Functional Programming

Faaiz Hussain Shah

University of Montpellier
(BforeAI)

11/10/2024

Agenda

Programming Paradigms

- Paradigm: In science, a *paradigm* describes distinct concepts or thought patterns in some scientific discipline.[1]
- A programming paradigm refers to a style or way of thinking about and structuring programs.
- Each paradigm provides a different approach to solving computational problems, and different languages often emphasize one or more paradigms.

[1] Martin Odersky

Programming Paradigms

Main programming paradigms:

- Imperative programming
- Functional programming
- Logic programming

Programming Paradigms: ”Imperative”

Definition

It is a paradigm where you write **step-by-step instructions** that tell the computer **how** to perform tasks. It focuses on how to achieve the desired result by explicitly managing the state of the program through variables, loops, and conditionals.

- Commands the computer to perform actions in a sequence of steps.
- Relies heavily on mutable state (variables that can be changed)
- Uses control flow structures such as loops (for, while) and conditionals (if, else).

Examples : C, Java, Python (can also be used imperatively).

Programming Paradigms: ”Functional”

Definition

It is a paradigm where **computation is treated as the evaluation of mathematical functions** and avoids changing state or mutable data. It emphasizes what to do rather than how to do it, relying on pure functions, immutability, and higher-order functions

- **Pure Functions:** Functions have no side effects; the output only depends on the inputs.
- **Immutability:** Data cannot be modified once created.
- **Function Composition:** Functions can be combined to create new functions.
- **First-class functions:** Functions are treated as first-class citizens and can be passed as arguments or returned as results

Examples : Haskell, Scala, Lisp, F#, and functional subsets of Python.

Programming Paradigms: ”Logic”

Definition

It is a paradigm that focuses on defining **facts and rules** that describe relationships between different entities, and then using a process of logical inference to solve problems or query the system. The programmer defines what needs to be achieved, and the language’s inference engine determines how to achieve it.

- Uses declarative statements instead of explicit instructions.
- Programs consist of facts (known information) and rules (logical conditions).
- Functions can be combined to create new functions.
- Solving problems involves logical inference based on the defined facts and rules.

Examples : Prolog

Object-Oriented Programming (OOP) - Orthogonal to Other Paradigms

Definition

(OOP) is often described as orthogonal to other paradigms, meaning it can be combined with other paradigms like imperative, functional, or logic programming. In OOP, programs are organized around objects, which are instances of classes that encapsulate both data (state) and behaviors (methods).

- **Encapsulation:** Objects encapsulate both state (fields) and behavior (methods).
- **Inheritance:** Objects can inherit properties and behavior from parent classes.
- **Polymorphism:** Methods can have different implementations based on the object's class.
- **Abstraction:** OOP abstracts complex systems into manageable objects.

Examples : Java, C++, Python (OOP subset), Scala (can combine OOP and FP).

Why Is OOP Considered Orthogonal?

OOP focuses on how data and behavior are bundled together in objects, but this can be combined with other paradigms:

- **Imperative OOP:** Managing object states through mutable fields (as in Java).
- **Functional OOP:** Using immutability and pure functions within objects (as in Scala).
- **Logic OOP:** Combining object-oriented models with logical inference engines.

Functional Programming

Difference Between a Function, Method, and Procedure ?

Functional Programming : ”Functions”

A function is a block of code that takes inputs (arguments), performs computations, and returns a value. Functions are generally ”pure,” meaning they do not have side effects, and their output is determined solely by their inputs.

Key Characteristics:

- Takes input and returns output
- Can be a pure function (output depends only on input, no side effects)
- Can be reused multiple times.

Example in Scala:

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

Functional Programming : ”Method”

A method is a function that is associated with an object or class. In object-oriented programming, methods are used to define behaviors for an object, operating on the object’s attributes (state).

Key Characteristics:

- Defined inside a class or object.
- Can access and modify the internal state (fields) of the object
- Methods can have side effects (e.g., changing object state).

Example in Scala:

```
class Calculator {
   def add(a: Int, b: Int): Int = a + b  // Instance method
}
```

Here, ”add” is a method of the ”Calculator” class and is called on an instance of Calculator.

Functional Programming : ”Procedure”

A procedure is a type of function or method that does not return any value. In some languages, a procedure is specifically defined as a block of code that performs an action but doesn’t explicitly return a result.

Key Characteristics:

- Does not return a value (in languages like Scala, it returns Unit, which is similar to void in Java or C).
- Often used for side effects (e.g., printing, updating a database, etc.).
- Methods can have side effects (e.g., changing object state).

Example in Scala:

```
1         def printMessage(msg: String): Unit = println(msg)
```

This procedure takes a string as input, prints it, but does not return anything (returns Unit).

Why Functional Programming Is Called ”Functional” ?

Functional programming is called ”functional” because the primary building blocks in this paradigm are functions. The core idea in functional programming (FP) is to treat computation as the evaluation of mathematical functions, and the focus is on using pure functions (without side effects) to build programs. In FP, functions::

- Are first-class citizens (can be passed as arguments, returned from other functions, stored in variables, etc.).
- Do not rely on or modify external state (this immutability helps eliminate side effects and makes programs easier to reason about).
- Encourage function composition (small, reusable functions can be combined to form more complex operations).

[1] Martin Odersky

Scala: Functional Programming

- Functional programming is a programming style that uses functions as a building block and avoids mutable variables, loops, and other imperative control structures
- It treats computation as an evaluation of mathematical functions, where the output of a function depends only on the arguments to the function.
- A program is composed of such functions.
- In addition, functions are first-class citizens in a functional programming language

Scala Multi-paradigm: Functional Programming

- Multi-paradigm languages are versatile, allowing developers to choose the best approach for a given problem
  - **Object-Oriented:** Organizes software design around data, or objects, rather than functions and logic. Objects represent data and methods to manipulaScala compiler enforces type safety at compile time. This helps reduce the number of bugs in an applicationte that data
  - **Functional:** Focuses on the evaluation of expressions rather than execution of commands. This paradigm emphasizes immutability (unchanging over time), higher-order functions (functions that take other functions as parameters or return them)
  - In addition, functions are first-class citizens in a functional programming language
- Scala integrates features from both object-oriented (like Java) and functional programming (like Haskell), allowing developers to use the most appropriate paradigm for their needs

Functional Programming Languages

- Besides Scala, there are several other programming languages that either fully support or have significant functional programming capabilities:
  - **Haskell:** Often considered the pure functional programming language due to its emphasis on immutability and pure functions.
  - **Erlang:** Used primarily for applications that require high availability, like telecoms and banking software
  - **F#:** A strongly typed, functional-first language that runs on the .NET framework.It supports both functional and object-oriented programming, making it a good choice for .NET developers looking to adopt functional techniques

Scala: Functions

- A function is a block of executable code
- In functional programming, an application is built entirely by assembling functions
- In functional programming languages, functions are first-class citizen.

Scala: Functions (First Class Citizens)

- A function has the same status as a variable or value
- It allows a function to be used just like a variable
- It is easier to understand this concept if you contrast FP functions with functions in imperative languages such as C
- Imperative languages treat variables and functions differently. For example, C does not allow a function to be defined inside another function. It does not allow a function to be passed as an input parameter to another function.

Scala: Functions (First Class Citizens)

In functional programming:

- a function to be passed as an input to another function
- It allows a function to be returned as a return value from another function
- A function can be defined anywhere, including inside another function
- It can be defined as an unnamed function literal just like a string literal and passed as an input to a function
- **Every statement is an expression** that returns a value. For example, the if-else control structure in Scala is an expression that returns a value.

Scala: Basic Scala Variable Types

Variable Type	Description
Byte	8-bit signed integer
Short	16-bit signed integer
Int	32-bit signed integer
Long	64-bit signed integer
Float	32-bit single precision float
Double	64-bit double precision float
Char	16-bit unsigned Unicode character
String	A sequence of Chars
Boolean	true or false

Figure: Basic Scala Variable Types [1]

[1] Guller, M. (2015). "Big data analytics with Spark", Apress.

What is Scala?

Scala is a modern (developed by Martin Odersky and released in 2004), multi-paradigm programming language designed to express common programming patterns in a concise, elegant, and type-safe way

Features:

- **Static Typing:** Scala is statically typed, which helps catch errors at compile-time rather than at run-time. Scala compiler enforces type safety at compile time. This helps reduce the number of bugs in an application
- **Object-Oriented:** Every value is an object and every operation is a method-call in Scala, similar to Java
- **Functional Programming:** Scala is also a functional language, supporting **first-class functions**, **immutability**, and **pattern matching**

Scala and Its Ecosystem

- **JVM Compatibility:** Scala runs on the Java Virtual Machine (JVM), allowing interoperability with Java and access to all Java libraries.
- **Concurrency & Distributed Computing:** It easier to build applications that are scalable and fault-tolerant ins Scala
- **Use Case:** Scala is frequently used in big data applications
- **Community and Adoption:** Scala has a strong community and is used by many companies worldwide for commercial applications and large-scale systems

Scala

- Objective here is not to be an expert in Scala, but to be able learn enough Scala to understand and write Spark applications in Scala
- To effectively use Scala, it is important to know **functional programming**

What is Apache Spark™?

Apache Spark is a unified analytics engine for large-scale data processing. [1]

It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. [1]

Apache Spark™ is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters.[2]

[1] https://spark.apache.org/docs/latest/index.html
[2] https://spark.apache.org/

Apache Spark™ Components and Libraries

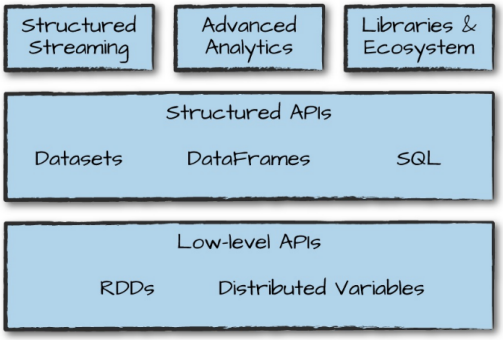

Figure: Spark Components & Libraries [1]

[1] Chambers, Bill, and Matei Zaharia. Spark: The definitive guide: Big data processing made simple

Conclusion

1. We reviewed Functional Programming basics
2. We studied about Scala as Functional Programming
3. We performed the installation of Scala and created worksheets
4. We learned basics of a Apache Spark App using Scala

Thank you