An Evaluation Paper on Kotlin, R, Go, & Ruby

submitted
in partial fulfillment of the requirements
for the course CSADPRG

Estrella, Katrina Isabel S.
Homssi, Yazan M.
Loria, Andrea Euceli F.
Stinson, Audrey Lauren G.

Mr. Romualdo M. Bautista Jr.
April 6, 2024

I. Introduction

A. Kotlin

Kotlin combines and supports all the major programming paradigms (functional, imperative, object-oriented, and procedural). The programming language was developed by JetBrains in 2010 and was initially released in July 2011 under the name Project Kotlin. The current version of Kotlin is version 1.9.23 which was released on March 7, 2024. This version merely released a bug fix for Kotlin 1.9.20, 1.9.21, and 1.9.22. New features were added in the 1.9.20 version released last November 1, 2023. These features included the following:

- Kotlin Multiplatform is now stable
- New default hierarchy template for setting up multiplatform projects
- Full support for the Gradle configuration cache in Kotlin Multiplatform
- Custom memory allocator enabled by default in Kotlin/Native
- Performance improvements for the garbage collector in Kotlin/Native
- New and renamed targets in Kotlin/Wasm
- Support for the WASI API in the standard library for Kotlin/Wasm

As for the future of Kotlin, according to Roman Elizarov who was the Project Lead for Kotlin, they are planning to:

- "Cultivate an ecosystem consisting of multiplatform, Kotlin-first libraries that work on multiple platforms with a wide range of abstractions and utilities for developers."
- "Start by defining things like collection literals and data in the source code easily, and then proceed to deconstruct this data later on."
- "Make it possible to write safer code that does not suffer from problems like sharing mutable data to other threads."
- "Expand the power of Kotlin inline functions with constant evaluation and propagation, as well as enable compiler plugins for advanced compile-time manipulation of the code."
- "Lay the groundwork for further evolution of the language without sacrificing performance."
- "Make it easier for tools to help you with your code."

As for which domain application Kotlin is most suited for, it is best suited for Android development, back-end development, or data science.

B. Ruby

Ruby is a pure object-oriented language where (mostly) everything in Ruby is an object. However, it has features which can support procedural, functional, and other programming paradigms. It was created in February 1993 by Yukihiro Matsumoto. Matsumoto, also known as Matz, described Ruby as “optimized for programmer happiness.” It is designed to be productive and fun rather than performant.

The latest version of Ruby is Ruby 3.3, released last December 25, 2023 (as tradition). Major features in this version include Ractors, an experimental feature introduced in Ruby 3.0 which allows for a thread-safe parallel execution of code and MJIT (Method Based Just-in-Time Compiler) which was added in Ruby 2.6 to improve performance by compiling Ruby code to YARV (Yet Another Ruby VM) instructions which are run by the Ruby Virtual Machine. Future plans for Ruby's development are mainly focused on performance optimization improvements. Although Ruby is flexible, it is most often used for web application development.

C. Go

In September 2007, discussions began about creating a new programming language to tackle engineering challenges encountered by Google software engineers. Robert Griesemer, Ken Thompson, and Rob Pike spearheaded the development of Go, which was publicly released in November 2009 (Pike, 2020). "Their aim was to craft a language with the speed of C++, the simplicity of Python, and the scalability of Java" (Singh, 2023). "Go is an open-source programming language that makes it simple to build secure, scalable systems" (Go, n.d.). Over the years, it has garnered attention and widespread adoption due to its ease of use, efficiency, and scalability, effectively meeting the goals set by its creators.

Recent updates, such as the release of Go 1.21.0 on August 8, 2023 introduced new features like the "clear" function, which efficiently removes all elements in a map, sets all elements to 0 in a slice, and operates depending on whether the type parameter is a set of maps or slices. Additionally, the "log/slog" package enhances structured logging for better processing and organization, gaining support from popular log analysis tools and services. On the side of the developers, Go's developer community has grown significantly. According to the 2020 Developer Ecosystem Survey there are around 1.1 million professional developers around the world using it as their primary language. This makes Go rank among the top 10 primary languages of professional developers at 7%. Furthermore, industries such as IT services, finance, fintech, and cloud computing/platforms are among the top users of Go (Zharova, 2021). Its versatility allows for various applications, including cloud and network services, command-line interfaces, web development, and site reliability engineering (Go, n.d.).

Companies besides Google, such as PayPal, Meta, Dropbox, Netflix, Riot Games, Twitch, and Cloudflare, have adopted Go for their projects (Go, n.d.). With consistent updates and release notes, Go continues to evolve with their most recent release, Go 1.22, enhancing developer productivity, code maintainability, and overall developer experience. With the rising prominence of cloud-native development, microservices, and the emergence of Artificial Intelligence (AI), Go is on track to remain a leading programming language in the foreseeable future (Geeksforgeeks, 2024).

D. R

R is a programming language initially developed by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, in the early 1990s. The project was conceived in 1992, and an initial version was released at around 1995, and the beta was released in 2000.

It's a language made for the purpose of statistical analysis, data and information visualization. According to the wikipedia page, R was actually built upon the foundations led by a language called 'S', developed by John Chambers in the 1970s. S was also designed for data analysis and statistical modeling, and R inherited many of its concepts and features (Wikipedia contributors, 2024b).

R actually supports a mixture of functional and object-oriented programming paradigms. When it comes to functional programming, it has first class functions, side-effect-free functions, and supports lazy evaluation of arguments– meaning it postpones the computation of an expression until its value is explicitly needed, it is quite the fascinating concept. As for the object-oriented side of R, it has 3 built-in object-oriented paradigms:

- S3 is a class that allows the user to overload functions by splitting them into methods, this makes S3 simple and flexible.
- S4 is a class that unlike S3– provides a more structured approach to OOP, it is best to use S4 as it is suitable for managing large systems like GUI apps or Web apps.
- Lastly, we have R5– which is officially called a "Reference Class" that when introduced in R version 2.12, addresses the need for mutable objects. It is best to use R5 in scenarios like GUI development.

When it comes to data science, R is one of the most popular languages used, ranked 7th as of April 2024, though it's natural that it would lag behind Python– maintaining its 1st position.

The future of R is quite promising. In the R manual, there are talks of "Long Vectors" though effectively impossible to implement under a 32-bit platform, this will only apply to 64-bit platforms. Another future implementation is "64-bit types" , the devs wanted the capability of being able to store larger integers in R, although the possibility of storing these as a `double` is overlooked, now the proposal is to add a new type to R such as `longint`.

Lastly we have "Large Matrices", matrices are stored as vectors but limited to 2^31-1 elements, longer vectors are allowed on 64-bit platforms, though the problem is that linear algebra is done by Fortran code compiled with 32-bit integer, it seems that the compilers currently used with R on a 64-bit platform allow matrices, where each dimension is less than 2^31 but with 2^31 or more elements and index them accordingly.

Although unlike languages such as Python, Java, and C#, R is not considered a general-purpose programming language. It is instead considered a

domain-specific language, which means its use is designed for specific uses, which in this case is statistical computing and analysis (Worsley, 2023). CRAN, which stands for "Comprehensive R Archive Network", is a network of servers around the world that store and distribute R packages and documentations that can help users, developers, and researchers with a multitude of tasks, where solutions and codes may already be available on the CRAN library. According to CRAN, there are nearly 20,000 packages available for public use (Worsley, 2023).

II. Language Comparison

A. Kotlin

1. Data Type Binding, Type Incompatibility Handling

Kotlin is a statistically typed language because a variable's data type is determined during compile time and remains the same throughout the execution. Regarding type binding, Kotlin makes use of both explicit declarations and implicit declarations for its variables. When assigning a variable, the data type is implicitly defined by Kotlin for the programmer. However, it is also possible for the programmer to explicitly define the data type of the variable by doing the following:

```
var x: Int = 4
var y: Float = -5.12F
```

There are four ways that Kotlin handles type incompatibility.

1. The first way is by explicit casting but this is deemed unsafe.
2. The second way is by using a safe casting operator (as?) and this is considered safe.
3. The third way is by using the built-in conversion functions that Kotlin has to offer.
4. The fourth way is for the programmer to create their own custom type conversion functions.

2. Binding Time

Whether the variable declaration is done explicitly or implicitly, the possible types for it is already known at the language definition time when the variable is assigned a value in the midst of writing the code. Once the data type is identified, the set of possible values for a variable will be known during language definition time because the IDE will identify an error if the value is incompatible with the variable type. Once the data type is identified, the set of possible values for a variable will be known during language definition time because the IDE will identify an error if the value is incompatible with the variable type. Finally, the binding time of the value of a variable will occur during compilation/translation time because an error will occur. For example, if x and y were not assigned any value, the values of x and y after an equal sign will result in an error: "Unresolved reference".

So, generally speaking, a variable can either be explicitly stated or implicitly stated by Kotlin during pre-runtime once a value has been assigned to the variable. The data type is already bound to its variable during pre-runtime if the variable was explicitly described. However, the data type is bound to a variable during runtime if the variable was implicitly described. Additionally, Kotlin has provided multiple ways for the data type to be changed, all during pre-runtime. Furthermore, during runtime, memory is allocated when a variable is created during runtime and is de-allocated when it goes out of scope, also during runtime.

3. Parameter Passing

By default, Kotlin functions make use of pass by value for their arguments. This means that if the value of the function's parameter is changed, its value outside the function is not affected. In the example below, it is seen that nRandom is not affected by the different inputs. In the myFunction, 30 will be the output, but in the different calls to the function with different inputs in main, the output is not 30 and are their expected outputs as seen below.

```kotlin
fun myFunction(nRandom: Int)
{
    nRandom = 20
    println(nRandom + 10)   // Outputs 30
}

fun main()
{
    myFunction(10)          // Outputs 20
    myFunction(5)           // Outputs 15
    myFunction(12)          // Outputs 22
}
```

However, if an object or non-primitive type arguments are passed, the reference is what the function copies. In the example below, the object is initialized to be equal to 0 but once the object is called after the function containing changes is called, the value of the object is changed. This is because of the values' shared reference.

```kotlin
data class SomeObj(var x: Int = 0)

fun modifyObject(someObj: SomeObj) {
    someObj.x = 3
}

fun main()
{
    val obj = SomeObj()

    println(obj.x)      // Outputs 0

    modifyObject(obj)

    println(obj.x)      // Outputs 3
}
```

B. Ruby

1. Data Type Binding, Type Incompatibility Handling

The following code swaps the values of a float and a String variable. As seen in the code, the values of different types of variables can be easily swapped by using an extra temp variable.

```
x= -234.654321190
y= "dollar interest"
temp = x
x = y
y = temp
print "x = ", x, " y = ", y
```

Ruby is dynamically typed therefore the type of a variable is determined when its value is assigned or changed during runtime. Regarding type handling, Ruby uses implicit type conversion in most operators and methods, for data types that are closely related to each other. For example in the expression **`x = 1.2 + 1`** which results in 2.2 as the value of x, 1 is implicitly converted to a float. Ruby also has methods for implicit type conversion such as to_str and to_ary. Meanwhile, Ruby supports explicit type conversion mainly for data types that are more different to each other using methods like to_f (to float) and to_s (to String). For example in **`x = "hello " + 1.0.to_s`** which results in "hello 1.0" as the value of x, 1.0 is explicitly converted to a String.

2. Binding Time

Since Ruby is dynamically typed, the data type of a variable x is only determined when x is assigned a value at runtime. A variable is dynamically bound to a data type after it is assigned a value. However, it can be bound to a different data type depending on its assigned value. Upon the value assignment, the variable is also initialized and its memory space is allocated. When the variable is no longer needed, it is automatically de-allocated by Ruby's garbage collector.

3. Parameter Passing

Parameter passing in Ruby fits neither the typical pass-by-value or pass-by-reference definitions. Technically, Ruby is pass-by-value because it passes copies of the actual parameter. However, these copies are actually copies of references to objects. The actual parameter is a reference to an object while the formal parameter contains a copy of a reference to the same object. A reference is passed but it is only a copy, therefore this can be interpreted as either pass-by-value or pass-by-reference. The terms "Pass-by-assignment", "Pass-by-reference-value", and

"Pass-by-object-reference" can also be used to describe Ruby's parameter passing.

In general, variables do not contain the objects themselves. They contain references to the objects.

Input:

```ruby
def bye(word)
    word # (This line can be omitted)
    # A variable named word is created
    word = "Bye"
    # Bye is assigned to this new variable
end

word = "Hello"
bye(word)
puts "After bye(word), word = " + word
```

Output:

```
After bye(word), word = Hello
```

After bye(word) is called, word still displays "Hello". This is because the function bye receives only a copy of a word. word is a reference to the "Hello" object so in other words, bye receives a copy of a reference to the "Hello" object. When a new word variable is created, word in bye is reassigned a new reference to a new object. It no longer points to the original object.

C. Go

1. Data Type Binding & Type Incompatibility Handling

Go is a statically typed language, wherein the variable type is determined before runtime or during compile time and remains consistent throughout execution. It allows for both explicit and implicit variable declarations (Golangbot, 2024).

In handling type incompatibility, Go employs methods like typecasting between float64 and int types. However, while functions like strconv.Atoi and strconv.ParseFloat can successfully convert numeric types to strings, type conversion may not be universally applicable to all types (Nilsson, n.d. & Geeksforgeeks, 2020). For instance, converting non-numeric values like strings to other types or vice versa can encounter issues, such as attempting to convert a word like "hello" to a numeric type. Therefore, careful error checking is necessary to handle such cases, as data

types have distinct behaviors that can affect the success of type conversion.

```go
func main() {
    str := "hello"
    float, err := strconv.ParseFloat(str, 64)
    if err != nil {
        fmt.Printf("Error: %v\n", err)
        fmt.Printf("String: %s\nFloat: %f\n", str, float)
        return
    }
    fmt.Printf("String: %s\nFloat: %f\n", str, float)
}
```

Results:

```
Error: strconv.ParseFloat: parsing "hello": invalid syntax
String: hello
Float: 0.000000
```

```go
func main() {
    str := "123"

    float, err := strconv.ParseFloat(str, 64)
    if err != nil {
        fmt.Printf("Error: %v\n", err)
    } else {
        fmt.Printf("String: %s\nFloat: %f\n", str, float)
    }

    integer, err := strconv.Atoi(str)
    if err != nil {
        fmt.Printf("Error: %v\n", err)
    } else {
        fmt.Printf("String: %s\nInteger: %d\n", str, integer)
    }
}
```

Results:

```
String: 123
Float: 123.000000
String: 123
Integer: 123
```

2. Binding Time

Go, being a statically typed language, the possible types for a variable `x` are determined during compilation or translation time, depending on the provided values or the explicitly declared data type. Whether explicitly or implicitly declared, the data type of a variable remains constant throughout execution. During compilation or translation time, a data type becomes bound to a variable, coinciding with the moment when statements are checked in Go's statically typed environment (Gopher Guides, 2019). Memory allocation occurs upon variable declaration, with memory space being deallocated once a function finishes executing or when it exceeds the function's scope (Can, 2022). Variables are initialized upon declaration, with the zero value of the variable's type assigned if no value is explicitly provided.

3. Parameter Passing

Go supports pass by value as its primary parameter passing method for all primitive data types. When a parameter is passed, it is copied into another location in the memory. Therefore, when modifying a variable only its copy is modified, not its original self.

```go
func add4(num int) int {
    return num + 4
}

func main() {
    ogNum := 10

    fmt.Printf("ogNum: %d\n", ogNum)

    add4(ogNum) //10 + 4

    fmt.Printf("add4(ogNum) result: %d\n", add4(ogNum))
    fmt.Printf("ogNum after calling add4(ogNum): %d\n", ogNum)
}
```

Results:

```
ogNum: 10
add4(ogNum) result: 14
ogNum after calling add4(ogNum): 10
```

On the other hand, pass by reference occurs when using pointers. In the code above, by using &ogNum, ogNum's address is passed to the add4 function. Inside the function *num was used to access the value at the given address, therefore, modifying the value of ogNum from 10 to 14 (Hassan, 2020).

```go
func add4(num *int) {
    *num = *num + 4
}

func main() {
    ogNum := 10

    fmt.Printf("ogNum: %d\n", ogNum)

    add4(&ogNum)

    fmt.Printf("ogNum after calling add4(&ogNum): %d\n", ogNum)
}
```

Results:

```
ogNum: 10
ogNum after calling add4(&ogNum): 14
```

D. R

1. Data Type Binding, Type Incompatibility Handling

In R, data type binding just refers to the process of combining two different data structures. Often done using functions like `rbind()` and `cbind`, which combine the data frames and/or matrices by rows and columns, respectively. It is useful when merging datasets with either similar structures, or simply when you want to add brand new data to an already existing dataset. Let's take an example at both functions:

`*rbind()*`:

```r
# CSADPRG MCO2 // R // Data Type Binding

# Using `rbind`
# 2 sample vectors
a <- c(1, 2, 3, 4, 5)
b <- c(6, 8, 7, 0, 9)

# rbind the 2 vectors into a matrix
new_matrix <- rbind(a, b)

# View matrix
new_matrix
```

```
> # Using `rbind`
> # 2 sample vectors
> a <- c(1, 2, 3, 4, 5)
> b <- c(6, 8, 7, 0, 9)
>
> # rbind the 2 vectors into a matrix
> new_matrix <- rbind(a, b)
>
> # View matrix
> new_matrix
  [,1] [,2] [,3] [,4] [,5]
a    1    2    3    4    5
b    6    8    7    0    9
>
```

`*cbind()*`:

```r
# CSADPRG MCO2 // R // Data Type Binding

# Using `cbind`
# 2 sample data frames
df_1 <- data.frame(a=c(1, 2, 3, 4, 5),
                   b=c(6, 7, 8, 9, 10),
                   c=c(11, 12, 13, 14, 15))
df_2 <- data.frame(d=c(12, 14, 16, 18, 22),
                   e=c(66, 35, 78, 36, 40))

# cbind the two data frames
new_df <- cbind(df_1, df_2)

# View new data frame
new_df
```

```
> df_2 <- data.frame(d=c(12, 14, 16, 18, 22),
+                    e=c(66, 35, 78, 36, 40))
>
> # cbind the two data frames
> new_df <- cbind(df_1, df_2)
>
> # View new data frame
> new_df
  a  b  c  d  e
1 1  6 11 12 66
2 2  7 12 14 35
3 3  8 13 16 78
4 4  9 14 18 36
5 5 10 15 22 40
>
```

R handles type incompatibility through coercion (GfG, 2022), where R would automatically convert a data type to another in order to complete an operation. With that said, if we try to add a numeric value to a character vector, R will automatically convert the numeric value to a character– and then perform concatenation. An example is shown below:

```r
# CSADPRG MCO2 // R // Type Incompatibility Handling

# Using the coercion method
# Numeric vector
num_vector <- c(11, 22, 33, 44, 55)

# Character vector
char_vector <- c("a", "d", "p", "r", "g")

# Combine the vectors
combined_vector <- c(num_vector, char_vector)

# Print the combined vector
print(combined_vector)
```

```
> # Using the coercion method
> # Numeric vector
> num_vector <- c(11, 22, 33, 44, 55)
>
> # Character vector
> char_vector <- c("a", "d", "p", "r", "g")
>
> # Combine the vectors
> combined_vector <- c(num_vector, char_vector)
>
> # Print the combined vector
> print(combined_vector)
 [1] "11" "22" "33" "44" "55" "a"  "d"  "p"  "r"  "g"
>
```

In the example above, R will coerce the numeric values in `*num_vector*` to character values in order to match the data type of `*char_vector*`. The output (right image) will be a character vector, yes all of them.

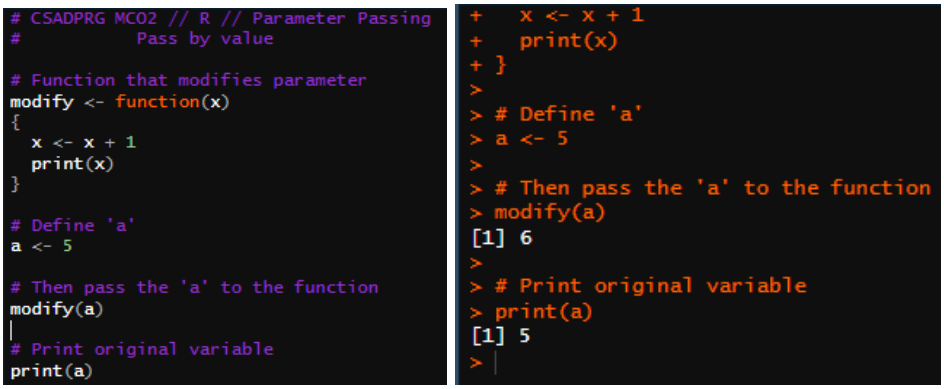
2. Binding Time

The R programming language's typing discipline is dynamic. With languages like R, variable types are determined during runtime, not at compile time. This means you don't have to explicitly declare the datatype of a variable before you can use it. R is also known to be interpreted just

like Python, as it executes code line-by-line by an interpreter during runtime, rather than being compiled into machine code beforehand like C.

3. Parameter Passing

R's parameter passing mechanism is known as "pass-by-value". Example shown below:



```
# CSADPRG MCO2 // R // Parameter Passing
#           Pass by value

# Function that modifies parameter
modify <- function(x)
{
  x <- x + 1
  print(x)
}

# Define 'a'
a <- 5

# Then pass the 'a' to the function
modify(a)

# Print original variable
print(a)
```

```
+   x <- x + 1
+   print(x)
+ }
>
> # Define 'a'
> a <- 5
>
> # Then pass the 'a' to the function
> modify(a)
[1] 6
>
> # Print original variable
> print(a)
[1] 5
>
```

In this example, the function `*modify*` adds 1 to its parameter x. However, when we print the variable 'a' after calling the function, we see that 'a' is still 5. This is because 'a' was *passed by value* to the function, so only a copy of a was modified, not a itself.

III. Developing the Weekly Payroll System Using Ruby (by Stinson)

The application is created using the Ruby language. There are two files used for the payroll system, payroll.rb and helper.rb. The system was created using object-oriented features such as classes and methods, different types of variables (global, class, instance, local), and other useful constructs.

A. Payroll File and Helper Module

The payroll.rb file is the main Ruby file which contains payroll-specific objects. The helper.rb file contains a Helper module which contains methods and variables that are not specific to the payroll, but are useful in different parts of the payroll program, for example, in input-checking and string-formatting.

B. Program Flow

The program starts at the menu where a number of options are available for the user to choose from. The user can change the default settings for the week before starting the week itself. After the week has started, the user can choose to change each day's default settings and input the OUT time. Then after the week has ended, information about the week is displayed, including the total week salary. To exit the program, the player can press [E] to exit the program anytime, or [4] to exit at the menu.

C. DayInfo class

Information about each day is stored in a class DayInfo. It contains class variables which store information that must remain the same for all days in the week, such as hour rate and maximum regular hours. It also contains instance variables which contain unique information per day of the week. This includes IN time, OUT time, day type, and so on.

```
class DayInfo
  attr_accessor :in_time, :out_time, :day_type, :hrs_ns, :hrs_ot, :hrs_nsot, :day_salary

  @@day_rate = 500.00
  @@reg_hrs = 8
  @@hr_rate = @@day_rate / @@reg_hrs

  def self.day_rate
    @@day_rate
  end

  def self.day_rate=(day_rate)
    @@day_rate = day_rate
  end

  def self.reg_hrs
    @@reg_hrs
  end
```

D. Calculation

In the method get_week_info, an array contains 7 instances of DayInfo to represent a week. For each day, the user answers a sequence of prompts to calculate the day's salary. Since the variables under DayInfo are initialized upon its creation, the user can just opt to enter OUT times for each day, with no change to any default settings. The day salary is added by the day rate, total overtime pay, and total night shift hour rate. Each day's salary is added to calculate the weekly salary.

```
def get_week_info
  days = []
  weekTotal = 0

  puts 'The week will start now.'

  for i in 1..7
    puts "\nDay #{i}"
    days[i] = DayInfo.new
    if i.between?(6, 7)
      puts 'Today is a Rest day.'
      days[i].day_type = 'Rest'
    end
    get_day_info(days[i])
    weekTotal += days[i].day_salary
  end

  puts "\nThe week is over."
  puts 'Displaying Week Information:'
  DayInfo.display_week_info
```

IV. Developing the Weekly Payroll System Using Other Programming Languages

A. Kotlin (by Estrella)

When using Kotlin in my IDE, I had to constantly create new files for every edit created so progress was slow. What it would run and compile would be what was initially compiled and ran (aka a test run which outputs “Hello World”). Thus, for every edit made in order to resolve an error or improve an aspect of the code, a new file had to be created which was not efficient. Thus, Kotlin was not our choice of programming language to use.

B. Go (by Loria)

Before creating the payroll program, I was confident that Go would be a great option for the project given its simplicity and my familiarity with C and Java (which are Go’s predecessors and inspiration). The process of developing the weekly payroll program was straightforward. I created functions addressing the various processes required, and some of them are shown below

‘getDaySalary’:

```go
func getDaySalary(inTime string, outTime string, maxHours int, dailySalary float64, dayType int) float64
    dailyRate := float64(dailySalary)
    var salary float64
    nightShiftHours := 0.0
    overtimeRate := 0.0
    nsOvertimeRate := 0.0

    inTimeN, inErr := strconv.Atoi(inTime)
    outTimeN, outErr := strconv.Atoi(outTime)
    _ = inErr
    _ = outErr

    isNightShift := (outTimeN >= 2200 || outTimeN < 600) || (inTimeN >= 2200 || inTimeN < 600)

    //check absence
    if inTimeN == outTimeN {
        if dayType == 0 {
            return 0.0 // Absent on a normal day
        } else {
            return 500.0 // Absent on a rest/holiday
        }
    }

    // check daytype, additional pay, overtime rates
    switch dayType {
    case 0:
        overtimeRate = OT
        if isNightShift {
            nsOvertimeRate = NSOT
        }
        break
    case 1:
        dailyRate *= addPayRD
        overtimeRate = RD_OT
        if isNightShift {
            nsOvertimeRate = RD_NSOT
        }
        break
```

The getDaySalary function starts with converting the time in data type string to integer then proceeds to check the day types. The remaining conditions in the switch case were for identifying the night shift rate and additional pay for the respective day types.

```
inHours := inTimeN / 100
outHours := outTimeN / 100
totalHours := float64(outHours - inHours)

regularHours := float64(maxHours) + 1 // Include 1 hour for lunch break

if totalHours <= regularHours { //No overtime
    salary = dailyRate

    if isNightShift {
        salary += (nightShiftHours * (dailyRate / float64(maxHours))) * NSD
    }
} else { //May overtime
    salary = dailyRate

    overtimeHours := totalHours - regularHours
    overtimeHours -= nightShiftHours
    fmt.Println("Overtime hours = ")
    fmt.Println(overtimeHours)
    fmt.Println("Night shift hours =")
    fmt.Println(nightShiftHours)
    fmt.Println("dailyRate = ")
    fmt.Println(salary)

    salary += (overtimeHours * (float64(dailySalary) / float64(maxHours))) * overtimeRate
    fmt.Println(salary)
    if isNightShift {
        salary += (nightShiftHours * (float64(dailySalary) / float64(maxHours))) * nsOvertimeRate
        fmt.Println(salary)
    }
}
```

Once the rates for overtime and any additional pay are established, the program proceeds to collect the total hours worked and categorizes them into regular hours, night shift hours, and overtime hours. Then, it computes the daily salary based on this information.

'askOutTime':

```
func askOutTime(daysOfWeek [7]string) [7]string {
    var outTimes [7]string

    for i := 0; i < len(daysOfWeek); i++ {
        fmt.Printf("Enter OUT time for %s: ", daysOfWeek[i])
        fmt.Scanln(&outTimes[i])
    }
    return outTimes
}
```

'changeDailyRate':

```go
func changeDailyRate(dailySalary *float64) {
    var newSalary float64
    for {
        fmt.Print("\033[H\033[2J")
        fmt.Printf("Daily salary: %.2f\n", *dailySalary)
        fmt.Printf("Enter NEW daily salary: ")
        fmt.Scanln(&newSalary)

        if newSalary > 0 {
            *dailySalary = newSalary
            break
        } else {
            showErrorMsg()
        }
    }

    fmt.Println()
    fmt.Printf("UPDATED daily salary: %.2f", *dailySalary)
    fmt.Scanln()
}
```

'Main Function case 1: generate payroll':

```go
switch menuChoice {
case 1:
    for {
        fmt.Print("\033[H\033[2J")
        fmt.Println("GENERATE PAYROLL")

        outTimes = askOutTime(daysOfWeek)

        for i := 0; i < len(outTimes); i++ {
            weekSalary[i] = getDaySalary(inTimes[i], outTimes[i], maxHours, dailySalary, dayTypes[i])
            totalSalary += weekSalary[i]
        }

        for {
            fmt.Print("\033[H\033[2J")
            showPayroll(daysOfWeek, weekSalary)
            fmt.Println()
            fmt.Printf("Total Salary = %.2f\n", totalSalary)
            fmt.Println("\n[1] Generate another payroll")
            fmt.Println("[2] <- GO BACK")
            fmt.Scanln(&payrollChoice)

            if payrollChoice == 1 {
                break
            } else if payrollChoice == 2 {
                break
            } else {
                showErrorMsg()
            }
        }

        if payrollChoice == 2 {
            break
        }

    }
```

'Main Function case 2: change settings':

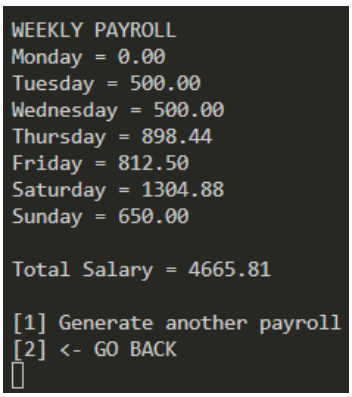
```go
case 2:
    for {
        fmt.Print("\033[H\033[2J")
        fmt.Println("SETTINGS")
        showSettings(dailySalary, maxHours, inTimes, dayTypes, daysOfWeek)

        fmt.Println("Would you like to change any settings?")
        fmt.Println("[1] Change daily rate")
        fmt.Println("[2] Change maximum regular hours")
        fmt.Println("[3] Change default IN time")
        fmt.Println("[4] Change day type/s")
        fmt.Println("[5] <- Go Back")

        fmt.Scanln(&settingsChoice)

        if settingsChoice == 1 {
            changeDailyRate(&dailySalary)
        } else if settingsChoice == 2 {
            changeMaxHours(&maxHours)
        } else if settingsChoice == 3 {
            changeInTime(&inTimes, daysOfWeek)
        } else if settingsChoice == 4 {
            changeDayType(&dayTypes, daysOfWeek)
        } else if settingsChoice == 5 {
            break
        }
    }
```

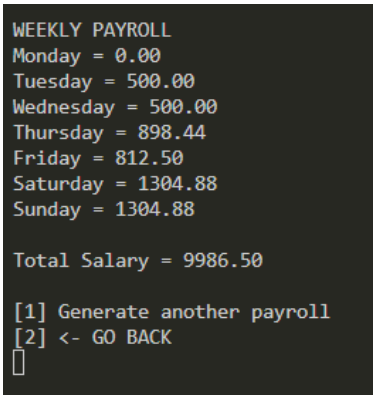
'Main Menu & Settings':

```
MENU
[1] Generate Payroll
[2] Settings
[3] Exit
Select an option
```

```
SETTINGS
Daily Rate: 500.00

Maximum regular hours of work per day: 8

Default IN time for the week:
Monday: 0900
Tuesday: 0900
Wednesday: 0900
Thursday: 0900
Friday: 0900
Saturday: 0900
Sunday: 0900

Default day types:
Monday: Normal day
Tuesday: Normal day
Wednesday: Normal day
Thursday: Normal day
Friday: Normal day
Saturday: Rest day
Sunday: Rest day

Would you like to change any settings?
[1] Change daily rate
[2] Change maximum regular hours
[3] Change default IN time
[4] Change day type/s
[5] <- Go Back
```

'Sample Output':





The outTimes input are: 0900, 1800, 1800, 2300, 2200, 0000, and 1800 for each day of the week. In previous runs, when "outTime" was set to 0000, it correctly yielded a daily salary of 500 for non-working days. However, I am unsure about why it's generating 1304.88 and where that came from. Additionally, there seems to be a problem where the total salary includes the previous week's salary when generating a new one, despite the total salary being calculated from the weeklyPayroll array.

The errors likely stemming from the structure of my code and the functions I implemented led us to not select Go as the language for our MCO1 project. However, I believe that someone proficient in Go could create a more streamlined and effective payroll program.

C. R (by Homssi)

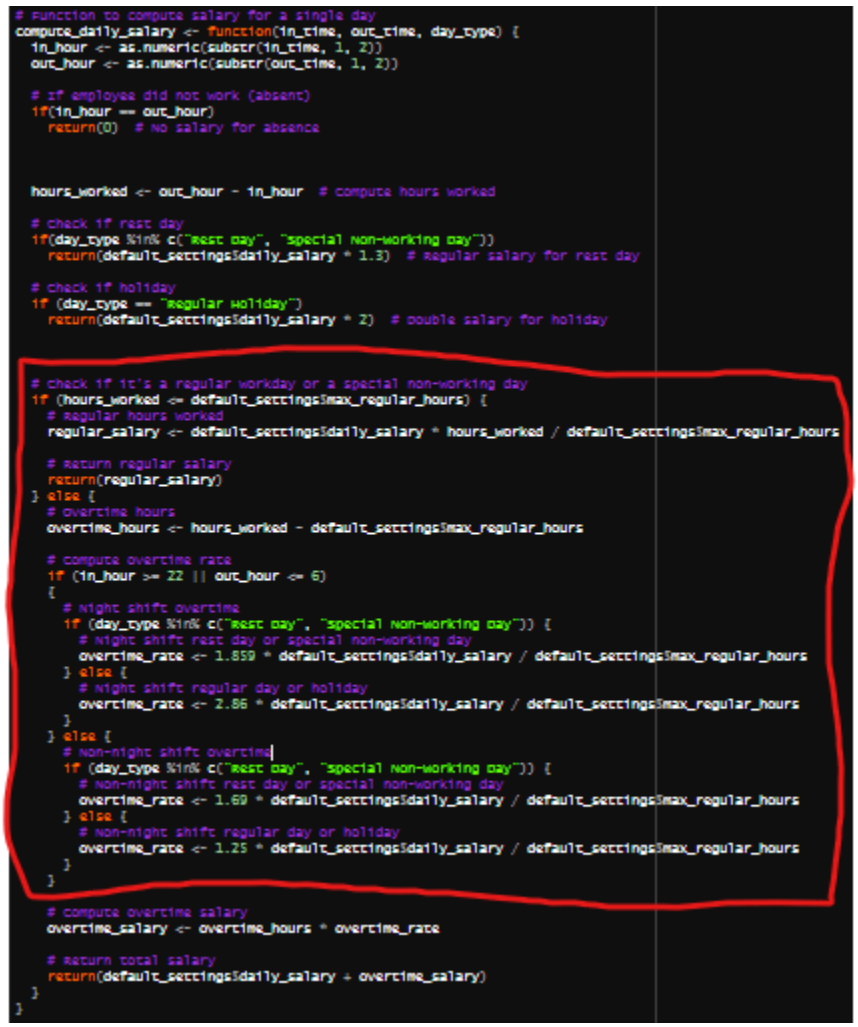
Before sharing some snippets of my implementation of the payroll system in R, I would like to state first that while yes, R is great for statistical analysis and data manipulation, it is unfortunately not the best choice for a general-purpose programming project, which in this case is our MCO "payroll system". Its syntax and ecosystem are more tailored for statistical computing rather than application development. Then again, I say that but someone might've implemented their MCO while using R, that just makes me quite mediocre in R.

`*default_settings*`:

```r
# CSADPRG MCO1 // R // Payroll System (Bad)

# Default configuration settings
default_settings <- list(
  daily_salary = 500,
  max_regular_hours = 8,
  workdays_per_week = 5,
  restdays_per_week = 2,
  in_time = "0900",
  out_time = "0900"
)
```

Program starts by defining a list of default settings according to the specs, which includes the daily salary of a measly 500, maximum regular hours, workdays and rest days per week, and the standard in/out time.

`*compute_daily_salary*`:



```r
# Function to compute salary for a single day
compute_daily_salary <- function(in_time, out_time, day_type) {
  in_hour <- as.numeric(substr(in_time, 1, 2))
  out_hour <- as.numeric(substr(out_time, 1, 2))

  # If employee did not work (absent)
  if(in_hour == out_hour)
    return(0)  # No salary for absence



  hours_worked <- out_hour - in_hour  # Compute hours worked

  # Check if rest day
  if(day_type %in% c("rest day", "special non-working day"))
    return(default_settings$daily_salary * 1.3)  # Regular salary for rest day

  # Check if holiday
  if (day_type == "regular holiday")
    return(default_settings$daily_salary * 2)  # Double salary for holiday


  # Check if it's a regular workday or a special non-working day
  if (hours_worked <= default_settings$max_regular_hours) {
    # Regular hours worked
    regular_salary <- default_settings$daily_salary * hours_worked / default_settings$max_regular_hours

    # Return regular salary
    return(regular_salary)
  } else {
    # Overtime hours
    overtime_hours <- hours_worked - default_settings$max_regular_hours

    # Compute overtime rate
    if (in_hour >= 22 || out_hour <= 6)
    {
      # Night shift overtime
      if (day_type %in% c("rest day", "special non-working day")) {
        # Night shift rest day or special non-working day
        overtime_rate <- 1.859 * default_settings$daily_salary / default_settings$max_regular_hours
      } else {
        # Night shift regular day or holiday
        overtime_rate <- 2.86 * default_settings$daily_salary / default_settings$max_regular_hours
      }
    } else {
      # Non-night shift overtime
      if (day_type %in% c("rest day", "special non-working day")) {
        # Non-night shift rest day or special non-working day
        overtime_rate <- 1.69 * default_settings$daily_salary / default_settings$max_regular_hours
      } else {
        # Non-night shift regular day or holiday
        overtime_rate <- 1.25 * default_settings$daily_salary / default_settings$max_regular_hours
      }
    }

    # Compute overtime salary
    overtime_salary <- overtime_hours * overtime_rate

    # Return total salary
    return(default_settings$daily_salary + overtime_salary)
  }
}
```

This function is the beating heart of my program. It is essential for determining how much an employee should be paid for a single workday, considering factors such as work hours, day type, and any overtime worked based on the configuration settings listed above. (Code in Red Box indicates the potential location of where the error is arising).

`*generate_weekly_payroll*`:

```r
# Function to generate payroll for the week
generate_weekly_payroll <- function() {

  daily_salaries <- list() # Placeholder for daily salaries
  total_weekly_salary <- 0 # Placeholder for total weekly salary

  # Loop through each day of the week ( day 1 to day 7 )
  for (day in 1:7) {
    out_time <- readline(prompt = paste("Enter OUT time for Day", day, ": "))

    # Placeholder for day type (normal, rest day, holiday)
    day_type <- ifelse(day > default_settings$workdays_per_week, "Rest Day", "Normal Day")

    # Compute salary for the day
    daily_salary <- compute_daily_salary(default_settings$in_time, out_time, day_type)

    # Store daily salary
    daily_salaries[[paste("Day", day)]] <- daily_salary

    # Add daily salary to total weekly salary
    total_weekly_salary <- total_weekly_salary + daily_salary
  }

  # Output daily salaries and total weekly salary
  cat("Daily Salaries:\n")
  print(daily_salaries)
  cat("Total Weekly Salary:", total_weekly_salary, "\n")
}
```

This function calculates the total weekly salary. It will loop through each day of the week, allowing the user to input the out time for each day, determine the type of day by itself, computes the daily salary, and adds it to the total weekly salary. It then outputs the daily salaries and the total weekly salary while the user is running the program.

`*main_menu*`:

```r
# Main menu function
main_menu <- function() {
  while(TRUE)
    {
    cat("\nPayroll System Menu:\n")
    cat("1. Generate Weekly Payroll\n")
    cat("2. Modify Default Configuration\n")
    cat("3. Exit\n")

    choice <- as.numeric(readline(prompt = "Enter your choice: "))

    if(choice == 1)
      {
      generate_weekly_payroll()
    }
    else if(choice == 2)
      {
      cat("no\n")
    }
    else if(choice == 3)
      {
      cat("Exiting the system. Goodbye!\n")
      break
    }
    else
      {
      cat("Invalid choice. Please enter a valid option.\n")
    }
  }
}
settings <- default_settings
main_menu()
```

Simple user interface for the payroll system. It presents the user with three options: generate the weekly payroll, modify the default configuration (I could not implement in time sorry), or exit the system. The user's choice determines which action is performed, which in this case, the only action relevant is choice 1.

Output of running code:

```
> # Run the main menu
> main_menu()

Payroll System Menu:
1. Generate Weekly Payroll
2. Modify Default Configuration
3. Exit
Enter your choice: 1
Enter OUT time for Day 1 : 0900
Enter OUT time for Day 2 : 1900
Enter OUT time for Day 3 : 1800
Enter OUT time for Day 4 : 2100
Enter OUT time for Day 5 : 0900
Enter OUT time for Day 6 : 1000
Enter OUT time for Day 7 : 1100
Daily Salaries:
$`Day 1`
[1] 0

$`Day 2`
[1] 656.25

$`Day 3`
[1] 578.125

$`Day 4`
[1] 812.5

$`Day 5`
[1] 0

$`Day 6`
[1] 650

$`Day 7`
[1] 650

Total Weekly Salary: 3346.875
```

Take note that the computation is wrong, due to a logical error in my second function: `*compute_daily_salary*`.

V. Conclusion

Our evaluation of Kotlin, Ruby, Go, and R highlights the diversity among these programming languages. Each language was designed with specific purposes and objectives in mind, from wanting to combine existing languages' features into one to optimizing code for developers' happiness. These purposes offer unique solutions to problems developers face. Moreover, the evaluation displayed the difference in data type bindings, type incompatibility handling, and parameter passing methods across the four languages. These disparities and characteristics demonstrate how these languages could be adopted to projects across the various sectors of the tech industry.

Ruby was chosen as the language of choice for MCO1 because of the completeness of the Ruby-developed program, the simplicity and ease of use of Ruby, and its object-oriented approach which was suitable for creating a payroll system. In comparison to the programs developed in the other programming languages, developing the Ruby payroll system went smoothly with correct calculations, apt input-checking, and only minor errors. Ruby was efficient and easy to use as a programmer, in contrast to, for example, the Kotlin program where every edit required the creation of new files. Ruby's syntax was also relatively easy to learn and apply. Its simple object-oriented approach was suitable for general purpose programs like the payroll system. Treating real-life concepts like days and salary as Ruby objects made programming in Ruby feel natural and even fun for a programmer.

While Ruby was the primary language, individual preferences and skills were important factors in the selection process. The non-utilization of other languages does not imply its inability to create an effective weekly payroll system. With adequate knowledge and skills, it is possible to develop a payroll system that may be more efficient than Ruby. Furthermore, given our experience with C and Java we consider them as potential languages suitable for developing such a payroll system.

VI. References

A. Kotlin References

Future. (n.d.). Kotlin Programming Language. https://kotlinlang.org/lp/10yearsofkotlin/future/

K, R. R. (n.d.). *Kotlin for Enterprise Applications using Java EE*. O'Reilly Online Learning. https://www.oreilly.com/library/view/kotlin-for-enterprise/9781788997270/ea4ec584-db64-4026-89a8-2086301eb9c5.xhtml#:~:text=Kotlin%20was%20developed%20by%20JetBrains,Java%2C%20including%20the%20Intellij%20Idea.

Kotlin, B. O., & Kotlin, B. O. (2024, January 30). *Is Kotlin Pass-By-Value or Pass-By-Reference? | Baeldung on Kotlin*. Baeldung on Kotlin. https://www.baeldung.com/kotlin/parameters-pass-value-reference#:~:text=Pass%2Dby%2Dreference%20is%20the,simulating%20pass%2Dby%2Dreference.

Kotlin Programming Language. (n.d.). Kotlin. https://kotlinlang.org/education/why-teach-kotlin.html

Kotlin releases | Kotlin. (n.d.). Kotlin Help. https://kotlinlang.org/docs/releases.html#release-details

Silverio, M. (2022, December 19). *What is Kotlin?* Built In. https://builtin.com/software-engineering-perspectives/kotlin#:~:text=Kotlin%20is%20designed%20to%20run,end%20development%20or%20data%20science.

What's new in Kotlin 1.9.20 | Kotlin. (n.d.-a). Kotlin Help. https://kotlinlang.org/docs/whatsnew1920.html

What's new in Kotlin 1.9.20 | Kotlin. (n.d.-b). Kotlin Help. https://kotlinlang.org/docs/whatsnew1920.html

B. Ruby References

Bagwala, M. (2022, April 12). How does Ruby manage memory? *Saeloun Blog*. https://blog.saeloun.com/2022/04/12/ruby-variable-width-allocation.rb/

Hogan, B. (2023, January 26). *Understanding data types in Ruby*. DigitalOcean. https://www.digitalocean.com/community/tutorials/understanding-data-types-in-ruby

MJIT Support in Ruby 2.6. (2019, March 5). BigBinary. https://www.bigbinary.com/blog/mjit-support-in-ruby-2-6

Olowode, A. (2022, August 24). An introduction to ractors in Ruby. *2021 AppSignal*. https://blog.appsignal.com/2022/08/24/an-introduction-to-ractors-in-ruby.html

Ruby: Optimized for programmer happiness | Learn Enough News & blog. (n.d.). Learn Enough News & Blog. https://news.learnenough.com/ruby-optimized-for-programmer-happiness#fig-language_creator

School, L. (2022, January 6). Object passing in ruby — pass by reference or pass by value. *Medium*. https://launchschool.medium.com/object-passing-in-ruby-pass-by-reference-or-pass-by-value-6886e8cdc34a

C. Go References

Can, A. (2022, March 12). *Memory management in Go*. Medium. https://medium.com/@ali.can/memory-optimization-in-go-23a56544ccc0

Geeksforgeeks. (2020, April 21). *strconv.Atoi() Function in Golang With Examples*. GeeksforGeeks. https://www.geeksforgeeks.org/strconv-atoi-function-in-golang-with-examples/

Go. (n.d.). *The Go Programming Language.* Go. https://go.dev/

Go. (2023, August 8). *Go 1.21 Release Notes.* Go. https://go.dev/doc/go1.21

Golangbot. (2024, April 1). *Variables*. Golangbot. https://golangbot.com/variables/

Gopher Guides. (2019, April 30). *Understanding data types in Go.* DigitalOcean. https://www.digitalocean.com/community/tutorials/understanding-data-types-in-go#declaring-data-types-for-variables

Hassan, A. H. (2020, March 15). *Pass by Value and Pass by Reference in Go.* Medium. https://betterprogramming.pub/pass-by-value-and-reference-in-go-94423b6accf1

Nilsson, S. (n.d.). *Convert between float and string*. Yourbasic. https://yourbasic.org/golang/convert-string-to-float/

Pike, R. (2020, August 27). *Using Go at Google.* Go. https://go.dev/solutions/google/

Singh, K. (2023, April 22). *Go Programming Language: A comprehensive Introduction and History of Google's Revolutionary Language.* Dev. https://dev.to/mavensingh/go-programming-language-a-comprehensive-introduction-and-history-of-googles-revolutionary-language-3d76

Zharova, E. (2021, February 3). *The State of Go.* JetBrains. https://blog.jetbrains.com/go/2021/02/03/the-state-of-go/

D. R References

Bind data.tables by row and column — bind_rows. (n.d.). https://markfairbanks.github.io/tidytable/reference/bind_rows..html

Cn, P. (2022, August 3). The rbind() function in R - Binding Rows Made Easy. DigitalOcean. https://www.digitalocean.com/community/tutorials/rbind-function-r

Creating classes in R: S3, S4, R5 (RC), or R6? (n.d.). Stack Overflow. https://stackoverflow.com/questions/27219132/creating-classes-in-r-s3-s4-r5-rc-or-r6

GfG. (2022, April 6). Explicit coercion in R programming. GeeksforGeeks. https://www.geeksforgeeks.org/explicit-coercion-in-r-programming/

GfG. (2023, December 21). R Programming Language Introduction. GeeksforGeeks. https://www.geeksforgeeks.org/r-programming-language-introduction/

Peng, R. D. (2022, May 31). 2 History and overview of R | R Programming for Data Science. https://bookdown.org/rdpeng/rprogdatascience/history-and-overview-of-r.html#what-is-r

Programmatically distinguishing S3 and S4 objects in R. (n.d.). Stack Overflow. https://stackoverflow.com/questions/6959846/programmatically-distinguishing-s3-and-s4-objects-in-r

R, pass-by-value inside a function. (n.d.). Stack Overflow. https://stackoverflow.com/questions/29831261/r-pass-by-value-inside-a-function

R Manuals :: R Internals - 12 Current and future directions. (n.d.). https://rstudio.github.io/r-manuals/r-ints/Current-and-future-directions.html

Rawat, V. S. (2022, April 4). Chapter 11 Pass by Value-Reference | Best Coding Practices for R. https://bookdown.org/content/d1e53ac9-28ce-472f-bc2c-f499f18264a3/reference.html

Smith, O. (2020, June 22). Objects and classes in R. https://www.datacamp.com/tutorial/r-objects-and-classes

What is the programming paradigm of R? (n.d.). Stack Overflow. https://stackoverflow.com/questions/6098810/what-is-the-programming-paradigm-of-r

Wikipedia contributors. (2024, March 26). R (programming language). Wikipedia. https://en.wikipedia.org/wiki/R_(programming_language)#History

Wikipedia contributors. (2024, March 27). Lazy evaluation. Wikipedia. https://en.wikipedia.org/wiki/Lazy_evaluation

Worsley, S. (2023, October 17). What is R? - An Introduction to The Statistical Computing Powerhouse. https://www.datacamp.com/blog/all-about-r