Kotlin Basics

1. Variable Declarations

Kotlin supports three main types of variable declarations: `var`, `val`, and `const`.

1.1 `var`

- Mutable (modifiable) variable.
- The value assigned to the variable can be changed later.

Example:

```
var age = 25
age = 30  // Allowed
println(age)  // Output: 30
```

1.2 `val`

- Immutable (read-only) variable.
- Once assigned, the value cannot be changed, but it can point to a mutable object.

Example:

```
val name = "Alice"
name = "Bob"  // Compile-time error
```

1.3 `const`

- Immutable variable whose value is known and assigned at **compile time**.
- Typically used for constants in the global scope or inside objects.

Example:

```
const val PI = 3.14159
println(PI)  // Output: 3.14159
```

---

2. Conditionals

Kotlin provides two types of conditionals for decision-making: `if-else` and `when`.

2.1 `if-else`

- Works similarly to other programming languages.
- Can be used as an expression (returns a value) or a statement.

Example:

```
val doNotDisturb = false
if (doNotDisturb) {
    println("Notifications are disabled")
} else {
    println("Notifications are enabled")
}
// Output: Notifications are enabled
```

2.2 `when`

- A more concise alternative to `if-else if` chains.
- Similar to `switch` in other languages but more powerful.

Example:

```
val expression = 2
val output = when (expression) {
    1 -> "one"
    2 -> "two"
    else -> "other"
}
println(output)  // Output: two
```

Advanced `when` Example:

- With multiple branches for the same condition:

```
when (expression) {
    1, 2, 3 -> println("one, two, or three")
    in 4..10 -> println("between four and ten")
    else -> println("other")
}
```

---

3. First-Class Functions

Kotlin treats functions as **first-class citizens**, allowing you to:

1. Assign functions to variables.
2. Pass functions as arguments to other functions.
3. Return functions from other functions.

3.1 Assigning Functions to Variables

```
fun test(str: String) {
    println("Test: $str")
}

val printTest = ::test
fun main() {
    printTest("Hello!")  // Output: Test: Hello!
}
```

3.2 Passing Functions as Parameters

```
fun testLambda(str: String, myLambda: (String) -> Unit) {
    myLambda(str)
}

fun main() {
    val myFunction = { println("Lambda in action!") }
    testLambda("Hello", myFunction)  // Output: Lambda in action!
}
```

---

4. Lambda Expressions

A **lambda expression** is an anonymous function that can be passed as a parameter or assigned to a variable.

4.1 Syntax

```
val lambdaName: (InputType) -> ReturnType = { parameters -> body }
```

4.2 Basic Lambda Example

```
val myLambda = { println("Lambda in action") }
fun main() {
```

```
    myLambda()  // Output: Lambda in action
}
```

4.3 Lambda with Parameters

```
val sum = { x: Int, y: Int -> x + y }
fun main() {
    println(sum(5, 10))  // Output: 15
}
```

4.4 Lambda with Higher-Order Functions

```
fun testLambda(str: String, myLambda: (String) -> Int) {
    println("Result: ${myLambda(str)}")
}

val myFunction: (String) -> Int = { str -> str.length }
fun main() {
    testLambda("Hello", myFunction)  // Output: Result: 5
}
```

---

5. Practical Examples

5.1 Example: Using `when` and Lambdas

```
fun main() {
    val myLambda = { num: Int -> num * 2 }
    val value = 5

    when (value) {
        in 1..10 -> println("Value: ${myLambda(value)}")  // Output: Value:
10
        else -> println("Other")
    }
}
```

5.2 Example: Combining `val` and First-Class Functions

```
fun greet(name: String) {
    println("Hello, $name!")
}
```

```kotlin
val greeting = ::greet
fun main() {
    greeting("Alice")  // Output: Hello, Alice!
}
```

Key Takeaways

1. **Variable Declarations**:
   - Use `var` for mutable variables.
   - Use `val` for immutable variables.
   - Use `const` for compile-time constants.
2. **Conditionals**:
   - `if-else` is used for simple branching.
   - `when` is a powerful alternative to `switch`.
3. **First-Class Functions**:
   - Functions can be passed, stored, and returned like regular variables.
4. **Lambda Expressions**:
   - Provide concise syntax for inline function declarations.
   - Commonly used with higher-order functions.
5. **Functional Programming**:
   - Kotlin encourages the use of lambdas and first-class functions for clean and efficient code.