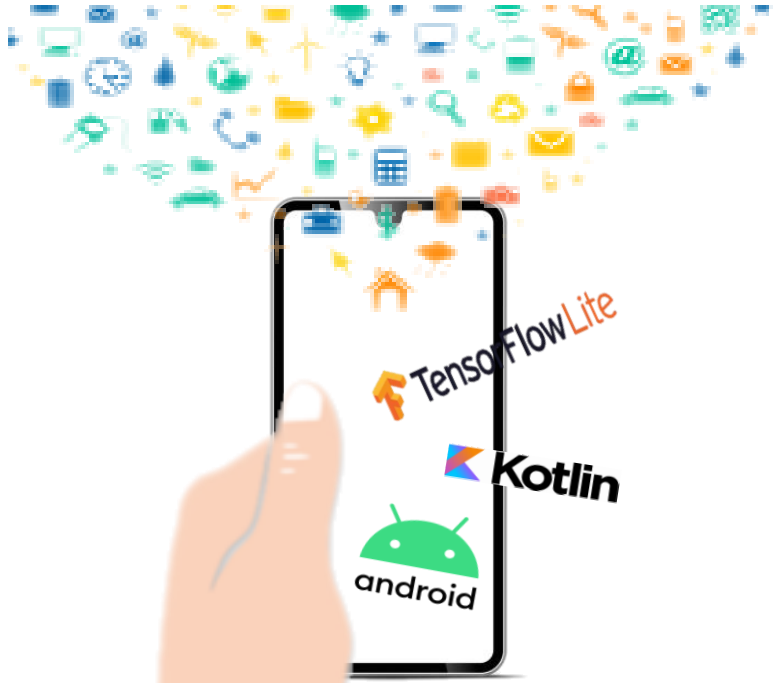
Android 기초 프로그래밍
- Kotlin 기초 문법 및 객체지향 프로그래밍

변수, 함수, 배열, 조건문, 반복문, 클래스



부산대학교 정보·의생명공학대학
정보컴퓨터공학부

이론

강의 목표와 구성

❖ Kotlin 기초 문법

- Kotlin 개요
- 변수와 함수
- 컬렉션 데이터 타입(배열, 리스트)
- 조건문
- 반복문

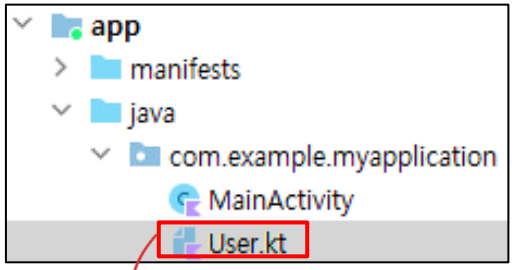
❖ Kotlin 객체지향 프로그래밍

- 클래스와 생성자
- 클래스 상속

Kotlin 개요

❖ Kotlin 이란?

- JVM(Java Virtual Machine)에서 동작하는 프로그래밍 언어
- 2017년 구글에서 **Android 공식언어**로 지정
- **객체지향**과 **함수형 프로그래밍** 스타일을 지원



code path

❖ Kotlin의 주요 특징: 안정성

- 앱 실행에 대한 **안정성**이 우수함
- 앱 비정상 종료의 원인이 되는 **Null Pointer Exception 를 완화**하기 위한 **Null Safe를 지원**
  - **Null Safe란**? : 객체의 널 상태를 컴파일러가 자동으로 해결하여 안정성을 확보
- 안정성을 기반으로, Android 앱의 **알고리즘 파트**를 구현

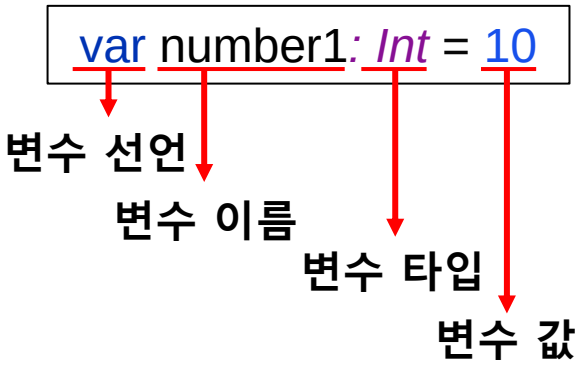
```kotlin
package com.example.myapplication

class FirstClass constructor() {
        fun print(){
                println("Hello world")
        }
}

fun main(args : Array<String>){
        var first_object : FirstClass = FirstClass()
        first_object.print()
}
```

```
"C:\Program Files\Android\Android
Hello world

Process finished with exit code 0
```

변수와 함수

❖ 변수의 선언

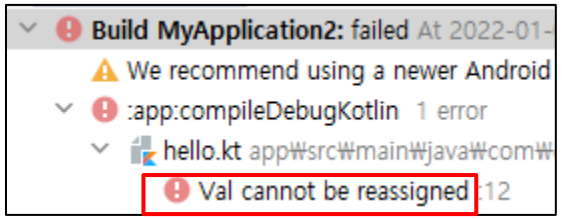
- 변수의 선언 방식은 **두 가지**가 존재함
  - **Var, Val** 로 변수를 선언 및 초기화




```
package com.example.myapplication

fun plus_number(number1: Int,number2: Int): String{
    var sum = (number1+number2).toString()
    return "number1과 number2를 더하면? : "+sum
}
fun main(){
    var number1: Int = 10
    val number2: Int = 10

    number1 = 20        → OK!
    number2 = 20        → Error!

    println(plus_number(number1,number2))
}
```

❖ Var과 Val의 차이점

- Val과 Var은 변수의 시작을 알리면서 변수가 **불변**인지 **가변**인지를 나타낸다.
- Val은 **불변** 타입 변수를 지정하며 초기에 값이 할당되면 나중에 값을 변경할 수 없다.
- Var은 **가변** 타입 변수를 지정하며 초기에 값이 할당되어도 나중에 값 변경이 가능하다.



변수와 함수

❖ 함수의 선언

- **예약어 fun** 으로 함수를 선언
- **파라미터 타입**과 **리턴 타입**의 설정이 필요함

```
fun plus_number(number1: Int,number2: Int): String{
```

함수 이름 / 파라미터 설정 / 리턴 타입

```
return "number1과 number2를 더하면? : "+sum
```

리턴 값

```
package com.example.myapplication

fun plus_number(number1: Int,number2: Int): String{
        var sum = (number1+number2).toString()
        return "number1과 number2를 더하면? : "+sum
}
fun main(){
        var number1: Int = 10
        var number2: Int = 20
        number1 = 20
        number2 = 20
        println(plus_number(number1,number2))
}
```

Modified!

OK!

❖ 함수 선언의 간소화

- 파라미터가 존재하지 않는다면, **파라미터 설정 부분 생략이 가능함**
- 리턴 값이 존재하지 않는다면, **리턴 과 리턴 타입 부분 생략이 가능함**

```
"C:\Program Files\Android\Android
number1과 number2를 더하면? : 40

Process finished with exit code 0
```

컬렉션 데이터 타입(배열, 리스트)

❖ 컬렉션 데이터 타입 이란?

- **연관된 데이터**를 하나의 변수로 관리하는 방법
- **종류**: Array, List, Set, Map
- **세 가지 단계**로 활용함
  - 선언과 초기화 -> 사용(값 참조, 출력) -> 수정(추가, 변경, 삭제)

❖ Array 타입

- **plus() 함수**를 이용하여 선언한 배열에 값을 추가함
- **선언과 초기화**: Array 클래스로 표현

```
val data1: Array<Int> = arrayOf(10, 20, 30)
```

배열 이름 / 정수 배열 / 초기화

❖ List 타입

- 순서가 있는 데이터 집합으로 데이터의 중복을 허용함
- 읽기 전용인 **List 클래스**와 수정가능한 **MutableList 클래스**가 있음
- **MutableList**를 사용하면 **add() 함수**를 사용하여 값을 추가함

선언과 초기화

요소 추가

출력

```
package com.example.myapplication
import java.util.*
fun main(){
    println("======= 배열 =========")
    var list1: Array<Int> = arrayOf(1,2,3,4,5,6,7,8)
    var list2: List<Int> = listOf(1,2,3,4,5,6,7,8)
    var list3: MutableList<Int> = mutableListOf(1,2,3,4,5,6,7,8)

    list1 = list1.plus( element: 9)
    list2 = list2.plus( element: 9)
    list3.add(9)

    println("list1 : ${Arrays.toString(list1)}")
    println("list2 : $list2")
    println("list2 : $list3")

    val filtered_list1 = list1.filter{it%3==0}
    val filtered_list2 = list2.filter{it%3==0}
    val filtered_list3 = list3.filter{it%3==0}

    println("list1의 3의 배수 출력 : $filtered_list1")
    println("list2의 3의 배수 출력 : $filtered_list2")
    println("list3의 3의 배수 출력 : $filtered_list3")
}
```

```
"C:\Program Files\Android\Android St
======= 배열 =========
list1 : [1, 2, 3, 4, 5, 6, 7, 8, 9]
list2 : [1, 2, 3, 4, 5, 6, 7, 8, 9]
list2 : [1, 2, 3, 4, 5, 6, 7, 8, 9]
list1의 3의 배수 출력 : [3, 6, 9]
list2의 3의 배수 출력 : [3, 6, 9]
list3의 3의 배수 출력 : [3, 6, 9]

Process finished with exit code 0
```

컬렉션 데이터 타입(배열, 리스트)

❖ 필터(filter)

- **조건식을 사용**하여서 원하는 요소를 추출함
- 필터 결과로 **List 타입의 배열**을 반환함

```
val filtered_list1 = list1.filter{it%3==0}
```

필터링 결과 — 필터 입력 — 조건식

❖ 필터 사용하기

- 조건식의 결과 값에 따라서, **참이면** 결과 배열에 넣고,
  거짓이면 결과 배열에 넣지 않음

```
package com.example.myapplication
import java.util.*
fun main(){
    println("======= 배열 =========")
    var list1: Array<Int> = arrayOf(1,2,3,4,5,6,7,8)
    var list2: List<Int> = listOf(1,2,3,4,5,6,7,8)
    var list3: MutableList<Int> = mutableListOf(1,2,3,4,5,6,7,8)

    list1 = list1.plus( element: 9)
    list2 = list2.plus( element: 9)
    list3.add(9)

    println("list1 : ${Arrays.toString(list1)}")
    println("list2 : $list2")
    println("list2 : $list3")

    val filtered_list1 = list1.filter{it%3==0}
    val filtered_list2 = list2.filter{it%3==0}
    val filtered_list3 = list3.filter{it%3==0}

    println("list1의 3의 배수 출력 : $filtered_list1")
    println("list2의 3의 배수 출력 : $filtered_list2")
    println("list3의 3의 배수 출력 : $filtered_list3")
}
```

배열 필터 적용

필터링 이후 배열 출력

```
"C:\Program Files\Android\Android St
======= 배열 =========
list1 : [1, 2, 3, 4, 5, 6, 7, 8, 9]
list2 : [1, 2, 3, 4, 5, 6, 7, 8, 9]
list2 : [1, 2, 3, 4, 5, 6, 7, 8, 9]
list1의 3의 배수 출력 : [3, 6, 9]
list2의 3의 배수 출력 : [3, 6, 9]
list3의 3의 배수 출력 : [3, 6, 9]

Process finished with exit code 0
```

조건문

❖ if – else

- **조건이 참일 때** 실행하는 **if 구문**과, **거짓일 때** 실행하는 **else 구문**으로 구성됨

```
fun main() {
        var grade: Array<String>
        grade = arrayOf("A+","A","B+","B","C+","C","D+","D","F")
        var ranking: Int = 1
        var idx: Int = 8

True!   if (ranking <= 5) idx = 0
        else if (ranking <= 10) idx = 1
        else if (ranking <= 15) idx = 2
        else if (ranking <= 20) idx = 3
        else if (ranking <= 25) idx = 4
        else if (ranking <= 30) idx = 5
        else if (ranking <= 35) idx = 6
        else if (ranking <= 40) idx = 7
Pass!   else if (ranking <= 45) idx = 8

        println("나의 학점은 ? : "+grade[idx])
}
```

❖ when

- 조건의 값을 **순차적으로 비교**하여 일치하는 부분을 찾음

```
fun main() {
        var grade: Array<String>
        grade = arrayOf("A+","A","B+","B","C+","C","D+","D","F")
        var ranking: Int = 1
        var idx: Int = 8
```

Style 1

```
        when(ranking / 5) {
Match!      0 -> idx = 0
            1 -> idx = 1
            2 -> idx = 2
            3 -> idx = 3
            4 -> idx = 4
            5 -> idx = 5
            6 -> idx = 6
Pass!       7 -> idx = 7
            8 -> idx = 8
        }
```

Style 2

```
        when(ranking) {
Match!      in 0 .. 5 -> idx = 0
            in 6 .. 10 -> idx = 1
            in 11 .. 15 -> idx = 2
            in 16 .. 20 -> idx = 3
            in 21 .. 25 -> idx = 4
            in 26 .. 30 -> idx = 5
            in 31 .. 35 -> idx = 6
            in 36 .. 40 -> idx = 7
Pass!       else -> idx = 8
        }

        println("나의 학점은 ? : "+grade[idx])
}
```

```
"C:\Program Files\Android\Android
나의 학점은 ? : A+

Process finished with exit code 0
```

❖ if-else와 when의 비교

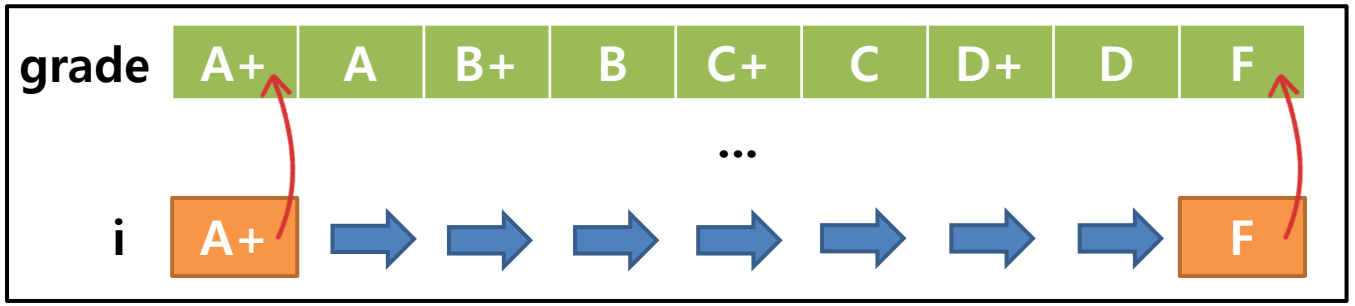
- **if-else 문**은 조건식으로 판별하고, **when 문**은 조건 값으로 판별 가능함
- 다른 언어의 switc와 달리, **when은 조건 값 외에 조건 범위로도 판별 가능함**

반복문

❖ for

- **지정된 범위 내에서 반복**을 수행함

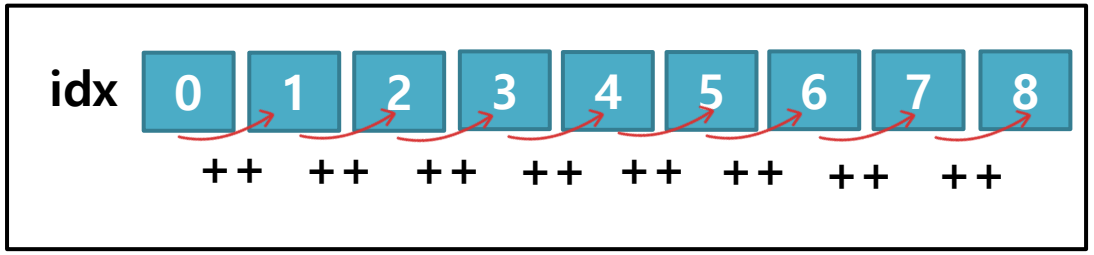
```
fun main() {
        var grade: Array<String>
        grade = arrayOf("A+","A","B+","B","C+","C","D+","D","F")

        print("학점의 종류는 ? ")
        for (i in grade) print(i + " ")
}
```



❖ While

- **조건식이 일치하는 동안 반복**을 수행함

```
fun main() {
        var grade: Array<String>
        grade = arrayOf("A+","A","B+","B","C+","C","D+","D","F")
        var idx: Int = 0

        print("학점의 종류는 ? ")
        while (idx < grade.size) print(grade[idx++] + " ")
}
```



❖ 반복문의 활용

- 반복문은 코드를 간결하게 만들어 줌
- **For문은 in 문과 결합**을 통해서 요소에 순차적으로 접근할 수 있어서, 배열을 조작하는 코드를 간결하게 함
- 주로, Android 프로그래밍에서는 **UI 위젯 배열에 대한 접근에 사용**됨

```
"C:\Program Files\Android\Android S
학점의 종류는 ? A+ A B+ B C+ C D+ D F
Process finished with exit code 0
```

클래스와 생성자

❖ 클래스 란?

- **동일한 속성과 메소드를 가진 데이터 구조**의 정의
- 관련된 데이터를 관리하기 유용함

```
public class Grade constructor(val name: String, score: Int) {
```

접근 제한자 | 클래스 이름 | 파라미터 설정 (생성자) | 파라미터 선언

- **생성자의 이름**은 클래스 이름과 동일하기에 **생략이 가능**함

- 클래스의 내부 변수 선언
  - 클래스의 **내부 변수를 선언하는 방법**은 **파라미터 선언, 내부 변수 선언** 2 가지 방법이 존재함

```
public class Grade constructor(val name: String, score: Int) {
        var score = score
        fun change_score(changed_score: Int){
                score = changed_score
        }
        fun print_score(){
                println(score)
        }
}
fun main(){
        var Hong = Grade( name: "Hong", score: 80)
        Hong.print_score()
        Hong.change_score( changed_score: 95)
        Hong.print_score()
}
```

내부변수 선언

객체 생성

```
"C:\Program Files\Android\Android
80
95

Process finished with exit code 0
```

클래스 상속

❖ 클래스 상속 이란?

- **부모 클래스의 속성과 메소드 구성을 자식 클래스가 동일하게 물려 받음**
- 상속을 이용하여서 **부모 클래스를 기반으로 하는 수정된 클래스 정의**가 가능함
- 클래스 상속 형식

```
open class Super{
}
Class Sub: Super() {
}
```

❖ 상속 예시

- **Person 부모 클래스**를 **상속받는 Grade 클래스**임
  - **부모클래스**는 **open 한정자**를 **사용하여 선언**함
  - **자식클래스**는 ***CLASSNAME*(param)**로 **부모를 명시**하여서 상속 받음

```
open class Person(school: String){
        var school = school
        open fun print_school(){
            println(school)
        }
}
public class Grade constructor
        (school: String,val name: String, score: Int):Person(school) {
        var score = score
        fun change_score(changed_score: Int){
                score = changed_score
        }
        fun print_score(){
                println(score)
        }
}
fun main(){
        var Hong = Grade( school: "Pusan Univ.", name: "Hong", score: 80)
        Hong.print_score()
        Hong.change_score( changed_score: 95)
        Hong.print_score()
        Hong.print_school()
}
```

```
"C:\Program Files\Android\Android
80
95
Pusan Univ.

Process finished with exit code 0
```

상속 허용

본 클래스의 부모 클래스 설정

부모 클래스 메소드 호출

호출 결과

실습

실습 목표와 구성

1. **기초(따라하기)**
   - 클래스를 담고 있는 배열 필터
   - 2차원 배열 클래스와 문자열
2. **응용(로직구현)**
   - 윤년 출력하기
3. **심화(완성하기)**
   - ATM 만들기

기초(따라하기) – 예제 1

❖ 클래스를 담고 있는 배열 필터

- 학년, 이름, 학번을 출력하는 클래스를 담고 있는 **배열을 배열 필터를 사용하여 출력**할 수 있다.

```
"C:\Program Files\Android\Android Studio\jre\
======= 배열 =========
YEAR : Freshman, NAME : Kim, ID : 202011679
YEAR : Freshman, NAME : Wang, ID : 202055028

Process finished with exit code 0
```

1. 클래스 정의
2. 객체 및 배열 선언
3. 배열 필터링

기초(따라하기) – 예제 1

1. 클래스 정의

- (line 4) 대학교 1학년~4학년의 Year을 가지는 **enum class** 선언
- (line 5) 학년, 이름, 학번을 매개변수로 가지는 **Student 클래스** 선언
- (line 6~8) information을 출력하는 메소드 선언

```
1  package com.example.myapplication
2  import java.util.*
3
4  enum class Year{ Freshman, Sophomore, Junior, Senior }
5  class Student(var year:Year, var name: String, var id: Int){
6      fun print_information(){
7          println("YEAR : ${year}, NAME : $name, ID : $id")
8      }
9  }
```

기초(따라하기) – 예제 1

2. 객체 및 배열 선언

- (line 12) Student 클래스 타입을 받는 배열 선언
- (line 14~18) 배열에 Student클래스 추가

```
10  fun main(){
11
12      var students: MutableList<Student> = mutableListOf()
13
14      students.add(Student(Year.Freshman, name: "Kim", id: 202011679))
15      students.add(Student(Year.Freshman, name: "Wang", id: 202055028))
16      students.add(Student(Year.Sophomore, name: "Kim", id: 201803049))
17      students.add(Student(Year.Junior, name: "Lee", id: 201646290))
18      students.add(Student(Year.Senior, name: "Hwangbo", id: 201412654))
19
```

기초(따라하기) – 예제 1

3. 배열 필터링

- (line 20) 필터를 통해 학년이 Freshman인 학생만 freshman_students 배열에 추가
- (line 22~24) 필터링 된 학생들 정보 출력

```
20      var freshman_students = students.filter { it.year == Year.Freshman }
21
22      for(student in freshman_students){
23          student.print_information()
24      }
25
26  }
```

기초(따라하기) – 예제 2

❖ 2차원 배열 클래스와 문자열

- **문자열 매트릭스를 2차원 배열로 변환하는 클래스를 작성**하여 **원하는 행/열을 출력**할 수 있다.

1. 클래스 정의
2. 문자열 to 2차원 배열 매트릭스
3. 행/열 출력

```
"C:\Program Files\Android\Android
매트릭스 문자열
1 2 3
4 5 6
7 8 9

매트릭스 클래스
[4, 5, 6]
[1, 4, 7]

Process finished with exit code 0
```

기초(따라하기) – 예제 2

1. 클래스 정의

- (line 5) 매개변수로 String 매트릭스를 받아오는 Matrix 정의
- (line 6) String 매트릭스를 List형태의 2차원 배열로 변환하기 위한 datas변수

```
5   class Matrix(private val matrixAsString: String) {
6       val datas: List<List<Int>>
```

기초(따라하기) – 예제 2

2. 문자열 to 2차원 배열 매트릭스

- (line 8) init 메소드로 클래스 선언 시, 초기 값 설정
- (line 9) 받아온 String 매트릭스 변수 matrixAsString 를 List형태의 2차원 배열로 변환시작
- (line 9~10) String 클래스의 확장함수인 .toRegex()로 "[space]" 빈 문자열이 "+" 1회 이상 등장할 때, " space "로 변환

  즉, val string = "1 2 3 4 5" 라는 String이 존재할 때, .replace와 .toRegex()을 통해 string = "1 2 3 4 5"로 변환가능

```
8    init {
9        datas = matrixAsString
10           .replace("[ ]+".toRegex(), replacement: " ")
```

기초(따라하기) – 예제 2

2. 문자열 to 2차원 배열 매트릭스

- (line 11) .split 으로 String을 "\n"로 나누어 리스트로 반환
- (line 12) <LIST>.map 으로 리스트 내부의 값을 변경하여 새로운 리스트를 생성
- (line 13) it은 변경하고자 하는 리스트 내부 값.
  .trim은 앞/뒤 공백제거.

  Ex) name = " 값1 값2 "
  name.trim() => "값1 값2"

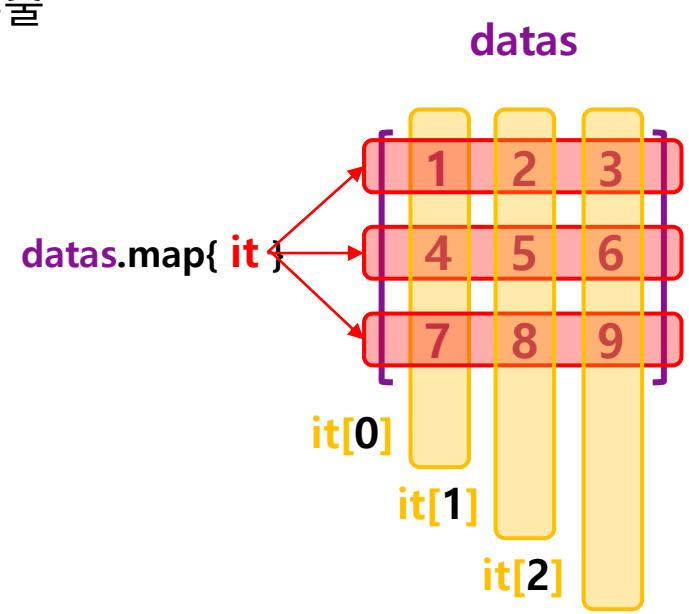
  .split()으로 내부의 값을 공백으로 나누어 2차원 배열 저장
  .map()으로 각각의 값을 Int형태로 변환

```
11              .split( ...delimiters: "\n")
12              .map {  it: String
13                  it.trim().split( ...delimiters: " ").map { it.toInt() }
14              }
15      }
```

기초(따라하기) – 예제 2

3. 행/열 출력

- (line 17~19) 2차원 배열 datas의 특정 컬럼을 반환. .map으로 해당 배열을 받아와 원하는 컬럼 index로 추출




```
17      fun column(colNr: Int): List<Int> {
18          return datas.map { it[colNr - 1] }
19      }
20
21      fun row(rowNr: Int): List<Int> {
22          return datas.get(rowNr - 1)
23      }
24  }
```

기초(따라하기) – 예제 2

3. 행/열 출력

- (line 28) 문자열 매트릭스 생성
- (line 30~31) 문자열 매트릭스 출력확인
- (line 33) 문자열 매트릭스를 2차원 배열로 변환하는 클래스의 매개변수로 String 삽입
- (line 35~37) 2차원 배열로 매트릭스를 변환한 클래스 내부 .row와 .column 함수로 원하는 행/열 값 출력

```
26  fun main(){
27
28      val inputString="1 2 3\n4 5 6\n7 8 9" // 매트릭스 문자열
29
30      println("매트릭스 문자열")
31      println(inputString)
32
33      val matrix = Matrix(inputString) // 매트릭스 클래스
34
35      println("\n매트릭스 클래스")
36      println(matrix.row( rowNr: 2)) // 원하는 로우 출력
37      println(matrix.column( colNr: 1)) // 원하는 칼럼 출력
38  }
```

응용(로직구현) – 예제 3

❖ 윤년 출력하기

- **주어진 메인함수를 사용**하여 연도가 주어졌을 때, **윤년이면 1, 아니면 0**을 반환하는 **Year 함수**를 작성하시오.
  - **윤년**
    - 연도가 4로 나누어 떨어지는 해
    - 연도가 400으로 나누어 떨어지는 해
  - **평년**
    - 연도가 100으로 나누어 떨어지지만 400으로 나누어 떨어지지 않는 해
  - **우선순위** : **400 > 100 > 4**

```
1   package com.example.myapplication
2   fun Year(number:Int):Int{
3
4
5
6
7
8
9
10
11
12
13  }
14  fun printer(number:Int){
15          if(number == 0) println("윤년이 맞습니다.")
16          else if(number == 1) println("윤년 아닙니다.")
17  }
18  fun main(){
19          println("2000년은 윤년 일까?")
20          printer(Year( number: 2000))
21
22          println("1900년은 윤년 일까?")
23          printer(Year( number: 1900))
24
25          println("2020년은 윤년 일까?")
26          printer(Year( number: 20020))
27
28          println("2013년은 윤년 일까?")
29          printer(Year( number: 2013))
30  }
```

TODO()

심화(완성하기) – 예제 4

❖ ATM 만들기

- 다음과 같은 **출력**이 나오도록 **MyAccount 클래스**와 **메인 함수**를 만들어야 한다.
- Class 내부 필요한 함수는 총 4개로 아래와 같다.
  - Deposite // 입금
  - Withdraw // 출금
  - downGrade // 신용 등급 하락
  - upGrade // 신용 등급 상승
- While문으로 계좌정보와 입금, 출금을 반복해서 받는다.
- I는 계좌정보를 나타내며 D와 W로 돈을 입금/출금 할 수 있다.
- 신용 등급은 enum class로 나타내어야 하며, 등급의 종류는 {**A, B, C, D, E, F**} 6가지가 존재한다.
- 입금 후, **계좌 잔액>= 0** 이면 신용 등급이 한단계 상승한다.
- 출금 후, 계좌 잔액이 -1000 이상 0 미만 이라면 신용등급이 한 단계 떨어진다.
- 출금 후, 계좌 잔액이 -1000 미만이라면 잔액이 부족하다는 에러를 출력한다.

출력1

```
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...
----선택하세요----
|(I) 계좌정보     |
|(D) 입금         |
|(W) 출금         |
|(E) 종료         |
-----------------
I
계좌정보는 다음과 같습니다
|이름:       Kim |
|계좌잔액:    0   |
|신용등급:    C   |
------------------

----선택하세요----
|(I) 계좌정보     |
|(D) 입금         |
|(W) 출금         |
|(E) 종료         |
-----------------
D
입금하실 금액을 말하세요.
1000
신용등급이 'C->B'로 한단계 상승합니다.
1000 원을 입금하였습니다. 잔액 : 1000
----선택하세요----
|(I) 계좌정보     |
|(D) 입금         |
|(W) 출금         |
|(E) 종료         |
-----------------
W
출금하실 금액을 말하세요.
1050
계좌가 마이너스 되었습니다.
신용등급이 'B->C'로 한단계 떨어집니다.1050 원을 출금하였습니다. 잔액 : -50
```

계좌정보

입금 금액 받기

입금 후, 신용등급 상승

출금 금액 받기

출금 후, 신용등급 하락

심화(완성하기) – 예제 4

❖ ATM 만들기

- 신용 등급이 F일 때, 계좌는 동결되며 **입금만 가능한 상태**가 된다.

출력2

```
----선택하세요----
|(I) 계좌정보     |
|(D) 입금         |
|(W) 출금         |
|(E) 종료         |
-----------------
W
출금하실 금액을 말하세요.
50
계좌가 마이너스 되었습니다.
신용등급이 'E->F'로 한단계 떨어집니다.50 원을 출금하였습니다. 잔액 : -450
----선택하세요----
|(I) 계좌정보     |
|(D) 입금         |
|(W) 출금         |
|(E) 종료         |
-----------------
W
출금하실 금액을 말하세요.
20
최저 등급의 신용을 가지고 있습니다.
계좌가 동결됩니다.
계좌가 마이너스 되었습니다.
20 원을 출금하였습니다. 잔액 : -470
----선택하세요----
|(I) 계좌정보     |
|(D) 입금         |
|(W) 출금         |
|(E) 종료         |
-----------------
W
동결된 계좌에서 출금하실 수 없습니다.
----선택하세요----
|(I) 계좌정보     |
|(D) 입금         |
|(W) 출금         |
|(E) 종료         |
-----------------
```

최저 신용 등급일 때, 계좌 동결

입금만 가능!

심화(완성하기) – 예제 4

❖ ATM 만들기

- 입금 후 동결된 계좌의 계좌 잔액이 0이상이라면, 계좌의 동결 상태를 해제한다.
- 하지만 0이상이 아니라 마이너스라면, 계속 동결한다.

출력3

```
최저 등급의 신용을 가지고 있습니다.
계좌가 동결됩니다.
계좌가 마이너스 되었습니다.
20 원을 출금하였습니다. 잔액 : -470
----선택하세요----
|(I) 계좌정보     |
|(D) 입금         |
|(W) 출금         |
|(E) 종료         |
----------------
W
동결된 계좌에서 출금하실 수 없습니다.
----선택하세요----
|(I) 계좌정보     |
|(D) 입금         |
|(W) 출금         |
|(E) 종료         |
----------------
D
입금하실 금액을 말하세요.
500
동결계좌가 열렸습니다.
신용등급이 'F->E'로 한단계 상승합니다.
500 원을 입금하였습니다. 잔액 : 30
----선택하세요----
|(I) 계좌정보     |
|(D) 입금         |
|(W) 출금         |
|(E) 종료         |
----------------
```

계좌 잔액 -470에서
+500입금 = 30
계좌 lock 해제

출력4

```
최저 등급의 신용을 가지고 있습니다.
계좌가 동결됩니다.
계좌가 마이너스 되었습니다.
100 원을 출금하였습니다. 잔액 : -470
----선택하세요----
|(I) 계좌정보     |
|(D) 입금         |
|(W) 출금         |
|(E) 종료         |
----------------
D
입금하실 금액을 말하세요.
400
400 원을 입금하였습니다. 잔액 : -70
----선택하세요----
|(I) 계좌정보     |
|(D) 입금         |
|(W) 출금         |
|(E) 종료         |
----------------
W
동결된 계좌에서 출금하실 수 없습니다.
----선택하세요----
|(I) 계좌정보     |
|(D) 입금         |
|(W) 출금         |
|(E) 종료         |
----------------
```

계좌 잔액 -470에서
+400입금 = -70
계속 계좌는 동결됨

별첨 – 코틀린 코드 실행하기

❖ 코틀린 코드 독립 실행 방법

1) 프로젝트 – 코틀린 코드 – 오른쪽 마우스 키 – Run

2) 코틀린 코드 화면 – 오른쪽 마우스키 - Run