Andrea Correia, Frances Gallagher and Rose Stefanidakis
Professor King
CSCI 324: Programming Languages and Implementation
23 April 2024

Ruby:
Report and Annotated Bibliography

For our project, we chose to use the programming language Ruby. Ruby was founded in 1995 with the goal of complete object oriented programming. Ruby treats essentially every part of code written as an individual object, giving them instance variables and methods, unlike some other languages, where some types, especially primitive types like integers, can not be treated as such. Other unique features of Ruby include the capability of blocking, duck typing, built-in hash tables, and regular expressions. As the first step of this project, we completed a common program: a game of hang-man where errors build a snowman. For the hang-man game, we were able to take advantage of the built-in hash tables that allowed us to easily read through the text document of possible words, convert them to lowercase, and then create keys for easy access to each word. Without this built in capability, the process of reading through, converting, and later accessing information would have been much longer and more difficult, and with a better run time. The below code is our implementation of the built-in hash tables, and the file input and conversion.

```
words_hash = {}

File.open(filename, 'r') do |file|
    file.each_line do |line|
        words = line.chomp.split(' ') # separated by spaces
        words.each do |word| # add to hashtable
            #words_hash[word] = true
            words_hash[word.downcase] = true  # Convert word to lowercase before
        end
    end
end
```

We were also able to take advantage of the use of regular expressions within Ruby. This is a unique feature where Ruby has an efficient method of validation, or pattern recognition within strings. In the snowman hang-man game, this process makes checking to see if a guessed letter by the user is in the given word extremely easy, and very efficient. The following photo is the example of the regular expression, which looks to confirm that the guessed character is a letter, upper or lower case, or contains a dash or apostrophe, as well as confirming only one letter could be guessed at once.

```
if letter.match?(/[a-zA-Z'-]/) && letter.length == 1
```

For the next part of our project, we began our creative project. We decided to create a game of blackjack, where the user plays against the dealer. In our game of blackjack, the user must first create a username to play. If the input username is not in the appropriate bounds of characters, the output will remove unapproved characters, and return the accepted ones. If the user enters the phrase “iwanttowin21” this triggers a cheat code, and the user automatically wins the game, prompting a new screen announcing this victory.

After pressing enter to input the username, the username will appear in the top left corner. The game has background music, and when the cards are initially shuffled, there is a corresponding audio. To begin playing, press ‘d’ for deal. When cards are passed out by the dealer, there is another sound that plays as each player, the dealer and the user receive two cards. The user then has the option to press ‘s’ for stand, or ‘h’ for hit. When you press ‘h’, a card gets added to the player’s deck, and the user total gets updated to reflect this addition. The corresponding card is shown as well. If the player has gone over 21, the game ends as the dealer has won, and a pop message comes up. When you press the character ‘s’ to stand, the dealer is

given a card, and after this card is given and their total is updated, whoever is closest to 21 without going over has won. In the case of a tie between sums after standing, this tie ends the game, and a corresponding message appears. Additionally, if either the player or dealer gets 21 after standing, they win, unless they both receive a 21, which defaults to a tie.

In our game, all number cards are their corresponding value, Jack, Queen and Kings are worth 10 points each, and the Ace card is either 1 or 11 points. The first Ace a player receives will be counted as 11 points, and any additional Aces received per game will be 1 additional point towards that goal sum of 21. At the end, the player with the sum closest but no greater than 21 has won, and this output will be announced through a message that pops onto the screen. In the case of a tie, there is a corresponding message announcing said tie.

To begin this game, we again used regular expressions to have the player create a username, which is then checked to make sure it is of appropriate syntax. The below code creates a string username, then through the variable "valid" determines whether the user input is composed of upper or lower case letters, has at least one character, noted by the + symbol, and can also use an underscore.

```
username = ""
valid = /\A[a-zA-Z0-9_]+\z/
game_started = false
prompt = Text.new("Enter a username!", x: 50, y:50, color:'white')
```

Without access to regular expressions, creating a valid username would be much more difficult, as checking to see if there were no other characters other than what was listed, or if there was any input given, would be much longer, take more time during program execution, and would require a lot more physical code to be written.

Another useful feature we were able to use with Ruby is blocking. Blocking is another facet of the flexibility Ruby's creator was striving for when creating the language. As anything in Ruby is considered to be an object, and therefore has instance variables and methods, another way of adding to these methods is to give more instructions about how said method will be executed. Blocks are able to be passed arguments, and also return a value. For our project, we used blocking to work with the dealer and player turns within the blackjack game. The code below demonstrates this functionality.

```
@player_hand.each do |card|
    player_total_value += card.value
    aces_count += 1 if card.rank == 'A'
end
```

```
def player_turn(&block)
    yield if block_given?
end
```

The block is created as part of the player_hand variable in this code, and the do… end contains the block. |card| is the parameter or argument being passed to the block, and when it is executed later in the code, it occurs where "yield if block_given" is, as this gets replaced and consequently executed with the block code if a block does get passed.

Another extremely important and useful feature within Ruby is the idea of polymorphism within duck typing. This means that we have access and can use the same methods within different classes to yield different results. When using polymorphism, as long as both classes have the same methods, Ruby, and as a result, us as we were developing this game, did not have to worry about specifying what type we were applying these functions to. In the Game class, we

relied on how certain classes respond to methods like “total”, so in essence we could focus on behaviors rather than what class it was. We did not have to have a shared superclass for User and Dealer because they have the methods that the Game class would rely on. This aided in code reusability and flexibility.

```ruby
# update player's total based on the cards they have
def update_player_total
    player_total_value = @player.total
    @text_player_total.text = "Total: #{player_total_value} "
end

# update dealer's total based on the cards they have
def update_dealer_total
    dealer_total_value = @dealer.total
    @text_dealer_total.text = "Total: #{dealer_total_value} "
end
```

We were also able to make use of the graphical interface possible with Ruby which means we could add shapes, colors, and sounds. We also included photos that would become the table and the cards. This allowed for us to simulate the game in a creative way because we could change aspects of the window using methods like Window.clear or Text.add.

A couple other interesting features of Ruby was that there are dynamic arrays. We did not have to worry about declaring a size for our arrays which was helpful when writing code. We simply could say cards = [] and this means that they are not fixed in size. Additionally, Ruby is dynamically typed so the variable types are determined at runtime. This means that we did not have to explicitly declare types and this helped with flexibility.

One aspect of Ruby that made it difficult to code was actually the downside of Ruby’s flexibility and preference for ‘plain English syntax’. Due to its lack of formality, finding and understanding errors within the code became somewhat difficult. Another issue we found is that compared to languages we are more familiar with, like Java and C++, Ruby is a slower language. Part of the reason for this is the ‘simplicity’ and flexibility of the language that was the goal of

creation. However, this concept of “metaprogramming” wherein the code is able to manipulate itself and other code while it runs, or writing programs where the output can be another program. Ruby is a language that does this, as classes and methods can be defined at run time. Metaprogramming is extremely complex, and this complexity slows down runtime. Additionally, metaprogramming makes it much harder to trace and find potential issues within a program, because of the complexity it creates at runtime. We on a much smaller scale experienced this issue in having a lot of difficulties narrowing down where an additional error was occurring within our dealer total. Because of the run time program metaprogramming creates, and the easy to read English syntax of the language, error detection and understanding where it is coming from is a large challenge within the language.

Annotated Bibliography

Wickramasinghe, Shanika. “Ruby Array 101: Primary Methods & How To Use Them.” *Learnenough.Com*, Learn Enough, 15 Jan. 2024, www.learnenough.com/blog/ruby-array#Ruby%20array%20uses%20and%20applications.

We used this source from Wickramasinghe to more firmly grasp a unique feature of Ruby. Arrays within Ruby are not of a fixed length, unlike languages like Java, which need a length determined at declaration. Another important distinction is that Ruby can have arrays with varying types being held within the same array. This was a useful feature in our code when reading the cards array to hold the various options within a deck of cards, as we could add elements without considering beforehand how long we would need the array to be, as seen also in dealer and player hand arrays.

Halmagean, Cezar. “Mastering Ruby Blocks in Less than 5 Minutes.” *Mix & Go*, Mix & Go, 19 Jan. 2015, mixandgo.com/learn/ruby/blocks.

We used this source to help us learn and understand the different syntaxes for creating and using a block. Following the many examples given within this source, we were able to tailor the code presented on the website for the type of block we used specifically in the part of our game code that deals with player and dealer turns, not only in the syntax we chose with the do | | end format, but also in the use of the word yield, and how this allows the block code to be executed at run time.

Castello, Jesus. “Mastering Ruby Regular Expressions.” *RubyGuides*, 24 Dec. 2022, www.rubyguides.com/2015/06/ruby-regex/.

This source walked through what a regular expression looks like within Ruby, and how they have different purposes and how they can be applied in different ways, whether just identifying if something is present or not, or looking for an exact match, as we do in our code when asking for a cheat code. The source walked through shorthands for looking for digits and letters, which we also used in our more general yet complex regular expression in checking to see if the player username is valid. The code provided on the website was used as a baseline to help create our regular expressions, using the general form of how regular expressions are generated.

Hogan, Brian. “Understanding Data Types in Ruby.” *DigitalOcean*, DigitalOcean, 26 Jan. 2023, www.digitalocean.com/community/tutorials/understanding-data-types-in-ruby.

This source was crucial in helping us to make hash tables and to understand the implications of dynamic typing. From here we learned that hash tables, like the one in our code, need to be contained within curly braces, and that the key-value relationships are noted through

(=>) assignment. In this way, we were able to adapt the example code on the website to our own code to create hash tables for cards in the blackjack game, and the snowman game as well.

Kumar, Ankit. “Type Checking and Duck Typing in Ruby.” *Code 360 by Coding Ninjas*, 26 Mar. 2024, www.codingninjas.com/studio/library/type-checking-and-duck-typing-in-ruby.

This spource was used to learn more about the concepts of duck typing within Ruby and the idea of type checking being done at run time, which was a new concept, and a unique feature to Ruby as a whole. The examples and conceptual parts of the website were useful as a basis for our duck typing, as well as in putting together an explanation for our presentation.

“Audio.” *Ruby 2D - Audio*, www.ruby2d.com/learn/audio/. Accessed 21 Apr. 2024.

This article comes from the Ruby2D website, and gave us information and basic code for adding audio into our program, as we wanted the cards to aubily shuffle and when the player is dealt a card. We adapted this code framework with our desired sounds for our creative program.

RubyCademy, Tech -. “The Yield Keyword in Ruby.” *RubyCademy*, Medium, 11 Apr. 2018, medium.com/rubycademy/the-yield-keyword-603a850b8921.

This website was another resource used to understand how the blocking functionality of Ruby works, and more specifically the use of the word “yield” within a closure. We wanted to have the block be optional, and dependent on whether or not a block is passed during a call to a method. We also learned from example code to understand how yield can receive arguments given in the creation of a block.

Baker, Ross. “What Is the Purpose of Attr_accessor in Ruby?” *Medium*, Medium, 19 Mar. 2020,
medium.com/@rossabaker/what-is-the-purpose-of-attr-accessor-in-ruby-3bf3f423f573.

This website helped us to understand how in Ruby we can modify the instance variables that are attached to each object within Ruby. Attr_accessor is used to simplify our code and save time by creating the “getter” and “setter” methods within Ruby. The code provided on the website was used as a framework for us to base our code on with our own instance variables and methods.

“Object-Oriented Programming in Ruby.” *RubyGuides*, 4 Apr. 2020,
www.rubyguides.com/ruby-tutorial/object-oriented-programming/.

This website was used in our initial research and compilation stage as we learned about complete object oriented programming, and the different ways this was achieved. We used this as a basis to learn more about classes and how almost everything in Ruby can be considered an object, and therefore has instance variables and methods.

"Ruby: Blocks." *GeeksforGeeks*, GeeksforGeeks, 25 Aug. 2022, www.geeksforgeeks.org/ruby-blocks/.

This source was used to help explain blocks and to give more concrete coding examples for our project and to further our understanding of how blocking can be used within Ruby. This code provided a framework we could compare our blocks to in order to make sure we were on the right track in terms of proper syntax and usage.

Miller, Stephan. "What Is Ruby Used For?" *Codecademy Blog*, 9 Apr. 2024, www.codecademy.com/resources/blog/what-is-ruby-used-for/.

This website was also used to give more background information on Ruby as we learned about what applications worked best for this programming language. This website provided information for the presentation about the best uses of Ruby, and its history in web development and static site generation. This website did not have any code, and was solely used for the presentation.

Britt, James. "Class Regexp." *Class Regexp - RDoc Documentation*, ruby-doc.org/3.2.2/Regexp.html. Accessed 22 Apr. 2024.

This website was used for helping us to generate our regular expressions. The website provided a lot of examples of code for different types of regular expressions, from matching to how to include different types of characters, like letters and numbers or symbols. We used the framework code they provided, as well as the detailed explanations of different shorthands for character groupings in order to create our two regular expressions; an exact match for the cheat code, as well as a more general validation for the username input.

"Object-Oriented Programming in Ruby: Set 1." *GeeksforGeeks*, GeeksforGeeks, 13 Aug. 2019, www.geeksforgeeks.org/object-oriented-programming-in-ruby-set-1/.

This source was used to help us understand the difference in syntax and uses for variable identification. We did not end up using any global variables, so in order to find an example of how a global variable is initialized and in the proper syntax, we took a screenshot of the code to place in our presentation. This website also helped to solidify the difference between

Todorovic, Nikola. "Ruby Metaprogramming Is Even Cooler than It Sounds: Toptal®." *Toptal Engineering Blog*, Toptal, 29 Oct. 2015, www.toptal.com/ruby/ruby-metaprogramming-cooler-than-it-sounds.

This article helped to explain metaprogramming in more detail as conceptually this is a complex subject matter. This article did not have any code, or really anything to do with the project, but provided more of an understanding as to what difficulties Ruby has from a programming perspective. Metaprogramming is an extremely interesting subtopic to consider for

potential issues, especially as we ran into some trouble not being able to tell where one of our errors was coming from.

Gault, Dustin. “Pros and Cons of the Ruby Programming Language in 2023.” *Cybersecurity Consulting, DevOps Services, & Custom Software Development*, NextLink Labs, 4 Oct. 2023, nextlinklabs.com/resources/insights/ruby-programming-language#:~:text=This%20means%20that%20in%20terms,more%20advanced%20or%20complex%20tasks.

This source was used to provide background into some of the pros and cons of Ruby as a language. We did a lot of background research into Ruby to determine unique features, and why this language would be chosen over other languages for specific purposes. This source helped to explain more of how Ruby was flexible, and began to explain the concept of metaprogramming, and some issues this could cause with the language.

“Ruby.” *About Ruby*, www.ruby-lang.org/en/about/. Accessed 22 Apr. 2024.

This website was used in our initial search and understanding of how Ruby works, and gave us a brief history of the language we used for our presentation. We learned about the creator, and some of the important details of Ruby as a programming language. This website gave a lot of information about unique features like the complete object orientation, and blocking, different types of variable scopes, that were useful in determining the type of project we wanted to create.