결정하며, 모든 value에 대해 매칭 되어야 한다. 만약 매칭되는 값이 없다면 default 키워드를 통해 기본 매칭값을 설정하며, default키워드는 마지막에 배치된다.

switch 예제

```
func sampleSwitch(someCharacter:Character)
{
    switch someCharacter {
    case "a":
        print("The first letter of the alphabet")
    case "z":
        print("The last letter of the alphabet")
    default:
        print("Some other character")
    }
}
```

문제

1. 시험 점수를 받아서 해당 점수의 등급(**Grade**)을 반환해주는 함수

***Grade** = A+, A, B+, B, C+...

2. **Grade** 받아서 **Point**로 변환해주는 함수

Point = 4.5, 4.0, 3.5, 3.0...

Interval Matching

- Switch 문의 상태는 단순 value매칭을 넘어 좀더 다양한 패턴을 통해 매칭이 가능하다.
- interval matching은 범위 연산자를 통해 해당 범위에 해당하는 value를 매칭 시킬수 있다.

Interval Matching 예제

```
func interSwitch(count:Int)
{
    let countedThings = "moons orbiting Saturn"
    let naturalCount: String
    switch count {
    case 0:
        naturalCount = "no"
    case 1..<5:
        naturalCount = "a few"
    case 5..<12:
        naturalCount = "several"
    case 12..<100:
        naturalCount = "dozens of"
    case 100..<1000:
        naturalCount = "hundreds of"
    default:
        naturalCount = "many"
    }
    print("There are \(naturalCount) \(countedThings).")
}
```

튜플매칭

- 튜플을 사용해서 여러개의 value를 동시에 확인 할수 있습니다.
- 사용 가능한 모든 값에 대한 매칭은 와일드 카드 (_)를 통해서 매칭 가능합니다.

튜플 예제

```
func getPoint(somePoint:(Int,Int))
{
    switch somePoint {
    case (0, 0):
        print("\(somePoint) is at the origin")
    case (_, 0):
        print("\(somePoint) is on the x-axis")
    case (0, _):
        print("\(somePoint) is on the y-axis")
    case (-2...2, -2...2):
        print("\(somePoint) is inside the box")
    default:
        print("\(somePoint) is outside of the box")
    }
}
```


값 바인딩

- case 내부에서 사용되는 임시 값으로 매칭 시킬수 있다.

값 바인딩 예제

```
func getPoint(somePoint:(Int,Int))
{
    switch somePoint {
    case (0, 0):
        print("\(somePoint) is at the origin")
    case (let x, 0):
        print("on the x-axis with an x value of \(x)")
    case (0, let y):
        print("on the y-axis with an y value of \(y)")
    case (-2...2, -2...2):
        print("\(somePoint) is inside the box")
    default:
        print("\(somePoint) is outside of the box")
    }
}
```

where문

- where 문의 추가로 추가 조건을 넣을수 있다.

where문 예제

```
func wherePoint(point:(int,Int))
{
    switch point {
    case let (x, y) where x == y:
        print("\(x), \(y)) is on the line x == y")
    case let (x, y) where x == -y:
        print("\(x), \(y)) is on the line x == -y")
    case let (x, y):
        print("\(x), \(y)) is just some arbitrary point")
    }
}
```