**A Report about Scala**

1. **Introduction**

Scala is a multi-paradigm general-purpose programming language designed to be concise, elegant, and type-safe [1]. Scala was invented by Martin Odersky and made its first release in 2003 [3]. Some of the design decisions made for Scala were aimed to address criticisms of Java [2]. The name “Scala” is an abbreviation on scalable language to signify the design purpose to grow with the demands of its users [2]. It provides a high-performance, concurrent-ready environment for object-oriented and functional programming language on Java Virtual Machine (JVM) [10]. Scala is a statically typed language, which is also optional since it can do type inference [1]. It also has many features that are common and explicit from Java and being more concise and readable from Java makes it a more comfortable and useful alternative.

1. **Programming Paradigm**

As a multi-paradigm programming language, Scala, integrates features of both object-oriented and functional languages to make it comfortable and easy for users to mix with each other.

* 1. **Dominant Paradigm**

Despite the seamless integration between the object-oriented and functional languages with-in, Scala is purely an object-oriented language in the sense that every value is an object [1]. Data types and behaviors of Scala are described by classes and traits [1]. Classes in Scala are used as blueprints for objects [4]. Traits in Scala are used as fields and a way to share interfaces between classes. Classes and objects can extend these traits [5].

* 1. **Other Paradigms**

Another paradigm of Scala is functional programming. It is a functional language in the sense that every function is a value [1]. Scala utilizes many features of functional programming languages, such as lazy evaluation, pattern matching, currying, type inference, and many others [2]. These features allow Scala to work like a functional language and help Scala mix the object-oriented style of programming with it.

1. **Features**

Since Scala is a multi-paradigm programming language, it has many features of both functional and object-oriented languages. These features give Scala a lot of different ways to mix and make it work together. Some of these features that, I think are useful are the following.

* 1. **Type Inference**

Scala utilizes type inference, which means users are not required to mention data type and function return type explicitly. The return function data type is determined by the type of last expression that is in the function [7]

* 1. **Lazy Evaluation**

Lazy evaluation is a functional program feature, that is used in Haskell by default, but with-in Scala it must be explicitly mentioned to be utilized. Users need to use “lazy” keyword when declaring a variable to do it. Lazy evaluation is used to increase the performance of a program by preventing it calculate variables that are not necessary.

* 1. **Currying**

Currying is a feature that lets the users convert functions that take multiple arguments into functions that take a sequence of single arguments.

* 1. **Pattern Matching**

Pattern matching is checking of values against cases of patterns. It is a more powerful version of the “switch” statement, and it can be utilized as an alternative to if/else statements [8].

* 1. **Concurrency Control**

Scala uses two different methods to do concurrency one being a standard library within Scala called “Actor” model and second one known as “Akka” [7]. Actor Model provides a higher level of abstraction when writing concurrency, it prevents the users from having to deal with locking and thread management [9].

* 1. **Singleton Object**

A singleton object, is a class with only one object with-in. A singleton object is declared by using “Object” instead of “Class” keyword [7].

* 1. **Immutability**

Scala makes a distinction between immutable and mutable variables by their declaration [2]. To make a variable mutable we declare it with “var” and to make it immutable we declare it with “val”. Variables that are immutable cannot be reassigned after initialization, while mutable ones can.

1. **Comparison with Java**

Since Scala was made in the image of Java it is easy to compare the differences between each other. Both have some strengths and weaknesses

* Lazy Evaluation: Scala supports but Java does not
* Concurrency Model: Scala uses actor model, java uses conventional thread-based model
* Operator Overloading: Scala supports operator overloading, Java does not
* Backward Compatibility: Scala does not support backward compatibility, Java does
* Compiling Process: Compiling the process of source code into byte code is for Scala is slow, Java is fast
* Object-Oriented Language: Scala supports object-oriented language, Java also does
* Functional Language: Scala supports functional language, Java only recently started to do
* Conciseness: Scala is more concise, java is in larger chunks of code
* Functions: Functions in Scala are variables, functions in java are objects
* Interpolater: Both languages are run on Java Virtual Machine
* Both statically typed [11]
  1. **Advantages of Java**
* Highly secure, very robust
* Can be run on almost any platform
* Features a garbage collector,
* Easy to learn
* There are a lot of documentations and resources online
* Popular and not found anywhere [12]
  1. **Advantages of Scala**
* Can run java libraries and code
* Uses expressive typing that makes abstraction secure and consistent
* Easy to learn for programmers with object-oriented background [12]
  1. **Disadvantages of Java**
* Significantly slower than natively compiled languages
* Consumes more memory [12]
  1. **Disadvantages of Scala**
* Limited community presence
* Steeper learning curve for programmers that are familiar with object-oriented languages
* Development tools are not as advanced or sophisticated as Java’s [12]

1. **Common Uses Compared to Java**
   1. **Application Uses of Scala**

1Scala has distinguished itself in designing the following applications:

* Batch data processing
* Concurrency and distributed data processing
* Data analytics
* Parallel processing
* Real-time data streaming with the Spark Framework
* Web applications and web pages [12]
  1. **Application Uses of Java**

Java is a well-established programming language used in the following applications:

* Cloud-based
* Desktop GUIs
* Enterprise apps
* Embedded systems
* Gaming
* Mobile
* Scientific
* Web servers and applications
* Web-based [12]
  1. **Positions to learn Scala for**

It would be better to learn Scala if you want a career in the following professions:

* Application developer
* Big data engineer
* Data scientist
* IT consultant
* Software developer
* Software engineer
* Spark engineer [12]
  1. **Positions to learn Java for**

It would be better to learn Java if you want a career in the following professions:

* Android development
* Back-end development
* Big data development
* Embedded devices development
* IT manager
* Junior programmer
* Senior programmer
* Systems architect [12]

1. **Data Types**

Scala has same data types as Java. And the data types as follows:

* Byte: 8 bit signed value
* Short: 16 bit signed value
* Int: 32 bit signed value
* Long: 64 it signed value
* Float: 32 bit IEEE 754 single-precision float
* Double: 64 bit IEEE 754 double-precision float
* Char: 16 bit unsigned Unicode character
* String: A sequence of Chars
* Boolean: Either literal True or False
* Unit: No Value
* Null: Null or empty reference
* Nothing: Subtype of every other type, no value
* Any: Supertype of any type
* AnyRef: Supertype of any reference type [13]

1. **Conclusion**

Scala is a general-purpose multi-paradigm programming language. Thanks to its features of both object-oriented and functional languages it makes the users able to mix them together. It was created to address criticisms of Java. It is utilized by many popular companies around the world such as Twitter, Netflix, Sony, and Apple [12].

**REFERENCES**

**[1]** [**https://docs.scala-lang.org/tour/tour-of-scala.html**](https://docs.scala-lang.org/tour/tour-of-scala.html)

**[2]** [**https://en.wikipedia.org/wiki/Scala\_(programming\_language)#Type\_inference**](https://en.wikipedia.org/wiki/Scala_(programming_language)#Type_inference)

**[3]** [**https://www.artima.com/weblogs/viewpost.jsp?thread=163733**](https://www.artima.com/weblogs/viewpost.jsp?thread=163733)

**[4]** [**https://docs.scala-lang.org/tour/classes.html**](https://docs.scala-lang.org/tour/classes.html)

**[5]** [**https://docs.scala-lang.org/tour/traits.html**](https://docs.scala-lang.org/tour/traits.html)

**[6]** [**https://www.newgenapps.com/blogs/advantages-of-using-scala/**](https://www.newgenapps.com/blogs/advantages-of-using-scala/)

**[7]** [**https://www.javatpoint.com/features-of-scala**](https://www.javatpoint.com/features-of-scala)

**[8]** [**https://docs.scala-lang.org/tour/pattern-matching.html**](https://docs.scala-lang.org/tour/pattern-matching.html)

**[9]** [**https://doc.akka.io/docs/akka/2.0.5/scala/actors.html**](https://doc.akka.io/docs/akka/2.0.5/scala/actors.html)

**[10]** [**https://data-flair.training/blogs/scala-features-comprehensive-guide/**](https://data-flair.training/blogs/scala-features-comprehensive-guide/)

**[11]** [**https://www.guru99.com/scala-vs-java.html**](https://www.guru99.com/scala-vs-java.html)

**[12]** [**https://www.simplilearn.com/scala-vs-java-article**](https://www.simplilearn.com/scala-vs-java-article)

**[13]** [**https://www.tutorialspoint.com/scala/scala\_data\_types.htm**](https://www.tutorialspoint.com/scala/scala_data_types.htm)