Washtenaw Community College
ANN ARBOR, MICHIGAN

CPS251

Android Development

by Scott Shaper

Kotlin Flow Control

Think of flow control in programming like following a recipe. Just as a recipe might say "if the sauce is too thick, add water" or "repeat steps 2-4 until the dough is smooth," flow control statements help your program make decisions and repeat actions. These are the building blocks that make your code dynamic and responsive to different situations.

In this lesson, we'll explore how to control the flow of your Kotlin programs using loops (for repeating actions) and conditional statements (for making decisions). Understanding these concepts is crucial for writing programs that can handle real-world scenarios and user interactions.

Quick Reference Table

Control Structure	Description	When to Use
`for` loop	Iterates over collections or ranges	When you know how many times to repeat
`while` loop	Repeats while condition is true	When you don't know how many iterations needed
`do-while` loop	Executes at least once, then checks condition	When you need to execute code at least once
`if` statement	Executes code based on a condition	For simple yes/no decisions
`when` statement	Handles multiple conditions	For complex decision trees
`break`/`continue`	Controls loop execution	When you need to exit or skip iterations

Loops

When to Use Different Loops

- Use `for` loops when you know the number of iterations
- Use `while` loops when you don't know how many iterations you'll need
- Use `do-while` loops when you need to execute code at least once

Loop Type	What It Does	When to Use It
for-in	Iterates over collections or ranges	Processing lists, arrays, or counting
while	Repeats while condition is true	Reading input until valid, game loops
do-while	Executes once, then repeats if condition true	Menu systems, input validation

Practical Examples

Copy Code

```
// ============================================
// FOR LOOP EXAMPLES
// ============================================

// Example 1: Iterating over a list of fruits
// This is useful when you have a collection of items to process
val fruits = listOf("apple", "banana", "cherry", "orange", "grape")
println("=== PROCESSING FRUIT LIST ===")
for (fruit in fruits) {
    // Each iteration, 'fruit' contains the next item from the list
    println("I like $fruit")
    // You can perform any operation on each item
    println("  - $fruit has ${fruit.length} letters")
}
// Output:
// I like apple
//   - apple has 5 letters
// I like banana
//   - banana has 6 letters
// ... and so on
// Example 2: Counting with ranges
// The '..' operator creates a range from 1 to 5 (inclusive)
println("\n=== COUNTING WITH RANGES ===")
for (i in 1..5) {
    // 'i' takes the values: 1, 2, 3, 4, 5
    println("Count: $i")

    // You can use the counter variable in calculations
    val square = i * i
    println("  $i squared is $square")
}

// Example 3: Processing with index
// Sometimes you need both the item and its position
println("\n=== PROCESSING WITH INDEX ===")
val colors = listOf("red", "green", "blue", "yellow")
for ((index, color) in colors.withIndex()) {
    // 'index' is the position (0, 1, 2, 3)
    // 'color' is the actual color value
    println("Color #${index + 1}: $color")
}

// Example 4: Counting backwards
// Use 'downTo' to count in reverse order
println("\n=== COUNTDOWN EXAMPLE ===")
for (countdown in 5 downTo 1) {
    println("T-minus $countdown seconds")
}
println("Blast off!")

// ============================================
// WHILE LOOP EXAMPLES
// ============================================
// Example 1: Input validation loop
// This loop continues until the user provides valid input
println("\n=== INPUT VALIDATION LOOP ===")
var age = -1  // Start with invalid value
var attempts = 0  // Track how many times user tries

while (age < 0 || age > 120) {
    attempts++
    print("Enter your age (0-120): ")

    // Simulate user input (in real app, this would be readLine())
    val userInput = when (attempts) {
        1 -> "150"      // First attempt: invalid (too high)
        2 -> "-5"       // Second attempt: invalid (negative)
        3 -> "abc"      // Third attempt: invalid (not a number)
        else -> "25"    // Fourth attempt: valid
    }

    // Try to convert input to integer
    age = userInput.toIntOrNull() ?: -1

    if (age < 0 || age > 120) {
        println("Invalid age: $userInput. Please try again.")
    }
}

println("Valid age entered: $age (after $attempts attempts)")

// Example 2: Game loop simulation
// This simulates a simple game that continues until certain conditions
println("\n=== GAME LOOP SIMULATION ===")
var playerHealth = 100
var enemyHealth = 80
var round = 1

while (playerHealth > 0 && enemyHealth > 0) {
    println("=== ROUND $round ===")
    println("Player Health: $playerHealth, Enemy Health: $enemyHealth")

    // Simulate combat
    val playerDamage = (10..20).random()  // Random damage between 10-20
    val enemyDamage = (8..18).random()    // Random damage between 8-18

    enemyHealth -= playerDamage
    playerHealth -= enemyDamage

    println("Player deals $playerDamage damage to enemy")
    println("Enemy deals $enemyDamage damage to player")

    round++

    // Add a small delay to make the output readable
    Thread.sleep(500)
}

// Determine winner
if (playerHealth > 0) {
    println("Player wins! Final health: $playerHealth")
} else {
    println("Enemy wins! Player defeated.")
}

// ============================================
// DO-WHILE LOOP EXAMPLES
// ============================================

// Example 1: Menu system
// This loop always shows the menu at least once
println("\n=== MENU SYSTEM EXAMPLE ===")
var choice: Int
var menuShown = 0

do {
    menuShown++
    println("\n--- MENU (shown $menuShown times) ---")
    println("1. Play Game")
    println("2. Settings")
    println("3. View High Scores")
    println("4. Exit")

    // Simulate user input
    choice = when (menuShown) {
        1 -> 5  // First time: invalid choice
        2 -> 2  // Second time: valid choice (Settings)
        else -> 4  // Third time: valid choice (Exit)
    }

    if (choice !in 1..4) {
        println("Invalid choice: $choice. Please select 1-4.")
    } else if (choice == 2) {
        println("Opening Settings...")
        // In a real app, this would open settings
    }

} while (choice !in 1..4)  // Continue until valid choice

println("Menu loop completed. Final choice: $choice")

// Example 2: Password validation
// This ensures the user gets at least one chance to enter a password
println("\n=== PASSWORD VALIDATION ===")
```

```
var password = ""
var attempts = 0

do {
    attempts++
    print("Enter password (attempt $attempts): ")

    // Simulate password input
    password = when (attempts) {
        1 -> ""           // First attempt: empty password
        2 -> "123"        // Second attempt: too short
        3 -> "password"   // Third attempt: too weak
        else -> "SecurePass123!"  // Fourth attempt: valid
    }

    // Check password requirements
    val isValid = password.length >= 8 &&
                 password.any { it.isUpperCase() } &&
                 password.any { it.isDigit() }

    if (!isValid) {
        println("Password too weak. Must be at least 8 characters with uppercase and number.")
    }

} while (!isValid && attempts < 5)

if (isValid) {
    println("Password accepted after $attempts attempts!")
} else {
    println("Too many failed attempts. Account locked.")
}
```

Loop Control Statements

When to Use

- Use `break` when you need to exit a loop early
- Use `continue` when you need to skip the current iteration
- Use labels when working with nested loops

Statement	What It Does	When to Use It
`break`	Exits the current loop	When you've found what you're looking for
`continue`	Skips to next iteration	When you want to skip certain items
`break@label`	Exits labeled loop	When working with nested loops
`continue@label`	Skips to next iteration of labeled loop	When skipping iterations in outer loops

Practical Examples

Copy Code

```
// ==========================================
// BREAK STATEMENT EXAMPLES
// ==========================================

// Example 1: Finding a specific number in a list
// This demonstrates how to exit a loop once you find what you need
println("=== FINDING A NUMBER WITH BREAK ===")
val numbers = listOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
val targetNumber = 7
var found = false
var searchCount = 0
```

```
for (num in numbers) {
    searchCount++
    println("Checking number: $num")

    if (num == targetNumber) {
        found = true
        println("Found $targetNumber at position ${searchCount - 1}!")
        break  // Exit the loop immediately - no need to check remaining numbers
    }
}

if (found) {
    println("Search completed in $searchCount steps")
} else {
    println("Number $targetNumber not found")
}

// Example 2: Processing until a condition is met
// This shows how to stop processing when certain criteria are met
println("\n=== PROCESSING UNTIL CONDITION MET ===")
val temperatures = listOf(72, 75, 68, 80, 85, 90, 95, 88, 82, 78)
var comfortableCount = 0

for (temp in temperatures) {
    if (temp > 85) {
        println("Temperature $temp°F is too hot! Stopping processing.")
        break  // Stop when we find an uncomfortable temperature
    }

    if (temp in 68..78) {
        comfortableCount++
        println("Temperature $temp°F is comfortable")
    } else {
        println("Temperature $temp°F is acceptable")
    }
}

println("Found $comfortableCount comfortable temperatures before stopping")

// ============================================
// CONTINUE STATEMENT EXAMPLES
// ============================================

// Example 1: Skipping even numbers
// This demonstrates how to skip certain iterations
println("\n=== SKIPPING EVEN NUMBERS ===")
for (i in 1..10) {
    if (i % 2 == 0) {
        continue  // Skip to next iteration when number is even
    }
    // This code only runs for odd numbers
    println("Processing odd number: $i")
    val square = i * i
    println("  $i squared is $square")
}

// Example 2: Processing only valid data
// This shows how to skip invalid or unwanted data
println("\n=== PROCESSING VALID DATA ONLY ===")
val userInputs = listOf("123", "abc", "456", "", "789", "def", "0")
var validNumbers = 0
var total = 0

for (input in userInputs) {
    // Skip empty strings
    if (input.isEmpty()) {
        println("Skipping empty input")
        continue
        }

        // Skip non-numeric inputs
        val number = input.toIntOrNull()
        if (number == null) {
            println("Skipping non-numeric input: '$input'")
            continue
        }

        // Skip zero values
        if (number == 0) {
            println("Skipping zero value")
            continue
        }

        // Process valid numbers
        validNumbers++
        total += number
        println("Processing valid number: $number (running total: $total)")
    }

    println("Processed $validNumbers valid numbers with total: $total")

    // ============================================
    // LABELED LOOPS EXAMPLES
    // ============================================

    // Example 1: Breaking out of nested loops
    // This shows how to exit multiple loop levels at once
    println("\n=== BREAKING OUT OF NESTED LOOPS ===")
    outerLoop@ for (i in 1..3) {
        println("Outer loop iteration: $i")

        for (j in 1..3) {
            println("  Inner loop: i=$i, j=$j")

            // Break out of both loops when specific condition is met
            if (i == 2 && j == 2) {
                println("  Breaking out of both loops!")
                break@outerLoop  // Exit the outer loop (and inner loop)
            }

            // Simulate some work
            Thread.sleep(200)
        }

        // This line won't execute when break@outerLoop is used
        println("Completed outer loop iteration $i")
    }

    println("Both loops completed")

    // Example 2: Continuing outer loop from inner loop
    // This demonstrates how to skip to the next iteration of an outer loop
    println("\n=== CONTINUING OUTER LOOP FROM INNER LOOP ===")
    mainLoop@ for (row in 1..4) {
        println("Processing row $row")

        for (col in 1..3) {
            println("  Processing column $col")

            // Skip to next row if we encounter a problem
            if (row == 2 && col == 2) {
                println("  Problem encountered! Moving to next row.")
                continue@mainLoop  // Skip to next iteration of mainLoop
            }

            // Simulate processing
            println("  Successfully processed position ($row, $col)")
        }

        // This line won't execute when continue@mainLoop is used
        println("Completed processing row $row")
    }

    println("All rows processed")
```

Conditional Statements

When to Use

- Use `if` for simple yes/no decisions
- Use `if-else` when you need two alternatives
- Use `if-else-if` for multiple conditions
- Use `when` for complex decision trees

Statement	What It Does	When to Use It
`if`	Executes code if condition is true	Simple yes/no decisions
`if-else`	Chooses between two alternatives	When you need two options
`if-else-if`	Handles multiple conditions	When you have many options
`when`	Matches value against patterns	Complex decision trees

Practical Examples

Copy Code

```
// ============================================
// IF STATEMENT EXAMPLES
// ============================================

// Example 1: Simple age check
// This demonstrates basic if statement usage
println("=== SIMPLE AGE CHECK ===")
val age = 20

if (age >= 18) {
    // This block only executes when age is 18 or greater
    println("You are an adult!")
    println("You can vote!")
    println("You can drive!")
}

// Example 2: Temperature-based clothing advice
// This shows how to use if statements for practical decisions
println("\n=== TEMPERATURE-BASED CLOTHING ADVICE ===")
val temperature = 75

if (temperature > 80) {
    println("It's hot outside!")
    println("Wear light clothing")
    println("Don't forget sunscreen!")
} else {
    // This block executes when temperature is 80 or less
    println("It's comfortable outside")
    println("Regular clothing should be fine")
}

// ============================================
// IF-ELSE-IF EXAMPLES
// ============================================

// Example 1: Grade calculation system
// This demonstrates handling multiple conditions in order
println("\n=== GRADE CALCULATION SYSTEM ===")
val score = 85

if (score >= 90) {
    println("Grade: A")
    println("Excellent work!")
} else if (score >= 80) {
    // This executes when score is 80-89
    println("Grade: B")
    println("Good job!")
} else if (score >= 70) {
    // This executes when score is 70-79
    println("Grade: C")
    println("Satisfactory")
} else if (score >= 60) {
    // This executes when score is 60-69
    println("Grade: D")
    println("Needs improvement")
} else {
    // This executes when score is below 60
    println("Grade: F")
    println("Failed")
}

// Example 2: Weather-based activity recommendations
// This shows practical decision-making with multiple conditions
println("\n=== WEATHER-BASED ACTIVITIES ===")
val weather = "rainy"
val temperature2 = 65
```

```
if (weather == "sunny" && temperature2 > 70) {
    println("Perfect weather for outdoor activities!")
    println("Go to the beach, have a picnic, or play sports")
} else if (weather == "sunny" && temperature2 <= 70) {
    println("Sunny but cool")
    println("Good for a walk or light outdoor activities")
} else if (weather == "rainy") {
    println("It's raining")
    if (temperature2 > 60) {
        println("Warm rain - good for indoor activities")
    } else {
        println("Cold rain - stay inside and stay warm")
    }
} else if (weather == "snowy") {
    println("It's snowing!")
    println("Great for winter sports or building snowmen")
} else {
    println("Weather is unclear")
    println("Check the forecast for better planning")
}

// ============================================
// WHEN STATEMENT EXAMPLES
// ============================================

// Example 1: Day of week decisions
// This demonstrates the when statement for multiple choices
println("\n=== DAY OF WEEK DECISIONS ===")
val day = "Monday"

when (day) {
    "Monday", "Tuesday", "Wednesday", "Thursday", "Friday" -> {
        // Multiple values can share the same action
        println("It's a weekday")
        println("Time to work or study")
        println("Set your alarm for tomorrow")
    }
    "Saturday", "Sunday" -> {
        println("It's the weekend!")
        println("Time to relax and have fun")
        println("Sleep in if you want")
    }
    else -> {
        // This handles any other values
        println("Invalid day: $day")
        println("Please enter a valid day of the week")
    }
}

// Example 2: Type-based processing
// This shows how when can handle different data types
println("\n=== TYPE-BASED PROCESSING ===")
val x: Any = "Hello World"

when (x) {
    // Check if value is in a specific range
    in 1..10 -> {
        println("Single digit number: $x")
        println("This is a small number")
    }
    in 11..99 -> {
        println("Double digit number: $x")
        println("This is a medium number")
    }
    // Check the type of the value
    is String -> {
        println("String value: '$x'")
        println("Length: ${x.length} characters")
        println("Uppercase: ${x.uppercase()}")
    }
    is Boolean -> {
        println("Boolean value: $x")
        if (x) {
            println("This is true")
        } else {
            println("This is false")
        }
    }
    // Handle null case
    null -> {
        println("Value is null")
        println("No processing possible")
    }
    // Default case for any other types
    else -> {
        println("Unknown type: ${x::class.simpleName}")
        println("Value: $x")
    }
}

// Example 3: Complex when with expressions
// This demonstrates advanced when statement features
println("\n=== COMPLEX WHEN WITH EXPRESSIONS ===")
val userInput = "admin"
val userLevel = when {
    // Multiple conditions can be combined
    userInput == "admin" && userLevel > 100 -> "Super Admin"
    userInput == "admin" -> "Administrator"
    userInput == "moderator" -> "Moderator"
    userInput == "user" -> "Regular User"
    userInput.isEmpty() -> "Guest"
    userInput.length < 3 -> "Invalid Username"
    else -> "Unknown User"
}

println("User level determined: $userLevel")

// Example 4: When as an expression
// When can also return values directly
println("\n=== WHEN AS AN EXPRESSION ===")
val score2 = 87
val grade = when (score2) {
    in 90..100 -> "A"
    in 80..89 -> "B"
    in 70..79 -> "C"
    in 60..69 -> "D"
    else -> "F"
}

println("Score: $score2")
println("Grade: $grade")

// You can also use when to assign multiple variables
val (status, message) = when (score2) {
    in 90..100 -> "Pass" to "Excellent work!"
    in 80..89 -> "Pass" to "Good job!"
    in 70..79 -> "Pass" to "Satisfactory"
    in 60..69 -> "Pass" to "Barely passing"
    else -> "Fail" to "Needs improvement"
}

println("Status: $status")
println("Message: $message")
```

Tips for Success

- Always use curly braces `{}` with if statements and loops, even for single lines
- Choose the right loop type based on whether you know the number of iterations
- Use `when` instead of long if-else chains for better readability
- Add comments to explain complex conditions or nested structures
- Use meaningful variable names in loop counters and conditions
- Consider using `break` and `continue` to simplify complex loops

Common Mistakes to Avoid

- Creating infinite loops by forgetting to update loop variables
- Using `=` instead of `==` in conditions
- Forgetting to handle all possible cases in when statements
- Not using proper indentation in nested structures
- Creating overly complex nested conditions that are hard to read
- Using loops when a built-in function would work better

Best Practices

- Keep loops and conditions simple and focused
- Use early returns or breaks to avoid deep nesting
- Consider extracting complex conditions into well-named functions
- Use range operators (`..`, `until`) for cleaner loop conditions
- Add comments to explain the purpose of complex control structures
- Test edge cases in your conditions (boundary values, null cases)