

# 7주차 (4/15~4/21)



2024-04-17 (수) 오프라인 미팅 15:00 ~ 19:00

2024-04-18 (목) 오프라인 미팅 16:00 ~ 19:00

총 활동 시간: 7시간

(1) 각자 코틀린 기본 개념 공부 및 정리

## 정리본

### ▼ 변수

## 1. 변수 선언하기

코틀린에서 변수는 **val**과 **var** 키워드를 사용하여 선언한다.

### 1. val 키워드

- **val** : value의 줄임말로 초깃값이 할당되면 바꿀 수 없는 변수, 즉 상수를 선언할 때 사용한다.

⇒ 선언 이후 값을 변경하려 하면 컴파일 오류가 발생한다.



### <변수 선언 방식>

val 변수명 : 타입 = 값

### 2. var 키워드

- var : variable의 줄임말로 초깃값이 할당된 후에도 값을 바꿀 수 있는 변수를 선언할 때 사용한다.



#### <변수 선언 방식>

```
var 변수명 : 타입 = 값
```

이 때, 변수를 초기화(만들 + 대입) 한다면, 들어오는 데이터를 보고 자동으로 데이터 타입을 추론한다. 이 경우 타입을 지정해주 않고 아래와 같이 선언할 수 있다.



```
val(var) 변수명 = 데이터
```

#### <추가 내용>

- 최상위에 선언한 변수나 클래스의 멤버변수는 선언과 동시에 초깃값을 할당해야 하며, 함수 내부에 선언한 변수는 선언과 동시에 초깃값을 할당하지 않아도 된다.

## 2. 변수의 초기화 미루기

변수의 초기화를 미룰 때에는 `lateinit` 과 `lazy` 를 사용한다.

단 조건이 있다.

- var 키워드로 선언한 변수에만 사용할 수 있다.
- int, Long, Short, Double, Float, Boolean, Byte타입에는 사용할 수 없다.

#### <lateinit 키워드>



lateinit var data1: String

### <lazy 키워드>

변수 선언문 뒤에 `by lazy{ }` 형식으로 선언하며, 소스에서 변수가 최초로 이용되는 순간 `{ }` 안 코드가 자동 실행되어 결과값이 변수의 초깃값으로 할당된다.

코드가 여러 줄인 경우 : 마지막 줄의 실행결과가 변수의 초깃값이 됨



lateinit var data1: String

### <코드 예시>

```
lateinit var data1 : String

val data2: Int by lazy{
    println("in lazy...")
    10
}

fun main(){

    println("in main...")
    println(data2 + 10)

}
```

## 3. 데이터 타입

코틀린의 모든 변수는 **객체**이다. 따라서 코틀린의 모든 데이터 타입은 기초 데이터 타입이 아닌 클래스이다.

우리가 알고 있는 데이터 타입의 첫 글자를 대문자로 쓰면 된다!

## 1. 기초 타입 객체

**Int, Short, Long, Double, Float, Byte, Boolean**

## 2. 문자와 문자열

**Char, String**

- Char : 작은 따옴표로 감싸서 표현
- String 큰 따옴표 혹은 삼중 따옴표로 감싸서 표현

### <문자열 템플릿>

**\$** : String 타입의 데이터에 변수값, 연산식의 결과값을 포함해야 하는 경우 사용한다.

```
//ex)

package com.metacoding.user

import java.util.*

fun main(){

    var str1 = "안녕하세요"
    println("Hello World")
    println("str1 : $str1")

}
```

## 3. Any - 모든 타입 가능

- 최상위 클래스이다. 따라서 Any타입의 변수에는 어떤 형태의 데이터라도 저장할 수 있다.

## 4. 널 허용과 불허용

코틀린에서 모든 변수는 객체 타입이므로 `null` 을 대입할 수 있다. `null`은 값이 할당되지 않은 상황을 의미하며, 변수를 선언할 때 `null`을 허용할 것인지 안할 것인지 구분해서 선언해야 한다.

?이 있으면 - 허용

?이 없으면 - 불허용



```
var data1: Int = 10 //널 불허용
```

```
var data2 : Int? = 10//널 허용
```

### ▼ 함수

## 1. 함수 선언하기

<키워드> : `fun`

코틀린에서 함수를 선언할 때에는 `fun` 라는 키워드를 사용한다.

반환타입을 선언할 수 있으며, 생략하는 경우 자동으로 `unit` 타입이 적용된다.

### 1. 함수 선언 형식



```
fun 함수명 (매개 변수명 : 타입) : 반환타입 { ... }
```

### 2. 반환 타입이 있는 함수 선언



```
fun some( data1 : Int ) : Int{
    return data1 * 10
}
```

### <참고사항>

- 함수의 매개 변수에는 var이나 val 키워드를 사용할 수 없다.
- 매개 변수에는 자동으로 **val**이 적용되며, 따라서 함수 안에서 **매개변수 값을 변경할 수 없다.**

### 3. 함수의 매개변수에 기본값 선언하기



```
fun some( data1 : Int, data2 : Int = 10 ) : Int{
    return data1 * data2
}

println( some( 10 ) )//data2 : 기본값 사용
println( some( 10, 20 ) )//기본값이 아닌 20 사용
```

매개변수에 기본값을 선언하면, 호출할 때 인자를 전달하지 않아도 된다.

이 때 선언문에 명시한 기본값이 적용된다.

### 4. 명명된 매개변수

함수의 매개변수가 여러 개면 호출할 때 전달한 인자를 순서대로 할당한다. 이 때 매개변수명을 지정하면 순서를 바꿔도 된다.



```
some(data2 = 20, data1 = 10)
```

⇒ 원래는 순서대로 data1 = 20, data2 = 10이 들어갔어야 함

## ▼ 컬렉션 타입

**컬렉션 타입의 정의 :** 여러 개의 데이터를 표현하는 방법

**종류:** Array, List, Set, Map

# 1. Array

## 1. 배열 선언 방법



```
val 배열의 이름 : Array<배열의 타입> = Array( 배열의 크기, 초깃값 )
```

## 2. 배열의 데이터에 접근하는 방법

배열의 데이터에 접근하는 방법은 세 가지가 있다.



```
(배열의 이름)[ 0 ] = 10 //배열의 0번째 요소에 10을 대입
```

```
(배열의 이름).set( 2, 30 ) //배열의 2번째 요소에 30 대입
```

```
(배열의 이름).get( 2, 30 ) //배열의 2번째 요소 갖고 오기
```

### 3. 배열을 선언하며 값 할당하기

`arrayOf` 라는 함수를 사용하면 배열을 선언할 때 값을 할당할 수 있다.



```
val 배열의 이름 : arrayOf < Int > ( 10, 20, 30 )
```

## 2. List, Set, Map

:인터페이스를 타입으로 표현한 클래스이며 통틀어 **컬렉션 타입 클래스**라고 한다.

- **List** : 순서가 있는 데이터의 집합. 데이터의 중복을 허용한다.
- **Set** : 순서가 없는 데이터의 집합. 중복을 허용하지 않는다.
- **Map** : 키와 값으로 이루어진 데이터 집합. 순서가 없으며 키의 중복은 허용하지 않는다.

### 1. List

List객체는 아래와 같은 방법으로 만들 수 있다. 이 때 어떤 데이터를 저장할지 제네릭 타입을 지정해야 한다.



```
var 집합이름 = ListOf<값 타입>( 초기값 )
```

List는 불변 타입이므로 size와 get함수만 제공하며, 데이터를 추가하거나 변경할 수 없다. 또한 list함수 대신 **list[ 0 ]**과 같이 배열처럼 접근할 수 있다.

### 2. MutableList



MutableList객체는 아래와 같은 방법으로 만들 수 있다. 이 때 어떤 데이터를 저장할지 제네릭 타입을 지정해야 한다.



```
var 집합이름 = MutableListOf<값 타입>( 초기값 )
```

MutableList는 가변 타입이므로 size와 get함수 외에도 add함수와 set함수를 제공한다.

### 3. map

map객체는 아래와 같은 방법으로 만들 수 있다. 이 때 키로 어떤 타입의 값을 쓸지, 어떤 데이터를 저장할지 제네릭 타입을 지정해야 한다.



```
var 집합이름 = mapOf<키 타입, 값 타입>( pair(키, 값), 키 to 값 )
```

map은 불변타입이므로 **size( )**함수와 **get( )**함수를 제공한다.

```
package com.metacoding.user

import java.util.*

fun main(){

    var map = mapOf<Int, String>(Pair(0,"first"),1 to "second",2 to "third")

    println(
        """
            0: ${map.get(0)}
            1: ${map.get(1)}
            2: ${map.get(2)}
        """)
}
```

```

        """.trimIndent()

    )

}

```

## ▼ 조건문

### 1. if ~ else문

#### <기본 형태>

```

fun main(){

    if(data > 0){

        println("data > 0")

    }else{

        println("data <= 0")

    }

}

```

if문을 만족 → if문 실행

if문을 만족 x → else부분 실행

#### <else if 추가 형태>

```

fun main(){

    if(data > 10){//10보다 큰 경우

```

```

        println("data > 10")

    }else if(data > 0){//10보다 작은 것중에 0보다 큰 경우

        println("10 >= data > 0")

    }else{//0보다도 작은 경우

        println("data <= 0")

    }
}

```

### <표현식>

- 표현식 : 결과값을 반환하는 계산식

아래 코드는 if문의 결과값을 result라는 변수에 초기화해준 예이다.

if - else문을 표현식으로 사용하기 위해선 항상 else문이 있어야 한다!

```

package com.metacoding.user

import java.util.*

fun main(){
    val data = -1

    val result = if(data >= 0){//10보다 큰 경우

        println("data >= 0")
        true//참일 때의 반환값

    }else{//0보다도 작은 경우

        println("data < 0")
    }
}

```

```

        false//거짓일 때의 반환값

    }
    println(result)
}

```

## 2. when

: 스위치 문과 동일하다.

when( ) 소괄호 안에 넣은 데이터의 값에 따라 각 구문을 실행한다. 이 때 소괄호 안에  
는 Int뿐 아니라 다른 타입의 데이터도들어갈 수 있다.

```

package com.metacoding.user

import java.util.*

fun main(){
    var data =10
    when(data){

        10 -> println("data is 10")//data가 10이면 실행
        20 -> println("data is 20")//data가 20이면 실행
        else ->{

            println("data is not valid data")//10도 20도 아님

        }
    }
}

```

### <다양한 유형의 조건>

- is 데이터 타입 : when에 들어온 데이터의 타입 확인 연산자
- a, b : a또는 b인지 확인하는 연산자

- in a .. b : a와 b 사이의 값인지

```
package com.metacoding.user

import java.util.*

fun main(){
    var data: Any = 10
    when(data){

        is String -> println("문자열 타입입니다")//data가 문자열
        20, 30 -> println("20또는 30입니다")//data가 20또는 30
        in 40..50 -> println("40에서 50 사이입니다")//data가 1
        else ->{

            println("data is not valid data")
        }
    }
}
```

## ▼ 반복문

### 1. for

#### <형태>

- 범위 연산자 : in
- for(i in 1 **until** 10) : 1부터 10보다 작을 때까지
- for(i in 1..10 **step** 2) : 1부터 10까지 2씩 증가
- for(i in 10..1 **downTo** 1) : 10부터 1까지 1씩 감소

```
fun main(){

    var sum = 0
```

```
    for(i in 1..10){  
        sum += i  
    }  
    println(sum)  
}
```

## 2. While문

```
var x = 0  
var sum = 0  
  
while( x < 10 ){  
  
    sum += ++x  
  
}
```

### (2) 앱 - 로그인

```

class LoginActivity : AppCompatActivity() {
    lateinit var binding: ActivityLogInBinding
    var username: String = ""
    var password: String = ""

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_log_in)

        binding = ActivityLogInBinding.inflate(layoutInflater)
        setContentView(binding.root)

        //id가 입력된 경우
        binding.idInputEditText.doAfterTextChanged {...}

        //pw가 입력된 경우
        binding.pwInputEditText.doAfterTextChanged {...}

        binding.signInBtn.setOnClickListener {...}
    }
}

```

```

        binding.signInBtn.setOnClickListener {...}
        //1. retrofit 만들기
        val retrofit = Retrofit.Builder()
            .baseUrl("https://book-service.inuappcenter.kr/")
            .addConverterFactory(GsonConverterFactory.create())
            .build()

        //2. retrofit 서비스 등록하기
        val retrofitService = retrofit.create(RetrofitService::class.java)

        binding.logInBtn.setOnClickListener {...}
    }
}

```

### (3) 라즈베리파이 SD 카드에 OS 설치



## 라즈비안 OS 설치

1. 라즈베리파이 이미저 다운로드 및 설치
2. 라즈베리파이 모델 선택
  - a. 장치선택 - 라즈베리파이 5
3. 라즈베리파이 운영체제 선택
4. 스토리지 선택
5. 이미저의 고급 옵션 사용하기
6. SD카드에 라즈베리파이 OS 쓰기
7. 라즈베리파이 부팅하기