**Scala Content**

**Class-1**

**Introduction to scala**

**Introduction to object oriented Programming language**

**Introduction to Functional Programming language**

**Class -2**

**Create Helloworld program using object , extends APP**

**Create a project in scala IDE**

**Scala Version: 2.12.2**

**Class-3**

**KeyWords: special keywords like ++, :+, etc..**

**Variables: var, val, multiple declarations**

**Variable Scopes**

**Class -4**

**Data Types:**

**Type Inference**

**SemiColon Preference**

**Class -5**

**Access Specifiers**

**Access Qualifiers**

**Strings**

* Testing String Equality
* Creating Multiline Strings
* Splitting Strings
* Substituting Variables into Strings
* Processing a String One Character at a Time
* Finding Patterns in Strings
* Replacing Patterns in Strings
* Extracting Parts of a String That Match Patterns
* Accessing a Character in a String
* Add Your Own Methods to the String Class

**Numbers**

* Parsing a Number from a String
* Converting Between Numeric Types (Casting)
* Overriding the Default Numeric Type
* Replacements for ++ and --
* Comparing Floating-Point Numbers
* Handling Very Large Numbers
* Generating Random Numbers
* Creating a Range, List, or Array of Numbers
* Formatting Numbers and Currency

**Control Structures**

* Looping with for and foreach
* Using for Loops with Multiple Counters
* Using a for Loop with Embedded if Statements (Guards)
* Creating a for Comprehension (for/yield Combination)
* Implementing break and continue
* Using the if Construct Like a Ternary Operator
* Using a Match Expression Like a switch Statement
* Matching Multiple Conditions with One Case Statement
* Assigning the Result of a Match Expression to a Variable
* Accessing the Value of the Default Case in a Match Expression
* Using Pattern Matching in Match Expressions
* Using Case Classes in Match Expressions
* Adding if Expressions (Guards) to Case Statements
* Using a Match Expression Instead of isInstanceOf
* Working with a List in a Match Expression
* Matching One or More Exceptions with try/catch
* Declaring a Variable Before Using It in a try/catch/finally Block
* Creating Your Own Control Structures

**Classes and Properties**

* Creating a Primary Constructor
* Controlling the Visibility of Constructor Fields
* Defining Auxiliary Constructors
* Defining a Private Primary Constructor
* Providing Default Values for Constructor Parameters
* Overriding Default Accessors and Mutators
* Preventing Getter and Setter Methods from Being Generated
* Assigning a Field to a Block or Function
* Setting Uninitialized var Field Types
* Handling Constructor Parameters When Extending a Class
* Calling a Superclass Constructor
* When to Use an Abstract Class
* Defining Properties in an Abstract Base Class (or Trait)
* Generating Boilerplate Code with Case Classes
* Defining an equals Method (Object Equality)
* Creating Inner Classes

**Methods**

* Controlling Method Scope
* Calling a Method on a Superclass
* Setting Default Values for Method Parameters
* Using Parameter Names When Calling a Method
* Defining a Method That Returns Multiple Items (Tuples)
* Forcing Callers to Leave Parentheses off Accessor Methods
* Creating Methods That Take Variable-Argument Fields
* Declaring That a Method Can Throw an Exception
* Supporting a Fluent Style of Programming

**Objects**

* Object Casting
* The Scala Equivalent of Java’s .class
* Determining the Class of an Object
* Launching an Application with an Object
* Creating Singletons with object
* Creating Static Members with Companion Objects
* Putting Common Code in Package Objects
* Creating Object Instances Without Using the new Keyword
* Implement the Factory Method in Scala with apply

**Packaging and Imports**

* Packaging with the Curly Braces Style Notation
* Importing One or More Members
* Renaming Members on Import
* Hiding a Class During the Import Process
* Using Static Imports
* Using Import Statements Anywhere

**Traits**

* Using a Trait as an Interface
* Using Abstract and Concrete Fields in Traits
* Using a Trait Like an Abstract Class
* Using Traits as Simple Mixins
* Limiting Which Classes Can Use a Trait by Inheritance
* Marking Traits So They Can Only Be Used by Subclasses of a Certain Type
* Ensuring a Trait Can Only Be Added to a Type That Has a Specific Method
* Adding a Trait to an Object Instance
* Extending a Java Interface Like a Trait

**Functional Programming**

* Using Function Literals (Anonymous Functions)
* Using Functions as Variables
* Defining a Method That Accepts a Simple Function Parameter
* More Complex Functions
* Using Closures
* Using Partially Applied Functions
* Creating a Function That Returns a Function
* Creating Partial Functions
* A Real-World Example

**Collections**

* Understanding the Collections Hierarchy
* Choosing a Collection Class
* Choosing a Collection Method to Solve a Problem
* Understanding the Performance of Collections
* Declaring a Type When Creating a Collection
* Understanding Mutable Variables with Immutable Collections
* Make Vector Your “Go To” Immutable Sequence
* Make ArrayBuffer Your “Go To” Mutable Sequence
* Looping over a Collection with foreach
* Looping over a Collection with a for Loop
* Using zipWithIndex or zip to Create Loop Counters
* Using Iterators
* Transforming One Collection to Another with for/yield
* Transforming One Collection to Another with map
* Flattening a List of Lists with flatten
* Combining map and flatten with flatMap
* Using filter to Filter a Collection
* Extracting a Sequence of Elements from a Collection
* Splitting Sequences into Subsets (groupBy, partition, etc.)
* Walking Through a Collection with the reduce and fold Methods
* Extracting Unique Elements from a Sequence
* Merging Sequential Collections
* Merging Two Sequential Collections into Pairs with zip
* Creating a Lazy View on a Collection
* Populating a Collection with a Range
* Creating and Using Enumerations
* Tuples, for When You Just Need a Bag of Things
* Sorting a Collection
* Converting a Collection to a String with mkString

**List, Array, Map, Set (and More)**

* Different Ways to Create and Populate a List
* Creating a Mutable List
* Adding Elements to a List
* Deleting Elements from a List (or ListBuffer)
* Merging (Concatenating) Lists
* Using Stream, a Lazy Version of a List
* Different Ways to Create and Update an Array
* Creating an Array Whose Size Can Change (ArrayBuffer)
* Deleting Array and ArrayBuffer Elements
* Sorting Arrays
* Creating Multidimensional Arrays
* Creating Maps
* Choosing a Map Implementation
* Adding, Updating, and Removing Elements with a Mutable Map
* Adding, Updating, and Removing Elements with Immutable Maps
* Accessing Map Values
* Traversing a Map
* Getting the Keys or Values from a Map
* Reversing Keys and Values
* Testing for the Existence of a Key or Value in a Map
* Filtering a Map
* Sorting an Existing Map by Key or Value
* Finding the Largest Key or Value in a Map
* Adding Elements to a Set
* Deleting Elements from Sets
* Using Sortable Sets
* Using a Queue
* Using a Stack
* Using a Range

**Files and Processes**

* How to Open and Read a Text File
* Writing Text Files
* Reading and Writing Binary Files
* How to Process Every Character in a Text File
* How to Process a CSV File
* Pretending that a String Is a File
* Using Serialization
* Listing Files in a Directory
* Listing Subdirectories Beneath a Directory
* Executing External Commands
* Executing External Commands and Using STDOUT
* Handling STDOUT and STDERR for External Commands
* Building a Pipeline of Commands
* Redirecting the STDOUT and STDIN of External Commands
* Using AND (&&) and OR (||) with Processes
* Handling Wildcard Characters in External Commands
* How to Run a Process in a Different Directory
* Setting Environment Variables When Running Commands
* An Index of Methods to Execute External Commands

**Actors and Concurrency**

* Getting Started with a Simple Actor
* Creating an Actor Whose Class Constructor Requires Arguments
* How to Communicate Between Actors
* Understanding the Methods in the Akka Actor Lifecycle
* Starting an Actor
* Stopping Actors
* Shutting Down the Akka Actor System
* Monitoring the Death of an Actor with watch
* Simple Concurrency with Futures
* Sending a Message to an Actor and Waiting for a Reply
* Switching Between Different States with become
* Using Parallel Collections

**Command-Line Tasks**

* Getting Started with the Scala REPL
* Pasting and Loading Blocks of Code into the REPL
* Adding JAR Files and Classes to the REPL Classpath
* Running a Shell Command from the REPL
* Compiling with scalac and Running with scala
* Disassembling and Decompiling Scala Code
* Finding Scala Libraries
* Generating Documentation with scaladoc
* Faster Command-Line Compiling with fsc
* Using Scala as a Scripting Language
* Accessing Command-Line Arguments from a Script
* Prompting for Input from a Scala Shell Script
* Make Your Scala Scripts Run Faster

**Web Services**

* Creating a JSON String from a Scala Object
* Creating a JSON String from Classes That Have Collections
* Creating a Simple Scala Object from a JSON String
* Parsing JSON Data into an Array of Objects
* Creating Web Services with Scalatra
* Replacing XML Servlet Mappings with Scalatra Mounts
* Accessing Scalatra Web Service GET Parameters
* Accessing POST Request Data with Scalatra
* Creating a Simple GET Request Client
* Sending JSON Data to a POST URL
* Getting URL Headers
* Setting URL Headers When Sending a Request
* Creating a GET Request Web Service with the Play Framework
* POSTing JSON Data to a Play Framework Web Service

**Databases and Persistence**

* Connecting to MySQL with JDBC
* Connecting to a Database with the Spring Framework
* Connecting to MongoDB and Inserting Data
* Inserting Documents into MongoDB with insert, save, or +=
* Searching a MongoDB Collection
* Updating Documents in a MongoDB Collection
* Accessing the MongoDB Document ID Field
* Deleting Documents in a MongoDB Collection
* A Quick Look at Slick

**Interacting with Java**

* Going to and from Java Collections
* Add Exception Annotations to Scala Methods to Work with Java
* Using @SerialVersionUID and Other Annotations
* Using the Spring Framework
* Annotating varargs Methods
* When Java Code Requires JavaBeans
* Wrapping Traits with Implementations

**The Simple Build Tool (SBT)**

* Creating a Project Directory Structure for SBT
* Compiling, Running, and Packaging a Scala Project with SBT
* Running Tests with SBT and ScalaTest
* Managing Dependencies with SBT
* Controlling Which Version of a Managed Dependency Is Used
* Creating a Project with Subprojects
* Using SBT with Eclipse
* Generating Project API Documentation
* Specifying a Main Class to Run
* Using GitHub Projects as Project Dependencies
* Telling SBT How to Find a Repository (Working with Resolvers)
* Resolving Problems by Getting an SBT Stack Trace
* Setting the SBT Log Level
* Deploying a Single, Executable JAR File
* Publishing Your Library
* Using Build.scala Instead of build.sbt
* Using a Maven Repository Library with SBT
* Building a Scala Project with Ant

**Types**

* Creating Classes That Use Generic Types
* Creating a Method That Takes a Simple Generic Type
* Using Duck Typing (Structural Types)
* Make Mutable Collections Invariant
* Make Immutable Collections Covariant
* Create a Collection Whose Elements Are All of Some Base Type
* Selectively Adding New Behavior to a Closed Model
* Building Functionality with Types

**Idioms**

* Create Methods with No Side Effects (Pure Functions)
* Prefer Immutable Objects
* Think “Expression-Oriented Programming”
* Use Match Expressions and Pattern Matching
* Eliminate null Values from Your Code
* Using the Option/Some/None Pattern