# Assignment-1:Exploring Programming Paradigms

**Presented by P.Jivan Prasadd**
**CB.EN.U4CYS21054**
**TIFAC-CORE in Cyber Security**
**Amrita Vishwa Vidyapeetham, Coimbatore Campus**

22nd January 2024


AMRITA
VISHWA VIDYAPEETHAM

# Contents



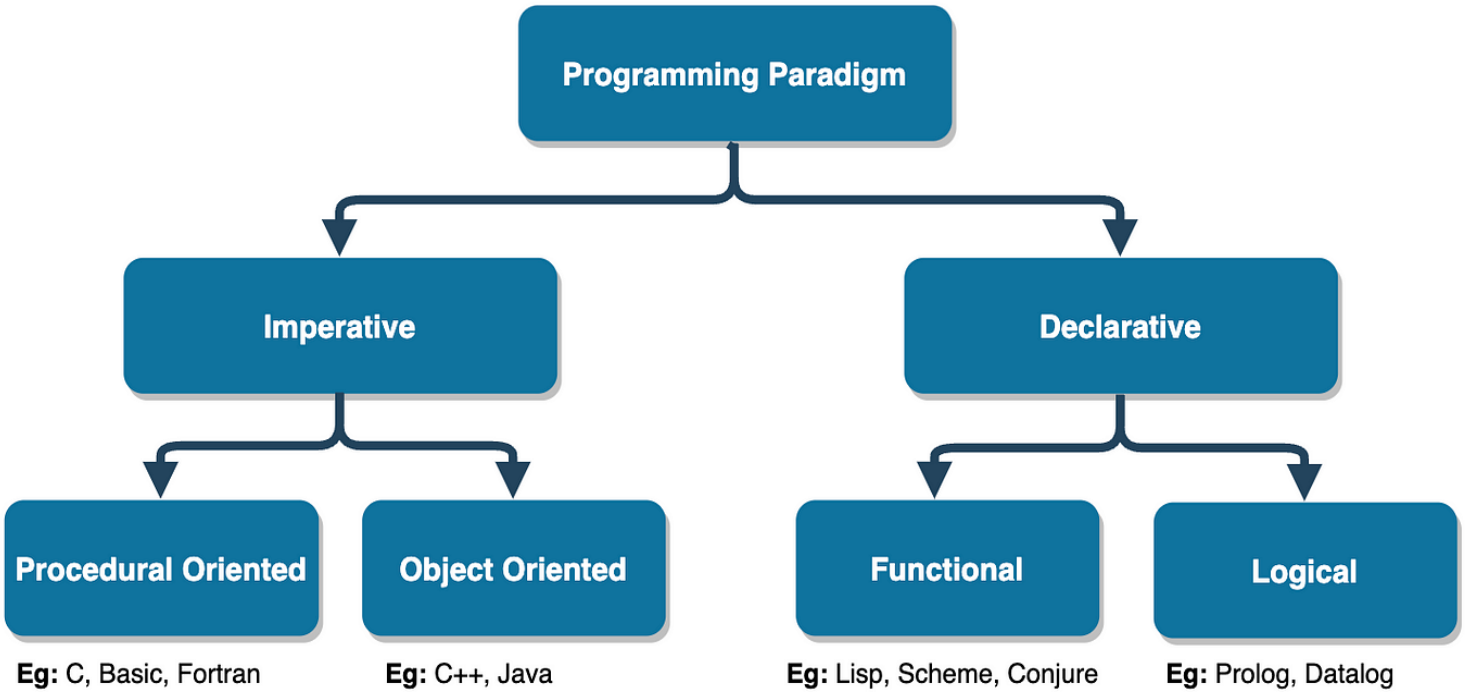
# Introduction

**Programming Paradigms:**Programming paradigms are different ways or styles in which a given program or programming language can be organized.

Topics of interest to us:

**Functional (Scala):** Emphasizes immutability and pure functions.

**Declarative (HiveQL):** Focuses on describing what should be done rather than how.



# Paradigm 1: Functional

**What is Functional Programming?**

- Functional programming (FP) is an approach to software development that uses pure functions to create maintainable software.
- It involves building programs by applying and composing functions.

**Types of Functions:**

- **Pure Functions:** Functions that always produce the same result for the same input and have no side effects.
- **Impure Functions:** Functions that may have side effects or produce different results for the same input due to state changes.

**Real World Use-cases:**

- Functional programming ensures accuracy in financial calculations through immutability and pure functions.
- Functional programming aids in efficient data processing, enabling flexible pipelines and supporting data-driven decision-making.

# Need for functional Programming

- Functional programming was conceived to overcome the limitations of imperative programming, where emphasis on mutable state and side effects often led to complex, error-prone code.

**Pros of functional Programming:**

- Functional programming is famous for its high-level abstractions that hide a large number of details of such routine operations like iterating.
- Lazy evaluation: Functional programming encourages lazy evaluation, which means that the value is evaluated and stored only when required.

**Cons of functional Programming:**

- Potentially poorer performance: Immutable values combined with recursion might lead to a reduction in performance.
- Coding difficulties:Though writing pure functions is easy, combining it with the rest of the application and I/O operations can be tough.

- **Pure Functions**
  1. Given the same inputs, always returns the same output.
  2. Has no side-effects.

```
// pure
function getSquare(x) {
   return x * x;
}

// impure
function getSquare(items) {
  var len = items.length;
  for (var i = 0; i < len; i++) {
    items[i] = items[i] * items[i];
  }
  return items;
}
```

- **No Side Effects**
  - Side effects, such as I/O, logging, errors, network calls, and alteration of external data, are considered undesirable.
  - Anything that makes a system unpredictable is avoided.
  - If a function contains side effects, it is called a procedure.

# Principles and concepts Functional Programming (Cont.)

- **Immutability**
  - At the core of functional programming.
  - Once a value is declared, it is unchangeable, making behaviors predictable.
- **Higher Order Functions**
  1. Takes one or more functions as arguments.
  2. Returns a function as its result.
- **Functions as First-Class Entities**
  - Functions can be passed as arguments, returned as values, stored in data structures, and assigned to variables.

```
function add(left, right) {
 return left + right;
}

const adder = add;

adder(2,3);
```

# What is Scala

- Scala is a strongly typed programming language.It is designed with functional principles in mind.
- Its strong type system and conciseness make it one of the most pleasant languages to work with.
- It runs on a Java virtual machine (JVM), making it compatible with all byte-code-related languages such as Java, Kotlin, etc.

**How is Scala Functional?**

- Scala is functional in the sense that each function is a value in Scala.
- Functions are first-class citizens in Scala and can be assigned names.
- Scala supports currying and anonymous functions, allowing programmers flexibility to write clean, concise, and elegant code.

# Scala's Characteristics

**Composing functions:** Function composition is the process of combining two or more functions to create a new function. This new function applies the result of one function as the input to another, forming a chain of operations.

**Higher Order function:** A higher-order function (HOF) is often defined as a function that

- takes other functions as input parameters or
- returns a function as a result. In Scala, HOFs are possible because functions are first-class values.

**Pattern Matching:** Pattern matching is a powerful feature of the Scala language. It allows for more concise and readable code while at the same time providing the ability to match elements against complex patterns.

```
def patternMatching(candidate: String): Int = {
  candidate match {
    case "One" => 1
    case "Two" => 2
    case _ => -1
  }
}
```

# Scala's Characteristics (Cont)

**Asynchronous and parallel programming:**

- The true non-blocking power comes from actions that do not block either you (the calling thread) or someone else (some secondary thread). Best exemplified with an Akka actor.
- The power of Akka comes from the fact that you can create a huge amount of actors (millions per GB of heap), so that a small number of threads can operate on them in a smart way, via scheduling.

**Extensible language:** The language comes with built-in features such as implicits, operator overloading, macros, etc., which allow you to create other Domain Specific Languages (DSL).

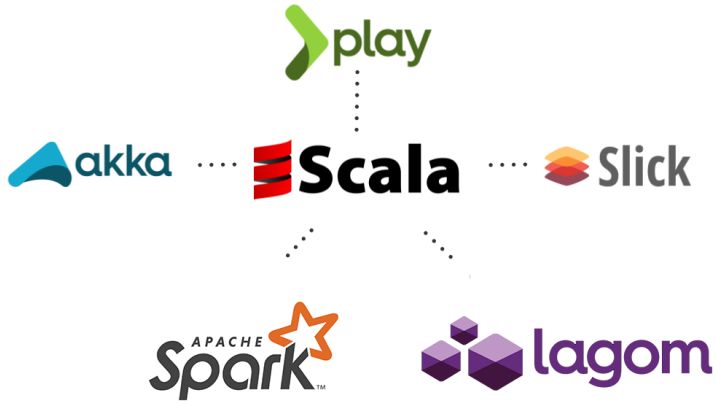



**Figure:** Applications

# Declarative Programming

**What is Declarative Programming:** Declarative programming is a programming paradigm in which the programmer defines what needs to be accomplished by the program without defining how it needs to be implemented.

**Types of Declarative Languages:**

1. **Logic Programming Languages:**
   - These languages state a program as a set of logical relations (e.g., a grandchild is the child of someone's child).
   - Logic programming languages are similar to the SQL database language.
2. **Functional Languages:**
   - These programming languages have a mathematical style.
   - You can build a functional program by applying functions to arguments.
   - Functional languages like LISP, ML, and Haskell are used as research tools in language development, in automated mathematical theorem provers, and even to some extent in commercial projects.

# Principles of Declarative Programming

- **Higher-order Abstractions:**
  - Constructs allowing expression of complex logic with simpler, concise code.
  - Promotes code reuse and modularity.
- **Immutable Data:**
  - Data structures cannot be changed after creation.
  - Prevents bugs from unintended side effects and enhances memory management.
- **Pure Functions:**
  - Always produce the same output with the same input, no side effects.
  - Facilitates easier testing and debugging due to predictability.
- **Declarative DSLs (Domain Specific Languages):**
  - Programming languages designed for specific domains.
  - Expresses domain requirements more concisely and naturally.

# HiveQL's Characteristics

- **Scalability and Performance:**
  - Hive is scalable, fast, and suitable for large datasets.
- **SQL-Inspired Language:**
  - Hive uses an SQL-inspired language (Hive Query Language - HQL) that leverages familiar relational database concepts, such as tables, columns, rows, and schema.
- **Data Partitioning:**
  - Hive uses directory structures to "partition" data, improving performance on specific queries, particularly in Hadoop's flat-file environment.
- **Hadoop Infrastructure Execution:**
  - Hive executes queries on Hadoop's infrastructure rather than on a traditional database, leveraging the power of distributed computing.
- **File Format Support:**
  - Hive supports various file formats including ORC, SEQUENCEFILE, RCFILE (Record Columnar File), and TEXTFILE.
- **Support for Partition and Buckets:**
  - Hive supports partitioning and buckets, providing fast and straightforward data retrieval options.
- **Separation of Schema and Data:**
  - Schema information is stored in a database, while processed data is stored in the Hadoop Distributed File System (HDFS).

# Applications of Declarative Programming

**Applications:**

- **SQL (Structured Query Language):** Used for managing relational databases.
- **HTML (HyperText Markup Language):** Used for structuring content on the web.
- **CSS (Cascading Style Sheets):** Used to style and format HTML documents.

**Advantages:**

- Declarative code is often easier to read and understand, as it tends to be more concise and less cluttered with implementation details.
- Declarative code can be more maintainable over time, as it tends to be more modular and less tightly coupled to specific implementation details.

**Disadvantages:**

- Declarative programming can be less precise and fine-grained than imperative programming, limiting control over the code's flow.
- It can be less performant than imperative programming, involving more steps to accomplish a particular task.
- It has a steeper learning curve than imperative programming, requiring developers to think about problems differently and learn new programming concepts.

# Similarities and Differences

## Similarities

- **Modularity:** Both functional and declarative programming promote modularity, making it easier to manage and maintain code by breaking it into smaller, independent pieces.
- **Abstraction:** Both functional and declarative programming languages emphasize abstraction, allowing developers to express high-level concepts without dealing with low-level implementation details.
- **Readability:** Both paradigms strive for code readability by minimizing boilerplate code and emphasizing clear, concise expressions of intent.

## Differences:

Functional programming is a subset of declarative programming.The differences lie in their specific focuses and the variety of declarative styles that extend beyond the functional paradigm.

**Focus on Functions:**

- **Functional Programming:** Centers around functions and mathematical operations.
- **Declarative Programming:** Encompasses various paradigms, focusing on expressing what should be done.

**Control Flow:**

- **Functional Programming:** Utilizes recursion and higher-order functions.
- **Declarative Programming:** Involves various control flow mechanisms (e.g., logical relations, constraints).

# References


https://techvidvan.com/tutorials/apache-hive-featurehttps://k21academy.com/big-data-engineer/introduction-to-hive-its-features-limitations/s/

https://www.digitalocean.com/community/tutorials/functional-imperative-object-oriented-programming-comparison

https://www.freecodecamp.org/news/the-principles-of-functional-programming/

https://pandaquests.medium.com/advantages-and-disadvantages-in-declarative-and-imperative-programming-in-

https://www.educba.com/hiveql/

https://www.upgrad.com/blog/apache-hive-architecture-commands/

https://www.turing.com/kb/introduction-to-functional-programming

https://www.slideshare.net/datamantra/functional-programming-in-scala-72060420

https://reintech.io/terms/tech/scala-programming-language?page_t=5

https://www.geeksforgeeks.org/scala-function-composition/

https://typeset.io/conferences/principles-and-practice-of-declarative-programming-2nqewraj

https://www.simplilearn.com/what-is-hive-article

https://www.freecodecamp.org/news/an-introduction-to-programming-paradigms/