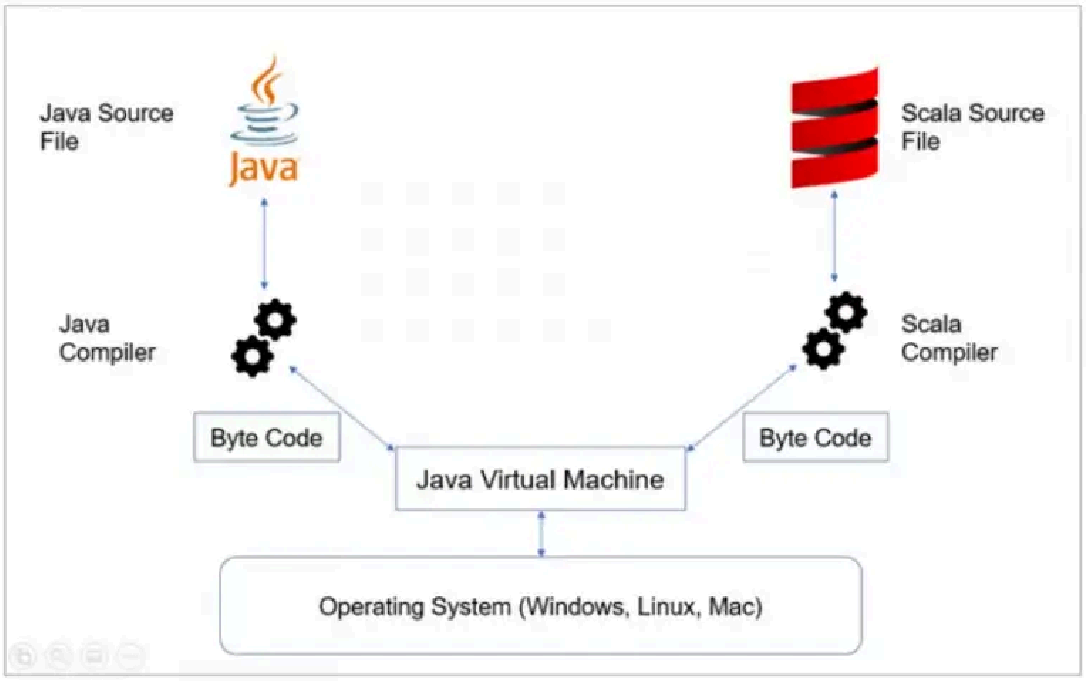
# Scala Fundamentals



Scala is a modern, high-level, multi-paradigm programming language that combines object-oriented and functional programming paradigms, designed to be concise, scalable, and run on the Java Virtual Machine (JVM) allowing seamless use of Java libraries.
Fundamental Scala concepts include:

- Basic Syntax: Scala is case-sensitive; class names start with uppercase letters, method names with lowercase. The semicolon at the end of lines is optional. Scala code is organized as objects and classes where an object is an instance of a class. A program starts execution from the main method within an object.
- Data Types: Scala has a rich type system with common types such as Int (32-bit integer), Double (64-bit floating point), String, Boolean, and Unit (void equivalent). Scala performs type inference so explicit type declarations are often optional.

- Variables: Scala supports immutable variables declared with `val` (cannot be reassigned) and mutable variables declared with `var` (can be reassigned). Functional programming encourages the use of immutable `val` variables in Scala.
- Control Structures: If-else expressions, match expressions (similar to switch), and loops (for, while, do-while) are core control flow constructs. Scala's if expressions return a value, making them useful in concise code.
- Functional Programming Support: Scala treats functions as first-class citizens. Functions can be assigned to variables, passed as arguments, and returned from other functions. Closures and higher-order functions are common in Scala.
- Classes, Objects, and Traits: Scala supports object-oriented design with classes (templates), objects (singletons), and traits (interfaces with concrete methods) for reusable code mixing. Methods contain logic and operate on data fields.
- Program Structure: A typical Scala program is a set of interacting objects. The source file name must match the object name containing the main method for the program to compile.

In summary, understanding Scala fundamentals involves mastering its syntax, type system, variable types, control flows, functional programming concepts, and object-oriented features like classes, traits, and objects. Scala is designed to take advantage of JVM while enabling expressive and concise code that blends object-oriented and functional programming styles
https://www.youtube.com/watch?v=I7-hxTbpscU

## 1. What is Scala?

Scala is a modern, concise, and flexible programming language that combines both object-oriented and functional programming paradigms. It runs on the Java Virtual Machine (JVM) and is fully interoperable with Java libraries.

## 2. Basic Syntax and Data Types

### Declaring Variables

- `val` for immutable values (like `final` in Java)
- `var` for mutable variables

```scala
val immutableValue: Int = 10
var mutableValue: String = "Hello"
mutableValue = "World"
```

## Common Data Types

- `Int` (integer numbers)
- `Double` (floating point)
- `Boolean` (`true` or `false`)
- `String`
- `Unit` (represents no meaningful value, similar to `void` in Java)

Example:

```scala
val isScalaFun: Boolean = true
val pi: Double = 3.14159
val name: String = "Scala"
```

## 3. Control Structures

### If-else Statement:

```
val x = 5
if (x > 0) println("Positive")
else println("Non-positive")
```

### Match Expression (like switch-case):

```
val day = "Monday"
day match {
  case "Monday"    => println("Start of week")
  case "Friday"    => println("End of work week")
  case _           => println("Midweek")
}
```

### For Loop

```
for (i <- 1 to 5) println(i)
```

## 4. Functions

Defining functions with or without return types:

```
def add(a: Int, b: Int): Int = a + b

def greet(name: String): Unit = {
  println(s"Hello, $name!")
}

println(add(3, 4))    // Output: 7
greet("Scala")        // Output: Hello, Scala!
```

Anonymous (lambda) function example:

```
val square = (x: Int) => x * x
println(square(5))  // Output: 25
```

## 5. Classes and Objects

Defining a simple class with fields and methods:

```
scala

class Person(var name: String, var age: Int) {
  def greet(): Unit = {
    println(s"Hi, my name is $name and I’m $age years
old.")
  }
}

val p = new Person("Shiva", 25)
```

```
p.greet()  // Output: Hi, my name is Shiva and I'm 25
years old.
```

Singleton object (like a static class):

```
object Main {
  def main(args: Array[String]): Unit = {
    println("Hello from the singleton object!")
  }
}
```

## 6. Collections

Basic list example:

```
val fruits = List("Apple", "Banana", "Cherry")
fruits.foreach(println)
```

Using map on collections:

```
val numbers = List(1, 2, 3)
val doubled = numbers.map(_ * 2)  // List(2, 4, 6)
```

## 7. Pattern Matching Example

```
def describe(x: Any): String = x match {
  case 5 => "Five"
  case "hello" => "Greeting"
  case _: Int => "Some integer"
  case _ => "Unknown"
}
```

```
println(describe(5))          // Five
println(describe("hello"))    // Greeting
println(describe(42))         // Some integer
println(describe(true))       // Unknown
```

## 8. Program Entry Point

Scala programs require an entry point defined by a main method inside an object:

```
object HelloWorld {
  def main(args: Array[String]): Unit = {
    println("Hello, Scala developer!")
  }
}
```

## Recap

- You declare values and variables with `val` and `var`.
- Control flow with `if`, `match`, and loops.
- Define functions easily, including anonymous ones.
- Work with classes for OOP, and objects for singletons.
- Use collections and powerful pattern matching.

If you want, I can provide a hands-on example project, detailed explanations on specific topics like functional programming, or more advanced Scala concepts!