화살이(가) 표시된 사진

자동 생성된 설명

iOS 5주차 레포트

|  |  |
| --- | --- |
| 과목명 : iOS 프로그래밍 | 담당교수 : 한성현 교수님 |
| 학과 : 컴퓨터소프트웨어학과 | 학년 : 2학년 |
| 학번 : 201912047 | 이름 : 정정욱 |
| 제출일 : 2022.10.04 |  |

**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

guard문은 기본적으로 특정 조건에 맞지 않을 경우에 현재의 함수나 반복문에서 빠져 나갈 수 있도록 하는 ‘조기 출구(early exit)’ 전략을 제공

**guard~let의 활용**

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

그래서 함수 뿐 아니라 반복문 등 특정 블록 내부에 있으면 사용 가능

물론 함수 내부에 있다면 보통 return을 써서 해당 함수를 조기에 빠져나오는 조기 출구 용도로 사용

실제 앱을 만들다 보면 옵셔널 바인딩 때문에 다중 if~else를 사용해야 하는데, guard~let을 사용하면 다중 루프 없는 훨씬 가독성이 좋은 코드가 가능해서 그런 경우 많이 사용

*// 이렇게 guard문을 활용하지는 않음*

**var** x = 1

**while** **true** {

**guard** x < 5 **else** { **break** }*//조건(x<5)이 거짓일 때 실행(break)*

print(x) *//1 2 3 4, 조건(x<5)이 참일 때 실행*

x = x + 1

}

*// 1*

*// 2*

*// 3*

*// 4*

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

**func** printName(firstName:String, lastName:String?){ *// 스트링 옵셔널 값이 들어올수 있기에 바인딩 해줘야함*

*// if let : 함수안에서 if let을 통한 바인딩*

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

print(lName,firstName)

}

**else** {

print("성이 없네요!")

}

}

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

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

*// 홍 길동*

*// 성이 없네요!*

*-------------------------------------------------------*

**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*

*// 조건이 안맞으면 바로 끝내버림 아래 print 안감 : 조기 출구(early exit)’ 전략*

*텍스트이(가) 표시된 사진

자동 생성된 설명*

실제로 앱 만들 때 guard let 좀더 많이 사용함

(옵셔널 바인딩 이용 등) (가독성이 올라가서)

**Switch-case 문**

**기본 틀**

**switch** 표현식

{

**case** match1:

구문 *// 주의 : 각 case문 마지막에 break가 자동으로 들어 있음*

**case** match2:

구문

**case** match3, match4:

구문

**default**:

구문

}

**활용**

**var** value = 0

**switch** value

{

**case** 0:

print("영") *// 조건에 부합하면 기본적으로 break 가 있어서 탈출*

**case** 1:

print("일")

**case** 2:

print("이")

**case** 3:

print("삼")

**default**:

print("4이상")

}

*// 영*

**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")

}

*// Prints "The last letter of the alphabet*

**사용시 주의 사항 !! + 여러가지 경우에 대하여 동일 코드 실행 방법**

**let** anotherCharacter: Character = "a"

**switch** anotherCharacter {

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

**case** "A":

print("The letter A")

**default**:

print("Not the letter A")

}

*//error: 'case' label in a 'switch' should have at least one executable statement*

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

*// 뭔가 기능을 넣어 줘야함*

아래와 같이 사용하여 오류 해결

*// 여러 가지 다른 경우에 대해서 동일한 코드를 실행해야 할 경우*

*// 실행될 공통 코드로 그룹 case 문을 만들 수 있음*

**let** anotherCharacter: Character = "a"

**switch** anotherCharacter {

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

print("The letter A")

**default**:

print("Not the letter A")

}

*//"The letter A"*

**var** value = 3

**var** days : Int = 0

**switch**(value)

{

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

print("31 일입니다")

**case** 4,6,9,11:

print("30 일입니다")

**case** 2:

print("28 or 29 일입니다")

**default**:

print("월을 잘못 입력하셨습니다")

}

*// 31 일입니다*

**let** num = 25

**let** count : String

**switch** num {

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

count = "한자리 수"

**case** 10...99: // 10부터 99를 포함 한 수 일 때 : 두 자리 수

count = "두자리 수"

**case** 100...999:

count = "세자리 수"

**default**:

count = "네자리 수 이상"

}

print("**\(**count**)**입니다.")

*// 두 자리 수입니다.*

**다중 if-else 문 : BMI계산 결과 판정을 switch-case 문으로 변경하기**

**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**)**")

*//BMI:20.761245674740483, 판정:정상*

**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**

* where절을 switch case에 부가적인 조건을 추가하기 위하여 사용
* 값이 속하는 범위 뿐만 아니라 그 숫자가 홀수인지 짝수인지도 검사

If 문 대신하여 조건을 줄 때 사용하는 것임

**var** temperature = 60

**switch** (temperature)

{

**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*

print("Cold and even")

**case** 50...79 **where** temperature % 2 == 0:

print("Warm and even")

**case** 80...110 **where** temperature % 2 == 0:

print("Hot and even")

**default**:

print("Temperature out of range or odd")

}

*// Warm and even*

*// where*

*// 특정 패턴과 결합하여 조건을 추가*

*// for문에서 사용 응용*

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

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

print(num)

}

*// 4*

*// 5*

추가 응용 찾아본 자료

*//옵셔널 패턴과 결합*

*//where 절은 옵셔널 패턴과도 결합 가능하다.*

**let** arr: [Int?] = [1, 2, 3, 4, 5, 6, 7, 8]

*// 옵셔널 패턴과 결합*

**for** **case** **let** i? **in** arr **where** i % 2 == 0 {

print(i)

}

*/\**

*2*

*4*

*6*

*8*

*\*/*

*추가 참고 문헌*

*where 절의 활용*

*패턴과 결합하여 조건 추가*

*타입에 대한 제약 추가*

<https://seons-dev.tistory.com/entry/Swift-%EA%B3%A0%EA%B8%89-%EB%AC%B8%EB%B2%95-where-%EC%A0%88-%ED%8A%B9%EC%A0%95-%ED%8C%A8%ED%84%B4%EA%B3%BC-%EA%B2%B0%ED%95%A9>

**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*

*// 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*

**Swift 함수 선언, 호출 방법과 다른 언어와의 차이점**

**용어정리**

* 함수 정의부의 값을 매개변수, 호출시의 값은 아규먼트라고 부름

#include <stdio.h>

void Fun( int Param ) *// parameter(매개변수, 인자), 형식 매개변수(formal parameter)*

{

   printf("%d",Param); *//Param 가 x 이면 x 라고 쓰면 되니까 (형식적) 형식 매개 변수라고 하는 것*

}

int main()

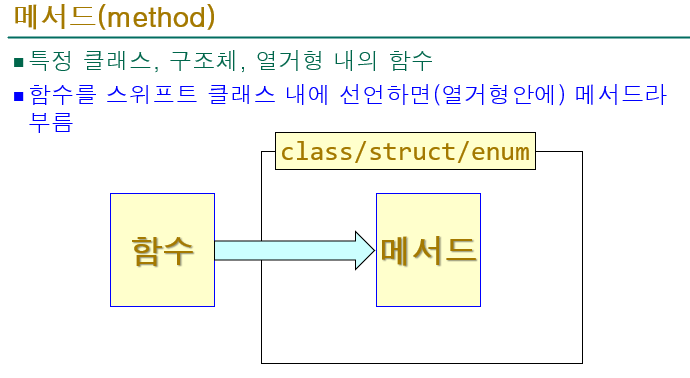
{

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

**return** 0;

}

함수? 메서드란?



**함수를 선언하는 방법**

**func <함수명> (<매개변수 이름>: <매개변수 타입>, <매개변수 이름>: <매개변수 타입>,... ) -> <반환값 타입> {**

**// 함수 코드**

**}**

*//매개변수를 받지 않으며 결과를 반환하지도 않고 오직 메시지만 출력*

**func** sayHello() { *//리턴값 없으면( -> Void ) 지정하지 않아도 됨 즉 생략 가능*

print("Hello") *// ; 생략 가능함 Swift는*

}

Void sayHello() { *//C, C++ 에서 사용 하는 것과 차이가 있음*

printf(“Hello”); *// ; 꼭 해줘야 함*

}

*//하나의 문자열과 하나의 정수를 매개변수로 받아서 문자열을 반환*

Func message(name: String, age: Int) -> String { *// 특징 매개변수 가 먼저 타입이 뒤에 나옴*

**return**(“\(name) \(age)”) *// -> 다음이 리턴 타입임*

}

함수 정의와 선언

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

print("Hello")

}

sayHello() *//호출*

*// 매개변수 없을 때는 다른 언어와 사용 하는게 비슷함*

c언어에서 함수 정의, 호출 시

int add(int x, int y) { *//C, C++*

**return**(x+y);

}

add(10,20);

Swift에서 사용시

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

**return**(x+y)

}

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

print(x)

*//main.swift:5:12: error: missing argument labels 'x:y:' in call*

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

**return**(x+y)

}

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

print(x)

*// 30*

*// 코틀린도 이렇게 사용함 -> 말고 : 찍으면 코틀린 문법임*

*// Swift 문법을 잘하면 코틀린을 잘 다룰 수 있음 정말 유사한 부분이 많음*

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)

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

이 함수는 매개변수도 없고 리턴 값도 없다 알 수 있음

**응용**

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

**return**"**\(**x**)** 와 **\(**y**)** 값을 넣은 사람은 **\(**z**)**"

}

print(add1(x:10, y:20, z:"정정욱"))

print(type(of:add1))

*// 10 와 20 값을 넣은 사람은 정정욱*

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

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

텍스트이(가) 표시된 사진

자동 생성된 설명

**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*

텍스트이(가) 표시된 사진

자동 생성된 설명

**4가지 함수 정의, 선언 방법 과 타입 출력**

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

**return**(x+y)

}

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

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

**return**(x+y)

}

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

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

**return**(x+y)

}

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

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

**return**(x+y)

}

print(add4(10, with:20))

print(type(of:add1))

print(type(of:add2))

print(type(of:add3))

print(type(of:add4))

*// 30*

*// 30*

*// 30*

*// 30*

*// (Int, Int) -> Int 어떻게 정의 하던 함수의 타입은 매개변수의 타입과 반환형 타입이 나옴*

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

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

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

*//4번째 방법*

*//오브젝트 c에서 함수 호출 방법과 닮아 있어서*

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

*//많이 사용됨*

**함수의 type은 모두 같고 함수명은 모두 다름**

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

print(**#function**)

**return**(x+y)

}

print(add(x:10, y:20))

print(type(of:add)) *// 함수 타입은 모두 같고 함수 명은 모두 다름*

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

print(**#function**)

**return**(x+y)

}

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

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

print(**#function**)

**return**(x+y)

}

print(add(10, 20))

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

print(**#function**)

**return**(x+y)

}

print(add(10, with:20))

*// add(x:y:)*

*// 30*

*// add(first:second:)*

*// 30*

*// add(\_:\_:)*

*// 30*

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

*// 30*

**--------------------------------------------**

*//함수이름 다른 언어에서 그냥 add라고함*

*//Swift에서 함수는 그냥 함수만 부르지 않고*

*//매개변수와 같이 해서 부름 같은 이름 함수 여러 개 올 수 있고*

*//매개변수에 따라 다른 함수임*

*//c언어에서 같은 이름 함수 여러 개 면 오류 나는데*

*//스위프트 함수 이름 중첩 가능 하기 때문에 오류 안 남*

*// \_가 외부 매개 변수 이름이 됨*

*// : 개수 보면 매개변수 개수를 알 수 있음*

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

print(**#function**) *//add(first:second:) 이게 함수 이름임*

**return**(x+y)

}

**let** a = add(first:10, second:20)

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

*//함수의 이름을 알 수 있음*

*//함수의 이름은 add(외부 매개변수 명: 외부 매개변수명:)*

텍스트이(가) 표시된 사진

자동 생성된 설명

<https://minmong.tistory.com/134> 참고 문헌

테이블이(가) 표시된 사진

자동 생성된 설명

Swift로 함수를 만든 다면

**func** ddd (x: Int) -> Int {

**return** x \* 2

}

fun setX (x:Int) -> void {

xx = x

}

fun getX() -> Int {

**return** x;

}

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

**return**(x+y)

}

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

xx=x;

yy=y;

}

이렇게 함수명을 아는 게 중요한 이유는 ex 앱 개발을 하는데 있어서 함수 명 다 tableView (함수 명 중첩) 임 거기서 외부 매개변수 이름이 다른 것과 같이 검색을 해야 이 함수가 어떤 역할을 하는지 검색 가능 하고 자료를 찾을 수 있기 때문임

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

*// 함수명 : tableView(\_:numberOfRowsInSection:)*

*// 자료형 : (UITableView, Int) -> Int*

*// 함수명 : tableView(\_:cellForRowAt:)*

*// 자료형 : (UITableView, IndexPath) -> UITableViewCell*

텍스트이(가) 표시된 사진

자동 생성된 설명

*// 함수명 : changeDatePicker(\_:)*

*// 자료형 : (UIDatePicker) -> ()*

텍스트이(가) 표시된 사진

자동 생성된 설명

*// 함수명 : pickerView(\_:rowHeightForComponent:)*

*// 자료형 : (UIPickerView, Int) -> CGFloat*

*// 함수명 : pickerView(\_:viewForRow:forcomponent:reusing:)*

*// 자료형 : (UIPickerView, Int , Int, Optional<UIView>) -> UIView*

텍스트이(가) 표시된 사진

자동 생성된 설명

*// 함수명 : pickerView(\_:didSelectRow:inComponent:)*

*// 자료형 : (UIPickerView, Int , Int) -> ()*

*/\* 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*

*// 2개의 정수를 입력 받아 가감승제,나머지 리턴*

*//과제: swift print format으로 구글링하여 소수점 3자리에서 반올림*

*//과제 : 이 함수의 자료형은?*

**import** **Foundation**

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

{

print(**#function**) *// 함수 이름*

**let** sum = x+y

**let** sub = x-y

**let** mul = x\*y

**let** remainder = x%y

**let** div = Double(x)/Double(y) *//같은 자료형만 연산 가능*

*//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*

**var** result = sss(x:10,y:3)

print(result.sum)

print(result.sub)

print(result.div)

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

print(result.mul)

print(result.remainder)

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

*// sss(x:y:)*

*// 13*

*// 7*

*// 3.3333333333333335*

*// 3.333*

*// 30*

*// 1*

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

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

print(myTuple.0)

print(type(of:myTuple)) *// 자료형은 튜플안에 선언 한 값의 자료형과*

*//튜플 자료형을 나타내는 ()로 감쌓아 나옴*

*// let (myInt, MyFloat, myString) = myTuple*

*// var (myInt, \_, myString) = myTuple //부동 소수점 수는 무시*

*//튜플을 생성할 때 각 값에 이름을 할당할 수도 있음 (레이블이라고 함 )*

**let** myTuple1 = (count: 10, length: 12.1, message: "Hi") *//과제 : myTuple의 자료형*

print(myTuple1)

print(type(of:myTuple1))*// 레이블이랑 같이 자료형이 출력됨*

print(myTuple1.message)

*// 즉 인덱스 번호를 사용할 수 있지만 값 별로 이름을 지정해서 이름을 불러 사용할 수 있는 것 임*

*// 10*

*// (Int, Double, String)*

*// (count: 10, length: 12.1, message: "Hi")*

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

*// Hi*

(이전 시간의 소스 자료입니다.)

**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*

*// 주소를 넘기는 것임*

*// c랑 똑같은 & 주소 넘기는 건 똑같은데*

*// 자료형 앞에 inout 넣어주기*

*// 그리고 이렇게 콜 바이 레퍼런스로 바꾼 자료형은 호출 이후에도 값 유지가 됨 주소안에 값을 바꿔 주었기 때문임*

추가 블로그 정리 글입니다.

<https://aloe-study.tistory.com/162>