# 안드로이드 프로그래밍

2025년 2학기

# 기본 자료형

# 변수 선언 (val, var)

- 코틀린에서 변수를 선언하는 방법은 크게 두 가지 키워드, val과 var를 사용
  - 재할당 가능 여부
- `val` (Value)
  - 읽기 전용이며, 한 번 값이 할당되면 재할당할 수 없음
  - 코드가 예상치 않게 변경되는 것을 막아 안정성을 높임

```
fun main() {
    val message = "안녕하세요, 코틀린!" // 선언과 동시에 값 할당
    println(message) // 출력: 안녕하세요, 코틀린!

    // message = "다시 만나요!" // 에러 발생! 'val'은 재할당할 수 없습니다.
}
```

# 변수 선언 (val, var)

- var (Variable)
  - 읽기/쓰기 가능하며, 언제든지 다른 값으로 재할당할 수 있음
  - 값이 변경될 필요가 있는 경우에 사용

```
fun main() {
    var count = 10 // 선언과 동시에 값 할당
    println(count) // 출력: 10

    count = 20 // 값 재할당 가능
    println(count) // 출력: 20
}
```

- Tip: 가능한 한 val을 우선적으로 사용, 값이 변경되어야 할 필요가 있을 때만 var를 사용
  - 코드를 더 안전하고 예측 가능하게 만듬

# 자료형 (기본 타입: Int, Double, Boolean, String)

- 코틀린의 기본 자료형은 객체 형태로 제공
- 자바의 원시 타입(primitive type)과 달리 모든 것이 객체이므로 편리하게 다양한 메서드를 호출할 수 있게 함
  - `Int`: 정수 값을 저장합니다. (예: 1, 100, -5)

```
fun main() {
    val number: Int = 123
    println(number) // 출력: 123
}
```

# 자료형 (기본 타입: Int, Double, Boolean, String)

- Double: 부동 소수점(실수) 값을 저장합니다. (예: 3.14, 0.5)

```
fun main() {
    val pi: Double = 3.141592
    println(pi) // 출력:
3.141592
}
```

- Boolean: 논리 값인 true 또는 false를 저장

```
fun main() {
    val isTrue: Boolean = true
    val isFalse: Boolean = false
    println(isTrue) // 출력: true
    println(isFalse) // 출력: false
}
```

# 자료형 (기본 타입: Int, Double, Boolean, String)

- String: 문자열을 저장합니다. 문자열은 큰따옴표(")로 감쌉니다.

```
fun main() {
    val greeting: String = "Hello, Kotlin!"
    println(greeting) // 출력: Hello,
Kotlin!
}
```

# 자료형 (기본 타입: Int, Double, Boolean, String)

- 자료형 추론: 변수에 값을 할당할 때 자료형을 자동으로 추론하는 기능
  - 대부분의 경우 자료형을 명시적으로 선언하지 않아도 됨

```
fun main() {
    val number = 123 // 자료형을 명시하지 않아도 Int로 추론
    val pi = 3.14 // Double로 추론
    val isReady = true // Boolean으로 추론
    val name = "Kotlin" // String으로 추론
}
```

# 자료형 (기본 타입: Int, Double, Boolean, String)

- 문자열 템플릿 ("Hello, $name")
  - 문자열 템플릿(String Template)은 문자열 안에 변수나 표현식의 값을 직접 포함시켜 동적인 문자열을 쉽게 만들 수 있는 기능
  - 변수 사용: 변수 이름 앞에 $를 붙여 사용

```
fun main() {
    val name = "Alice"
    val age = 30

    // $name을 사용하여 변수 값 포함
    val greeting = "안녕하세요, 제 이름은 $name입니다."
    println(greeting) // 출력: 안녕하세요, 제 이름은 Alice입니다.

    val introduction = "저는 $age살입니다."
    println(introduction) // 출력: 저는 30살입니다.
}
```

# 자료형 (기본 타입: Int, Double, Boolean, String)

- 문자열 템플릿 ("Hello, $name")
  - 문자열 템플릿(String Template)은 문자열 안에 변수나 표현식의 값을 직접 포함시켜 동적인 문자열을 쉽게 만들 수 있는 기능
  - 표현식 사용: 변수뿐만 아니라, {} 안에 계산식이나 함수 호출과 같은 표현식을 포함시킬 수 있음

```
fun main() {
    val a = 10
    val b = 5

    // ${a + b}를 사용하여 표현식의 결과 포함
    println("10 + 5 = ${a + b}") // 출력: 10 + 5 = 15

    val text = "Kotlin"
    // ${text.length}를 사용하여 함수 호출 결과 포함
    println("문자열의 길이는 ${text.length}입니다.") // 출력: 문자열의 길이는 6입니다.
}
```

# Null-safety (?, !!, ?: 엘비스 연산자)

- 코틀린의 가장 강력한 기능 중 하나
- 컴파일 시점에 NullPointerException(NPE)을 방지하도록 설계
- 변수가 Null 값을 가질 수 있는지 여부를 명확하게 구분
  - ? (Null 허용 타입): 변수 뒤에 ?를 붙이면 해당 변수가 Null 값을 가질 수 있다는 의미
    - Null 허용 변수에 접근할 때는 반드시 안전 호출(?.)을 사용

```
fun main() {
    var nullableName: String? = "John" // Null을 허용하는 String 타입
    nullableName = null // Null 값 할당 가능

    val length1 = nullableName?.length // 안전 호출. 'nullableName'이 null이면 null 반환, 아니면 길이
반환
    println(length1) // 출력: null

    nullableName = "Kotlin"
    val length2 = nullableName?.length // 'Kotlin'은 null이 아니므로 길이 반환
    println(length2) // 출력: 6
}
```

# Null-safety (?, !!, ?: 엘비스 연산자)

- !! (Null 아님 단언 연산자): 해당 변수가 절대 Null이 아님을 확신할 때 사용
  - 변수가 Null이면 NullPointerException이 발생
  - 매우 위험하므로 최후의 수단으로 사용해야

```
fun main() {
    val nullableName: String? = null

    // val length = nullableName!!.length // 실행 시 NullPointerException 발생
    // 이 코드는 위험하므로 실제로는 사용을 피해야 합니다.
}
```

# Null-safety (?, !!, ?: 엘비스 연산자)

- ?: (엘비스 연산자): Null 허용 변수가 Null일 경우 대신 사용할 기본값을 지정
  - A ?: B는 'A가 Null이 아니면 A의 값을, Null이면 B의 값을 사용하라'는 의미

```
fun main() {
    val nullableName: String? = null
    val name: String = nullableName ?: "Unknown" // nullableName이 null이므로 "Unknown"이 할당됨
    println(name) // 출력: Unknown

    val nullableName2: String? = "Alice"
    val name2: String = nullableName2 ?: "Unknown" // nullableName2가 null이 아니므로 "Alice"가 할당됨
    println(name2) // 출력: Alice

    val length = nullableName?.length ?: 0 // nullableName이 null이므로 0이 할당됨
    println(length) // 출력: 0
}
```

- Null-safety를 통해 코틀린은 안전하고 견고한 애플리케이션을 만들 수
- Null 가능성을 명시적으로 다루기 때문에 예측 불가능한 런타임 오류를 크게 줄일 수

# 실습 1

- 문제 1: 정수와 실수 연산

  - 요구 사항

    1. numInt라는 이름의 Int 변수에 10을 할당하세요.
    2. numDouble이라는 이름의 Double 변수에 3.14를 할당하세요.
    3. 두 변수를 더하여 결과를 sumResult라는 이름의 Double 변수에 저장하세요.
    4. sumResult 변수의 값을 출력하세요.

# 실습 1

- 문제 2: 문자열 변환과 출력

  - 요구 사항

    1. name이라는 이름의 String 변수에 여러분의 이름을 할당하세요.
    2. age라는 이름의 Int 변수에 여러분의 나이를 할당하세요.
    3. "안녕하세요, 제 이름은 [이름]이고, 나이는 [나이]살입니다." 형식의 문장을 출력하세요.

# 실습 1

- 문제 3: 논리형(Boolean) 연산

  - 요구 사항

    1. isKotlinFun이라는 이름의 Boolean 변수에 true를 할당하세요.
    2. isWeatherGood이라는 이름의 Boolean 변수에 false를 할당하세요.
    3. isKotlinFun과 isWeatherGood이 모두 true인지를 확인하는 논리곱(&&) 연산을 수행하여 결과를 bothTrue 변수에 저장하세요.
    4. 두 변수 중 하나라도 true인지를 확인하는 논리합(||) 연산을 수행하여 결과를 atLeastOneTrue 변수에 저장하세요.
    5. bothTrue와 atLeastOneTrue 변수의 값을 각각 출력하세요.

# 실습 1

- 문제 4: 자료형 변환 (Type Casting)

  - 요구 사항

    1. priceDouble이라는 이름의 Double 변수에 99.99를 할당하세요.
    2. 이 값을 Int 자료형으로 변환하여 priceInt 변수에 저장하세요.
    3. priceInt 변수의 값을 출력하세요.

# 실습 1

- 문제 5: 문자(Char)와 문자열(String) 다루기

  - 요구 사항

    1. firstLetter라는 이름의 Char 변수에 여러분 이름의 첫 글자를 할당하세요.
    2. fullSentence라는 이름의 String 변수에 "코틀린 첫 글자는 A입니다."라는 문장을 만드세요. 단, 'A' 부분은 위에서 선언한 firstLetter 변수를 사용해야 합니다.
    3. fullSentence 변수의 값을 출력하세요.

# 실습 2

- 문제 1: Nullable 변수 선언 및 Safe Call 연산자 사용

  - 요구 사항
    - nullableString이라는 이름의 String? 타입 변수에 null을 할당하세요.
    - 이 변수의 길이를 safe call 연산자(?.)를 사용하여 출력하세요.

# 실습 2

- 문제 2: Safe Call과 Elvis 연산자 (?:) 사용

  - 요구 사항

    1. text라는 이름의 String? 변수에 "코틀린"을 할당하세요.
    2. safe call 연산자와 Elvis 연산자를 함께 사용하여, 변수가 null이 아닐 경우 길이를, null일 경우 0을 반환하도록 코드를 작성하고 출력하세요.
    3. text 변수에 null을 다시 할당한 후, 같은 코드를 실행하여 결과를 확인하세요.

# 실습 2

- 문제 3: Null 체크와 Non-null Assertion (!!)
    - 요구 사항
        - anotherString이라는 이름의 String? 변수에 "안전한 문자열"을 할당하세요.
        - if문을 사용하여 변수가 null이 아닐 경우, "안전한 문자열의 길이: "와 함께 길이를 출력하세요.
        - (주의!) anotherString에 null을 할당하고, Non-null Assertion(!!) 연산자를 사용했을 때 어떤 결과가 나오는지 확인하세요.
            - 이 연산자의 사용은 매우 위험하므로, 꼭 필요한 경우가 아니면 지양해야 합니다.

# 실습 2

- 문제 4: 함수 파라미터로 Nullable 타입 사용하기
  - 요구 사항
    1. String? 타입을 파라미터로 받는 safeLength 함수를 아래에 정의하세요.
    2. 이 함수는 파라미터가 null일 경우 0을, 아닐 경우 문자열의 길이를 반환해야 합니다.
    3. 아래의 호출 코드를 통해 결과를 확인하세요.

# 제어문

# 조건문

- 특정 조건에 따라 다른 코드 블록을 실행하는 데 사용
- `if`
  - 가장 기본적인 조건문
  - 자바와 달리, 코틀린의 `if`는 표현식으로 사용될 수 있어 값을 반환할 수 있음
  - 형식

```
val result = if (condition) {
    // 조건이 참일 때 실행되는 코드
    "true"
} else {
    // 조건이 거짓일 때 실행되는 코드
    "false"
}
```

# 조건문

- if
  - 예제

```
fun main() {
    val score = 85
    val grade: String

    if (score >= 90) {
        grade = "A"
    } else if (score >= 80) {
        grade = "B"
    } else {
        grade = "C"
    }

    // if를 표현식으로 사용
    val grade2 = if (score >= 90) "A" else if (score >= 80) "B" else "C"

    println("점수: $score, 학점: $grade")
    println("if 표현식으로 구한 학점: $grade2")
}
```

# 조건문

- when
  - 여러 조건에 대해 복잡한 if-else if 체인을 대체하는 더 강력하고 유연한 조건문
  - switch 문과 유사하지만 훨씬 더 많은 기능을 제공
  - 형식

```
when (subject) {
    value1 -> // value1일 때 실행
    value2 -> // value2일 때 실행
    in range -> // 범위 내일 때 실행
    is Type -> // 타입이 일치할 때 실행
    else -> // 그 외의 경우 실행
}
```

# 조건문

- when
  - 예제

```
fun main() {
    val day = 3

    when (day) {
        1 -> println("월요일")
        2 -> println("화요일")
        3, 4 -> println("수요일 또는 목요일") // 여러 값 조건
        in 5..7 -> println("주말") // 범위 조건
        else -> println("유효하지 않은 요일")
    }

    val result = when (day) {
        1 -> "월요일"
        2 -> "화요일"
        else -> "다른 요일"
    }
    println("when 표현식 결과: $result")
}
```

# 반복문

- 특정 코드 블록을 여러 번 실행하는 데 사용
- for
  - for 루프는 컬렉션(리스트, 배열 등)이나 범위의 각 요소에 대해 반복
  - 형식

```
for (item in collection) {
    // 각 item에 대해 실행
}
```

# 반복문

- for
  - 예제

```
fun main() {
    val fruits = listOf("사과", "바나나", "오렌지")

    // 컬렉션 반복
    for (fruit in fruits) {
        println(fruit)
    }

    // 인덱스와 함께 반복
    for (index in fruits.indices) {
        println("fruits[$index] = ${fruits[index]}")
    }

    // withIndex() 사용 (더 코틀린스럽다)
    for ((index, value) in fruits.withIndex()) {
        println("인덱스 $index: $value")
    }
}
```

# 반복문

- while
  - while과 do-while 루프는 특정 조건이 참인 동안 코드 블록을 반복 실행
    - while: 조건이 참인 동안 반복
    - do-while: 코드 블록을 먼저 실행한 후 조건 검사 (최소 한 번 실행 보장)
  - 예제

```
fun main() {
    var i = 0
    while (i < 3) {
        println("while 루프: $i")
        i++
    }

    var j = 0
    do {
        println("do-while 루프: $j")
        j++
    } while (j < 3)
}
```

# 반복문

- 범위 표현식
  - for 루프에서 숫자의 범위를 지정하거나 when 문에서 조건을 정의할 때 매우 유용하게 사용
  - .. 연산자
    - .. 연산자는 두 값 사이의 닫힌 범위를 생성 (양 끝 값 모두 포함)
    - 예제

```
fun main() {
    // 1부터 5까지 (1, 2, 3, 4, 5)
    for (i in 1..5) {
        print("$i ")
    }
    println()
}
```

# 반복문

- 범위 표현식
  - downTo
    - downTo는 숫자를 감소시키며 범위를 생성
    - 예제

```
fun main() {
    // 5부터 1까지 (5, 4, 3, 2, 1)
    for (i in 5 downTo 1) {
        print("$i ")
    }
    println()
}
```

# 반복문

- 범위 표현식
  - step
    - step은 반복 간격(증가 또는 감소)을 지정
    - 예제:

```
fun main() {
    // 1부터 10까지 2씩 증가 (1, 3, 5, 7, 9)
    for (i in 1..10 step 2) {
        print("$i ")
    }
    println()

    // 10부터 1까지 3씩 감소 (10, 7, 4, 1)
    for (i in 10 downTo 1 step 3) {
        print("$i ")
    }
    println()
}
```

# 실습 3

- 문제 1: if-else를 이용한 숫자 판별

  - 요구 사항

    1. number 변수가 양수, 음수, 0 중 무엇인지 판별하여 출력하세요.
    2. 출력 형식: "이 숫자는 양수입니다." 또는 "이 숫자는 음수입니다." 또는 "이 숫자는 0입니다."

# 실습 3

- 문제 2: when을 이용한 문자 타입 판별

  - 요구 사항

    1. char 변수가 모음(a, e, i, o, u)인지, 자음인지, 또는 알파벳이 아닌지 판별하여 출력하세요.
    2. 출력 형식: "이 문자는 모음입니다." 또는 "이 문자는 자음입니다." 또는 "이 문자는 알파벳이 아닙니다."

# 실습 3

- 문제 3: for 반복문을 이용한 합계 계산

  - 요구 사항

    1. for 반복문을 사용하여 1부터 10까지의 모든 정수 합계를 계산하고 출력하세요.
    2. 출력 형식: "1부터 10까지의 합계는 55입니다."

# 실습 3

- 문제 4: while 반복문을 이용한 조건 만족

  - 요구 사항

    1. while 반복문을 사용하여 1부터 시작하여 2로 나누어 떨어지는 첫 번째 숫자를 찾아 출력하세요.
    2. 출력 형식: "2로 나누어 떨어지는 첫 번째 숫자는 2입니다."

# 실습 4

- 문제 1: 1부터 5까지 출력하기

  - 요구 사항

    1. for 반복문과 범위를 사용하여 1부터 5까지의 숫자를 순서대로 출력하세요.

# 실습 4

- 문제 2: 5부터 1까지 역순으로 출력하기

  - 요구 사항

    1. for 반복문과 downTo 연산자를 사용하여 5부터 1까지의 숫자를 역순으로 출력하세요.

# 실습 4

- 문제 4: 1부터 5까지 (5를 포함하지 않고) 출력하기

  - 요구 사항

    1. for 반복문과 until 연산자를 사용하여 1부터 4까지의 숫자를 출력하세요.

# 실습 4

- 문제 1: 1부터 5까지 출력하기

  - 요구 사항

    1. for 반복문과 범위를 사용하여 1부터 5까지의 숫자를 순서대로 출력하세요.