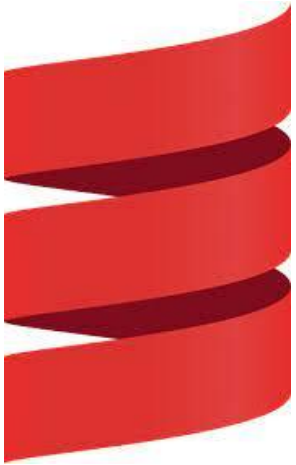


Scala Programming Language

Scala is an acronym for **Scalable Language**

Is so named because it was designed to grow with the demands of its users.

What is Scala

Scala compiles to Byte Code in the JVM. Scala programs compile to JVM bytecodes. Their run-time performance is usually on par with Java programs.

- You can write a .class being java or scala code.
- Interoperability with Java, So you can easily use the Java Ecosystem

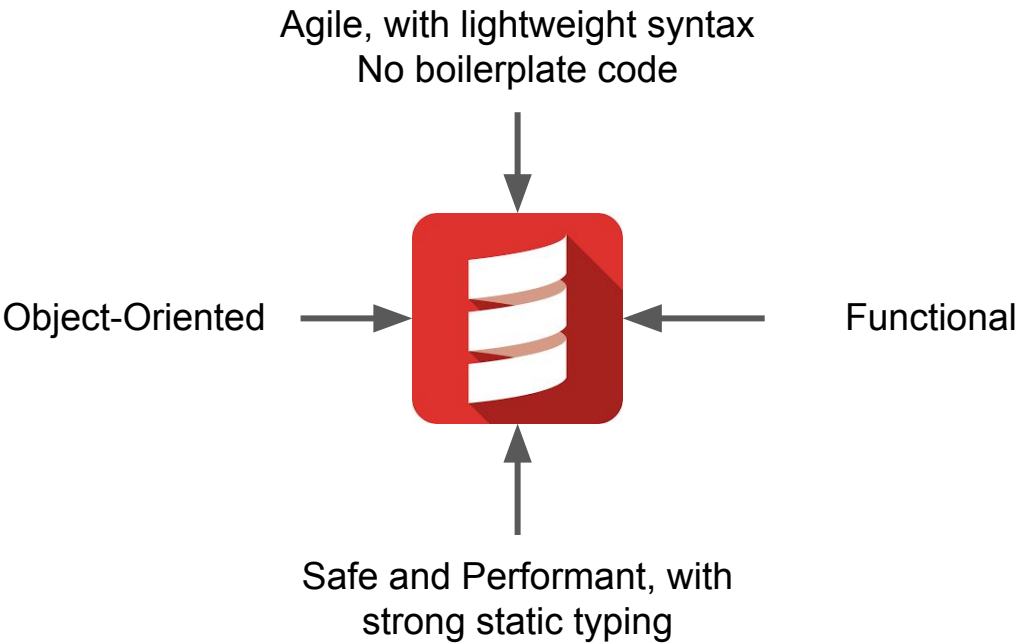
Why new language?

The work on Scala was motivated by **two hypotheses**:

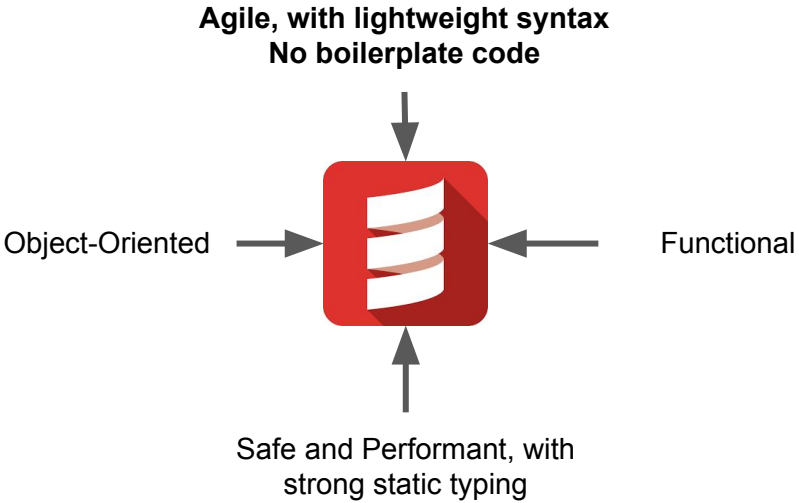
Hypothesis 1: A general-purpose language needs to be scalable; the same concepts should describe small as well as large parts.

Hypothesis 2: Scalability can be achieved by unifying and generalizing functional and object-oriented programming concepts.

Why new language?



Why new language?



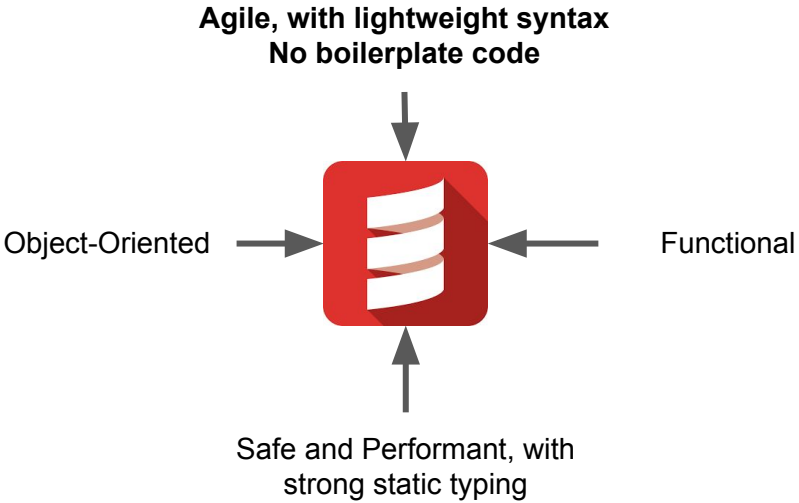
Java

```
public class HelloJava {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Scala

```
object HelloScala {
    def main(args: Array[String]): Unit = {
        println("Hello World!")
    }
}
```

Why new language?



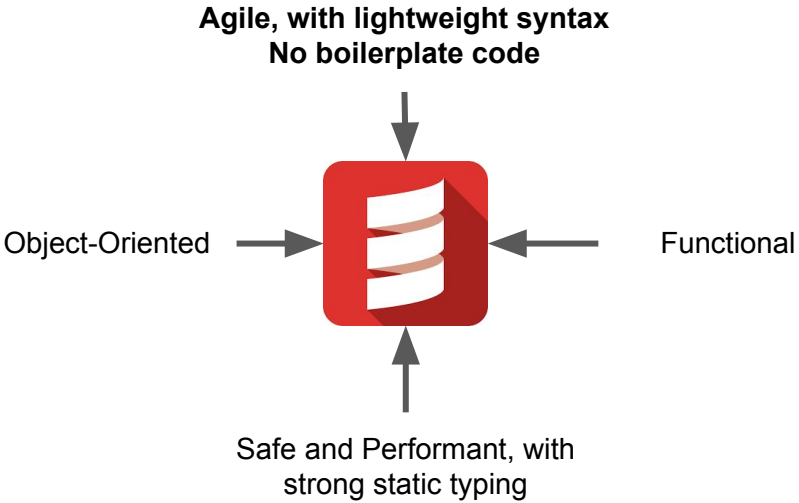
Java

```
List<String> list = new ArrayList<String>();
list.add("1");
list.add("2");
list.add("3");
```

Scala

```
val list = List("1", "2", "3")
```

Why new language?



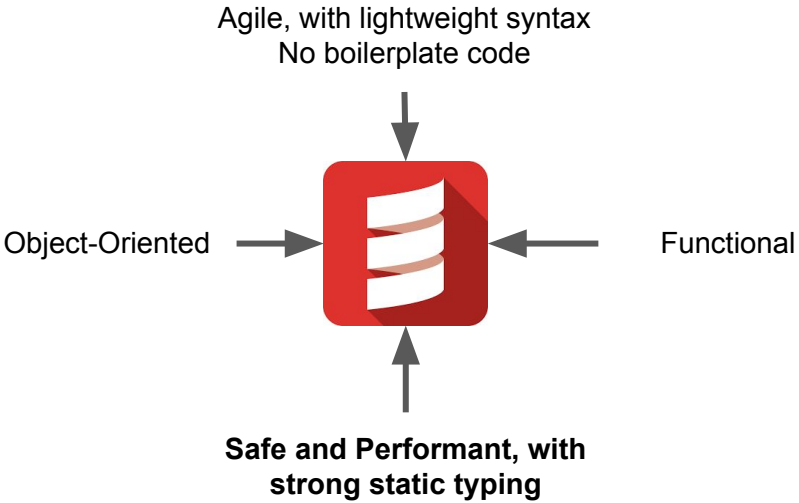
Java

```
List<Integer> ints = new ArrayList<Integer>();
for (String s : list) {
    ints.add(Integer.parseInt(s));
}
```

Scala

```
val ints = list.map(s => s.toInt)
```

Why new language?



Dynamic Typing

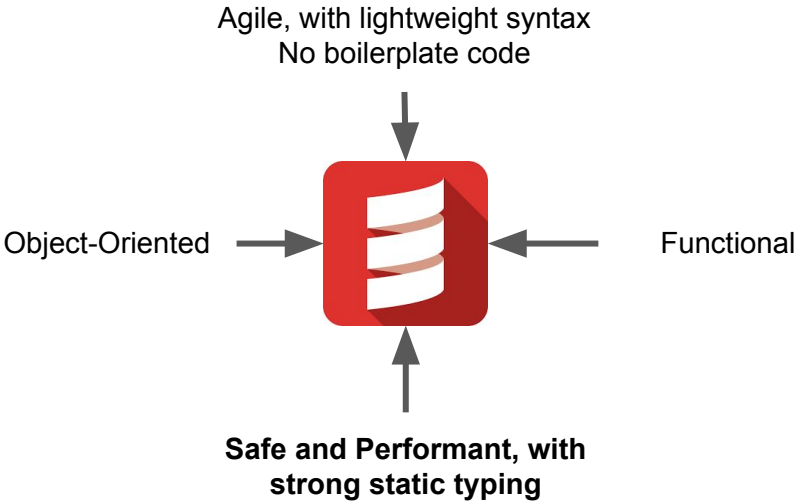
Python

```
def foo(a):
    if a > 0:
        print 'Hi'
    else:
        print "3" + 5
```

Javascript

```
function foo(a) {
    if ( a > 0 ) {
        console.log('Hi')
    } else {
        console.log("3" + 5)
    }
}
```

Why new language?

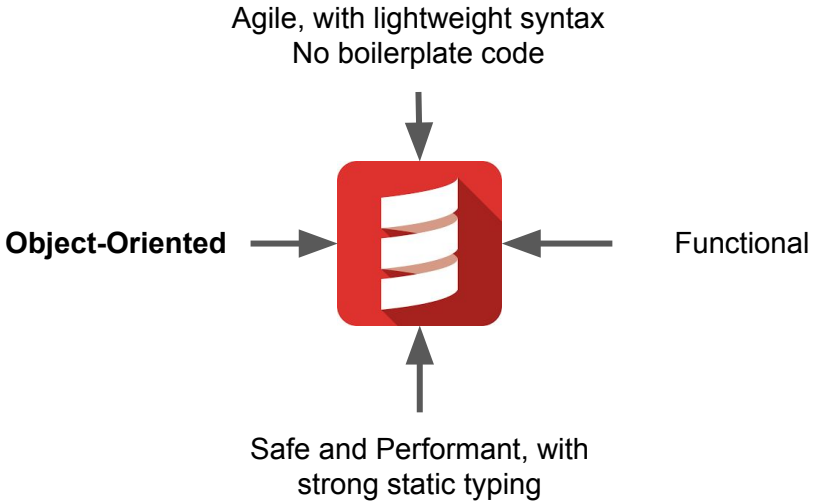


Static Typing

Scala

```
def foo(a:Int): Any = {
    if ( a > 0 ) {
        println("Hi");
    } else {
        println("3" + 5);
    }
}
```

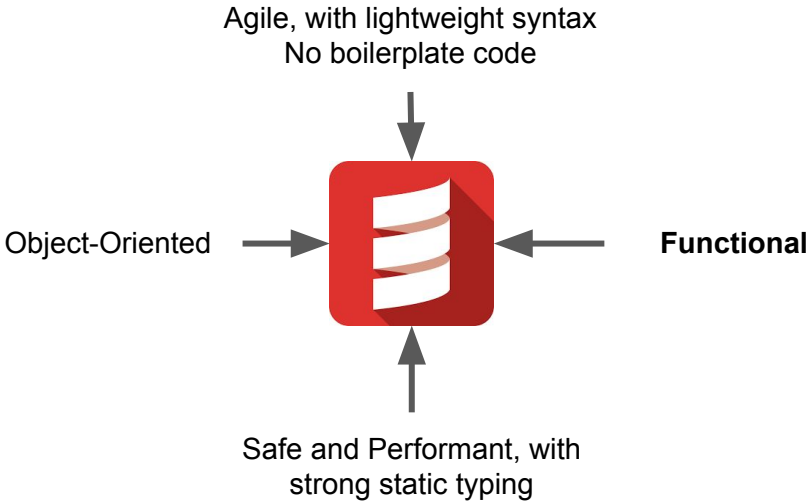
Why new language?



Pure O.O.

```
// Every value is an object
1.toString

// Every operation is a method call
1 + 2 + 3  (1).+(2).+(3)

// Can omit . and ( )
"abc" charAt 1  "abc".charAt(1)

// Classes (and abstract classes) like Java
abstract class Language(val name:String) {}

// Example implementations
class Scala extends Language("Scala")

// Anonymous class
val scala = new Language("Scala") {/*empty*/}
```

Why new language?

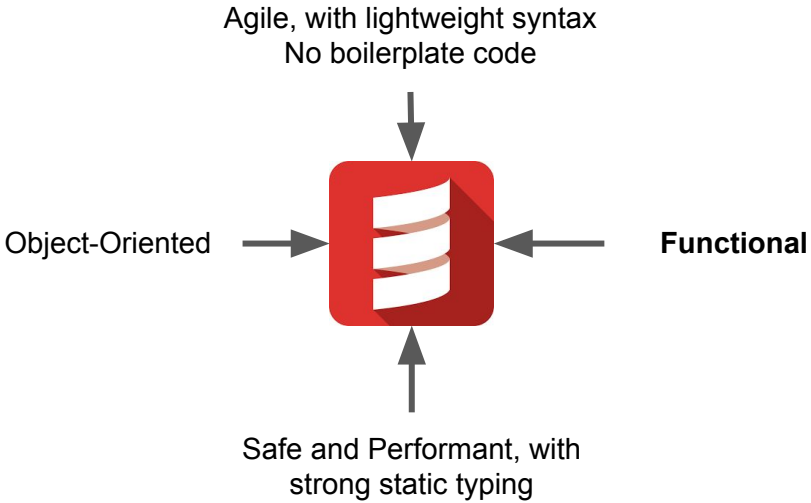


Functional Programming

Functional Programming language is one which does not have mutable variables, assignments, control structures.

It enables the construction of elegant programs that focus on functions.

Why new language?



Immutable

Examples:

```
- val age = 18 // In Scala
- final int age = 18 // Java
- const int age = 18 // C
```

Mutable

Examples:

```
- var age = 20 //In Scala
- age = 18 //Java, C, C++, C
```

Use Cases


samza


APACHE
Spark
TM


Flink


APACHE
kafka
®


play


Lift


akka


spray

Basic Syntax

If you have good understanding on Java, then it will be very easy for you to learn Scala.

Scala program can be defined as a collection of objects that communicate via invoking each others methods.

- **Object** - Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors - wagging, barking, eating. An object is an instance of a class.
- **Class** - A class can be defined as a template/blueprint that describes the behaviors/states that object of its type support.
- **Methods** - A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Fields** - Each object has its unique set of instant variables, which are called fields. An object's state is created by the values assigned to these fields.

- **Case Sensitivity -** Scala is case-sensitive, which means identifier **Hello** and **hello** would have different meaning in Scala.
- **Class Names -** For all class names, the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case. Example class **MyFirstScalaClass**
- **Method Names -** All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case. Example def **myMethodName()**
- **Program File Name -** Name of the program file should exactly match the object name. When saving the file, **you should save it using the object name** (Remember scala is case-sensitive) and append '.scala' to the end of the name (if the file name and the object name do not match your program will not compile). Example: Assume **'HelloWorld' is the object name**. Then, the file should be saved as **'HelloWorld.scala'**
- **def main(args: Array[String]) -** Scala program processing starts from the **main()** method, which is a mandatory part of every Scala Program.

Comments

```
object HelloWorld {
      /* This is my first java program.
       * This will print 'Hello World' as the output
       * This is an example of multi-line comments.
       */
      def main(args: Array[String]) {
            // Prints Hello World
            // This is also an example of single line comment.
            println("Hello, world!")
      }
}
```

New Lines

```
val s = "hello"; println(s)
```

Scala Packages

```
package com.liftcode.stuff
```

Import other packages, for example imports the contents of the scala.xml package

```
import scala.xml._
```

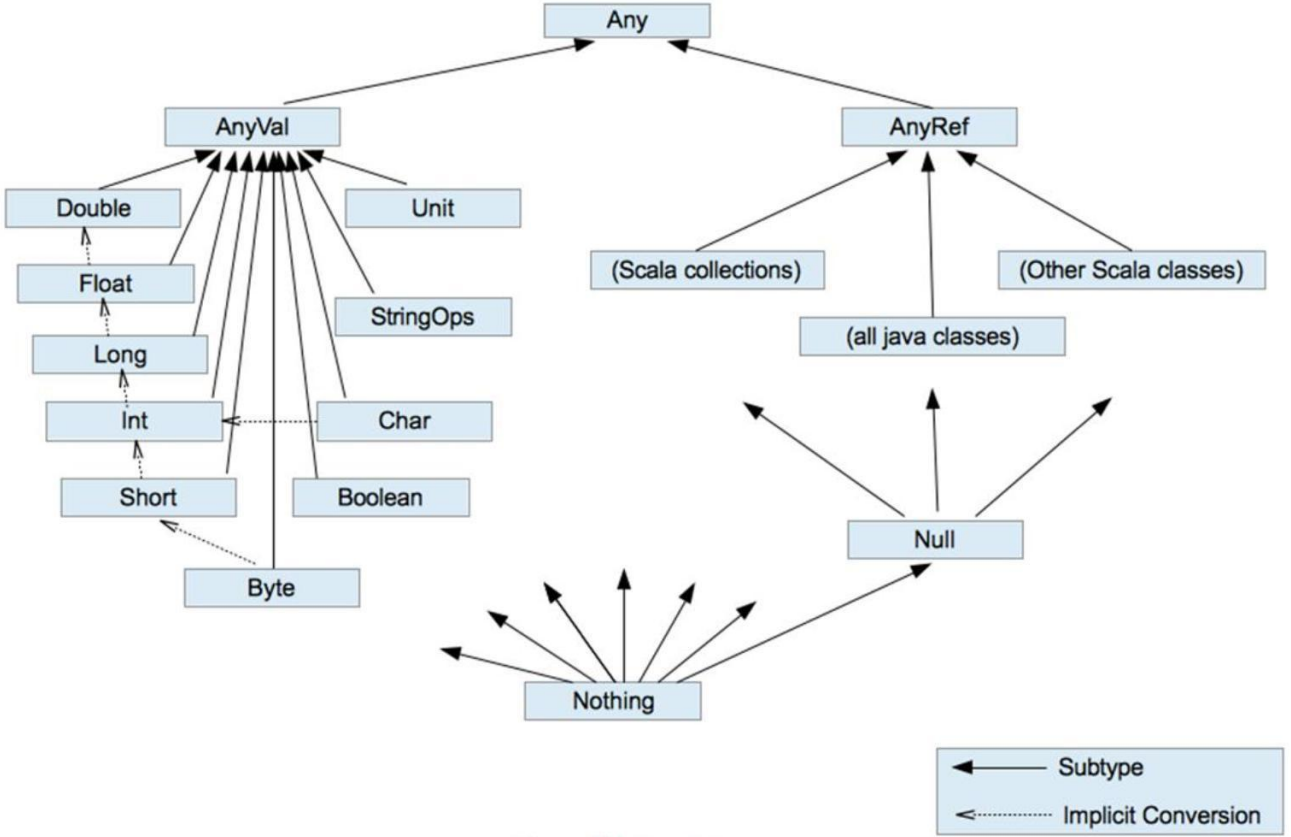
Import Single Class or Object, for example, HashMap from the scala.collection.mutable package:

```
import scala.collection.mutable.HashMap
```

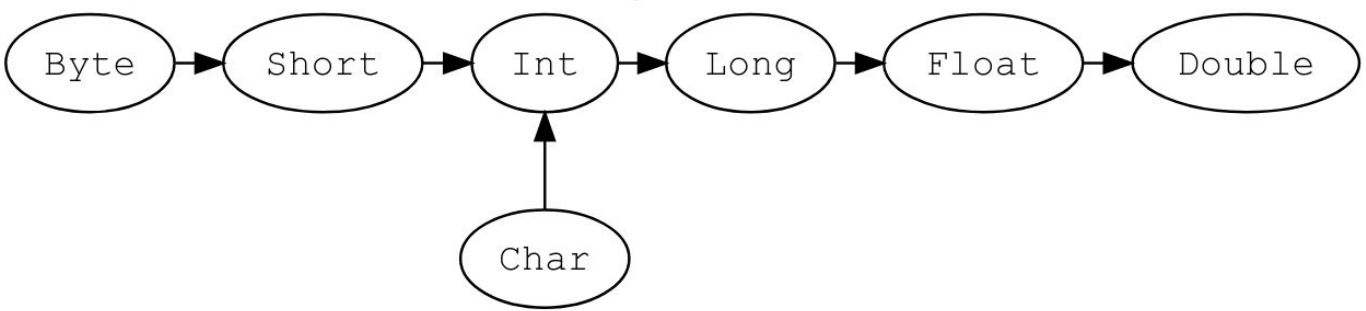
Import more than one class or object from a single package, for example, TreeMap and TreeSet from the scala.collection.immutable package:

```
import scala.collection.immutable.{TreeMap, TreeSet}
```

Type Hierarchy



Type Casting



Variables Declaration

The type of a variable is specified after the variable name and before equals sign. You can define any type of Scala variable by mentioning its data type as follows:

```
val or val variableName : DataType [= Initial Value]
```

If you do not assign any initial value to a variable, then it is valid as follows:

```
var myVar :Int;
val myVal :String;
```

Or with initial value

```
var myVar : String = "Foo"
val myVal : String = "Foo"
```

Variable Type Inference

```
var myVar = 10;

val myVal = "Hello, Scala!";
```

Multiple assignments

Scala supports multiple assignments. If a code block or method returns a Tuple, the Tuple can be assigned to a val

variable. [Note: We will study Tuple in subsequent chapters.]

```
val (myVar1: Int, myVar2: String) = Pair(40, "Foo")
```

Scala Access Modifiers

Members of **packages**, **classes** or **objects** can be labeled with the access modifiers **private** and **protected**, and if we are not using either of these two keywords, then access will be assumed as **public**.

- **private** member is visible only inside the class or object that contains the member definition
- A **protected** member is only accessible from subclasses of the class in which the member is defined.
- **public** member is every member that not labeled **private** or **protected** is public. There is no explicit modifier for public members.

Private Examples

```
class Outer {
    class Inner {
        private def f() { println("f") }
        class InnerMost {
            f() // OK
        }
    }
    (new Inner).f() // Error: f is not accessible
}
```

Protected Examples

```
package p {
     class Super {
          protected def f() { println("f") }
     }
     class Sub extends Super {
          f()
     }
     class Other {
          (new Super).f() // Error: f is not accessible
     }
}
```

Public Examples

```
class Outer {
    class Inner {
        def f() { println("f") }
        class InnerMost {
            f() // OK
        }
    }
    (new Inner).f() // OK because now f() is public
}
```

Scala Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

Scala is rich in built-in operators and provides the following types of operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators

Blocks

You can combine expressions by surrounding them with {}. We call this a block. The result of the last expression in the block is the result of the overall block, too:

```
println({
  val x = 1 + 1
  x + 1
}) // 3
```

The if Statement

An if statement consists of a Boolean expression followed by one or more statements .

```
if (Boolean_expression) {
// Statements will execute if the Boolean expression is true
}
```

The if … else Statement

An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.

```
if (Boolean_expression) {
// Statements will execute if the Boolean expression is true
} else {
// Statements will execute if the Boolean expression is true
}
```

The if...else if...else Statement

An if statement can be followed by an optional else if...else statement, which is very useful to test various conditions using single if...else if statement.

```
if (Boolean_expression 1) {
  //Executes when the Boolean expression 1 is true
} else if (Boolean_expression 2) {
  //Executes when the Boolean expression 2 is true
} else if(Boolean_expression 3) {
  //Executes when the Boolean expression 3 is true
} else {
  //Executes when the none of the above condition is true.
}
```

if … else assignment

```
var x = 30;
val str = if( x < 20 ) "if statement" else "else statement"
```

Loops

while loop

Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

do...while loop

Like a while statement, except that it tests the condition at the end of the loop body

for loop

Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.