# Conditional Statements and Loops

## Scala Fundamentals (Beginner Friendly)

## Learning Objectives

After this module, you will be able to:

- Understand conditional control flow in Scala
- Use `if`, `if-else`, and nested conditions
- Work with `for`, `while`, and `do-while` loops
- Apply conditions and loops in practical programs

## Part 1: Conditional Statements

## What are Conditional Statements?

Conditional statements allow a program to:

- Make **decisions**
- Execute code **based on conditions**

They control the **flow of execution**.

## if Statement – Theory

- Executes a block of code **only if condition is true**
- Condition must return a Boolean value (`true` or `false`)

Syntax:

```
if (condition) {
  // code
}
```

## if Statement – Example

```
val x = 10

if (x > 5) {
  println("x is greater than 5")
}
```

## if–else Statement – Theory

- Executes one block if condition is true
- Executes another block if condition is false

Syntax:

```
if (condition) {
  // true block
} else {
  // false block
}
```

## if–else Statement – Example

```
val marks = 45

if (marks >= 50) {
  println("Pass")
} else {
  println("Fail")
}
```

## if–else–if Ladder – Theory

- Used to test **multiple conditions**
- Conditions are checked **top to bottom**

## if–else–if Ladder – Example

```
val score = 85

if (score >= 90) {
  println("Grade A")
} else if (score >= 75) {
  println("Grade B")
} else if (score >= 50) {
  println("Grade C")
} else {
  println("Fail")
}
```

## Nested if – Theory

- One `if` inside another `if`
- Used for **complex decision-making**

## Nested if – Example

```kotlin
val age = 20
val hasID = true

if (age >= 18) {
  if (hasID) {
    println("Allowed")
  } else {
    println("ID Required")
  }
} else {
  println("Not Allowed")
}
```

## if as an Expression (Important)

In Scala:

- `if` returns a value
- Can be assigned to a variable

## if Expression – Example

```
val num = 7
val result = if (num % 2 == 0) "Even" else "Odd"
println(result)
```

## Part 2: Looping Statements

## What are Loops?

Loops are used to:

- Execute a block of code **multiple times**
- Reduce code repetition

## for Loop – Theory

- Used to iterate over a **range or collection**
- Most commonly used loop in Scala

Syntax:

```scala
for (i <- range) {
  // code
}
```

## for Loop – Example (Range)

```scala
for (i <- 1 to 5) {
  println(i)
}
```

## for Loop – Example (Collection)

```scala
val fruits = List("Apple", "Banana", "Mango")

for (fruit <- fruits) {
  println(fruit)
}
```

## for Loop with Condition (Guard)

```
for (i <- 1 to 10 if i % 2 == 0) {
  println(i)
}
```

## for Yield – Theory

- Used to create a **new collection**
- Returns values instead of printing

## for Yield – Example

```scala
val squares = for (i <- 1 to 5) yield i * i
println(squares)
```

## while Loop – Theory

- Executes code **while condition is true**
- Condition checked before execution

## while Loop – Example

```
var i = 1
while (i <= 5) {
  println(i)
  i += 1
}
```

## do–while Loop – Theory

- Executes code **at least once**
- Condition checked after execution

## do-while Loop – Example

```
var i = 1
do {
  println(i)
  i += 1
} while (i <= 5)
```

## Loop Control: break (Using Breaks)

Scala does not support `break` directly.
It uses `Breaks` object.

## break Example

```
import scala.util.control.Breaks._

breakable {
  for (i <- 1 to 10) {
    if (i == 5) break
    println(i)
  }
}
```

## Comparison of Loops

| Loop | Use Case |
|---|---|
| for | Fixed iterations, collections |
| while | Condition-based repetition |
| do-while | Executes at least once |

## Practical Example (Combined)

```scala
for (i <- 1 to 10) {
  if (i % 2 == 0) {
    println(s"Even: $i")
  } else {
    println(s"Odd: $i")
  }
}
```

## Summary

- Conditional statements control decision-making
- Loops control repetition
- Scala treats `if` as an expression
- `for` loop is preferred over `while`

## End of Module