

독학 Kotlin 1

[독학 Kotlin 2](#)

Kotlin 기본 문법 완벽 가이드

Kotlin 문법 기본 규칙


기본 문법 특징

특징	설명
세미콜론	붙이지 않아도 됨 (선택사항)
변수 명명	파스칼/카멜 표기법 권장
null 안전성	기본적으로 null을 허용하지 않음

변수 선언

변수 선언 키워드

키워드	특징	Java 비교
var	일반적으로 통용 (변경 가능)	일반 변수
val	선언 시에만 초기화 가능 (변경 불가)	final 변수

 **권장사항:** Runtime 시 변경하지 않는 값은 `val` 로 선언

변수 타입 분류

```
class MyClass {  
    var classProperty: String = "클래스 속성" // Property (속성)  
  
    fun myFunction() {  
        var localVariable: Int = 10 // Local Variable (로컬 변수)  
    }  
}
```

Null 안전성

Kotlin은 기본적으로 **null**을 허용하지 않습니다.

```
fun main() {  
    var a: Int = 123    // ✅ 정상  
    print(a)  
  
    var b: Int          // ❌ 초기화 없이 사용하면 컴파일 에러  
    b = 456  
    print(b)  
}
```

🎯 **장점:** NullPointerException 등을 컴파일 시점에 방지

🔄 형변환 (Type Conversion)

⚠️ Kotlin의 형변환 특징

언어	암시적 형변환 지원
다른 언어들	✅ 지원
Kotlin	❌ 지원하지 않음

🎯 명시적 형변환

개발자가 직접 변환될 자료형을 지정해야 합니다.

```
val intValue: Int = 10  
val longValue: Long = intValue.toLong()    // 명시적 변환 필요  
val doubleValue: Double = intValue.toDouble()  
val stringValue: String = intValue.toString()
```

📋 주요 형변환 메서드

```
// 숫자 타입 변환  
toByte(), toShort(), toInt(), toLong()  
toFloat(), toDouble(), toChar()  
  
// 문자열 변환  
toString()
```

📁 배열 (Array)

배열 생성 방법

```
// 1 Array 생성자 사용
val numbers = Array(5) { 0 } // 크기 5, 모든 요소 0으로 초기화

// 2 arrayOf 함수 사용
val fruits = arrayOf("사과", "바나나", "오렌지")

// 3 타입별 전용 배열
val intArray = intArrayOf(1, 2, 3, 4, 5)
val boolArray = booleanArrayOf(true, false, true)
```

배열 사용 예제

```
fun main() {
    val numbers = arrayOf(10, 20, 30, 40, 50)

    // 인덱스로 접근
    println(numbers[0]) // 10

    // 배열 크기
    println(numbers.size) // 5

    // 배열 순회
    for (number in numbers) {
        println(number)
    }
}
```

타입 추론 (Type Inference)

변수나 함수 선언 시 자료형을 명시하지 않아도 Kotlin이 자동으로 추론합니다.

타입 추론 예제

```
// 명시적 타입 선언
val name: String = "Kotlin"
val age: Int = 25
val height: Double = 175.5

// 타입 추론 (권장)
val name2 = "Kotlin" // String으로 추론
val age2 = 25 // Int로 추론
val height2 = 175.5 // Double로 추론

// 함수에서도 타입 추론 가능
fun add(a: Int, b: Int) = a + b // 반환 타입 Int로 추론
```

함수 (Method/Function)

특정한 동작을 하거나 원하는 결과값을 연산하는 데 사용합니다.


기본 함수 구문

```
fun functionName(parameter1: Type1, parameter2: Type2): ReturnType {  
    // 함수 본문  
    return result  
}
```

단일 표현식 함수

간단한 기능의 경우 변수에 결과값을 할당하듯 기술할 수 있습니다.

```
// 일반 함수  
fun add(a: Int, b: Int): Int {  
    return a + b  
}  
  
// 단일 표현식 함수  
fun add2(a: Int, b: Int) = a + b  
  
// 더 복잡한 예제  
fun max(a: Int, b: Int) = if (a > b) a else b
```

 **Kotlin 관점:** 함수는 내부적으로 기능을 가진 형태이지만, 외부에서 볼 때는 Parameter를 넣는다는 점 외에는 **자료형이 결정된 변수**라는 개념으로 접근

조건문

if문

Java와 동일하지만, **결과값을 변수에 할당**할 수 있습니다.

```
// 기본 if문  
val score = 85  
if (score >= 90) {  
    println("A 등급")  
} else if (score >= 80) {  
    println("B 등급")  
} else {  
    println("C 등급")  
}  
  
// if문의 결과를 변수에 할당
```

```

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

// 단일 표현식으로 사용
val result = if (score >= 60) "합격" else "불합격"

```

🎯 when문 (switch문 대체)

다른 언어의 **switch문**을 **when**으로 사용합니다.

```

fun main() {
    val day = 3

    // 기본 when문
    when (day) {
        1 -> println("월요일")
        2 -> println("화요일")
        3 -> println("수요일")
        4 -> println("목요일")
        5 -> println("금요일")
        else -> println("주말")
    }

    // when문의 결과를 변수에 할당
    val dayName = when (day) {
        1 -> "월요일"
        2 -> "화요일"
        3 -> "수요일"
        4 -> "목요일"
        5 -> "금요일"
        else -> "주말"
    }
}

```

☀️ when문의 다양한 활용

```

// Any 자료형 사용 (최상위 자료형)
fun describe(obj: Any): String = when (obj) {
    1 -> "One"
    "Hello" -> "Greeting"
    is Long -> "Long number"
    !is String -> "Not a string"
    else -> "Unknown"
}

// 범위 조건

```

```

val score = 85
val grade = when (score) {
    in 90..100 -> "A"
    in 80..89 -> "B"
    in 70..79 -> "C"
    else -> "F"
}

// 여러 조건
when (day) {
    1, 2, 3, 4, 5 -> println("평일")
    6, 7 -> println("주말")
}

```

반복문

반복문 분류

타입	특징	종류
조건형 반복문	조건이 참인 경우 반복 유지	while , do-while
범위형 반복문	반복 범위를 정해 반복 수행	for

1. while문

```

fun main() {
    var i = 1
    while (i <= 5) {
        println("i = $i")
        i++
    }
}

```

2. do-while문

```

fun main() {
    var i = 1
    do {
        println("i = $i")
        i++
    } while (i <= 5)
}

```

💡 **차이점:** `do-while` 은 최초 1번 `do` 블록을 실행한 후, `while` 로 조건을 체크합니다.

3. 🔄 for문

📊 기본 증가 반복

```
// 기본 for문 (1씩 증가)
for (i in 1..5) {
    println("i = $i")    // 1, 2, 3, 4, 5
}

// until 사용 (마지막 값 제외)
for (i in 1 until 5) {
    println("i = $i")    // 1, 2, 3, 4
}
```

⚡ step을 이용한 증가값 변경

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

// 3씩 증가
for (i in 0..15 step 3) {
    println("i = $i")    // 0, 3, 6, 9, 12, 15
}
```

📉 downTo를 이용한 감소

```
// 1씩 감소
for (i in 5 downTo 1) {
    println("i = $i")    // 5, 4, 3, 2, 1
}

// 2씩 감소
for (i in 10 downTo 1 step 2) {
    println("i = $i")    // 10, 8, 6, 4, 2
}
```

🗂 컬렉션 순회

```
val fruits = arrayOf("사과", "바나나", "오렌지")

// 값만 순회
for (fruit in fruits) {
    println(fruit)
}
```

```
// 인덱스와 값 함께 순회
for ((index, fruit) in fruits.withIndex()) {
    println("$index: $fruit")
}
```

4. 🎮 흐름 제어

🔴 break - 반복문 종료

```
for (i in 1..10) {
    if (i == 5) {
        break    // i가 5일 때 반복문 종료
    }
    println("i = $i")    // 1, 2, 3, 4 출력
}
```

▶▶ continue - 다음 반복으로 건너뛰기

```
for (i in 1..10) {
    if (i == 3 || i == 5 || i == 7) {
        continue    // 3, 5, 7일 때 건너뛰기
    }
    println("i = $i")    // 1, 2, 4, 6, 8, 9, 10 출력
}
```

🏷️ Label을 이용한 다중 반복문 제어

Kotlin만의 독특한 기능으로, 원하는 루프의 흐름을 제어할 수 있습니다.

```
fun main() {
    loop@ for (i in 1..5) {
        for (j in 1..5) {
            if (i == 3 && j == 2) {
                break@loop    // 바깥쪽 loop를 빠져나감
            }
            println("i = $i, j = $j")
        }
    }
    println("반복문 종료")
}
```

🎯 Label 활용 예제

```
outer@ for (i in 1..3) {
    inner@ for (j in 1..3) {
        if (i == 2 && j == 2) {
```



```
        continue@outer    // 바깥쪽 루프의 다음 반복으로
    }
    println("$i, $j")
}
}
```

코드 출력

문자열 템플릿

변수명 앞에 `$` 를 붙이면 변수 값을 출력할 수 있습니다.

```
fun main() {
    val name = "Kotlin"
    val version = 1.9
    val isStable = true

    // 기본 문자열 템플릿
    println("언어: $name")
    println("버전: $version")
    println("안정화: $isStable")

    // 표현식 사용 (중괄호 필요)
    val a = 10
    val b = 20
    println("$a + $b = ${a + b}")

    // 복잡한 표현식
    val numbers = arrayOf(1, 2, 3, 4, 5)
    println("배열 크기: ${numbers.size}")
    println("첫 번째 요소: ${numbers[0]}")
    println("합계: ${numbers.sum()}")
}
```

출력 결과

```
언어: Kotlin
버전: 1.9
안정화: true
10 + 20 = 30
배열 크기: 5
첫 번째 요소: 1
합계: 15
```

🎯 핵심 요약 정리

✅ Kotlin 문법의 핵심 특징

특징	장점	예제
타입 추론	코드 간소화	<code>val name = "Kotlin"</code>
null 안전성	런타임 에러 방지	<code>var data: String?</code>
표현식 지향	함수형 프로그래밍	<code>val result = if (a > b) a else b</code>
간결한 문법	생산성 향상	<code>fun add(a: Int, b: Int) = a + b</code>

🔍 자주 사용하는 패턴

```
// 1 안전한 null 처리
val length = text?.length ?: 0

// 2 범위 체크
if (score in 80..90) { /* ... */ }

// 3 컬렉션 처리
val evenNumbers = numbers.filter { it % 2 == 0 }

// 4 문자열 템플릿
val message = "Hello, $name! You are ${age} years old."

// 5 when 표현식
val grade = when (score) {
    in 90..100 -> "A"
    in 80..89 -> "B"
    else -> "C"
}
```

🚀 **Kotlin 문법 마스터하기!** 이 가이드를 통해 Kotlin의 핵심 문법을 완벽하게 이해하고 실무에 활용해보세요!