UNIVERSIDAD NACIONAL DE ASUNCIÓN

FACULTAD POLITÉCNICA

“TRABAJO PRÁCTICO FINAL DE ASIGNATURA”

Asignatura: Estructura de los lenguajes de programación.

Profesor: Phd Christian Daniel von Lücken Martínez

Alumno: Juan Andres Gonzalez Arevalos

Semestre: 5to semestre.

AÑO: 2024

INTRODUCCIÓN

El propósito de este informe es analizar la implementación y resolución de tres problemas clásicos de programación utilizando cinco lenguajes: **Python**, **R**, **Ruby**, **C#**, y **Pascal**. Cada problema fue diseñado para explorar aspectos clave del diseño e implementación de algoritmos en diferentes paradigmas de programación, evaluando cómo cada lenguaje maneja estructuras de datos, flujo de control y manejo de errores.
El análisis se basa en los **criterios propuestos por Robert Sebesta** en su libro *Concepts of Programming Languages*, específicamente:

Facilidad de uso: ¿Qué tan intuitivo y directo es implementar el problema en cada lenguaje?

Eficiencia: ¿Qué tan rápido y óptimo es el desempeño del lenguaje en la resolución de problemas?

Expresividad: ¿Qué lenguaje permite escribir soluciones claras y comprensibles con menor esfuerzo?

Además, el informe incluye una **reflexión final** sobre las ventajas y desventajas de los lenguajes utilizados, así como una comparación crítica basada en la experiencia obtenida durante el desarrollo de las soluciones.

La importancia de este análisis radica en que los lenguajes de programación tienen fortalezas y debilidades específicas que los hacen más o menos adecuados para ciertos tipos de problemas. Al implementar estos algoritmos en diversos lenguajes, se adquiere un entendimiento más profundo sobre cómo elegir el lenguaje más apropiado para resolver problemas en escenarios del mundo real.

DESARROLLO

Problema 1: Ordenamiento Manual con MergeSort

Descripción de la Solución

La solución al problema de ordenamiento manual de registros utilizando el algoritmo **MergeSort** se implementó en cinco lenguajes de programación: Python, R, Ruby, C# y Pascal. El objetivo fue ordenar una lista de estudiantes (registros con atributos como nombre, edad y calificación) por el atributo **edad**, de menor a mayor.

Python:

Se utilizó una lista de diccionarios para representar los registros.

El algoritmo MergeSort se implementó mediante funciones recursivas para dividir la lista en mitades y luego fusionarlas de manera ordenada.

La flexibilidad de Python para manejar listas y su sintaxis compacta facilitaron la implementación.

R:

Los registros se representaron como una lista de listas.

El algoritmo MergeSort se implementó mediante funciones recursivas similares a las de Python.

Aunque R está optimizado para cálculos estadísticos, el manejo de estructuras de datos más generales, como listas de registros, requirió un poco más de esfuerzo.

Ruby:

Se utilizó un arreglo de objetos para representar los registros, aprovechando las capacidades orientadas a objetos de Ruby.

El enfoque recursivo de MergeSort fue sencillo de implementar gracias a la sintaxis clara de Ruby.

La capacidad de Ruby para operar con estructuras dinámicas simplificó el trabajo.

C#:

Se usaron listas genéricas (`List<T>`) para almacenar los registros como objetos de una clase personalizada.

El algoritmo MergeSort fue implementado como un conjunto de métodos recursivos, con control explícito sobre los índices y las sublistas.

C# exigió una mayor cantidad de código debido a su tipado estático, pero proporcionó una mayor claridad en la estructura del programa.

Pascal:

Los registros se implementaron como un arreglo dinámico de `records`, siguiendo el enfoque clásico de Pascal.

MergeSort se implementó manualmente con bucles y procedimientos recursivos.

Aunque Pascal requiere un manejo más explícito de los índices y estructuras, la claridad en el flujo del algoritmo es notable.

Comparación entre Lenguajes (Criterios de Sebesta)

Criterio	Python	R	Ruby	C#	Pascal
Facilidad de Uso	Alto. Las listas y la recursión en Python son muy intuitivas.	Moderado. El manejo de listas de listas en R requiere mayor conocimiento.	Alto. La sintaxis clara y las estructuras dinámicas facilitan la implementación.	Moderado. La implementación es robusta pero más verbosa debido al tipado estático.	Bajo. El manejo de índices y estructuras explícitas requiere mayor esfuerzo.
Eficiencia	Moderada. Python no es el más rápido para cálculos intensivos.	Baja. La eficiencia en R es limitada para estructuras no especializadas.	Moderada. Ruby es dinámico pero menos eficiente que C#.	Alta. El tipado estático y las optimizaciones del compilador mejoran el rendimiento.	Alta. Pascal es eficiente, pero el manejo manual de estructuras aumenta la complejidad.

Expresividad	Alta. Código conciso y fácil de entender.	Moderada. El código es menos claro debido a la estructura de listas anidadas.	Alta. La sintaxis y el paradigma orientado a objetos hacen el código legible.	Alta. La claridad del código mejora con tipado explícito y estructura clara.	Moderada. Aunque claro, el manejo manual reduce la expresividad.

Problema 2: Búsqueda Binaria

Descripción de la Solución

La búsqueda binaria se utilizó para localizar un registro en una lista ordenada de productos (con atributos como nombre, precio y cantidad). El algoritmo fue implementado manualmente, sin funciones predefinidas, verificando casos como listas vacías y elementos no encontrados.

Python:

Usó listas nativas para representar los registros.

El algoritmo se implementó con un enfoque iterativo y control de errores mediante excepciones (`try-except`).

R:

Los registros se manejaron como listas de listas.

Se utilizó una estructura iterativa para la búsqueda, con manejo de errores mediante la función `tryCatch`.

Ruby:

Los registros se representaron como objetos en un arreglo.

La búsqueda binaria se implementó mediante bucles, con mensajes de error personalizados.

C#:

Se utilizó una lista genérica (`List<T>`) para representar los productos como objetos.

La búsqueda binaria se implementó con un manejo estricto de índices y un control de errores robusto mediante `try-catch`.

Pascal:

Los registros se representaron como arreglos dinámicos de `records`.

La implementación se realizó mediante bucles, con un manejo explícito de casos de error utilizando `writeln` y `halt`.

Comparación entre Lenguajes (Criterios de Sebesta)

Criterio	Python	R	Ruby	C#	Pascal
Facilidad de Uso	Alta. Las listas y manejo de excepciones son intuitivos.	Moderada. El manejo de listas anidadas complica el desarrollo.	Alta. La estructura orientada a objetos facilita la claridad.	Moderada. Código más detallado por el tipado estático.	Baja. Requiere mayor esfuerzo por el manejo explícito de índices y errores.
Eficiencia	Moderada. Python es eficiente, pero no optimizado para grandes listas.	Baja. La velocidad en R es limitada para búsquedas iterativas manuales.	Moderada. Ruby tiene un buen desempeño, aunque no es el más eficiente.	Alta. El rendimiento es notable debido a las optimizaciones del compilador.	Alta. Pascal es eficiente, pero la gestión manual de índices aumenta la carga.
Expresividad	Alta. Código claro y fácil de entender.	Moderada. Las listas anidadas dificultan la lectura del código.	Alta. La implementación es concisa y legible.	Alta. La estructura del código es clara pero más extensa.	Moderada. Aunque claro, la verbosidad reduce la expresividad.

Problema 3: Multiplicación de Matrices

Descripción de la Solución

La solución al problema de multiplicación de matrices se implementó manualmente en cinco lenguajes de programación: Python, R, Ruby, C# y Pascal. En cada lenguaje, se verificó la compatibilidad de dimensiones de las matrices (el número de columnas de la primera matriz debe ser igual al número de filas de la segunda), y el cálculo se realizó utilizando bucles anidados.

Python:

Las matrices se representan como listas de listas.

La implementación consistió en un doble bucle para calcular los valores de la matriz resultado y otro bucle para recorrer los índices comunes.

El manejo de errores para verificar dimensiones incompatibles se implementó con excepciones (`raise`).

R:

Las matrices se representaron como objetos `matrix`, que son nativos en R.

Aunque R incluye operadores predefinidos como `%*%` para multiplicación de matrices, la implementación se realizó manualmente usando bucles para cumplir con las instrucciones.

Se utilizó `stop` para manejar errores en caso de incompatibilidad de dimensiones.

Ruby:

Las matrices se representan como arreglos bidimensionales.

Se utilizó un enfoque iterativo similar al de Python, con tres bucles anidados.

Los errores se manejaron con mensajes explícitos y terminación de la ejecución.

C#:

Se usaron arreglos bidimensionales (`int[,]`) para las matrices.

La implementación incluyó tres bucles anidados para calcular los valores de la matriz resultado.

Los errores de compatibilidad de dimensiones se gestionaron mediante excepciones (`throw` y `try-catch`).

Pascal:

Las matrices se implementaron como arreglos dinámicos bidimensionales (`array of array`).

La lógica de multiplicación incluyó tres bucles, y los errores se manejaron con mensajes (`writeln`) y terminación del programa (`halt`).

Aunque menos intuitivo que en lenguajes modernos, Pascal permitió un control total sobre los cálculos.

Comparación entre Lenguajes (Criterios de Sebesta)

Criterio	Python	R	Ruby	C#	Pascal
Facilidad de Uso	Alta. Las listas y su flexibilidad simplificaron la implementación.	Moderada. Aunque R maneja matrices de forma nativa, los bucles manuales son menos intuitivos.	Alta. Ruby tiene una sintaxis clara y un manejo sencillo de arreglos.	Moderada. El tipado estático exigió una mayor cantidad de código.	Baja. El manejo manual de arreglos y errores hace que sea más complejo.
Eficiencia	Moderada. Python no está optimizado para cálculos intensivos en matrices grandes sin bibliotecas externas.	Moderada. R es eficiente para operaciones matriciales nativas, pero menos en bucles manuales.	Moderada. Ruby ofrece un rendimiento aceptable, pero no está diseñado para operaciones de alto rendimiento.	Alta. C# es eficiente gracias a las optimizaciones del compilador.	Alta. Pascal es eficiente para operaciones de bajo nivel y algoritmos manuales.
Expresividad	Alta. El código es claro y compacto, facilitando su comprensión.	Moderada. El uso de estructuras y bucles hace que el código sea menos claro.	Alta. La sintaxis de Ruby permite escribir código legible y elegante.	Alta. La estructura de C# garantiza claridad en el código, aunque más verboso.	Moderada. Aunque claro, requiere más trabajo debido al manejo explícito.

Observaciones Generales

Python:

El lenguaje es altamente expresivo y permite escribir código sencillo y claro. Sin embargo, no es el más eficiente para operaciones matriciales manuales sin bibliotecas como `numpy`.

R:

Aunque R está diseñado para trabajar con matrices, este problema resalta las limitaciones del lenguaje al implementar operaciones manuales, ya que su fuerza está en el uso de funciones predefinidas.

Ruby:

Ruby es fácil de usar y su flexibilidad permite implementar operaciones matriciales de forma intuitiva. Sin embargo, al no ser un lenguaje orientado a cálculos matemáticos intensivos, no es la opción más eficiente.

C#:

La implementación en C# Es robusta y eficiente gracias a las optimizaciones del compilador. Sin embargo, la verbosidad del código puede ser un inconveniente para usuarios que busquen soluciones rápidas y compactas.

Pascal:

Aunque Pascal es eficiente para operaciones básicas, su sintaxis y manejo explícito de estructuras lo hacen menos adecuado para problemas de cálculo matricial comparado con lenguajes modernos.

Reflexión Final

La experiencia de implementar y resolver los problemas propuestos en cinco lenguajes diferentes proporcionó una visión clara sobre las fortalezas y limitaciones de cada uno. Cada lenguaje destaca en aspectos específicos que lo hacen más adecuado para ciertas tareas, lo que subraya la importancia de elegir el lenguaje correcto según el problema a resolver.

Python

Python fue el lenguaje más accesible y expresivo para implementar las soluciones. Su sintaxis sencilla y su capacidad para manejar estructuras de datos dinámicas, como listas de listas, lo convierten en una herramienta ideal para resolver problemas complejos de manera rápida y legible. Aunque no es el más eficiente en términos de desempeño, su soporte extensivo para bibliotecas como `numpy` lo hace altamente competitivo en aplicaciones prácticas, especialmente para cálculos matriciales. Python resultó ser el lenguaje más interesante para quienes valoran la claridad y rapidez en el desarrollo.

R

R fue eficiente en operaciones nativas, como trabajar con matrices, pero mostró limitaciones en flexibilidad y expresividad al implementar algoritmos manualmente. Aunque no es el lenguaje más intuitivo para programar fuera de su dominio principal (análisis estadístico), su capacidad para manejar cálculos matriciales predefinidos destaca. Sin embargo, su uso puede ser más desafiante para desarrolladores sin experiencia en análisis de datos.

Ruby

Ruby sorprendió por su combinación de sintaxis clara y soporte para estructuras dinámicas. Su enfoque orientado a objetos permitió implementar soluciones de manera intuitiva, aunque su rendimiento es inferior al de lenguajes más optimizados como C#. Ruby se destacó por ser expresivo y amigable para programadores que buscan un equilibrio entre legibilidad y simplicidad.

C#

C# ofreció un balance interesante entre eficiencia y claridad. Aunque requiere escribir más código debido a su tipado estático y estructura detallada, esto contribuyó a una mayor robustez en las soluciones. Las optimizaciones del compilador y el soporte para bibliotecas avanzadas, como **MathNet.Numerics**, lo convierten en una excelente opción para proyectos complejos donde el rendimiento es crucial. Este lenguaje destacó como uno de los más completos y profesionales para resolver los problemas propuestos.

Pascal

Pascal, aunque más limitado en comparación con los otros lenguajes, permitió entender mejor los fundamentos de los algoritmos gracias a su estructura clara y su enfoque en el manejo manual de datos. Sin embargo, su sintaxis más detallada y la falta de soporte moderno lo hicieron menos adecuado para resolver problemas complejos de manera eficiente. A pesar de estas limitaciones, fue interesante trabajar con un lenguaje que obliga a desarrollar soluciones meticulosas y detalladas, destacándose como un excelente lenguaje para propósitos educativos.

El lenguaje más interesante resultó ser **Python**, principalmente debido a su combinación de simplicidad y flexibilidad. Su capacidad para manejar estructuras de datos complejas sin una sintaxis engorrosa lo convierte en una opción preferida para problemas de programación algorítmica. Aunque C# mostró un rendimiento superior y Ruby fue notable por su legibilidad, Python demostró ser el lenguaje que mejor equilibra facilidad de uso, expresividad y capacidad para adaptarse a distintos tipos de problemas.

CONCLUSIÓN

El desarrollo e implementación de los tres problemas en cinco lenguajes de programación permite explorar, desde diferentes perspectivas, la relación entre los paradigmas de programación, las características propias de cada lenguaje y las necesidades específicas de los problemas planteados. Este ejercicio no solo evidenció las fortalezas y limitaciones de cada lenguaje, sino también la importancia de comprender cómo seleccionar la herramienta adecuada para cada escenario.

Lecciones Aprendidas

1. **No hay un lenguaje único para todo**: La diversidad de herramientas subraya que cada lenguaje está diseñado para resolver ciertos problemas de manera más efectiva que otros. Por ejemplo, mientras que Python es excelente para la programación general, C# brilla en aplicaciones donde la eficiencia es crítica, y R domina en el ámbito del análisis estadístico.
2. **Compromisos entre simplicidad y control**: Lenguajes como Python y Ruby priorizan la simplicidad y expresividad, pero a menudo sacrifican el rendimiento. Por otro lado, Pascal y C# Ofrecen un mayor control sobre las estructuras y los recursos, pero requieren un esfuerzo adicional en términos de desarrollo.
3. **Importancia del contexto**: Seleccionar el lenguaje adecuado depende del tipo de problema a resolver. Si el objetivo es aprender fundamentos algorítmicos, Pascal o C# ofrecen un enfoque riguroso. Si se priorizan resultados rápidos y código legible, Python o Ruby son las mejores opciones. En el caso de análisis estadístico o científico, R es claramente superior.

BIBLIOGRAFÍA

- Sebesta, R. W. (2015). *Concepts of Programming Languages* (11th ed.). Pearson Education.

Este libro fue la referencia principal para analizar y comparar los lenguajes de programación según criterios como facilidad de uso, eficiencia y expresividad.

- Python Software Foundation. (2023). *Python Documentation*. Obtenido de https://docs.python.org/

Referencia utilizada para comprender las estructuras de datos y sintaxis de Python durante la implementación de las soluciones.

- The R Project for Statistical Computing. (2023). *R Documentation*. Obtenido de https://www.r-project.org/

Utilizado para consultar detalles sobre el manejo de matrices y estructuras de datos en R.

- Ruby Programming Language. (2023). *Ruby Documentation*. Obtenido de https://ruby-doc.org/

Referencia para el manejo de arreglos y objetos en Ruby, utilizados en las implementaciones.

- Microsoft. (2023). *C# Documentation*. Obtenido de https://learn.microsoft.com/en-us/dotnet/csharp/

Utilizado para implementar soluciones en C# y comprender estructuras como listas genéricas y excepciones.

- Free Pascal. (2023). *Free Pascal Documentation*. Obtenido de https://www.freepascal.org/docs.html

Fuente para comprender las características del compilador Free Pascal y el manejo de arreglos bidimensionales.

- NumPy Developers. (2023). *NumPy Documentation*. Obtenido de https://numpy.org/doc/

Utilizado como referencia adicional para la multiplicación de matrices en Python con bibliotecas avanzadas.

- OnlineGDB. (2023). *Free Pascal Online Compiler*. Obtenido de https://www.onlinegdb.com/

Plataforma utilizada para probar y ejecutar el código en Pascal.

- DotNetFiddle. (2023). *C# Online Compiler*. Obtenido de https://dotnetfiddle.net/

Plataforma utilizada para probar y ejecutar el código en C#.

- Replit. (2023). *Online IDE for Python, Ruby, and Other Languages*. Obtenido de https://replit.com/

Plataforma utilizada para probar las implementaciones en Python y Ruby.