# An Evaluation Paper on
# Go, Kotlin, R, and Ruby

Submitted in partial fulfillment of the course requirements
For CSADPRG

Daniel Gavrie Clemente
Jan Robee Feliciano
Erik Andrew Rodriguez
Darryl Johnson Ty

Romualdo M. Bautista Jr.
November 23, 2023

# Chapter 1: Introduction

Programming languages have been in history for many decades. In the 1800s, Ada Lovelace and Charles Babbage joined forces to birth the first-ever programming language. Babbage unleashed the Analytical Engine in 1839, a marvel capable of all sorts of mathematical and logical feats. Lovelace, captivated by this invention, translated an 1842 paper by Italian mathematician Luigi Federico Menabrea, etching her name as the world's inaugural computer programmer. Despite the Analytical Engine remaining a blueprint, Lovelace's work laid the groundwork for future tech enthusiasts. (Cambridgeblog) (scientificamerican)

Fast forward to today's tech landscape, where programming languages have undergone a remarkable evolution to meet the dynamic needs of developers. Fueled by new computing paradigms, a quest for user-friendly coding, and a heightened awareness of tech's societal impact, the future of programming is on the cusp of groundbreaking changes (blog.emb.global). As technology advances, so do programming languages, necessitating developers to adapt and learn new languages for continued efficacy.

Ruby, a language that wears many hats, supports three programming styles: Object-Oriented, Functional, and Procedural. It made its debut in 1995 and has seen updates over the years, introducing new features and enhancements. What makes Ruby appealing is its clean and easily readable syntax, making it a favorite among beginners. It's particularly popular in web development, notably with the Ruby on Rails framework, and in tasks involving scripting and automation. (RubyCademy)

Enter Go, or Golang, a statically typed and compiled language supporting procedural, concurrent, and object-oriented paradigms. Born at Google in 2009, Go is recognized for its simplicity and efficiency, making it an ideal choice for system-level programming and high-performance web servers. Its influence extends widely into cloud infrastructure and web development. (Springer Link)

Kotlin, a statically typed language from JetBrains, embraces functional, procedural, and object-oriented programming. Introduced in 2011, Kotlin has seen multiple updates, with the latest version, Kotlin 1.9, arriving in July 2023. Known for its interoperability with Java, it has become a popular contender for Android app development. Additional features like nested functions and contracts contribute to its expressive capabilities. (TechRepublic)

R, primarily a language for statistical computing and graphics, leans towards a functional paradigm. Debuting in 1976, it has evolved with time, maintaining its relevance in data analysis and visualization. The simplicity of R's syntax and its flexible data structures make it a go-to choice. User-created packages further enhance its capabilities, offering statistical techniques, graphical tools, import/export functionalities, reporting, and more. (rOpenSci)

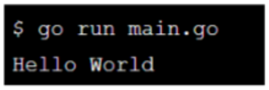
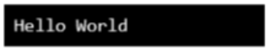
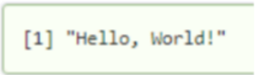
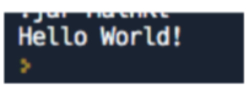
In the future, these languages are expected to evolve and incorporate new features to meet the changing demands of developers and the programming landscape. Ruby and Go are anticipated to expand their ecosystems and enhance performance. Kotlin, backed by Google, is likely to continue gaining traction in Android development. R is projected to persist in the field of data science and analytics, with ongoing improvements in packages and performance.

In terms of applications, Ruby finds its forte in web development, particularly with the Ruby on Rails framework. Go excels in cloud infrastructure and web development. Kotlin is primarily associated with Android app development but is versatile enough for full-fledged web applications. R takes the lead in data science and analytics but can extend its reach to other domains requiring statistical computing and graphics.

In this paper, we will explore and compare four popular programming languages: Ruby, R, Go, and Kotlin. We shall go over their prominent characteristics, advantages, and disadvantages. Additionally, we will contrast their resources, syntax, structure, performance, and scalability, as well as how appropriate they are for various project kinds. We will also cover their history, current state, future directions, and application domains.

## Chapter 2: Language Comparison

Go : Write the code using the fmt library to print "Hello, World!".
Kotlin : Write the code using the println() function to print "Hello, World!".
R : Use the cat() function to print "Hello, World!".
Ruby : Use the puts command to print "Hello, World!" .

| Programming Language | Source Code | Screenshot of output | Programming Paradigm |
|---|---|---|---|
| Go | package main<br><br>import "fmt"<br><br>func main() {<br>    fmt.Println("Hello World")<br>} | <br>$ go run main.go<br>Hello World | Object-Oriented<br>Imperative<br>Concurrent |
| Ruby | puts (“Hello World”) | <br>Hello World | Object-Oriented<br>Functional<br>Procedural |
| R | print("Hello, World!") | <br>[1] "Hello, World!" | Object-Oriented<br>Functional<br>Procedural |
| Kotlin | fun main() {<br>    println(“Hello World!”)<br>} | <br>Hello World!<br>> | Object-oriented<br>Functional<br>Procedural<br>Imperative |

### Binding Time

Given the generic statement **x = x + y** (i.e. the values of x and y are added and assigned to x), identify the binding time of each of the questions listed below. Is the information known at <u>language definition time</u>, <u>language implementation time</u>, <u>compilation/translation time</u>, or <u>runtime</u>?

| **Questions** | **Kotlin** | **Ruby** | **Go** | **R** |
|---|---|---|---|---|
| a. Set of possible types of x? | compilation time | compilation/translation time | compilation/translation time | language definition time |
| b. Data type of x? | runtime | runtime | compilation/translation time | runtime |
| c. Set of possible values of x? | runtime | runtime | compilation/translation time | runtime |
| d. Value of x? | runtime | runtime | runtime | runtime |
| e. Meaning of +? | language definition time | language definition time | language definition time | language definition time |

2. Provide the appropriate answers to the questions in the table below.

| Questions | Kotlin | Ruby | Go | R |
|---|---|---|---|---|
| How is a variable's data type specified? | In Kotlin, the variable's data type is specified with certain keywords such as Char, String, etc. | Ruby is a dynamically typed language, therefore the data type of a variable is based on the kind of value it is assigned to (known as "duck typing"). | In Go, we can define a variable with the keyword var followed by the name of the variable and the data type desired. | In R, you don't have to specify a variable's data type when you create it. Instead, the language figures out the data type based on the values you assign to the variable as you go along. |
| When is a data type bound to a variable? | In Kotlin, a data type is bound to a variable the moment it is initialized and assigned a value. | In Ruby, the data type is bound to a variable at the moment when you assign a value to that variable. | Since Go is a statically typed language, data types are bound to variables rather than values. | In R, a variable's data type is established during runtime, meaning it gets determined as the code runs, not during the writing or compiling stages. |
| Can the data type of a variable change? | In Kotlin, there is a way to change | In Ruby, the data type of a variable | Go is statically typed, meaning | In R, a variable's |

| | | | | |
|---|---|---|---|---|
| | the data type of a variable since Kotlin is the successor to the Java programming language. | can freely change at any point in time depending on what kind of value it is holding at that moment. | that once a variable type is defined, it can only store data of that type. | data type can change during runtime. R is dynamically typed, so a variable's data type isn't fixed until the code is actually executed. |
| When is memory space allocated to the variable? | Memory space is allocated to the variable when the variable is declared. | Memory space is allocated to a variable when you assign a value to that variable. Additionally, Ruby has a garbage collection system that then reclaims memory when it is no longer needed. | Stack memory is automatically allocated when a function is called. | In R, memory space is allocated to a variable either upon its creation or when a value is assigned to it. |
| When is memory space de-allocated from a variable? | Memory space is de-allocated from a variable when the variable is out of scope. | In Ruby, memory space is de-allocated as part of its garbage collection process. In general, memory space deallocation from a variable occurs when that variable is out of scope, no longer referenced, or is reassigned to a different value. | Whenever the function call is over, the memory for the variables is de-allocated. Typically, function parameters and local variables are allocated on the stack. | In R, memory space is de-allocated from a variable when it's no longer required, and this occurs when the garbage collection process is activated. Garbage |

| | | | | |
|---|---|---|---|---|
| | | | | collection is responsible for releasing memory that is no longer in use by the program. |
| When are variables initialized? | Variables are initialized at the start of the code and they should be assigned the correct data type. In Kotlin, if the value does not match the variable data type, the compiler would return an Error: Type Mismatch. | Variables are initialized when they are assigned a value. In Ruby, if no value for the variable is assigned, then that variable is assigned the value of nil by default. | If a variable is not assigned any value, Go automatically initializes it with the zero value of the variable's type. | In R, variables are initialized when they are assigned a value. If a variable is created but not assigned a value, its initial value is 'NULL'. |

1a. In each of the indicated PLs, write a program that swaps the values in two variables **x** and **y** using a temporary variable **temp**. Write the code in the table below. Your program should use all three variables (and only the three variables indicated) and initialize the **x** and **y** variables with the values as shown in the **Inputs** column. If the PL requires variable types to be explicitly declared, the **Data Type** column indicates the data type for both variables. Otherwise, you can disregard the **Data Type** column.

| **Data Type** | **Inputs** | **Kotlin** | **Ruby** | **Go** | **R** |
|---|---|---|---|---|---|
| int | x= 5<br>y= 6.5 | fun main() {<br><br>var x = 5<br>var y = 6.5<br><br>var temp = x<br>x = y<br>y = temp | x = 5<br>y = 6.5<br><br># Swapping<br>temp = x<br>x = y<br>y = temp<br><br># Check Swap | x := 5<br>y := 6<br><br>// swap<br>temp := x<br>x = y<br>y = temp | x <- 5<br>y <- 6.5<br><br># Swapping<br>temp <- x<br>x <- y<br>y <- temp |

| | | | | | |
|---|---|---|---|---|---|
| | | println("x = $x")<br>println("y = $y")<br><br>} | puts ("x = #{x}")<br>puts ("y = #{y}") | fmt.Printf(" x = %d and y = %d\n", x, y) | # Check Swap<br>cat("x =", x, "\n")<br>cat("y =", y, "\n") |
| float | x= -456.23456<br>y= “dollar” | fun main() {<br><br>var x = -456.23456f<br>var y = “dollar”<br><br>var temp = x<br>x = y<br>y = temp<br><br>println("x = $x")<br>println("y = $y")<br><br>} | x = -456.23456<br>y = “dollar”<br><br># Swapping<br>temp = x<br>x = y<br>y = temp<br><br># Check Swap<br>puts ("x = #{x}")<br>puts ("y = #{y}") | x := -456.23456<br>y := "dollar"<br><br>var temp interface{}<br>temp = x<br>x = float64(len(y))<br>y = fmt.Sprintf("%f", temp)<br><br>fmt.Printf(" x = %s and y = %.5f\n", y, x) | x <- -456.23456<br>y <- "dollar"<br><br># Swapping<br>temp <- x<br>x <- y<br>y <- temp<br><br># Check Swap<br>cat("x =", x, "\n")<br>cat("y =", y, "\n") |
| String | x= -234.654321190<br>y= “dollar interest” | fun main() {<br><br>var x = -234.654321190f<br>var y = “dollar interest”<br><br>var temp = x<br>x = y<br>y = temp<br><br>println("x = $x")<br>println("y = $y")<br><br>} | x = -234.654321190<br>y = "dollar interest"<br><br># Swapping<br>temp = x<br>x = y<br>y = temp<br><br># Check Swap<br>puts ("x = #{x}")<br>puts ("y = #{y}") | x := -234.654321190<br>y := "dollar interest"<br><br>//swap<br>temp := fmt.Sprintf("%.10f", x)<br>x = y<br>y = temp<br><br>/temp<br>fmt.Println("x:", x)<br>fmt.Println("y:", y) | x <- -234.654321190<br>y <- "dollar interest"<br><br># Swapping<br>temp <- x<br>x <- y<br>y <- temp<br><br># Check Swap<br>cat("x =", x, "\n")<br>cat("y =", y, "\n") |
| char | x= 4<br>y= -5 | fun main() {<br><br>var x: Char = 5<br>var y: Char = 6.5 | x = 4<br>y = -5<br><br># Swapping | x := '4'<br>y := '-5'<br>var temp rune | x <- 4<br>y <- -5<br><br># Swapping |

| | | | | | |
|---|---|---|---|---|---|
| | | var temp = x<br>x = y<br>y = temp<br><br>println("x = $x")<br>println("y = $y")<br><br>} | temp = x<br>x = y<br>y = temp<br><br># Check Swap<br>puts ("x = #{x}")<br>puts ("y = #{y}") | // Swap<br>temp = x<br>x = y<br>y = temp<br><br>fmt.Println("x:", string(x))<br>fmt.Println("y:", string(y)) | temp <- x<br>x <- y<br>y <- temp<br><br># Check Swap<br>cat("x =", x, "\n")<br>cat("y =", y, "\n") |

The program needs to define the integer variables x and y for Kotlin and initialize them with the specified values. The application should define and initialize variables x and y in Ruby and Go, without defining the data type. The programs in Kotlin and Ruby function as expected, but the program in Go fails because one of the variables was declared as a float on an integer data type. For R, the program should initialize x and y as numeric values. The table displays the program code for each language and whether the swap program was successful. Because one of the variables in the R program had a float on an integer data type, the program also failed.

1.b. Did the swapping program work for each of the input cases? In other words, were you able to come up with a solution in the indicated PL that successfully swapped the variables without error? If not, briefly explain in the table below why it is not possible or if additional instructions/commands might be needed. You're encouraged to use terms discussed in the Type Bindings lesson to articulate your thoughts. If there is an error, what is the error and when was it detected (e.g. compile time, runtime)?

| **Data Type** | **Inputs** | **Kotlin** | **Ruby** | **Go** | **R** |
|---|---|---|---|---|---|
| int | x= 5<br>y= 6.5 | No, Kotlin returned a Type mismatch. | Yes | No, Go returned a Type mismatch. | Yes |
| float | x= -456.23456<br>y= “dollar” | No, Kotlin returned a Type mismatch. | Yes | No, Go returned a Type mismatch. | Yes |
| String | x= -234.654321190<br>y= “dollar interest” | No, Kotlin returned a Type mismatch | Yes | No, Go returned a Type mismatch. | Yes |
| char | x= 4<br>y= -5 | No, Kotlin returned an error that says the literal does not conform to the expected type. | Yes | No, Go returned an error. more than one character in rune literal | Yes |

2.a. Replace your swap algorithm in Problem 1 with the algorithm or program segment below. The given program segment is supposed to perform the swap but does not use a temporary variable and leverages arithmetic operations to perform the swap. Like question 1.a, write the source code in the table below. Kindly refer to all notes found in 1.a.

```
x = x + y;
y = x - y;
x = x - y;
```

| Data Type | Inputs | Kotlin | Ruby | Go | R |
|---|---|---|---|---|---|
| int | x= 5<br>y= 6.5 | fun main() {<br>    var x = 5<br>    var y = 6.5<br><br><br>    x = x + y<br>    y = x - y<br>    x = x - y<br><br>    println("x = $x")<br>    println("y = $y")<br>} | x = 5<br>y = 6.5<br><br># Provided Program Segment<br>x = x + y;<br>y = x - y;<br>x = x - y;<br><br># Check Swap<br>puts ("x = #{x}")<br>puts ("y = #{y}") | x := 5<br>y := 6<br><br><br>x = x + y<br>y = x - y<br>x = x - y<br><br>fmt.Printf("x = %d and y = %d\n", x, y) | x <- 5<br>y <- 6.5<br><br># Provided Program Segment<br>x <- x + y<br>y <- x - y<br>x <- x - y<br><br># Check Swap<br>cat("x =", x, "\n")<br>cat("y =", y, "\n") |
| float | x= -456.23456<br>y= “dollar” | fun main() {<br>    var x= -456.23456f<br>    var y= "dollar"<br><br><br>    x = x + y<br>    y = x - y<br>    x = x - y<br><br>    println(x)<br>    println(y)<br>} | =begin<br>This program results int a TypeError:<br>x= -456.23456<br>y= "dollar"<br><br># Provided Program Segment<br>x = x + y;<br>y = x - y;<br>x = x - y;<br><br># Check Swap<br>puts ("x = #{x}")<br>puts ("y = #{y}")<br>=end<br><br># A workaround for this would be | x := -456.23456<br>y := "dollar"<br>    ‘<br>x, y = float64(len(y)), fmt.Sprintf("%f", x)<br><br>fmt.Printf("x = %s and y = %.5f\n", y, x) | x <- -456.23456<br>y <- "dollar"<br><br># Provided Program Segment<br>x <- x + y<br>y <- x - y<br>x <- x - y<br><br># Check Swap<br>cat("x =", x, "\n")<br>cat("y =", y, "\n") |

| | | | | | |
|---|---|---|---|---|---|
| | | | to convert the value of y to a float<br>x= -456.23456<br>y= "dollar".to_f<br><br># Provided Program Segment<br>x = x + y;<br>y = x - y;<br>x = x - y;<br><br># Check Swap<br>puts ("x = #{x}")<br>puts ("y = #{y}")<br><br># This results in “dollar” converting to its float equivalent of 0.0 | | |
| String | x= -234.654321190<br>y= “dollar interest” | fun main() {<br>    var x = -234.654321190<br>    var y = "dollar interest"<br><br><br>    x = x + y<br>    y = x - y<br>    x = x - y<br><br>    println(x)<br>    println(y)<br>} | =begin<br>This program does not work as a way to swap two strings:<br>x = -234.654321190<br>y = "dollar interest"<br><br># Provided Program Segment<br>x = x + y;<br>y = x - y;<br>x = x - y;<br><br># Check Swap<br>puts ("x = #{x}")<br>puts ("y = #{y}")<br>=end<br><br># A workaround for this is to alter the provided program segment | x := -234.654321190<br>y := "dollar interest"<br><br>// Swap x and y<br>x, y = float64(len(y)), fmt.Sprintf("%.10f", x)<br><br>// Print the swapped values<br>fmt.Println("x:", x)<br>fmt.Println("y:", y) | x <- -234.654321190<br>y <- "dollar interest"<br><br># Provided Program Segment<br>x <- x + y<br>y <- x - y<br>x <- x - y<br><br># Check Swap<br>cat("x =", x, "\n")<br>cat("y =", y, "\n") |

| | | | | | |
|---|---|---|---|---|---|
| | | | in the context of string concatenation<br><br># x's value is converted to a string just like with the last workaround<br>x = -234.654321190.to_s<br>y = "dollar interest"<br><br># Provided Program Segment but in the context of string concatenation<br>x = x + y;<br>y = x[0, x.length - y.length];<br>x = x[y.length, x.length - y.length];<br><br># Check Swap<br>puts ("x = #{x}")<br>puts ("y = #{y}") | | |
| char | x= 4<br>y= -5 | fun main() {<br>    var x: Char = 4<br>    var y: Char = -5<br><br>    x = x + y<br>    y = x - y<br>    x = x - y<br><br>    println(x)<br>    println(y)<br>} | x = 4<br>y = -5<br><br># Provided Program Segment<br>x = x + y;<br>y = x - y;<br>x = x - y;<br><br># Check Swap<br>puts ("x = #{x}")<br>puts ("y = #{y}") | x := '4'<br>y := '-5'<br><br>x, y = y, x<br><br>fmt.Printf("x: %c\n", x)<br>fmt.Printf("y: %c\n", y) | x <- 4<br>y <- -5<br><br># Provided Program Segment<br>x <- x + y<br>y <- x - y<br>x <- x - y<br><br># Check Swap<br>cat("x =", x, "\n")<br>cat("y =", y, "\n") |

The following table provides an overview of how the specified swap technique was implemented in four distinct programming languages (Kotlin, Ruby, Go, and R) utilizing varying input data types (int, float, string, and char) without the need for a temporary variable. Due to faults or mismatched data types, the swap algorithm did not function in Kotlin and Go, however it did in Ruby and R. For instance, in Ruby, the swap algorithm failed while attempting to conduct arithmetic operations as one variable was a string and the other was a float. Similar to this, because the variables in R were undefined, the algorithm did not function for the forchar data type.

2.b. Like in question 1.b., did the swap algorithm work with the provided input cases? Provide answers to the cells of your respective PL. Refer to question 1.b. for the notes on how to answer this question.

| **Data Type** | **Inputs** | **Kotlin** | **Ruby** | **Go** | **R** |
|---|---|---|---|---|---|
| int | x= 5<br>y= 6.5 | No, Kotlin returned a Type Mismatch statement saying inferred type is a Double but regarded as an Int. | Yes | No, Go returned a Type mismatch. | Yes |
| float | x= -456.23456<br>y= “dollar” | No, Kotlin returned a Type Mismatch | No. Despite Ruby being a dynamically typed language, a TypeError will still occur at runtime if the original provided program segment is implemented without any additional workarounds. | No, Go returned a Type mismatch. | No, R returned an error because you cannot add, subtract, or perform arithmetic operations between different data types.<br>“Error |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | in x + y : non-numeric argument to binary operator Execution halted” |
| String | x= -234.654321190<br>y= “dollar interest” | No, Kotlin returned a Type Mismatch | No. Despite Ruby being a dynamically typed language, a TypeError will still occur at runtime if the original provided program segment is implemented without any additional workarounds. | No, Go returned a Type mismatch. | No, R returned an error because you cannot add, subtract, or perform arithmetic operations between different data types.<br>“Error in x + y : non-numeric argument to binary |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
|  |  |  |  |  | operator Execution halted ” |
| char | x= 4<br>y= -5 | No, Kotlin returned a Type Mismatch | Yes | No, Go returned a Type mismatch. | Yes |

3. The tables above show the behavior of how a programming language behaves when encountering incompatible types. Discuss your selected language handles type incompatibility. You’re encouraged to use terms discussed in the Type Bindings lesson to articulate your thoughts.

| **Language** | **How is type incompatibility handled?** |
|---|---|
| Kotlin | Type incompatibility in Kotlin is managed by informing the user of the incompatibility and displaying numerous warnings. The compiler produced messages stating that this or that type was required when the code was ran through it. Changing the data type will enable the compiler to read and process the code, which will resolve the issue. |
| Ruby | Ruby is a dynamically typed language, variables can contain values of various data types at different points in the program because their data type is not fixed. Furthermore, variables' data types are not stated clearly. This explains why the initial swapping procedure that used the temp variable worked in every circumstance. Nevertheless, if you try to operate on two incompatible data types (like float and string), Ruby will still raise a TypeError. |
| Go | The  Go language is strongly typed and statically typed. In order to prevent operations between incompatible types and guarantee that type-related errors are discovered prior to runtime, type compatibility is strictly enforced at compile time. Variable types in Go must be declared explicitly, but type inference is also included, enabling the compiler to infer the type from the assigned value. Code safety and predictability are enhanced by this strong mechanism for handling type incompatibility, which combines type inference, strong typing, and static typing. |
| R | Automatic type conversion is a common way to handle type incompatibility in R. This indicates that R will attempt to convert one variable to the type of the other variable before performing the operation if it is applied to two variables of different types. |

**Application**

| **Language** | **compute_hours_difference** |
|---|---|

| | |
|---|---|
| Kotlin | fun parseMilitaryTime(timeString: String): Date {<br>    val format = SimpleDateFormat("HH:mm:ss")<br>    format.timeZone = TimeZone.getTimeZone("UTC")<br>    return format.parse(timeString)<br>}<br><br>fun computeHoursDifference(startTime: String, endTime: String): Int {<br>    val startDateTime = parseMilitaryTime(startTime).time<br>    val endDateTime = parseMilitaryTime(endTime).time<br><br>    val differenceInHours = Math.max((endDateTime - startDateTime) / 3600000, 0).toInt()<br>    return differenceInHours |
| Ruby | def compute_hours_difference(start_time, end_time)<br>  # Parse the input strings into Time objects<br>  start_time = parse_military_time(start_time)<br>  end_time = parse_military_time(end_time)<br>   # Calculate the time difference in seconds and convert to hours<br>  difference_in_hours = [(end_time - start_time) / 3600, 0].max<br>   difference_in_hours.to_i  # Convert to integer<br>end |
| Go | func parseMilitaryTime(timeString string) time.Time {<br>        parsedTime, _ := time.Parse("15:04:05", timeString)<br>        return parsedTime<br>}<br><br>func computeHoursDifference(startTime string, endTime string) int {<br>        startTimeObj := parseMilitaryTime(startTime)<br>        endTimeObj := parseMilitaryTime(endTime)<br><br>        differenceInHours := int(endTimeObj.Sub(startTimeObj).Hours())<br>        if differenceInHours < 0 {<br>                return 0<br>        }<br>        return differenceInHours<br>} |
| R | computeHoursDifference <- function(start_time, end_time) {<br>  # Parse the input strings into POSIXct objects<br>  start_time <- as.POSIXct(start_time, format="%H:%M:%S", tz="UTC")<br>  end_time <- as.POSIXct(end_time, format="%H:%M:%S", tz="UTC") |

| | |
|---|---|
| |   # Calculate the time difference in hours<br>  difference_in_hours <- as.numeric(difftime(end_time, start_time, units="hours"))<br>  difference_in_hours <- ifelse(difference_in_hours < 0, 0, difference_in_hours)<br><br>  return(floor(difference_in_hours))<br>}<br><br># Example usage<br>start_time <- "08:00:00"<br>end_time <- "12:30:00"<br>result <- computeHoursDifference(start_time, end_time)<br>print(result) |

## Chapter 3: Developing the Weekly Payroll System using Ruby

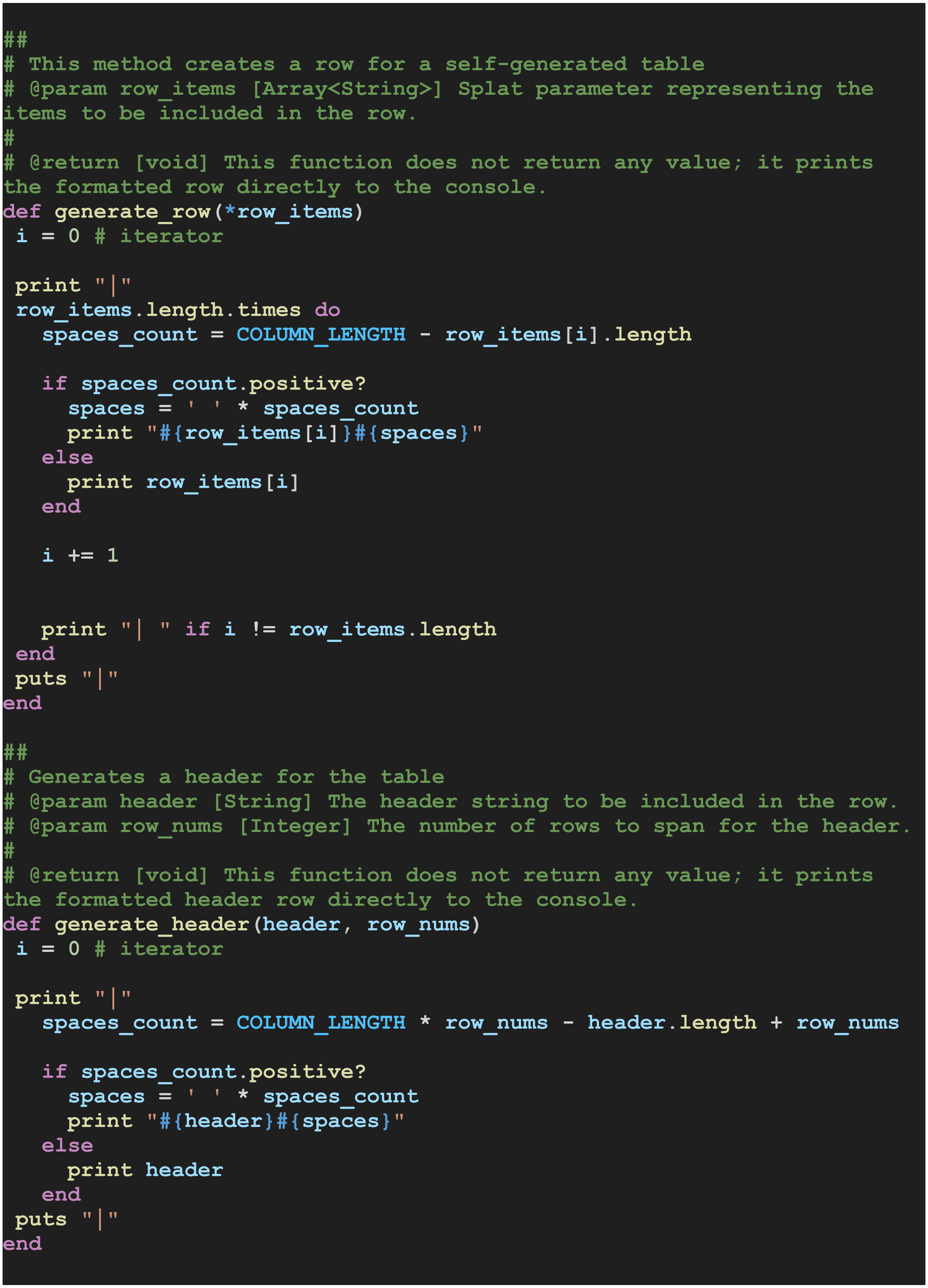
Our Weekly Payroll System was programmed in Ruby. It has all of the key functions needed for completing all of the requirements as defined in the specifications of this project. Its core features include:

- Automation of payroll computation considering factors including but not limited to type of day and overtime pay
- Ability to manually adjust default configurations
- Daily and weekly payroll generation

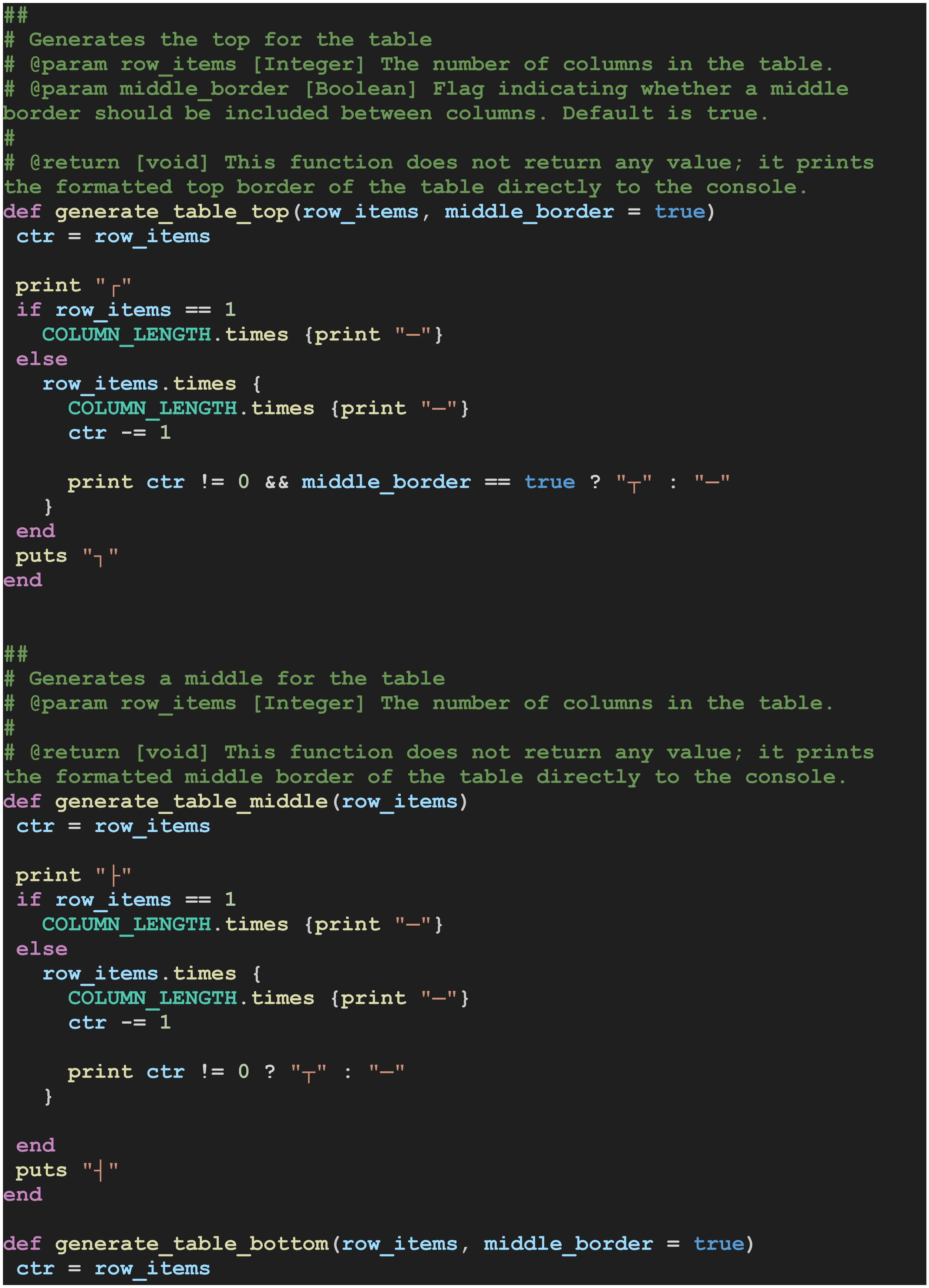
Ruby was the programming language of choice due to its simplicity and developer-friendly syntax. It shines most in short programs due to its convenient built-in methods, examples of which include .times and .sum that does a block n times and computes the sum of the values in an array respectively. Additionally, Ruby is a dynamically-typed language, which also makes it more convenient for programming but also provides its own cons.

The main drawback experienced in using Ruby for the development of the program is that many of the development stage's errors occurred in runtime, leading to an inconvenient debugging experience. Ruby is an interpreted language, so it provides a different experience from Java and C, both of which are compiled languages which CSADPRG students would be more familiar with prior to learning Ruby. Part of the learning experience was working with these drawbacks to better understand how to use the language more efficiently and effectively.

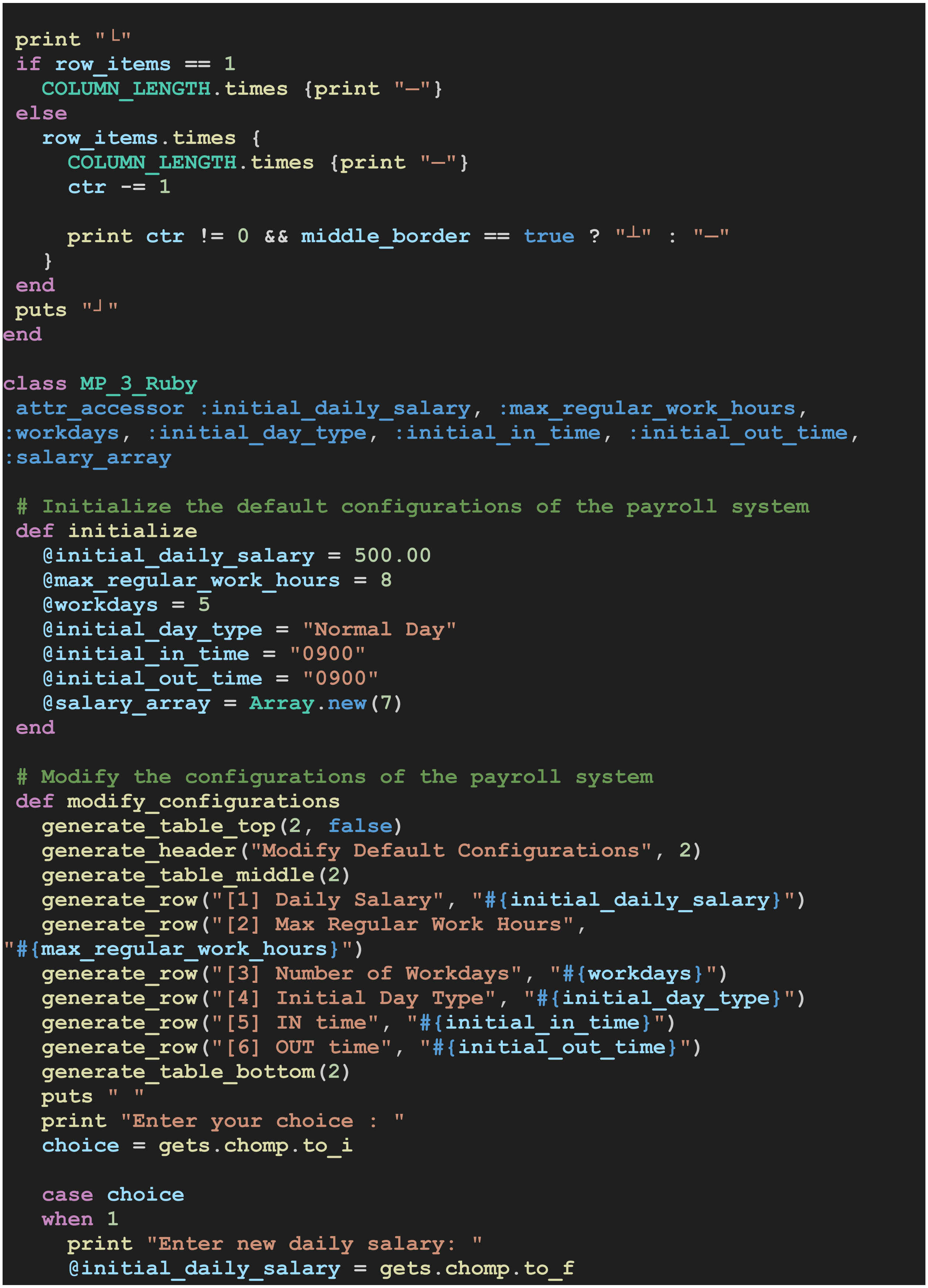
```ruby
=begin
Last names: Feliciano, Ty, Rodriguez, Clemente
Language: Ruby
Paradigm(s): Object-Oriented, Functional
=end

require 'time' #for parsing time

# Define the character length for each table row
COLUMN_LENGTH = 40
```



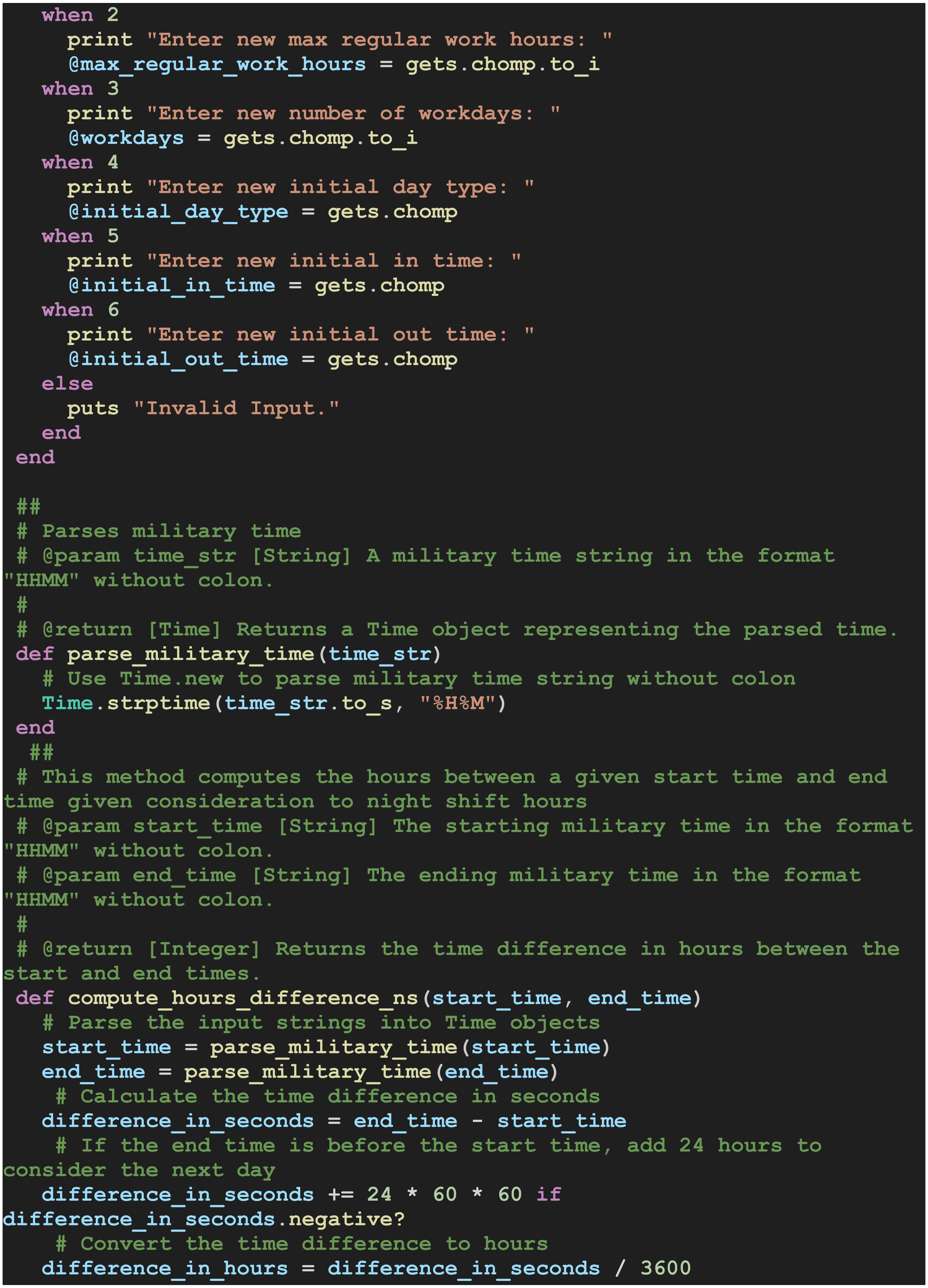
```ruby
##
# This method creates a row for a self-generated table
# @param row_items [Array<String>] Splat parameter representing the
items to be included in the row.
#
# @return [void] This function does not return any value; it prints
the formatted row directly to the console.
def generate_row(*row_items)
 i = 0 # iterator

 print "|"
 row_items.length.times do
   spaces_count = COLUMN_LENGTH - row_items[i].length

   if spaces_count.positive?
     spaces = ' ' * spaces_count
     print "#{row_items[i]}#{spaces}"
   else
     print row_items[i]
   end

   i += 1


   print "| " if i != row_items.length
 end
 puts "|"
end

##
# Generates a header for the table
# @param header [String] The header string to be included in the row.
# @param row_nums [Integer] The number of rows to span for the header.
#
# @return [void] This function does not return any value; it prints
the formatted header row directly to the console.
def generate_header(header, row_nums)
 i = 0 # iterator

 print "|"
   spaces_count = COLUMN_LENGTH * row_nums - header.length + row_nums

   if spaces_count.positive?
     spaces = ' ' * spaces_count
     print "#{header}#{spaces}"
   else
     print header
   end
 puts "|"
end
```



```
##
# Generates the top for the table
# @param row_items [Integer] The number of columns in the table.
# @param middle_border [Boolean] Flag indicating whether a middle
border should be included between columns. Default is true.
#
# @return [void] This function does not return any value; it prints
the formatted top border of the table directly to the console.
def generate_table_top(row_items, middle_border = true)
 ctr = row_items

 print "┌"
 if row_items == 1
   COLUMN_LENGTH.times {print "─"}
 else
   row_items.times {
     COLUMN_LENGTH.times {print "─"}
     ctr -= 1

     print ctr != 0 && middle_border == true ? "┬" : "─"
   }
 end
 puts "┐"
end


##
# Generates a middle for the table
# @param row_items [Integer] The number of columns in the table.
#
# @return [void] This function does not return any value; it prints
the formatted middle border of the table directly to the console.
def generate_table_middle(row_items)
 ctr = row_items

 print "├"
 if row_items == 1
   COLUMN_LENGTH.times {print "─"}
 else
   row_items.times {
     COLUMN_LENGTH.times {print "─"}
     ctr -= 1

     print ctr != 0 ? "┬" : "─"
   }

 end
 puts "┤"
end

def generate_table_bottom(row_items, middle_border = true)
 ctr = row_items
```

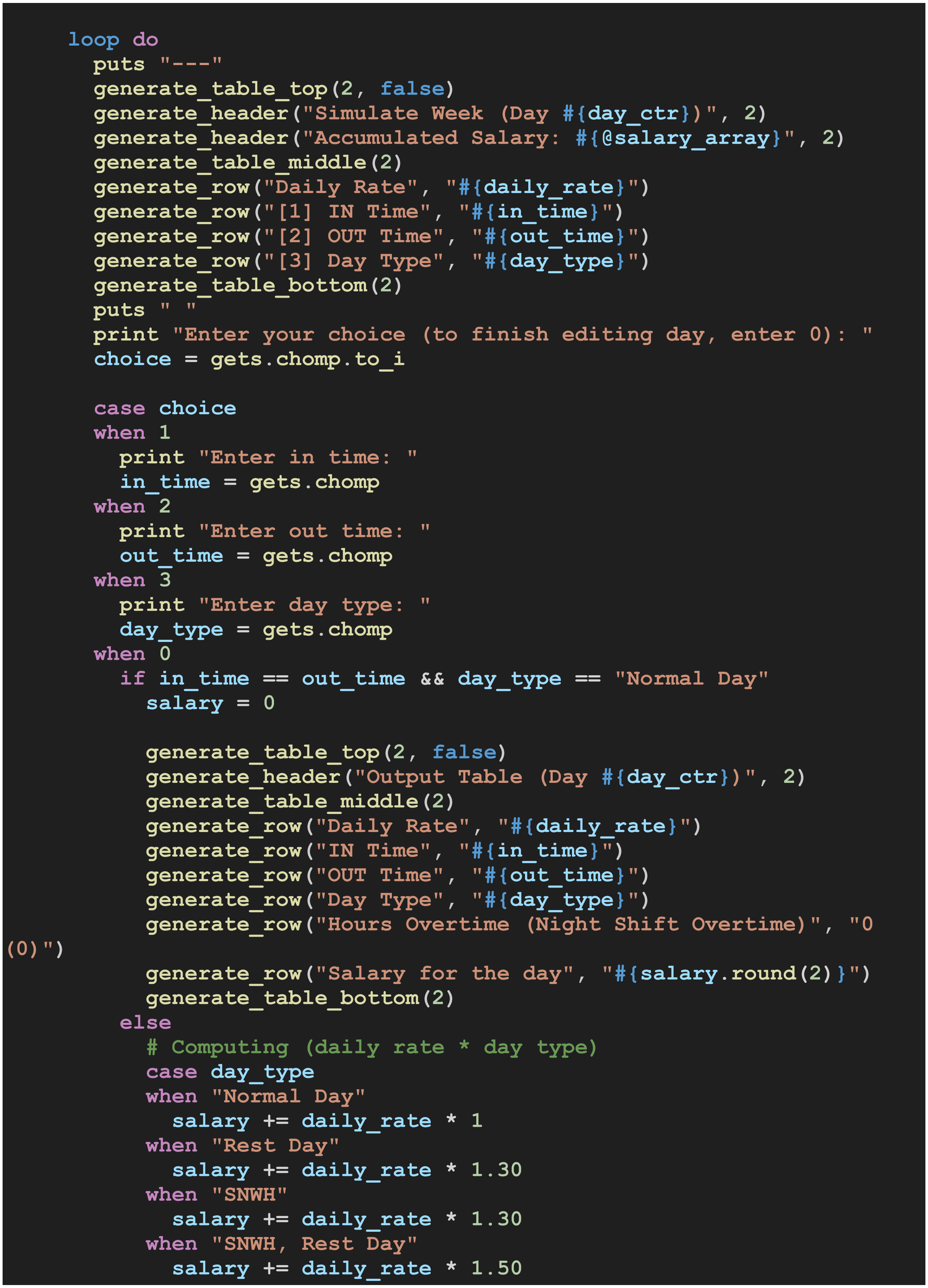


```
 print "└"
 if row_items == 1
   COLUMN_LENGTH.times {print "─"}
 else
   row_items.times {
     COLUMN_LENGTH.times {print "─"}
     ctr -= 1

     print ctr != 0 && middle_border == true ? "┴" : "─"
   }
 end
 puts "┘"
end

class MP_3_Ruby
 attr_accessor :initial_daily_salary, :max_regular_work_hours,
:workdays, :initial_day_type, :initial_in_time, :initial_out_time,
:salary_array

 # Initialize the default configurations of the payroll system
 def initialize
   @initial_daily_salary = 500.00
   @max_regular_work_hours = 8
   @workdays = 5
   @initial_day_type = "Normal Day"
   @initial_in_time = "0900"
   @initial_out_time = "0900"
   @salary_array = Array.new(7)
 end

 # Modify the configurations of the payroll system
 def modify_configurations
   generate_table_top(2, false)
   generate_header("Modify Default Configurations", 2)
   generate_table_middle(2)
   generate_row("[1] Daily Salary", "#{initial_daily_salary}")
   generate_row("[2] Max Regular Work Hours",
"#{max_regular_work_hours}")
   generate_row("[3] Number of Workdays", "#{workdays}")
   generate_row("[4] Initial Day Type", "#{initial_day_type}")
   generate_row("[5] IN time", "#{initial_in_time}")
   generate_row("[6] OUT time", "#{initial_out_time}")
   generate_table_bottom(2)
   puts " "
   print "Enter your choice : "
   choice = gets.chomp.to_i

   case choice
   when 1
     print "Enter new daily salary: "
     @initial_daily_salary = gets.chomp.to_f
```

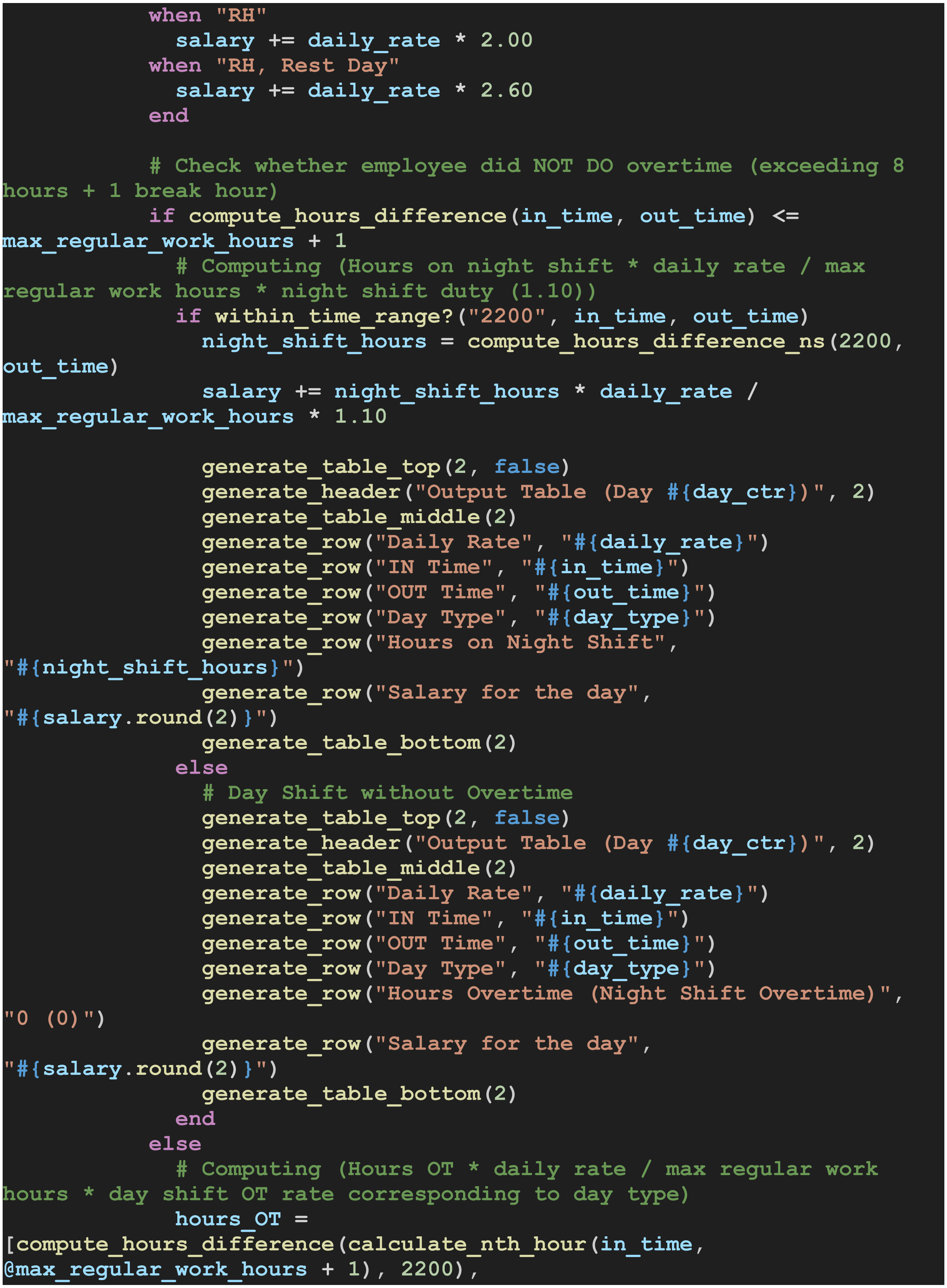


```ruby
   when 2
     print "Enter new max regular work hours: "
     @max_regular_work_hours = gets.chomp.to_i
   when 3
     print "Enter new number of workdays: "
     @workdays = gets.chomp.to_i
   when 4
     print "Enter new initial day type: "
     @initial_day_type = gets.chomp
   when 5
     print "Enter new initial in time: "
     @initial_in_time = gets.chomp
   when 6
     print "Enter new initial out time: "
     @initial_out_time = gets.chomp
   else
     puts "Invalid Input."
   end
 end

 ##
 # Parses military time
 # @param time_str [String] A military time string in the format "HHMM" without colon.
 #
 # @return [Time] Returns a Time object representing the parsed time.
 def parse_military_time(time_str)
   # Use Time.new to parse military time string without colon
   Time.strptime(time_str.to_s, "%H%M")
 end
  ##
 # This method computes the hours between a given start time and end time given consideration to night shift hours
 # @param start_time [String] The starting military time in the format "HHMM" without colon.
 # @param end_time [String] The ending military time in the format "HHMM" without colon.
 #
 # @return [Integer] Returns the time difference in hours between the start and end times.
 def compute_hours_difference_ns(start_time, end_time)
   # Parse the input strings into Time objects
   start_time = parse_military_time(start_time)
   end_time = parse_military_time(end_time)
    # Calculate the time difference in seconds
   difference_in_seconds = end_time - start_time
    # If the end time is before the start time, add 24 hours to consider the next day
   difference_in_seconds += 24 * 60 * 60 if difference_in_seconds.negative?
    # Convert the time difference to hours
   difference_in_hours = difference_in_seconds / 3600
```

```
    difference_in_hours.to_i
 end

 ##
 # Computes the time difference in hours between two military time
strings.
 # @param start_time [String] The starting military time in the format
"HHMM" without colon.
 # @param end_time [String] The ending military time in the format
"HHMM" without colon.
 #
 # @return [Integer] Returns the time difference in hours between the
start and end times.
 def compute_hours_difference(start_time, end_time)
   # Parse the input strings into Time objects
   start_time = parse_military_time(start_time)
   end_time = parse_military_time(end_time)
    # Calculate the time difference in seconds and convert to hours
   difference_in_hours = [(end_time - start_time) / 3600, 0].max
    difference_in_hours.to_i  # Convert to integer
 end
  ##
 # Checks if a target hour is within the specified time range.
 # @param target_hour_str [String] The target hour in the format "HH".
 # @param start_hour_str [String] The starting hour of the time range
in the format "HH".
 # @param end_hour_str [String] The ending hour of the time range in
the format "HH".
 #
 # @return [Boolean] Returns true if the target hour is within the
specified time range; otherwise, returns false.
 def within_time_range?(target_hour_str, start_hour_str, end_hour_str)
   target_hour = target_hour_str.to_i
   start_hour = start_hour_str.to_i
   end_hour = end_hour_str.to_i
    if start_hour <= end_hour
     # Range is within a single day
     return target_hour.between?(start_hour, end_hour)
   else
     # Range spans across midnight (e.g., 1800 to 0300)
     return target_hour >= start_hour || target_hour <= end_hour
   end
 end

 ##
 # Calculates the time after a specified number of hours from a given
start time.
 # @param start_time [String] The starting time in the format "HHMM"
without colon.
 # @param n [Integer] The number of hours to calculate.
 #
 # @return [String] Returns the calculated time after n hours from the
start time in the format "HHMM".

 def calculate_nth_hour(start_time, n)
   # Convert start_time to DateTime object
   start_datetime = DateTime.strptime(start_time, "%H%M")
    # Calculate nth hour
   nth_hour_datetime = start_datetime + Rational(n, 24)
    # Format nth_hour_datetime as HHMM string
   nth_hour = nth_hour_datetime.strftime("%H%M")
    return nth_hour
 end

 ##
 # Simulates the work week with each day configured according to
configurations by default
 #
 # @return [void] This function does not return any value; it provides
an interaction between the user and the payroll system and facilitates
some computations
 def simulate_week
   # Assumptions (from specs):
   # - Employees are not late
   # - Employees work for at least 8 hours
   # - Employees may be absent

   # General formula for computing daily salary (I made this myself so
it might be wrong):
   # IF normal day and employee did not work (in time = out time)
   #   then salary = 0
   # ELSE
   #   then salary = (daily rate * day type)
   #   IF NOT OT (in time to out time <= max regular work hours + 1
rest hour)
   #     + (Hours on night shift * daily rate / max regular work hours
* night shift duty (1.10))
   #   ELSE
   #     + (Hours OT * daily rate / max regular work hours * day shift
OT rate corresponding to day type)
   #     + (Night shift Hours OT * daily rate / max regular work hours
* night shift OT rate corresponding to day type)


   day_ctr = 1

   salary_array.length.times do
       # Initialize table
       daily_rate = initial_daily_salary
       in_time = initial_in_time
       out_time = initial_out_time
       day_type = initial_day_type
       salary = 0
```

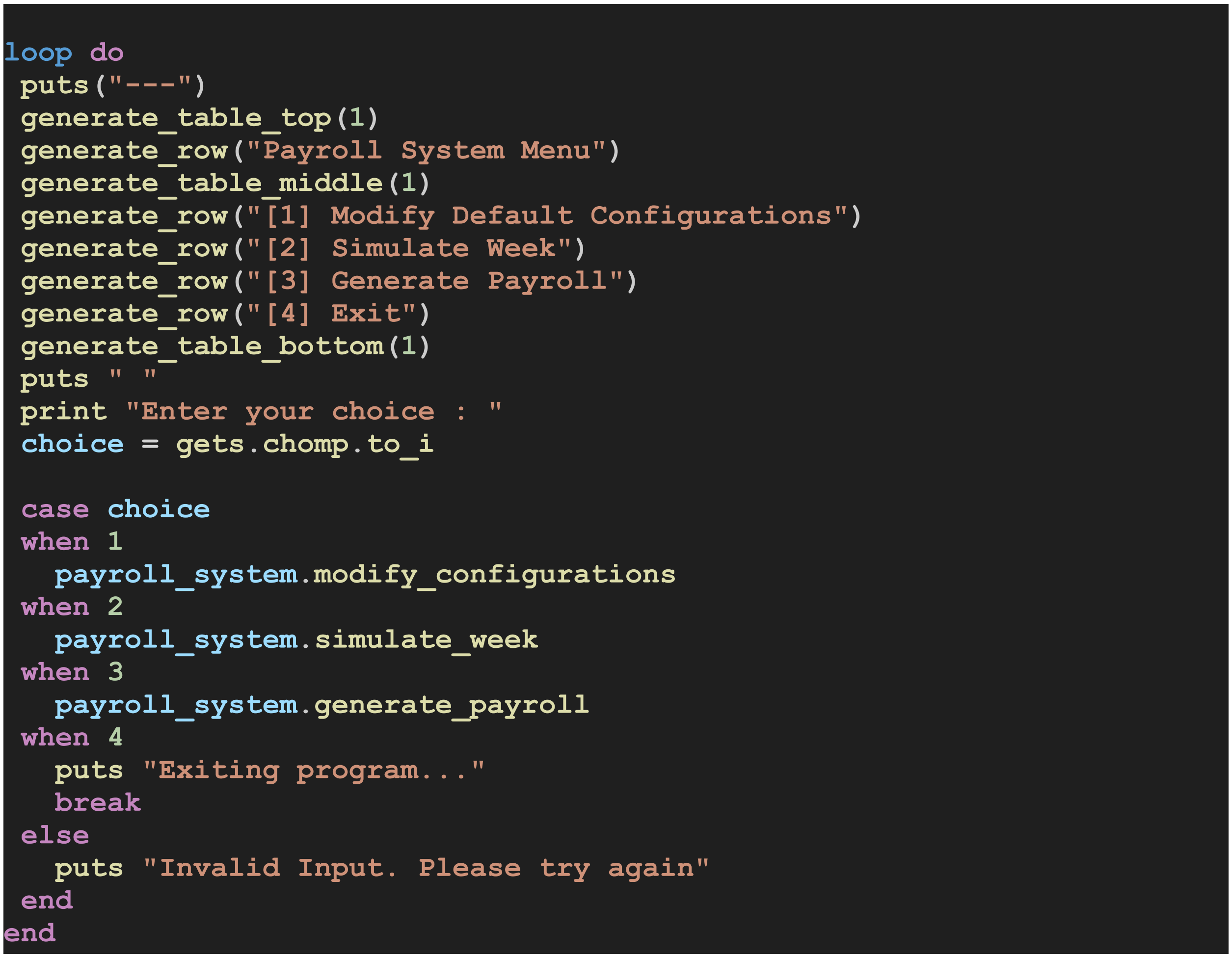


```
    loop do
      puts "---"
      generate_table_top(2, false)
      generate_header("Simulate Week (Day #{day_ctr})", 2)
      generate_header("Accumulated Salary: #{@salary_array}", 2)
      generate_table_middle(2)
      generate_row("Daily Rate", "#{daily_rate}")
      generate_row("[1] IN Time", "#{in_time}")
      generate_row("[2] OUT Time", "#{out_time}")
      generate_row("[3] Day Type", "#{day_type}")
      generate_table_bottom(2)
      puts " "
      print "Enter your choice (to finish editing day, enter 0): "
      choice = gets.chomp.to_i

      case choice
      when 1
        print "Enter in time: "
        in_time = gets.chomp
      when 2
        print "Enter out time: "
        out_time = gets.chomp
      when 3
        print "Enter day type: "
        day_type = gets.chomp
      when 0
        if in_time == out_time && day_type == "Normal Day"
          salary = 0

          generate_table_top(2, false)
          generate_header("Output Table (Day #{day_ctr})", 2)
          generate_table_middle(2)
          generate_row("Daily Rate", "#{daily_rate}")
          generate_row("IN Time", "#{in_time}")
          generate_row("OUT Time", "#{out_time}")
          generate_row("Day Type", "#{day_type}")
          generate_row("Hours Overtime (Night Shift Overtime)", "0
(0)")
          generate_row("Salary for the day", "#{salary.round(2)}")
          generate_table_bottom(2)
        else
          # Computing (daily rate * day type)
          case day_type
          when "Normal Day"
            salary += daily_rate * 1
          when "Rest Day"
            salary += daily_rate * 1.30
          when "SNWH"
            salary += daily_rate * 1.30
          when "SNWH, Rest Day"
            salary += daily_rate * 1.50
```



```
           when "RH"
             salary += daily_rate * 2.00
           when "RH, Rest Day"
             salary += daily_rate * 2.60
           end

           # Check whether employee did NOT DO overtime (exceeding 8
hours + 1 break hour)
           if compute_hours_difference(in_time, out_time) <=
max_regular_work_hours + 1
             # Computing (Hours on night shift * daily rate / max
regular work hours * night shift duty (1.10))
             if within_time_range?("2200", in_time, out_time)
               night_shift_hours = compute_hours_difference_ns(2200,
out_time)
               salary += night_shift_hours * daily_rate /
max_regular_work_hours * 1.10

               generate_table_top(2, false)
               generate_header("Output Table (Day #{day_ctr})", 2)
               generate_table_middle(2)
               generate_row("Daily Rate", "#{daily_rate}")
               generate_row("IN Time", "#{in_time}")
               generate_row("OUT Time", "#{out_time}")
               generate_row("Day Type", "#{day_type}")
               generate_row("Hours on Night Shift",
"#{night_shift_hours}")
               generate_row("Salary for the day",
"#{salary.round(2)}")
               generate_table_bottom(2)
             else
               # Day Shift without Overtime
               generate_table_top(2, false)
               generate_header("Output Table (Day #{day_ctr})", 2)
               generate_table_middle(2)
               generate_row("Daily Rate", "#{daily_rate}")
               generate_row("IN Time", "#{in_time}")
               generate_row("OUT Time", "#{out_time}")
               generate_row("Day Type", "#{day_type}")
               generate_row("Hours Overtime (Night Shift Overtime)",
"0 (0)")
               generate_row("Salary for the day",
"#{salary.round(2)}")
               generate_table_bottom(2)
             end
           else
             # Computing (Hours OT * daily rate / max regular work
hours * day shift OT rate corresponding to day type)
             hours_OT =
[compute_hours_difference(calculate_nth_hour(in_time,
@max_regular_work_hours + 1), 2200),
compute_hours_difference(calculate_nth_hour(in_time,
@max_regular_work_hours + 1), out_time)].min

             case day_type
             when "Normal Day"
               salary += hours_OT * daily_rate /
max_regular_work_hours * 1.25
             when "Rest Day"
               salary += hours_OT * daily_rate /
max_regular_work_hours * 1.69
             when "SNWH"
               salary += hours_OT * daily_rate /
max_regular_work_hours * 1.69
             when "SNWH, Rest Day"
               salary += hours_OT * daily_rate /
max_regular_work_hours * 1.95
             when "RH"
               salary += hours_OT * daily_rate /
max_regular_work_hours * 2.60
             when "RH, Rest Day"
               salary += hours_OT * daily_rate /
max_regular_work_hours * 3.38
             end


             # Computing (Night shift Hours OT * daily rate / max
regular work hours * night shift OT rate corresponding to day type)
             hours_OT_NS = compute_hours_difference(2200, out_time)

             case day_type
             when "Normal Day"
               salary += hours_OT_NS * daily_rate /
max_regular_work_hours * 1.375
             when "Rest Day"
               salary += hours_OT_NS * daily_rate /
max_regular_work_hours * 1.859
             when "SNWH"
               salary += hours_OT_NS * daily_rate /
max_regular_work_hours * 1.859
             when "SNWH, Rest Day"
               salary += hours_OT_NS * daily_rate /
max_regular_work_hours * 2.145
             when "RH"
               salary += hours_OT_NS * daily_rate /
max_regular_work_hours * 2.860
             when "RH, Rest Day"
               salary += hours_OT_NS * daily_rate /
max_regular_work_hours * 3.718
             end

             generate_table_top(2, false)
             generate_header("Output Table (Day #{day_ctr})", 2)
```

```
             generate_table_middle(2)
             generate_row("Daily Rate", "#{daily_rate}")
             generate_row("IN Time", "#{in_time}")
             generate_row("OUT Time", "#{out_time}")
             generate_row("Day Type", "#{day_type}")
             generate_row("Hours Overtime (Night Shift Overtime)",
"#{hours_OT} (#{hours_OT_NS})")
             generate_row("Salary for the day", "#{salary.round(2)}")
             generate_table_bottom(2)
           end
         end

         @salary_array[day_ctr - 1] = (salary.round(2)) #pushes sthe
salary for the day to the array


         day_ctr += 1 #moves on to the next day

         break #ends the loop and begins the next day
       end
     end
   end
 end

 # Generates a payroll report table.
 #
 # This function prints a formatted table with daily salaries and the
total salary for the week.
 #
 # @return [void] This function does not return any value; it prints
the formatted payroll report directly to the console.
 def generate_payroll

   generate_table_top(2, false)
   generate_header("Generate Payroll", 2)
   generate_table_middle(2)
   for i in 1..7
     generate_row("Day #{i}", "#{@salary_array[i - 1]}")
   end

   if @salary_array.all? { |element| element.nil? }
     generate_row("Total Salary for the Week", "0")
   else
     generate_row("Total Salary for the Week", "#{@salary_array.sum}")
   end

   generate_table_bottom(2)
 end
end

# Main program
payroll_system = MP_3_Ruby.new
```



```
loop do
 puts("---")
 generate_table_top(1)
 generate_row("Payroll System Menu")
 generate_table_middle(1)
 generate_row("[1] Modify Default Configurations")
 generate_row("[2] Simulate Week")
 generate_row("[3] Generate Payroll")
 generate_row("[4] Exit")
 generate_table_bottom(1)
 puts " "
 print "Enter your choice : "
 choice = gets.chomp.to_i

 case choice
 when 1
   payroll_system.modify_configurations
 when 2
   payroll_system.simulate_week
 when 3
   payroll_system.generate_payroll
 when 4
   puts "Exiting program..."
   break
 else
   puts "Invalid Input. Please try again"
 end
end
```

# Chapter 4: Conclusion

Programming the Payroll application has been quite challenging as the group has tried using four different programming languages to create the software application. Some insights the group was able to gain was that each language has a purpose in programming. Each programming language has its strengths and weaknesses which is why our group decided to use Ruby for the software development of the Payroll application. For example, R is used in data analysis and Kotlin is mainly used in app development. By using the Ruby programming language the group was able to accomplish the requirements given by the specifications of the project. In conclusion, the software application is doable by the four languages but to our group's opinion, we thought best that Ruby was the best choice in programming the software application.

## Reference List

EMB Global. (2023, November 7). *The Future of Programming Languages: Trends and Innovations. EMB Global Blog.* https://blog.emb.global/future-of-programming-languages/

Cambridge Blog. (2017). *Ada Lovelace: Programming Pioneer. Cambridge Blog.* http://www.cambridgeblog.org/2017/11/ada-lovelace-programming-pioneer/

Scientific American. (2015). In Celebration of Ada Lovelace, the First Computer Programmer. Scientific American. https://www.scientificamerican.com/article/in-celebration-of-ada-lovelace-the-first-computer-programmer/

Rubycademy. (2019, February 15). *Ruby is a multi-paradigm programming language.* Medium. https://medium.com/rubycademy/ruby-is-a-multi-paradigm-programming-language-49c8bc5fca80

Clark, K. L., & McCabe, F. G. (2004). *Go! — A Multi-Paradigm Programming Language for Implementing Multi-Threaded Agents*. Annals of Mathematics and Artificial Intelligence, 41, 171–206. https://doi.org/10.1023/B:AMAI.0000031195.87297.d9\

TechRepublic. (n.d.). *Programming languages: How to learn Kotlin with these resources for developers.* TechRepublic. https://www.techrepublic.com/article/programming-languages-how-to-learn-kotlin-with-these-resources-for-developers/

Kotlin. *Home.* Kotlin. https://kotlinlang.org/docs/home.html

LaZerte, S. E., & Albers, S. (2018). *How to cite R and R packages.* rOpenSci. https://ropensci.org/blog/2021/11/16/how-to-cite-r-and-r-packages/

*Face prep: The right place to prepare for placements*. FACE Prep | The right place to prepare for placements. (2023, April 21). https://www.faceprep.in/web-development/what-is-kotlin/

HP. (2018, October 15). *Computer history: A timeline of computer programming languages*. HP® Tech Takes. https://www.hp.com/us-en/shop/tech-takes/computer-history-programming-languages