Learning a New Language: Ruby

Alden Luc R. Hade
De La Salle University-Manila
2401 Taft Avenue,
Manila, Philippines 1004
alden_luc_hade@dlsu.edu.ph

1. INTRODUCTION

Ruby is an object-oriented programming language that is used in many fields. Developed by Yukihiro Matsumoto in the mid-1990s in order to make games, Ruby supports many programming paradigms, such as functional and object-oriented. It is weakly-typed.

It can be used in many fields, but is best known as a language for web applications because of the Ruby on Rails framework built on top of Ruby. However, its general purpose makes it suitable of a wide variety of tasks, similar to Java and Python.

This paper discusses Ruby and evaluates it through the implementation of quicksort and merge sort algorithms. The language is also compared to a work-in-progress programming language called MLG. The source code described in this paper is available at https://github.com/aldsTM/RUBY.

2. IMPLEMENTATION

As an exercise meant to aid in the learning and evaluation of Ruby, simple merge sort and quicksort algorithms were implemented.

2.1. Merge Sort

The merge sort is a well-known efficient sorting algorithm invented by John von Neumann in 1945. It is a divide-and-conquer algorithm, evident in its two parts: one for splitting the array into arrays that are trivially sorted, and another for merging the arrays in the correct order.

The top-down implementation is discussed in this paper. Algorithm 2.1 illustrates the implementation as done in Ruby.

```
def mergesort(array)
    if array.count <= 1
        return array
    end

    mid = array.count / 2
    left_partition = mergesort array.slice(0, mid)
    right_partition = mergesort array.slice(mid, array.count - mid)

    array = []
    left_lead = 0
    right_lead = 0
    while left_lead < left_partition.count &&
          right_lead < right_partition.count
        a = left_partition[left_lead]
        b = right_partition[right_lead]

        if a <= b
            array << a
            left_lead += 1
        else
            array << b
            right_lead += 1
        end
    end

    while left_lead < left_partition.count
        array << left_partition[left_lead]
        left_lead += 1
    end

    while right_lead < right_partition.count
        array << right_partition[right_lead]
        right_lead += 1
    end

    return array
end
```

Algorithm 2.1. *Top-down merge sort coded in Ruby.*

2.1.1 The Divide

The merge sort algorithm first splits the (assumedly unsorted) list into two lists, until each list only has one element. Since each list now only has one element, it is considered sorted by default.

2.1.2 The Conquer

Once the lists have been trivially sorted, they are then merged to produce new sorted lists, until there is only one list remaining. This one list is the sorted version of the initially unsorted list.

The merging compares the first element of each list. The element which is lower is then pushed into a new, bigger list. This process is done continuously until one or both of the lists are emptied out; in the former case, all elements left in the other array are simply pushed to the new list. The merging is done until the original list's length is replicated.

2.2. Quicksort

Quicksort is another divide-and-conquer sorting algorithm well-known for its efficiency. It was developed by Tony Hoare in 1959, and published 1961. When implemented well, it can be faster than merge sort in the average use case.

Quicksort divides a list into two smaller sublists: a list of the low elements and a list of the high elements. Algorithm 2.2 shows the algorithm for partitioning. It can then recursively sort the sublists.

```
def partition(array, low, high)
    i = low
    j = high + 1
    pivot = array[low]
    while true
        begin
            i += 1
            break if i==high
        end while array[i] < pivot
        begin
            j -= 1
            break if j == low
        end while array[j] > pivot
        break if i >= j
        temp=array[i]
        array[i]=array[j]
        array[j]=temp
    end
    temp = array[low]
    array[low] = array[j]
    array[j] = temp
    return j
end
```

Algorithm 2.2. *The partitioning function used by quicksort.*

The low and high lists are determined based by a chosen *pivot* element. For this implementation, this pivot is the leftmost element (this is known as the original Hoare partition scheme). Algorithm 2.3 shows the implementation as done in Ruby.

```
def quicksort(array, low=0, high=nil)
    if high == nil
        high = array.count - 1
    end

    if low < high
        p = partition array, low, high
        quicksort array, low, p-1
        quicksort array, p+1, high
    end

    return array
end
```

Algorithm 2.3. *The main quicksort function.*

2.2.1 The Divide

The quick sort algorithm partitions a list into two based on the pivot element. (Three lists if the pivot is counted as a separate list.)

2.2.2 The Conquer

Once the list has been partitioned, it's easy to sort the sublists. All elements with less value than the pivot are pushed to the list with the low elements, and all elements with higher value than the pivot get sent to the other list.

3. EVALUATION

3.1. Evaluation Criteria

The evaluation criteria used are influenced by the book, *Concepts of Programming Languages.* The criteria are as follows:

1. Readability, or how easy the average piece of code is to be understood by humans;
2. Writability, or how seamless or straightforward the process of coding can be in this language;
3. Reliability, or how consistently the language performs when subjected to different situations; and
4. Cost, or how easy the language is to learn as well as how costly it can be to maintain the functionality of existing programs in that language.

Evaluation is done with knowledge learned from the experience of coding the sorting algorithms as well as information derived from references.

3.2. Ruby, an Evaluation

3.2.1 Readability

Ruby has two things that make it stand out from other programming languages: it is [1] object-oriented, and [2] weakly-typed at the same time. This is similar to Python and gives them both an instant advantage: they are easier to read at a glance. Ruby's blocks and the dynamic typing make it seem like a programmer has written pseudocode. It is very simple to understand and makes it easier to make changes later on if need be.

Another highlight of the code blocks is that very simple procedural programs are easy to process by first-time programmers who are not so familiar with OOP languages. For comparison:

- Java:

```
class Main {
     public static void main(String args[]){
          System.out.print("Hello world!");
     }
}
```

- Ruby:

```
print "Hello world!"
```

Ruby also does not have need for terminating symbols, making it slightly faster to read since the mind does not have to process semicolons, only indentions. The weak typing, however, can make it confusing when debugging because it can be hard to track down the exact value and exact datatype of a variable at any given time without

using an IDE's watch function. In addition, the fact that functions and comparisons can be called with or without parentheses can make it confusing to read code written by different people without a common coding standard.

```
def quicksort (array, low, hi)
     ...
end

quicksort array, low, hi
quicksort(array,low,hi)
quicksort (array, low, hi)

if (5>4) && 5 < 6
    ...
elsif (5>25 && 5 < 20)
    ...
elsif 5 < 7
    ...
end
```

Blocks are done without the use of braces but rather indentation levels, meaning that code looks very clean when done correctly.

3.2.2 Writability

Ruby is similar to Python in that it is very easy to open up a blank text editor and start programming from scratch. The notations are concise, so programs can be written quickly. Declaring the main function is implied (see above example). All basic constructs are present and easy to write, such as declaring variables (see subsection above for drawback in weakly-typed variables). For comparison:

- C:

```
char first_name[50] = "Alds";
char last_name[50] = "Hade";
int id_num = 11789190;
char[7] course_code = "MSCS";
printf("%d - %s %s", id_num, first_name, last_name);
```

- Ruby:

```
first_name = "Alds"
last_name = "Hade"
id_num = 11789190
course_code = "MSCS"
print id_num.to_s + first_name + last_name
```

Another showcase of the conciseness of Ruby:

- C:

```
int i;
for (i = 1; i <= 10; i++){
     printf("%d", i);
}
```

- Ruby:

```
for i in 1...10
     print i
end
```

However, one glaring omission is the lack of the increment and decrement operators. Instead of being able to use ++ and --, programmers have to use += and -= .

3.2.3 Reliability

The readability and writability of this programming language influence its reliability in the sense that it is very easy to write programs and read code and thus come up with the correct, intended program.

Ruby is advertised as being a flexible programming language in the sense that [1] programmers would have more options in solving problems using different methods, but more importantly [2] Ruby programs keep running until they hit an error; they do not throw errors before executing most of the time.

Because Ruby's variables are weakly- or dynamically-typed, there is no type checking done when changing the value of a variable to another. However, there is type checking done when variables are used in functions. For example, printing requires all variables to be strings, and arithmetic requires integers or decimals.

As a modern language, Ruby supports exceptions. Programmers can even very easily catch custom exceptions by the use of the raise keyword. Ruby does not support pointers.

3.2.4 Cost

Ruby is free to download and does not require the use of a custom IDE in order to work—as soon as Ruby is installed, programmers can use a combination of the command line interface as well as their favorite text editor in order to start making programs. It is very easy to get into Ruby in that regard.

Ruby is also, for reasons previously mentioned, easy to get into and learn; however, like all programming languages, mastering it requires the mastery of all the tips, tricks, and shortcuts provided by the designer in order to make the code shorter or easier to read. Ruby is even recommended in some circles for complete beginners. Therefore, in terms of training costs, it does not take much to learn Ruby at all.

In terms of computing costs, Ruby has been reported to be slower than other languages due to its weak typing, which means that it would have to do a lot of referencing to find out the correct context of lines of codes.

Finally, in terms of possible loss due to unreliability, there are many cases and anecdotes online which show that the use of Ruby for many applications, from education to healthcare, is fine, and no major cases have been cited regarding Ruby's unreliability.

3.3. MLG, an Evaluation

The Major League Gaming programming language is a strongly-typed procedural programming language initially developed by Alds Hade with Arvin Reyes and Yun Sup Lee later on contributing valuable addenda to the language. It is designed to emulate the basic functionality of C as beginners see it.

3.3.1 Readability

The MLG language is procedural in nature; however, it clearly delineates the main function and other functions declared by the user; in addition, all lines require the use of terminators, and all code blocks require the use of braces. Therefore, the language looks uniform.

A big problem comes from the tokens. There are five data types: mmr (int), kdr (float), SAMPLETEXT (String), char,

and isMLG (boolean). The keyword for DO is WEED, and the keyword for WHILE is EVERYDAY. The terminator is an exclamation point, which can be confused for a factorial.

The word names used in the language are based on meme culture and L33T gaming culture of the early 2010s, during the seventh console generation of video games. Many readers who try to read MLG code today will not recognize the references that the tokens represent. They are not intuitive.

3.3.2 Writability

Writability is a big challenge for MLG. Basic features in other programming languages such as C and Ruby are not present here. These are some examples:

1. Data types are explicit and not dynamic, and the data type names (e.g. kdr, mmr) are not intuitive;
2. Variables cannot be assigned values on the same line as their declarations (e.g. "mmr x = 10" throws an error);
3. There are no increment and decrement operators;
4. Code blocks with a single line cannot be written without the braces flanking them (so even a simple conditional with one line needs to have braces).
5. Functions cannot be declared before the main function. Functions must be declared AFTER.
6. Errors are not intuitive.

These point to issues with orthogonality, syntax design, feature multiplicity, and expressivity.

3.3.3 Reliability

The readability and writability of this programming language influence its reliability badly—it is hard to be able to use this in a professional setting wherein the software to be developed is usually more complex than simple procedural scripts.

The programming language also allows aliasing, which is defined to be dangerous. Finally, the error handling in the language is very limited; it works perfectly if the code written is correct, but if it isn't, it is very hard to parse the error messages that show up in order to correct mistakes. This is related to cost as well.

3.3.4 Cost

MLG does not come with a price tag and it is simple. However, it is very simple to a fault. It is not powerful enough for the time it takes to learn it. It is only procedural and has very limited room for programmers to express themselves (the fact that one cannot instantiate variables in the same line they are declared is a big hit). Therefore, in order to code a relatively simple function like a sorting algorithm, longer codes would be required. And because they cannot be shortcut (i.e. there is no orthogonality), developers are stuck with repetitiveness, which costs time and money.

5. IN CONCLUSION

In this activity, the author was able to learn a new programming language, Ruby. After using Ruby to write a couple of efficient sorting algorithms, as well as upon doing further research to delve into the depth of the programming language and its power, Ruby has shown itself to be a powerful object-oriented programming language on par with the likes of Java, C#, and Python. Ruby also comes with design choices that may improve usability in certain cases.

However, the weak/dynamic typing means that it will be hard to track down the exact values of variables, and therefore it does not scale as well as something like C# or Java.

Ruby is ideal for beginners looking to get started with making interactive games or build their own websites because of its great readability as well as the lack of issues with its reliability. It is also flexible, which means that beginners quickly see their progress when it comes to writing bug-free programs or debugging bug-ridden ones.

In comparison, having analyzed the power of Ruby, the shortcomings of MLG have become more apparent. Table 1 shows a summary of comparison between the two languages. MLG is not ready for professional use. However, the activity has shown that Ruby is a decent standard worth aiming for when designing a new programming language that others can appreciate.

Category	Ruby	MLG
Simplicity	Simple but powerful	Hard to learn, not powerful
Orthogonality	Just enough	Too limited
Data Types	Dynamic	int / float / char / string / boolean
Syntax Design	Clean, concise, clear	Poor
Abstraction	Supported	Supported
Expressivity	Good expressivity	No variations
Type Checking	Only in functions	Yes
Error Handling	Very flexible	Yes
Aliasing	No aliasing	Yes

Table 1. *Comparison of Ruby and MLG.*

6. REFERENCES

1. ruby-lang.org (2008). *About Ruby.* Accessed at https://www.ruby-lang.org/en/about/.
2. Knuth, Donald (1998). "Section 5.2.4: Sorting by Merging". *Sorting and Searching.* The Art of Computer Programming. 3 (2nd ed.). Addison-Wesley. pp. 158–168. ISBN 0-201-89685-0.
3. Hoare, C. A. R. (1961). "Algorithm 64: Quicksort". Comm. ACM. 4 (7): 321. doi:10.1145/366622.366644.
4. Sebesta, R. (2013). *Concepts of Programming Languages* (10th Edition).
5. Codementor (n.d.). *Why Learn Ruby?* Accessed at http://www.bestprogramminglanguagefor.me/why-learn-ruby.