**Imperative programing:**
is a programming paradigm that uses statements that change a program's state.

**OOPS programing language:**
is a programming paradigm based on the concept of "objects", which are data structures that contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods.

Java based functional programming languages like Scala, Clojure, Groovy etc. looks into these problems with a different angle and provides less complex and less error-prone solutions as compared to imperative programming. They provide immutability concepts out of the box and hence eliminate need of synchronization and associated risk of deadlocks or livelocks. Concepts like Actors, Agents and DataFlow variables provide high level concurrency abstraction and makes very easy to write concurrent programs.

**Team OOP** argues that
the concept of inheritance and encapsulation makes it easier to manage and manipulate data.

**Inheritance**: new objects taking on the attributes/methods of existing objects letting us reuse more code
**Encapsulation**: the data and methods related to a certain object being bound together, creating independent, protected entities

**Functional programing language** is a programming paradigm, a style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.

**Functional Programming** is a concept which treats functions as first class citizens. At the core of functional programming is immutability. It emphasizes on application of functions in contrast to imperative programming style which emphasizes on change in state. Functional programming has no side effects whereas programming in imperative style can result in side-effects.

- There is a complete separation between the data of a program, and the behaviors of a program
- All objects created in functional programming are immutable (once something is created, it can't be changed)
- Shared state is avoided (objects do not share scope with other objects)

**Immutable state** - The state of an object doesn't change and hence need not be protected or synchronized.

**Functions as first class citizens** - There was a major shift in the way programs were written when Object oriented concepts came into picture. Everything was conceptualized as object and any action to be performed was treated as method call on objects. Hence there is a series of method calls executed on objects to get the desired work done. In functional programming world, it's more about thinking in terms of communication chain between functions than method calls on objects. This makes functions as first class citizens of functional programming since everything is modelled around functions.

**Higher-order functions** - Functions in functional programming are higher order functions since following actions can be performed with them.

1. Functions can be passed within functions as arguments.
2. Functions can be created within functions just as objects can be created in functions
3. Functions can be returned from functions

**Functions with no side-effects -** if a function takes an input and returns some output, multiple invocation of that function at different point of time will always return same output as long as input remains same.

Jvm based Functional Programming languages:

- Scala
- Clojure
- Groovy
- Lambda Expressions in Java 8.

A pure function is a function where:

1. The return value only depends on the input (if you input the same value, you will always return the same value)
2. There are no side effects (for example: no network or database calls which could affect the return value)
3. They do not alter the data that was passed into them. We only want to describe how the input will be changed (think destructive vs non-destructive)

**Team FP** argues that
the separation of data and methods, as well as the high level of abstraction leave less room for errors.

**Groovy** is a dynamic language with some support for functional programming. Groovy can be considered weakest in terms of functional programming features.

However because of its dynamic nature and close resemblance to Java, it has been widely accepted and considered good alternative to Java.

**Scala** is a statically typed multi-paradigm programming language designed to integrate features of object oriented programming and functional programming.

Since it is static, one cannot change class definition at run time i.e. one cannot add new methods or variables at run-time.

However Scala does provide functional programming concepts i.e. immutability, higher-order functions, nested functions etc.

Apart from supporting Java concurrency model, it also provides concept of Actor model out of the box for event based asynchronous message passing between objects.

The code written in Scala gets compiled into very efficient bytecode which can then be executed on JVM.

**Groovy** does not provide immutable objects out of the box but has excellent support for higher order functions. Immutable objects can be

| | |
|---|---|
| In Groovy functions can be passed around just as any other variable in the form of Closures | created with annotation @Immutation, but it's far less flexible than immutability support in Scala and Clojure. |

Note: Information gathered in this document has been collected from various sources on the Internet.