16039231

Reflective Report

Programming Languages: Principles and Design (6G6Z1110)

Pritam Sangani

January 14, 2019

Abstract

This short report will cover a critical comparison of the features of Pharo, Ruby, JavaScript, Clojure and Haskell. It will also cover the key concepts and nuances of each language that have been learnt by completing the laboratory exercises and portfolios. This report will start off by individually describing the features of each of the above languages before doing an overall comparison using a number of categories to compare against. The next section of this report will provide a personal evaluation and reflect on the structure of each of the programming languages studied, before ending the report with a brief discussion on how studying and working with the principles of these languages will help in furthering a career in software development.

Comparison of Programming Languages

Pharo

Introduction

Pharo is an open-source programming language and environment, influenced by the principles of the Smalltalk programming language. Pharo describes its environment, which is used to interact with the language as an 'Integrated Development Environment (IDE) and OS rolled into one'.

Paradigm

Pharo is a pure object-oriented programming language, which means everything in Pharo is an object. It also takes principles from the reflective programming paradigm.

Language Features and Support for Data Types

Pharo is a powerful language, yet its full set of built-in syntax can 'fit on one postcard'. Pharo has no constructor declarations for classes, no primitive types, such as `int` and `String` in Java, and is dynamically typed, so data types are not declared. Pharo uses `messages` to allow communication between Objects - this model is used throughout the language. Inheritance exists in Pharo, but only single inheritance is supported.

Ruby

Introduction

Ruby is an open-source programming language, influenced by the principles of the Perl, Smalltalk, Eiffel, Ada and Lisp programming languages. Ruby focuses on being simplistic and increasing developer productivity.

Paradigm

Ruby is a multi-paradigm language, that takes principles from the object-oriented, imperative, functional and reflective programming paradigms. Like Pharo, everything in Ruby are objects and, as such, is a pure object-oriented programming language.

Language Features and Support for Data Types

The founder of Ruby, Yukihiro Matsumoto, aimed for Ruby to have syntax that was natural to read. One of the ways Ruby makes its syntax natural to read is in its flexibility, which allows developers to add methods to its built-in classes with more natural names. For example, addition in Ruby is performed with the `+` operator. Ruby's flexibility allows for developers to add a method to its built-in `Numeric` class, with a more natural name, `plus`.

```
class Numeric
   def plus(x)
      self.+(x)
   end
end
# results in 11 being assigned to y
y = 5.plus 6
```

Another aspect of Ruby that offers flexibility is its `blocks`. `Blocks` in Ruby are `closures`, which developers can attach to any method to describe how the method should execute. Ruby `closures` are a functional paradigm aspect inspired from Lisp.

Ruby only supports single inheritance, however it supports modules, which any class can mixin, using the keyword `include`, to give access to all the methods in the module.

Ruby is dynamically typed, so variable types are not declared. However, Ruby syntactically differentiates variables with different levels of scope.

```
# local variable
name
# instance variable - removes the need to use ‘self.‘
@name
# global variable
$name
```

JavaScript

Introduction

JavaScript (shortened to JS) is a lightweight, high-level language that is best known as being the scripting language for the Web, but is also used in some non-browser environments, such as Node.js and Adobe Acrobat. The language conforms to the ECMAScript specification, which is updated annually.

Paradigm

JavaScript is a multi-paradigm language that supports object-oriented, imperative and declarative programming principles. The main declarative programming style that JavaScript supports is functional programming.

Language Features and Support for Data Types

JavaScript's syntax is similar to Java and C++. However, unlike Java and C++, JavaScript is dynamically typed and so variables can hold data of any type at any point in time.

JavaScript also supports prototype-based programming principles, which means that objects' definitions can be extended at run-time, whereas in languages using class-based principles, such as Java, objects' definitions can only be extended at compile-time, as a new class will have to be created.

JavaScript also supports first-class functions. This means that functions can be passed as arguments to other functions, returned as values from other functions, assigned to variables and stored as fields in data structures.

Clojure

Introduction

Clojure Docs describe Clojure as being a "***robust***, ***practical***, *and* ***fast*** *programming language with a set of useful features that together form a* ***simple***, ***coherent***, *and* ***powerful*** *tool.*" The language is a dialect of the Lisp programming language and is a general-purpose programming language, so can be used in a wide variety of domains.

Paradigm

Clojure is a purely functional programming language, and emphasises the use of recursive iteration over the use of loops with side-effects.

Language Features and Support for Data Types

Clojure provides developers with inbuilt features to avoid the need of creating mutable states. One of the main tools is the collection of immutable data structures that Clojure provides. As these collections are immutable, adding or removing to/from it will create a new collection with the change, whilst retaining the old version.

Clojure is hosted on the Java Virtual Machine (JVM) and so all Java libraries are available to be called within Clojure code. Clojure utilises the Java concurrent thread system and as the core data structures are immutable they can be easily shared between threads, without having to worry about deadlocks. Clojure also provides mechanisms to allow state changes providing developers with tools to avoid conflicts when manually using locks.

Clojure is a dynamically typed and so data types do not need to be declared. However, type hints can be assigned to fields, using `deftype`, and if the type hint is of primitive type, it will be used as a field constraint, otherwise the type hint will be used to optimise methods.

Haskell

Introduction

Haskell is a high-level, open-source language and is described by Haskell Docs as allowing *"rapid development of* ***robust, concise, concise*** *software."*

Paradigm

Haskell is a purely functional language and even operations that have side effects, such as I/O operations have to be written as pure code. Variables cannot be mutated and functions have to be written as expressions, not statements like in imperative languages.

Language Features and Support for Data Types

Haskell is a statically-typed language so types are checked at compile-time. However, type declarations are optional as types can be inferred by the compiler.

Haskell can also call function from other languages, as well as, have Haskell function be called using Haskell's `Foreign Function Interface`. This can be done by rewriting the foreign function header in the Haskell equivalent. For example,

```
double exp(double);
-- exponential function from the C library can be rewritten as...
foreign import ccall "exp" c_exp :: Double -> Double

triple :: Int -> Int
triple x = 3*x
-- This function can be exported to be used in other languages by...
foreign export ccall triple :: Int -> Int
```

Haskell also has a large and wide-ranging library of open-source contributed packages that can be used to make development easier and quicker.

Haskell is also very popular for concurrent programming due to side-effects not being allowed. GHC, Haskell's compiler also includes a high-performance garbage collector, which can be used in parallel threads and cores, and a lightweight concurrency library which contains a collection of useful tools for concurrent programming.

Overall Comparison

Readability

Pharo is a very readable language due to its short collection of syntax, which means code will be more consistent and therefore easier to interpret what the code is doing.

Ruby is a very readable language due to its focus on providing syntax that is natural to read.

JavaScript is considered to be a very readable language due to the fact that its syntax is very similar to that of the popular Java and C++. There are a number of ways operations can be implemented, which makes it more expressive. However, this allows code to be written in more obscure ways which could affect its readability.

With Clojure being functional, the code is easily readable as it is easy to step through the expressions in the function to see what is being evaluated. Also as all functions are separated with an outer pair of brackets, it is easy to see the structure of the code.

Like Clojure, Haskell is also written purely using functions, which makes it easy to see what code is being evaluated first.

Writability

Pharo can be quite tricky to write for beginners due to its messages system, but once this is understood, Pharo can be considered to be fairly writable due to its small amount of syntax. However, due to it being very orthogonal errors can be quite hard to find and fix.

Ruby is a very writable language due to the flexibility that Ruby offers, which allows the extension of implementations to suit individual needs.

JavaScript is considered to be a writable language due to its support for abstraction and prototype-base programming principles.

With Clojure being functional, the developer will be forced to think mathematically and logically and so the code that is written tends to be very well structured and well written.

Haskell supports function composition and application, which makes the code more writable and readable as less brackets are used.

Reliability

Pharo is not very reliable as it is quite hard to do error checking and it is common to get runtime errors when first running code.

Ruby includes tools for error handling and garbage collection so programs can run more reliably and efficiently.

JavaScript supports error handling which can handle invalid types or type conflicts.

Clojure is reliable as its built in data structures are immutable so side effects are not an issue when using these data structures. However, type conflicts can be an issue as it is dynamically typed.

Haskell is very reliable as it is statically typed so type conflicts are caught at compile time. Also side effects are not allowed so unexpected behaviour is less of an issue.

Popularity

Pharo is not a very popular language and it doesn't feature in the Top 50 of the TIOBE index which measures and records the most popular programming languages annually. Pharo can only be used in Pharo's own IDE, but documentation is fairly good and there is a full course available for free and a number of books available for purchase with some being free.

Ruby is a very popular language and it has featured in the Top 10 of the TIOBE index for the past decade. There are a number of different implementations of Ruby and the community is strong with various events, including conferences, being held annually.

JavaScript is one of the most popular languages as it is the most commonly used scripting language for the web. The documentation is excellent and there are a number of sites offering different styles of documentation, such as MDN and W3Schools. JavaScript has been a common feature in the top 10 since the rise of the web around the millennium.

Clojure isn't a hugely popular language for individual developers, however, Clojure Docs lists a huge number of companies that use the language which is increasing as more and more companies look to languages that offer reliable concurrent programming tools.

Haskell is also not very popular with individual developers, but its community is strong and has many events, such as, meetups and conferences listed on its community page on Haskell Docs.

Evaluation and Reflection

Pharo

I found Pharo a bit tricky at first as I had not experienced a language with syntax like Pharo's. However, the cheat sheet on Pharo Docs made it easier to understand and and get to grips with the syntax. I did not like the IDE as it was quite unstable and it crashed a couple of times. I also did not like that it ran on a Virtual Machine as it meant that my laptop slowed down and that if the IDE crashed or I did not save my work to my host system, I could not recover my work. I also did not like that there wasn't a comprehensive documentation website as the Pharo Docs only listed the books and the cheat sheet. However, the books were useful as they contained a lot of examples.

Ruby

I enjoyed programming in Ruby as the syntax made my code rally easy to write and elegant looking. I also liked that there was good documentation online which explained fully, with examples, how to use the various core functions and libraries. I also like the IRB as it allowed me to evaluate and test that my functions work very easily.

JavaScript

I enjoyed programming in JavaScript as I have programmed in it quite a bit previously and it allowed me to use the portfolio as a refresher, as well as, giving me the opportunity to practice writing JavaScript using the ES2016 standard, which I hadn't used previously. I like the JavaScript documentation, with my favourite and goto being the documentation on MDN.

Clojure

I found it quite difficult to code in Clojure as this was my first experience of programming in a functional language. It took some time to get used to thinking in a much different way than I do when programming in imperative language and also getting used to putting brackets on the outside of functions. I also found it weird that the operation argument order was like it is in Clojure with the operand being first followed by its arguments. However, once I got used to the way of thinking I enjoyed structuring my code in a way that it was easy to work myself up the chain of functions evaluating the expressions until I got to the function prototype which evaluates the final result. I found that this is a much better way of writing reliable and easy to understand code.

Haskell

As I had already written code in Clojure, this was less of a steep learning curve and I focussed more on researching and using parts of Haskell that was not taught in the lectures as I enjoy writing the best and most efficient code that I can. I enjoyed refactoring my code to use function applications and compositions as I could see how much cleaner and more maintainable my code became.

Overall Reflection

Overall, I found that I have many strengths that I discovered during the course of this semester. One of these being my ability to pick up new languages pretty quickly and being able to interpret documentation and find relevant help online on forums such as Stack Overflow when the documentation does not answer my questions. I also discovered some weaknesses as well that I need to work on. One of these was that I often got fixated on finding unique ways of implementing a particular function or operation. I often spent a lot of time on finding a solution when I could have been working on something else. However, I found that if I spent some time away from the problem, this cleared my head and I often found that I quickly found a solution when I came back to solving the problem.

Discussion

Going through the portfolios using the 5 languages discussed in this report, I have gained an understanding of different programming principles and added more tools to my toolbox which I can use if I have to program in one of these languages during my career. I would like to learn and gain more experience in functional programming and I would like to gain experience in programming in Scala as I have attended a couple of workshops during hackathons where financial companies, in particular, have talked about the benefits of programming in Scala.

Bibliography

Listed below are the resources that have helped in writing this report:

`https://pharo.org/`
`https://pharo.org/about`
`https://www.ruby-lang.org/en/`
`https://www.ruby-lang.org/en/about/`
`https://developer.mozilla.org/en-US/docs/Web/javascript`
`https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript`
`https://clojure.org/`
`https://clojure.org/about/rationale`
`https://www.haskell.org/`
`https://www.tiobe.com/tiobe-index/`