Comparing and Contrasting Ruby & Python

New Mexico Institute of Mining & Technology
Department of Computer Science
cs324@cs.nmt.edu

ABSTRACT
In this paper, we critique the Python and Ruby programming languages from a language designer's point-of-view. Additionally, each language's features are reviewed, and the inclusion or exclusion of a feature in each language is reasoned. The applications of each language are also discussed to support the evaluation.

Categories and Subject Descriptors
D.3.3 [**Programming Languages**]: *ALL*

General Terms
Design, Reliability, Experimentation, Languages, and Theory.

Keywords
Abstraction, Garbage Collection, Concurrency, Efficiency, Syntax Design, Exceptions, Interpreted, Languages, Maintenance, Orthogonal, Portability, Power, Python, Readability, Robust, Ruby, Safety, Security, Simplicity, Types.

1. INTRODUCTION
Since the creation of the first computer, there has been a need for a way to control computers. Initially, the languages that were designed focused on the efficiency of the programs in execution and neglected the efficiency of the actual programmer. After the cost of computation decreased below that of the programmer, languages began sharing their focus between execution time and the time a programmer spent coding. Some of the largest factors that influence the time taken to write a program are the features in a language. Further, every language is designed with a target application area, meaning a specific design goal.

2. POWER
Power in a programming language refers to how easy it is to express complex ideas in a simple manner. Many factors of the language influence how powerful it is. Some of the factors include what abstractions are provided, the data types included, the typing system, the syntax, and support for high level features like concurrency.

2.1 Abstraction
One of the most important parts of a language is abstraction. By nature, the brain of a programmer can only think about so many things at once. Abstraction is all about hiding as many of the unimportant things as possible so the programmer only has to focus on the important things. However, this does not mean that working at the highest possible level of abstraction is always best. As the level of abstraction increases, the amount of control over lower level systems decreases. This can be a benefit or a disadvantage depending on the application, which makes it important to pick a language at the right level of abstraction.

New Mexico Institute of Mining & Technology
Department of Computer Science
cs324@cs.nmt.edu

2.1.1 Ruby
Ruby operates at a very high level of abstraction. A number of factors put it at this high level including its object oriented nature, support for closures, operator overloading, and open classes including the core language classes.

In Ruby, everything is an object. This pure approach to object orientation allows for easy manipulation and extension of all objects, even what would be considered basic types in other languages (integers, floating point numbers, etc). For example, you can change the base of a number like so:

```
5651.to_s(2) => 1011000010011
```

This is calling the to_s method with parameter 2 on the number 5651. The full scope of Ruby's object oriented nature extends far beyond this simple example but it functions as large part of Ruby's abstraction by allowing complex behavior to be encapsulated inside classes.

Closures are a core part of Ruby's power as well. Blocks, denoted with "def" and "end", can be assigned to variables and then called using the variable. This feature allows functions to be treated as first class objects and be assigned to variables, passed as arguments, and modified, all seamlessly and abstracted away within the language.

Operator overloading also adds to the power of Ruby by allowing the use of a very simple syntax to represent complex behavior defined by the programmer. This is often used when creating a class that logically works with the available operators. For instance, a vector class that holds angle and magnitude would benefit from the "+" operator being overloaded to contain the appropriate formula for adding vectors.

Another feature that potentially ties together all of the aforementioned features is the ability to modify the core behavior of Ruby itself. One potential use of this feature is to add new methods to existing core functionality such as adding a new number function to convert it into roman numerals for instance. By adding this method to the Fixnum (Ruby's number class) class the new method can be used like any other provided Fixnum method that is already included. This ability to modify the language itself is very powerful but comes with consequences. Since the language features are being changed and existing methods can be overridden, unintended results can occur if for instance one developer changes addition to subtraction and fails to mention it to anyone.

2.1.2 Python
Python operates at a similar high level of abstraction that Ruby does. Some of Python's features that improve abstraction are lambda functions, chained comparisons, and list comprehensions.

Lambda functions are simply anonymous functions which are typically used with the filter, map, and reduce functions in Python. For example, the filter takes two arguments, the first is the function to filter by and the second is the list to filter and it

returns a list of all items from the input list that meet the filter function criteria. Without lambda functions, a function would have to be defined for the sole purpose of using in the filter function even if it was only to be used a single time. However, with this feature a lambda function can be defined inside the parameter list of filter to be used just for the filter. This feature is abstracting the process of creating a function for limited use.

Another abstraction that Python provides is chaining comparison operators. An example of this is:

```
y = 2
1 < y < 5 < 10 = True
```

This evaluates to (1 < y and y < 5 and 5 < 10). Each of the sub-expressions evaluates to true and therefore the result of the entire expressions is also true. This is an example of the language simplifying a common chain of expressions into a simpler format that hides the underlying details.

The creation of lists is another feature that is abstracted by Python. In programming language without list comprehension creating a list of perfect square numbers would require using a loop to calculate the next perfect square and add it to the list. With list comprehension this is simplified to:

```
[ n*n for n in range(5) ]
```

which results in the list [0, 1, 4, 9, 16]. This greatly simplifies the creation of lists and gives the programmer power to create lists easily.

2.1.3 Critique

This is where the examined language features would be criticized with an effort to improve upon the language keeping a design goal in mind. You may wish to target an application area. Make sure to justify your statements as you feel necessary.

2.2 Simplicity and Orthogonality

2.3 Control Structures

2.4 Data Types & Structures

2.5 Typing Systems

2.6 Exception Handling

2.7 Syntax Design

2.8 Interpretation

2.9 Concurrency

3. SECURITY

4. ROBUSTNESS

5. EFFICIENCY

6. PLATFORM INDEPENDENCE

7. REFERENCES

[1] http://www.ruby-lang.org/en/documentation/ruby-from-other-languages/

[2] http://www.secnetix.de/olli/Python/

[3] Ruby From Novice to Professional, 2/E, Peter Cooper.