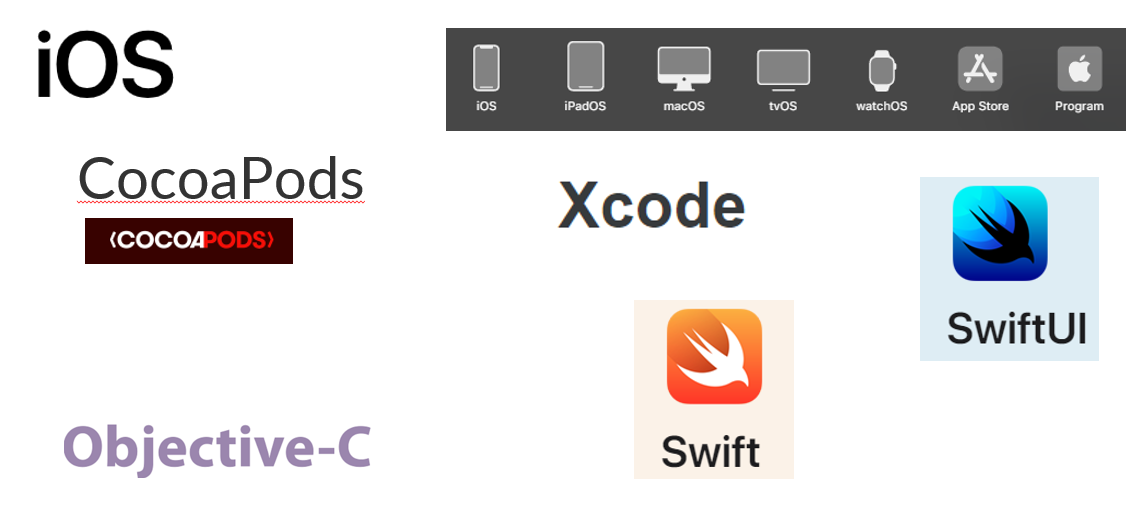
테이블이(가) 표시된 사진

자동 생성된 설명

**가장 많은 카테고리**

**게임, 비즈니스, 교육**

****

**스펠링 중요**

**상수와 변수 선언하고 값 수정하기**

애플은 코드의 효율성과 실행 성능을 높이기 위해서 변수(var)보다는 상수(let)를 사용하라고 권장

*// let으로 만든 상수는 초기값에서 변경이 불가능함*

**let** x = 10

print(x)

//x = 20

//print(x)

변수

Swift 모든 변수는 var, let으로 시작함

var myNumber : Int = 10

// var myNumber = 10 위와 같이 초기값이 있으면 컴파일러가 타입 추론(type inference)을 하므로 데이터 타입을 명시할 필요 없음

자료형의 종류와 크기 궁금 할 때

**var** x = 10

print(type(of:x)) *//Int :변수의 타입을 출력*

**let** s = MemoryLayout.size(ofValue: x) *//8 변수가 메모리를 얼마나 차지하나 출력*

**let** t = MemoryLayout.<Int>size *//8 <> 안 자료형이 메모리를 얼마나 차지하나 출력*

print(s, t)

Int 데이터 타입은 해당 코드가 실행되는 플랫폼에 맞는 정수 크기를 사용(ex 32, 64bit 컴퓨터 등)

위에서 8 이 나오지만 다른 데서 4 나올 수 있음

**문자열 데이터 타입 : String**

**var** userName = "Kim" *// : String 생략하는 것이 일반적임*

*//이런 식으로 문자열 안에 원하는 변수 상수를 넣을 수도 있음*

**var** x = 10;

print("x 값은 : **\(**x**)** ===> **\(**x+1**)**")

*// \() 키워드 안에 변수 상수값을 넣으면 콘솔 창에 출력됨*

Int **형의 크기**

*// INT형은 플랫폼(OS/컴파일러)의 따라 크기가 다름*

print("Int32 Min = **\(**Int32.min**)** Int32 Max = **\(**Int32.max**)**")

*// 32 bit 인트 값의 최대 최소값*

print("Int Min = **\(**Int.min**)** Int Max = **\(**Int.max**)**")

*// 출력 되는것을 보니까 현재 온라인 컴파일러는 64비트 즉 8바이트 만큼 크기를 가짐*

**문자 데이터 타입 : Character 선언 시 중요 포인트**

**var** myChar2 = ":"

**var** myChar3 : Character = "X" *//:Character 생략불가,생략하면 String형임*

*// 그래서 Character을 써야하는 상황일때는 꼭 : Character 써줘야함*

*/\**

*String*

*Character*

*X*

*\*/*

*//유니코드를 이용하여 변수에 문자 'X'를 할당 \u 다음에 코드 값 넣으면 됨*

**var** myChar4 = "\u{0058}"

print(myChar4)

var userCount : Int = 10 // : Int가 type annotation

즉 직접 선언할 때 타입 명시하는 것을 타입 어노 테이션이라고함 (: int) 직접 쓰는 것

위에 : Int 를 생략하는게 타입 타입 추론(type inference) 이라고 함

* 1. var signalStrength = 2.231 // var signalStrength : Double = 2.231
  2. let companyName = "My Company"
  3. Double 타입(스위프트의 타입 추론에서 모든 부동 소수점 수는 Double 타입)
  4. companyName이라는 상수는 String 타입으로 간주

var myChar2 = 20.0

var myChar3 : Float = 30.0

print(type(of: myChar2))

print(type(of: myChar3))

// Double

// Float

**튜플(Tuple)**

**let** myTuple = (10, 12.1, "Hi")

print(myTuple.0)

**var** myString = myTuple.2

print(myString)

*// 튜플을 이용해서 변수를 3개를 한번에 초기화 함*

**let** myTuple2 = (10, 12.1, "Hi") *//: myTuple의 자료형*

**let** (myInt2, myFloat2, myString2) = myTuple2

print(myTuple2)

*// (10,12.1,“Hi”)*

*//무시하고 싶은 값에 밑줄을 사용하면 그 값은 무시*

**let** myTuple3 = (10, 12.1, "Hi")

**var** (myInt3, **\_**, myString3) = myTuple3 *//부동 소수점 수는 무시*

*//튜플을 생성할 때 각 값에 이름을 할당할 수도 있음*

**let** myTuple4 = (count: 10, length: 12.1, message: "Hi")

print(type(of:myTuple4)) *// 레이블이랑 자료형 같이 출력*

print(myTuple4.message)

*// 10*

*// Hi*

*// (count: Int, length: Double, message: String)*

*// Hi*

1. **옵셔널 타입만이 값을 갖지 않는다는 의미의 nil 값을 가질 수 있음**

*// 옵셔널*

*// Int형 값을 저장 or 값이 없음(nil)*

*// nil 앱 개발 하거나 값을 처리하는데 있어서 값이 없을 때를 처리하기 위해 고안된 자료형*

print(10)

print(Int("12"))

print(Int("hi"))

*// 10*

*// Optional(12)*

*// nil*

**본격 옵셔널 값 추출 방법들 : 옵셔널을 푸는 방법들**

***옵셔널 타입 강제 언래핑(forced unwrapping) 1***

**var** x : Int? *//옵셔널 정수형 변수 x 선언*

**var** y : Int = 0

x = 10

print(x) *// Optional(10)*

print(x!) *// forced unwrapping해서 10이 나옴*

print(y) *// 0*

*//x = x+2 //가능? X*

x = x! + 2 *// 옵셔널 값은 그냥연산 못함 풀어서 사용해야함 ! 사용하는forced unwrapping 사용 그리고 다시 옵셔널 변수에 할당*

print(x) *// Optional(12)*

*//y = x //가능? x*

y = x! *// 일반 변수에 넣을때도 값을 풀어서 넣어줘야함 일반 자료형 변수에*

*// 옵셔널값 그냥 못 넣음*

print(y)

*// Optional(10)*

*// 10*

*// 0*

*// Optional(12)*

*// 12* // 일반 변수에 넣는 거니 12 출력

**옵셔널 값을 풀 때 주의 할점!**

**var** x : Int? *//옵셔널 변수는 초기값을 안주면 nil이 들어간다*

*//x = 10 // 주석처리하면?*

print(x) // nil

*//print(x!) // nil 값이 들어간 옵셔널 변수를 강제로 풀면 Crash가 난다*

**스위프트에서 옵셔널을 풀때 가장 많이 쓰는 방법1**

**If-else 문과 *forced unwrapping***

**var** x : Int?

x = 10 // 값 안주면 nil 출력

**if** x != **nil** { *// 옵셔널은 닐이 아닐때만 풀어야 하기때문에 이렇게 if문에 의해서*

print(x!) *// nil이 아닌지 체크하고 풀어야함*

*// 주의 : if x!=nil 이라고 쓰면 안됨*

}

**else** {

print("nil")

}

**스위프트에서 옵셔널을 풀때 가장 많이 쓰는 방법2**

**옵셔널 바인딩(optional binding)을 이용하여 옵셔널에 할당된 값을 임시 변수 또는 상수에 할당**

**var** x : Int?

x = 10

**if** **let** xx = x { *//옵셔널 변수 x가 값(10)이 있으므로 언래핑해서 일반 상수 xx에 대입하고 if문 실행 xx는 이 블락 안에서만 사용 가능함*

print(x, xx) *// 키워드가 let이 아니고 var면 안에서 값을 가지고 놀겠다는 뜻임*

}

**else** {

print("nil")

}

**if** **let** x = x { *// 옵셔널 바인딩시 이렇게 같은 이름을 줘도 상관 없음*

**if** **let** x { *// if let x 이걸로 옵셔널 바인딩을 한줄에 끝내버림 최신 업데이트*

*//Optional(10) 10*

**여러 옵셔널을 언래핑**

//여러 옵셔널 변수를 한번에 언래핑하는 방법은 콤마 사용

+ let 키워드를 둘 다 붙여줘야함

**var** pet1: String?

**var** pet2: String?

pet1 = "cat"

pet2 = "dog"

**if** **let** firstPet = pet1, **let** secondPet = pet2 {

*// 스위프트에서 ,는 and 임 즉 두개다 참이여야함 하나라도 nill이면 else쪽으로감*

print(firstPet, secondPet)

} **else** {

print("nil")

}

*//cat dog*

즉 둘다 nil이 아니여야 참이 됨

텍스트, 스크린샷, 전자기기, 디스플레이이(가) 표시된 사진

자동 생성된 설명

+

**as로 upcasting : 자식인스턴스를 부모클래스로 변환**

**as! as?로 downcasting:부모인스턴스를 자식클래스로 변환**

**대부분의 Swift 연산자는 다른 언어들과 비슷하지만 그중 몇가지 특징을 살펴보겠다.**

**증감 연산자 유무**

* 1. x++ // x를 1 증가시킴, Swift 3에서 없어짐, x+=1
  2. x-- // x를 1 감소시킴, Swift 3에서 없어짐, x-=1

Swift에서는 증감 연산자가 Swift3 버전에서 사라졌다.

*// 범위 연산자 실습*

*// x...y*

*// x에서 시작하여 y로 끝나는 범위에 포함된 숫자*

*//One-Sided Ranges*

**let** names = ["A", "B", "C", "D"]

**for** name **in** names[2...] { *//[...2], [..<2] //과제: 실행 결과*

print(name)

} *// C*

*// D*

**for** name **in** names[...2] { // 0부터 끝깢; 3.. 3부터 끝까지임 반대 x

*// A*

*// B*

*// C*

**for** name **in** names[..<2] { *//x부터 시작하여 y가 포함되지 않는 모든 숫자*

*// A*

*// B*

*//닫힌 범위 연산자(closed range operator)*

*//x...y*

*//x에서 시작하여 y로 끝나는 범위에 포함된 숫자*

**(없어진 문법) 시험**

for var i = 0; i < 10; i+=1 { // for i in 0..<10 로 수정해야 함

print(i)

} //error: C-style for statement has been removed in Swift 3

**for** name **in** 1..<10 {

print(name)

}

1

.

9

**Nil-Coalescing Operator (Nil합병연산자) ?? 시험**

**let** defaultAge = 1

**var** age : Int?

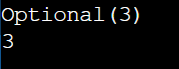
age = 3

print(age) *//과제:값은?*

**var** myAge = age ?? defaultAge *// 닐을 풀때 사용하는 연산자임 닐이라면 우쯕 값이 들어감*

*//age가 nil이 아니므로 언래핑된 값이 나옴*

print(myAge) *//과제: 값은?*



**장점 옵셔널 값을 따로 (• Optional Binding - 옵셔널 바인딩, • Force Unwrapping - 강제 추출)할 필요 없이 연산자로 한번에 풀 수 있음**

**let** defaultColor = "black"

**var** userDefinedColor: String? *// defaults to nil*

**var** myColor = userDefinedColor ?? defaultColor

*//nil이므로 defaultColor인 black으로 할당됨*

print(myColor) *//black*

userDefinedColor = "red"

myColor = userDefinedColor ?? defaultColor

*//nil이 아니므로 언래핑된 red가 할당됨*

print(myColor) *//red, 주의 optional(red)가 아님*

For-in 반복문 사용과 \_ 참조체

**for** **\_** **in** 1...5 {

print("안녕")

}

*// 안녕*

*// 안녕*

*// 안녕*

*// 안녕*

*// 안녕*

**반복문에서 빠져나오기(break) 주의**

**for** i **in** 1..<10 {

**if** i > 5 {**break**} *//if 조건 다음에 실행줄이 한줄이라도 중괄호 사용해 줘야함*

*//중요!! 에러 수정 : error: expected '{' after 'if' condition*

print(i)

}



**if문 조건에서 콤마의 의미(중요)**

**if** a < b && d > c {

print("yes")

}

**if** a < b, d > c { *//과제, if문에서 콤마는 AND(&&)의 의미*

print("yes")

}

**다중 if-else문 : BMI 계산 결과 판정**

**let** weight = 60.0

**let** height = 170.0

**let** bmi = weight / (height\*height\*0.0001) *// kg/m\*m*

**var** body = ""

**if** bmi >= 40 {

body = "3단계 비만"

} **else** **if** bmi >= 30 && bmi < 40 {

body = "2단계 비만"

} **else** **if** bmi >= 25 && bmi < 30 {

body = "1단계 비만"

} **else** **if** bmi >= 18.5 && bmi < 25 {

body = "정상"

} **else** {

body = "저체중"

}

print("BMI:**\(**bmi**)**, 판정:**\(**body**)**")



**guard문(조건식이 거짓이면 실행)**

**guard <불리언 표현식> else {**

**// 표현식이 거짓일 경우에 실행될 코드**

**<코드 블록을 빠져 나갈 구문>**

**}**

**// 표현식이 참일 경우에 실행되는 코드는 이곳에 위치**

guard문은 표현식이 거짓(false)으로 판단될 경우에 수행될 else 절을 반드시 포함해야 함

else 절에 속한 코드는 현재의 코드 흐름을 빠져 나갈 수 있는 구문(return, break, continue, throw 구문)을 반드시 포함해야 함

**guard~let의 활용**

guard는 return, break, continue, throw 등 제어문 전환 키워드를 쓸 수 있는 상황이라면 사용이 가능

**함수 안에서 if-let 과 guard~let 을 통한 바인딩**

**func** printName(firstName:String, lastName:String?){

**if** **let** lName = lastName { *// lastName이 nil이 아니면*

print(lName,firstName)

}

**func** printName(firstName:String, lastName:String?){

*// guard let : 함수안에서 guard let 을 통한 바인딩*

**guard** **let** lName = lastName **else** { *// lastName이 nil이면*

print("성이 없네요!")

**return** *// early exit*

}

print(lName,firstName)

}

printName(firstName: "길동", lastName:"홍")

printName(firstName: "길동", lastName:**nil**)

*// 홍 길동*

*// 성이 없네요!*

*// lName은 아래 블락에서도 사용가능함 굉장히 중요*

*// if let은 if 블락에서만 사용 가능하지만*

*// 가드 문은 아래 블락에서도 사용이 가능함*

**Switch-case 문**

**활용**

**let** someCharacter: Character = "z" *// + 케릭터형 정의시 : Character 생략 불가*

**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")

}

*주의*

**case** "a": *// Invalid, the case has an empty body*

*// 적어도 하나 이상의 실행 가능한 문장이 있어야 하는데 없어서 오류 남*

*// 뭔가 기능을 넣어 줘야함*

**case** "a", "A": *// 이와 같이 대문자 소문자 던 a 일 경우 아래 문장을 출력하라고 하면 오류 안남*

print("The letter A")

**case** 1,3,5,7,8,10,12: *// , 로 동일 조건(경우) 명시*

print("31 일입니다")

*// 31 일입니다*

**case** 0...9: // 0 부터 9를 포함 한 수 일 때 : 즉 한자리

count = "한자리 수"

**함수 만들수 있어야함**

**let** weight = 60.0

**let** height = 170.0

**let** bmi = weight / (height\*height\*0.0001) *// kg/m\*m*

**var** body = ""

**switch** (bmi) {

**case** 40 ... 1000: // case 조건에는 bmi >= 40 이렇게 true false 값이 들어가는게 아니라 범위가 들어가야함

body = "3단계 비만"

**case** 30 ..< 40:

body = "2단계 비만"

**case** 25 ..< 30:

body = "1단계 비만"

**case** 18.5 ..< 25:

body = "정상"

**default**:

body = "저체중"

}

print("BMI:**\(**bmi**)**, 판정:**\(**body**)**")

*// BMI:20.761245674740483, 판정:정상*

**+**

그리고 스위치 케이스문에 {}블럭 넣으면 오류 남 주의 !

**함수를 만들어 더블형 매개변수 2개에 값을 저장해**

**더블형 case별로 비교하여 결과를 도출 하는 코드**

**import** **Foundation**

**func** calcBMI (weight : Double, height : Double) { *//Void형*

**let** bmi = weight / (height\*height\*0.0001) *// kg/m\*m*

**let** shortenedBmi = String(format: "%.1f", bmi)

**switch** bmi {

**case** 0.0..<18.5:

print("BMI:**\(**shortenedBmi**)**,판정:저체중")

**case** 18.5..<25.0 :

print("BMI:**\(**shortenedBmi**)**,판정:정상")

**case** 25.0..<30.0 :

print("BMI:**\(**shortenedBmi**)**,판정:1단계 비만")

**case** 30.0..<40.0 :

print("BMI:**\(**shortenedBmi**)**,판정:2단계 비만")

**default** :

print("BMI:**\(**shortenedBmi**)**,판정:3단계 비만")

}

}

calcBMI(weight:62.5, height: 172.3)

**switch-case에서 where절 사용하기**

**case** 0...49 **where** temperature % 2 == 0:

*// where 절은 부가적인 조건을 주는것임*

*// 0부터 49 를 포함한 사이에서 2로 나눈 나머지가 0 즉 짝수이면 and 조건이라고 생각*

*// for문에서 사용 응용*

**var** numbers: [Int] = [1, 2, 3, 4, 5]

**for** num **in** numbers **where** num > 3 {

print(num)

}

*// 4*

*// 5*

**fallthrough**

C/C++, Objective-C와 달리 case문 다음에 break 문을 포함할 필요가 없음

Swift는 일치하는 case문을 만나면 자동으로 빠져나옴

case별로 빠져 나가지 않고 아래로 계속 내려가게 하려면 fallthrough 문 사용

**var** value = 4

**switch** (value)

{

**case** 4:

print("4")

**fallthrough**

**case** 3:

print("3")

**fallthrough**

**case** 2:

print("2")

**fallthrough**

**default**:

print("1")

}

*//빠져나오지 않고 쭉 진행하고 싶을 때 사용*

*//계속 내려 가는 것 임 fallthrough 연결되어 있는데 까지는*

*// 4*

*// 3*

*// 2*

*// 1*

**var** value = 4

**switch** (value)

**case** 4:

print("4")

**fallthrough**

**case** 3:

print("3")

**case** 2:

print("2")

**fallthrough**

**default**:

print("1")

}

*// 4*

*// 3*

함수

**용어정리**

1. 함수 정의부의 값을 매개변수, 호출시의 값은 아규먼트라고 부름

내 동전 : 전달인자: 아규먼트

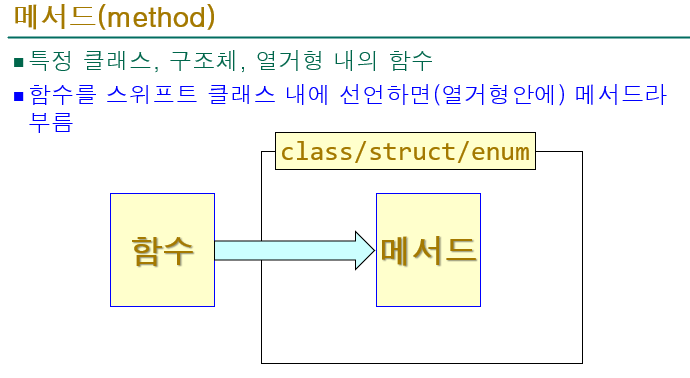
**자판기 안에서 동전 : 매개변수 파라메타**

void Fun( int Param )

*// parameter(매개변수, 인자), 형식 매개변수(formal parameter)*

Fun( 10 ); *// 10은 argument(인수), 실 매개변수(actual parameter)*

함수? 메서드란?



**함수의 타입**

**(자료형,자료형,…) -> 리턴형**

**(Int, Int) -> Int**

**리턴형이 Void형이면 ()**

Swift 함수 타입 확인

**func** add(x: Int, y: Int) -> Int {

**return**(x+y)

}

**func** sayHello() { *//->Void*

print("Hello")

}

**let** x = add(x:10, y:20) )*// 호출 할 때 이렇게 이름(매개변수)을 넣어야 함*

print(type(of:add)) *// 함수 타입 확인*

print(type(of:sayHello))

print(type(of:x))

print(x)

*// (Int, Int) -> Int //* 매개변수 타입과 리턴 타입이 출력됨

*// () -> ()*

*// Int*

*// 30*

매개변수도 없고 리턴 값도 없는 함수는 저렇게 나옴

**내부 매개변수(parameter name) 이름과 외부 매개변수(argument label) 이름**

**func** add(first x: Int, second y: Int) -> Int {

*//외부 내부:자료형,외부 내부:자료형 -> 리턴형*

**return**(x+y) *//함수 정의할 때는 내부 매개변수명을 사용*

} *//return(first+second)은 오류*

**let** x = add(first:10, second:20) *//add(x:10, y:20)은 오류*

*//함수 호출할 때는 외부 매개변수명을 사용*

print(x) *// 30*

*// 외부 매개변수 외부 호출 시 사용할 이름*

*// 즉 만들 때 외부 매개변수, 내부 매개변수, 자료형 이렇게 사용 함*

**func** add(x: Int, y: Int) -> Int {

*// add(x x: Int, y y: Int) 이렇게 쓴 거랑 똑같은 것임*

**return**(x+y)

}

**let** x = add(x:10, y:20)

*//외부 매개변수명 생략하면 내부 매개변수명이 외부 매개변수명까지 겸함*

print(x)

*// 30*

**4가지 함수 정의, 선언 방법 과 타입 출력**

*//4가지 함수 정의, 선언 방법 과 타입 출력*

**func** add1(x: Int, y: Int) -> Int {

print(**#function**) *// 함수명 얻을 수 있음*

**return**(x+y)

}

print(add1(x:10, y:20)) *// 내부 외부 같이 사용 하겠다.*

**func** add2(first x: Int, second y: Int) -> Int {

print(**#function**) *// 함수명 얻을 수 있음*

**return**(x+y)

}

print(add2(first:10, second:20))

**func** add3(**\_** x: Int, **\_** y: Int) -> Int {

print(**#function**) *// 함수명 얻을 수 있음*

**return**(x+y)

}

print(add3(10, 20)) *// 요렇게 사용하면 다른 언어처럼 값만 보낼 수 있음*

**func** add4(**\_** x: Int, with y: Int) -> Int {

print(**#function**) *// 함수명 얻을 수 있음*

**return**(x+y)

}

print(add4(10, with:20))

print(type(of:add1))

print(type(of:add2))

print(type(of:add3))

print(type(of:add4))

*//함수의 type은 모두 같고 함수명은 모두 다름*

*// add1(x:y:)*

*// 30*

*// add2(first:second:)*

*// 30*

*// add3(\_:\_:)*

*// 30*

*// add4(\_:with:)*

*// 30*

*// (Int, Int) -> Int*

*// (Int, Int) -> Int*

*// (Int, Int) -> Int*

*// (Int, Int) -> Int*

*// \_가 외부 매개 변수 이름이 됨*

*// : 개수 보면 매개변수 개수를 알 수 있음*

*//어떤 함수 안에서 #function 이렇게 하는 걸 expression 이라고 함*

*//함수의 이름을 알 수 있음*

*//함수의 이름은 add(외부 매개변수 명: 외부 매개변수명:)*

*// 어떻게 정의 하던 함수의 타입은 매개변수의 타입과 반환형 타입이 나옴*

*//4번째 방법*

*//오브젝트 c에서 함수 호출 방법과 닮아 있어서*

*//애플에서 만들어진 함수 보면 이런 식으로 만들어진 함수 굉장히 많음*

*//많이 사용됨*

*/\* 추가적으로*

*func add1(x: Int, y: Int?){ // (Int, Optional<Int>) -> () 옵셔널은 이렇게 나옴*

*}*

*print(type(of:add1))*

*\*/*

**함수로부터 여러 개의 결과 반환하기**

*//함수는 여러 결과 값들을 튜플로 감싸서 반환할 수 있음*

*//인치 단위의 수를 매개변수로 받아 야드, 센티미터, 미터로 변환하고 이 세 개의 값을 하나의 튜플에 넣어 반환*

**func** converter(length: Float) -> (yards: Float, centimeters: Float, meters: Float) {

**let** yards = length \* 0.0277778

**let** centimeters = length \* 2.54

**let** meters = length \* 0.0254

**return** (yards, centimeters, meters)

*// 리턴이 와야하는데 () 이거 보니까 튜플임*

*// 간단하게 튜플 같은거 이용하면 여러가지를 한번에 리턴 가능*

}

**var** lengthTuple = converter(length:10)

print(lengthTuple)

print(lengthTuple.yards)

print(lengthTuple.centimeters)

print(lengthTuple.meters)

*// (yards: 0.277778, centimeters: 25.4, meters: 0.254)*

*// 0.277778*

*// 25.4*

*// 0.254*

**import** **Foundation**

**func** **sss**(**x** : **Int**, **y** : **Int**) -> (**sum** : **Int**, **sub** : **Int**, **div** : **Double**, **mul** : **Int**, **remainder**:**Int** )

{

**let** div = Double(x)/Double(y) *//같은 자료형만 연산 가능 : 소수점 만들기 위해*

*//c 나 다른 언어에서는 서로 다른 자료형도 연산가능 인트형과 더블형 연산하면 더 큰 더블 형으로 처리되지만 Swift는 반드시 같은 자료형 연산 해야함*

**return** (sum, sub, div, mul, remainder)

}

print(type(of:sss))*// 함수 타입*

print(String(format: "%.3f", result.div)) *//import Foundation 추가해야함 사용하려면*

)

*// (Int, Int) -> (sum: Int, sub: Int, div: Double, mul: Int, remainder: Int)*

이전 수업 시간에도 이야기하셨던 것처럼 튜플은 레이블을 입력하면 반환 시 레이블이랑 같이 반환되기에 함수의 타입을 확인할 때도 같이 나옵니다.

디폴트 매개변수(아규면트) 정의

텍스트, 스크린샷, 모니터이(가) 표시된 사진

자동 생성된 설명

**가변 매개변수(variadic parameter)**

*// 함수가 가변 매개변수를 받는다는 것을 가리키기 위해서 세 개의 점(…)을 사용*

*// 지정되지 않은 개수의 String 값을 매개변수로 받아서 콘솔에 출력*

**func** displayStrings(strings: String...)

{

**for** string **in** strings {

print(string)

}

}

displayStrings(strings: "일", "이", "삼", "사")

displayStrings(strings: "one", "two")

*// 일*

*// 이*

*// 삼*

*// 사*

*// one*

*// two*

**func** add (numbers:Int ...){

**var** sum:Int = 0 // 초기화 필수

**for** num **in** numbers{

sum += num

}

print(sum)

}

add(numbers:1,2,3) *//6*

add(numbers:2,2,2,2,2) *//10*

add(numbers:1,1,1,1,1,1,1,1,1,1) *//10*

add(numbers:1,1,1,1) *//4*

텍스트이(가) 표시된 사진

자동 생성된 설명

*//Swift 3이후*

**var** myValue = 10

**func** doubleValue (value: **inout** Int) -> Int {

value += value

**return**(value)

}

print(myValue)

print(doubleValue(value : &myValue)) *//출력 값? 레포트*

print(myValue)

*// 10*

*// 20*

*// 20*