Una caricatura de una persona

Descripción generada automáticamente con confianza baja

**FACULTAD DE INFORMÁTICA ELECTRÓNICA Y COMUNICACIÓN**

**LICENCIATURA EN INGENIERÍA EN INFORMÁTICA**

**MATERIA:**

**ESTRUCTURAS DE LOS LENGUAJES DE PROGRAMACIÓN**

**LABORATORIO #6**

**Criterios de Comparación y Evaluación de los Lenguajes de Programación**

**PROFESOR: SAÚL ARDINES**

**NOMBRE:**

**ASHLY MENDIETA EC-106-1297**

**FECHA DE ENTREGA: 07 DE NOVIEMBRE DE 2024**

**SEGUNDO SEMESTRE**

**AÑO LECTIVO 2024**

Contenido

[**Introducción** 3](#_Toc181721683)

[**1.** **Breve Reseña Histórica** 4](#_Toc181721684)

[**1.1** **Lenguaje de programación OBJETIVE-C** 4](#_Toc181721685)

[**1.2** **Lenguaje de programación C#** 4](#_Toc181721686)

[**2.1** **Características del lenguaje Cobol** 5](#_Toc181721687)

[**3. Objetos de datos** 6](#_Toc181721688)

[3.1 Tipos de datos primitivos 6](#_Toc181721689)

[3.1.1 Variables y constantes 6](#_Toc181721690)

[3.1.2 Tipos de datos numéricos 6](#_Toc181721691)

[3.1.3 Tipos de datos de caracteres 6](#_Toc181721692)

[3.1.4 Tipos de datos booleanos 7](#_Toc181721693)

[3.2 Tipos de datos estructurados 7](#_Toc181721694)

[3.2.1 Tipos de datos de arreglos 7](#_Toc181721695)

[3.2.2 Tipos de datos de registro 7](#_Toc181721696)

[3.2.3 Apuntadores o Punteros 7](#_Toc181721697)

[3.2.4 Archivo de entrada/salida 8](#_Toc181721698)

[3.3 Tipos de datos definidos por el usuario 8](#_Toc181721699)

[**4. Control de secuencia** 8](#_Toc181721700)

[4.1 Expresiones 8](#_Toc181721701)

[4.2 Enunciados 8](#_Toc181721702)

[4.1.1 Enunciados de entrada y salida 8](#_Toc181721703)

[4.1.2 Enunciados compuestos 9](#_Toc181721704)

[4.1.3 Enunciados condicionales 9](#_Toc181721705)

[4.1.4 Enunciados de iteración 9](#_Toc181721706)

[**5. Subprogramas y gestión de almacenamiento** 10](#_Toc181721707)

[**6. Ejemplos prácticos** 11](#_Toc181721708)

[**7. Cuadro de Comparación y Evaluación de los lenguajes de programación** 13](#_Toc181721709)

[**8. Evaluación y conclusiones** 18](#_Toc181721710)

[**Referencias Bibliográficas** 19](#_Toc181721711)

# **Breve Reseña Histórica**

## **Lenguaje de programación Objetive-c**

Es un lenguaje de programación orientado a objetos que fue desarrollado en la década de 1980 por Brad Cox y Tom Love. Es una extensión del lenguaje C, añadiendo capacidades de programación orientada a objetos mediante la inclusión de clases, métodos y el sistema de mensajes. El lenguaje se caracteriza por su sintaxis única, que combina la notación de C con una estructura de mensajería dinámica basada en Smalltalk. Esto significa que en Objective-C, los métodos no se invocan de la manera convencional (como en C o C++), sino que se envían mensajes a los objetos. Fue el principal lenguaje de desarrollo para las plataformas de Apple (macOS, iOS, iPadOS, watchOS y tvOS) durante muchos años, hasta la introducción de Swift en 2014. Sin embargo, sigue siendo importante en muchos proyectos existentes y en el mantenimiento de código legado. Aunque Swift ha tomado su lugar como lenguaje preferido para nuevas aplicaciones, Objective-C sigue siendo una herramienta poderosa para los desarrolladores que trabajan con la infraestructura de Apple.

## **Lenguaje de programación RPG**

Es un lenguaje de programación orientado a objetos que fue desarrollado en la década de 1980 por Brad Cox y Tom Love. Es una extensión del lenguaje C, añadiendo capacidades de programación orientada a objetos mediante la inclusión de clases, métodos y el sistema de mensajes. El lenguaje se caracteriza por su sintaxis única, que combina la notación de C con una estructura de mensajería dinámica basada en Smalltalk. Esto significa que en Objective-C, los métodos no se invocan de la manera convencional (como en C o C++), sino que se envían mensajes a los objetos. Fue el principal lenguaje de desarrollo para las plataformas de Apple (macOS, iOS, iPadOS, watchOS y tvOS) durante muchos años, hasta la introducción de Swift en 2014. Sin embargo, sigue siendo importante en muchos proyectos existentes y en el mantenimiento de código legado. Aunque Swift ha tomado su lugar como lenguaje preferido para nuevas aplicaciones, Objective-C sigue siendo una herramienta poderosa para los desarrolladores que trabajan con la infraestructura de Apple.

**OBJETIVE-C VS C#**

|  |  |  |  |
| --- | --- | --- | --- |
| **Criterios** | **Razones** | **Calificación**  **OBJETIVE-C** | **Calificación**  **C#** |
| 1. **Expresividad** | **1.Sintaxis de Declaración y Lectura de Código:** Objective-C utiliza una sintaxis más compleja y basada en símbolos, lo que puede dificultar la lectura y comprensión rápida del código, especialmente para aquellos no familiarizados con sus convenciones. La sintaxis de C#, en cambio, es más moderna, clara y compacta, lo que facilita la lectura del código y permite que las declaraciones reflejen de forma más directa el propósito de la programación. Esto mejora la expresividad, ya que el código es más fácil de entender y mantiene su significado de manera más inmediata.   1. **Manejo de Memoria y Gestión Automática:** En Objective-C, la gestión de memoria, aunque ha mejorado con ARC, sigue siendo relativamente compleja y requiere que el programador se ocupe explícitamente de la memoria, lo que puede hacer que el código sea más difícil de mantener y comprender. En C#, el recolector de basura (garbage collection) automatiza el manejo de memoria, permitiendo que el programador se enfoque más en la lógica del programa, lo que resulta en un código más limpio, directo y expresivo. 2. **Características de Lenguaje (LINQ y Expresiones Lambda en C# vs. Objective-C):** Objective-C no cuenta con características avanzadas para manipular colecciones de manera declarativa y compacta, lo que obliga a escribir más código imperativo para lograr lo mismo. C# incluye LINQ, que permite realizar consultas y transformaciones de colecciones de manera declarativa, haciendo que las operaciones sean más fáciles de expresar en pocas líneas de código, lo que refuerza la expresividad y facilita la comprensión del código al reflejar claramente las intenciones del programador. 3. **Paradigma de Programación y Facilidad para la Programación Orientada a Objetos:** Objective-C sigue un enfoque orientado a objetos con una sintaxis basada en mensajes, lo que puede hacer que la estructura y el flujo del código sean menos claros para programadores acostumbrados a lenguajes con una sintaxis más directa. C#, por otro lado, ofrece una sintaxis orientada a objetos más estándar y coherente, lo que facilita la expresión de conceptos como herencia, polimorfismo y encapsulamiento de manera clara y concisa, ayudando a reflejar el significado del programa con mayor precisión y menos confusión. 4. **Soporte para Programación Asíncrona:** La programación asíncrona en Objective-C se realiza mediante bloques y GCD, pero la sintaxis y el patrón de uso pueden resultar complejos y difíciles de seguir, especialmente cuando hay múltiples tareas asíncronas. En C#, el soporte para programación asíncrona con las palabras clave async y await permite escribir código asíncrono de manera casi idéntica al código secuencial, haciendo que el flujo del programa sea más claro y que el propósito del código asíncrono se exprese de forma más directa. | **3.6** | **4.4** |
| **2. Bien-Definido** | 1. **Claridad en la Sintaxis y Semántica: Objective-C** extiende la sintaxis de C con una notación única para la comunicación con objetos, lo que introduce una semántica más compleja que puede generar ambigüedad, especialmente si no se está familiarizado con el modelo de mensajería de **Objective-C** o el marco de trabajo **Cocoa**. En cambio, **C#** ofrece una sintaxis clara y moderna, con reglas semánticas bien definidas, facilitando la comprensión del lenguaje. La semántica de **C#** está basada en un sistema de tipos estáticos y objetos claramente definidos, lo que elimina muchas de las ambigüedades presentes en otros lenguajes y facilita el aprendizaje para los programadores. 2. **Especificación Completa y Definición Formal del Lenguaje:** Mientras que **Objective-C** tiene una especificación formal, su integración con plataformas específicas de Apple como Cocoa y Cocoa Touch significa que muchos aspectos de su semántica dependen de las implementaciones de estas librerías. Esto hace que, en algunos casos, el lenguaje no esté completamente especificado fuera del ecosistema de Apple. En contraste, **C#** ofrece una especificación detallada y formal que cubre todos los aspectos del lenguaje, desde la sintaxis hasta la semántica de tipos y objetos, lo que lo convierte en un lenguaje más predecible y fácil de entender, con una documentación ampliamente accesible para desarrolladores. 3. **Transportabilidad del Lenguaje: Objective-C** está vinculado principalmente al ecosistema de Apple, por lo que su portabilidad fuera de macOS y iOS es limitada. Aunque existen implementaciones alternativas, la mayoría de las librerías y herramientas están diseñadas específicamente para estas plataformas, lo que dificulta su uso en otras configuraciones. Por otro lado, C# fue diseñado para ser portable desde sus inicios, con la plataforma .NET permitiendo que las aplicaciones escritas en **C#** se ejecuten de manera efectiva en una amplia variedad de sistemas operativos, incluyendo Windows, macOS y Linux, lo que lo hace mucho más adecuado para proyectos multiplataforma. 4. **Facilidad para Realizar Pruebas y Depuración:** La depuración en **Objective-C** puede ser más difícil debido a su modelo de mensajería y a la gestión manual de memoria (aunque con el uso de ARC esta complejidad ha disminuido). Los errores relacionados con la memoria o el manejo de objetos pueden ser difíciles de rastrear, especialmente en proyectos grandes y complejos. En cambio, **C#** se beneficia de un ecosistema robusto de herramientas de desarrollo, como Visual Studio, que facilita la depuración con capacidades avanzadas como la depuración en tiempo real, la visualización de estructuras de datos y las pruebas unitarias integradas. Además, la gestión automática de memoria en **C#** mediante el garbage collector reduce muchos de los errores típicos relacionados con la memoria, simplificando las pruebas y el mantenimiento del código. 5. **Interoperabilidad con Otros Lenguajes y Sistemas: Objective-C** está bien integrado dentro del ecosistema de Apple, pero fuera de este, la interoperabilidad con otros lenguajes es limitada. Aunque se pueden utilizar bibliotecas en C o interactuar con Swift, la integración con otros lenguajes o plataformas no es tan directa y puede requerir soluciones complejas. En cambio, **C#** ofrece una interoperabilidad más amplia gracias a su plataforma .NET, que permite interactuar fácilmente con otros lenguajes y sistemas mediante herramientas como P/Invoke y COM Interop. Además, C# puede integrarse con sistemas que no están basados en .NET, como aplicaciones en Java o servicios en Python, lo que lo hace más flexible para trabajar en entornos heterogéneos | **3.7** | **4.5** |
| 1. **Tipos y estructuras de datos** | 1. **Soporte de Tipos Primitivos y Tipos Básicos: Objective-C** hereda los tipos primitivos de **C** y permite trabajar con objetos a través de clases como NSString y NSArray. Sin embargo, el manejo de punteros y la relación entre tipos primitivos y objetos puede ser confuso. **C#** también soporta tipos primitivos y distingue entre tipos por valor y por referencia, lo que facilita su manejo. Además, ofrece tipos anulables y enumeraciones, lo que le da mayor flexibilidad. 2. **Arreglos y Registros: Objective-C** utiliza clases como NSArray para arreglos dinámicos, pero no tiene soporte directo para arreglos de tamaño fijo. Los registros se implementan mediante structs, pero se usan menos que las clases. En C#, se pueden usar tanto arreglos fijos como dinámicos, además de List<T>. Las structs en C# son más potentes y flexibles que las de Objective-C, y también permite trabajar con tipos anónimos. 3. **Estructuras de Datos Dinámicas (Listas Enlazadas, Pilas, Colas, Árboles, etc.):** En **Objective-C**, las estructuras de datos dinámicas deben implementarse manualmente, ya que las clases estándar como NSMutableArray no están optimizadas para casos complejos. **C#** proporciona una amplia variedad de colecciones avanzadas como **List<T>, Queue<T>, Stack<T> y Dictionary<TKey, TValue>**, que están optimizadas y son fáciles de usar. 4. **Gestión de Memoria y Punteros: Objective-C** permite el control manual de memoria mediante punteros, lo que ofrece flexibilidad, pero requiere cuidado para evitar errores. Aunque ARC facilita la gestión, sigue existiendo el riesgo de fugas de memoria. **C#** usa garbage collection, lo que automatiza la gestión de memoria, reduciendo el riesgo de errores, pero limita el acceso directo a punteros. 5. **Flexibilidad y Extensibilidad de las Estructuras de Datos: Objective-C** permite crear estructuras de datos personalizadas, pero requiere implementaciones manuales. En cambio, **C#** ofrece mayor flexibilidad mediante genéricos, LINQ y tipos anónimos, lo que facilita la creación y manipulación de colecciones y estructuras de datos complejas de manera más eficiente y reutilizable. | **3.7** | **4.8** |
| 1. **Modularidad** | 1. **Apoyo al uso de funciones y procedimientos (Subprogramación): Objective-C** facilita la modularidad mediante el uso de funciones y métodos dentro de las clases, lo que permite dividir el código en unidades más pequeñas y manejables. La comunicación entre estas unidades se realiza mediante el paso de parámetros, lo que permite la reutilización del código y la reducción de la complejidad. Sin embargo, su enfoque en punteros y referencias puede hacer que la interacción entre métodos sea más propensa a errores y más difícil de gestionar para desarrolladores novatos. En comparación, **C#** también soporta la modularidad a través de métodos dentro de clases, pero va un paso más allá con el uso de **delegados** y **eventos**, que permiten una interacción más flexible y menos dependiente de la estructura de clases. Esto facilita el diseño de sistemas más dinámicos, donde las funciones pueden ser pasadas como parámetros y ejecutadas de manera asíncrona o en respuesta a eventos, lo que refuerza aún más la modularidad del sistema. 2. **Extensibilidad mediante operadores y tipos definidos por el programador: Objective-C** permite la creación de tipos personalizados mediante clases y categorías, lo que ofrece una buena base para la modularidad. Sin embargo, la extensión de operadores y la definición de nuevos tipos de datos es más limitada que en otros lenguajes, ya que no tiene un soporte tan completo para la sobrecarga de operadores. Esto significa que, aunque es posible extender el lenguaje, la flexibilidad está restringida a ciertas estructuras de datos y métodos. En **C#**, la extensibilidad se lleva a otro nivel, ya que no solo permite la creación de nuevos tipos de datos mediante clases, structs y enumeraciones, sino que también soporta la **sobrecarga de operadores**. Esta característica permite al programador redefinir operadores para tipos personalizados, haciendo que las operaciones sobre esos tipos sean más naturales y legibles, lo que resulta en una mayor modularidad y flexibilidad. Además, las herramientas avanzadas como **delegados**, **expresiones lambda** y **eventos** en **C#** enriquecen aún más la extensibilidad, permitiendo que nuevas funcionalidades sean añadidas de manera eficiente. 3. **Modularidad a través de bibliotecas y namespaces**: En **Objective-C**, la modularidad se implementa principalmente mediante bibliotecas y frameworks como **Cocoa** y **Cocoa Touch**, que contienen clases y métodos predefinidos que los desarrolladores pueden utilizar para construir aplicaciones. Estas bibliotecas se integran fácilmente con el código a través de la directiva #import, lo que promueve la reutilización del código dentro del ecosistema **Apple**. Sin embargo, la modularidad de **Objective-C** está bastante centrada en el entorno específico de **macOS** y **iOS**, lo que limita su interoperabilidad con otros ecosistemas. En contraste, **C#** ofrece una modularidad más robust gracias a su estructura basada en **namespaces** y **assemblies** dentro del ecosistema **.NET**, lo que permite organizar y distribuir el código de manera eficiente. Los **namespaces** facilitan la gestión de nombres y la separación de funciones dentro de grandes proyectos, mientras que los **assemblies** permiten la reutilización de código entre diferentes aplicaciones. Además, **C#** puede integrarse fácilmente con otras plataformas y lenguajes, lo que amplía aún más las posibilidades de modularidad y reutilización en contextos variados. 4. **Control y visibilidad de los elementos modulares: Objective-C** proporciona mecanismos para controlar la visibilidad de los métodos y propiedades dentro de las clases mediante modificadores como @private, @protected, y @public, lo que permite encapsular los datos y asegurar que solo las partes relevantes del código puedan acceder a ellos. Además, las **categorías** permiten agregar nuevas funcionalidades a clases existentes sin alterar su código base, promoviendo la reutilización del código. Sin embargo, el uso de estas características puede ser menos transparente y más propenso a errores si no se gestiona correctamente. **C#** ofrece un control más robusto y explícito de la visibilidad mediante los modificadores public, private, protected, y internal, lo que proporciona una estructura más clara y segura para el acceso a los miembros de las clases. Además, las propiedades automáticas y la mayor integración con herramientas de encapsulación hacen que el código en **C#** sea más fácil de gestionar, y su visibilidad es más flexible y clara, lo que facilita la modularidad en proyectos de gran escala. 5. **Soporte para la programación funcional y modular: Objective-C** es un lenguaje orientado principalmente a objetos, con soporte limitado para la programación funcional. Aunque los bloques permiten realizar algunas operaciones de manera funcional, como pasar funciones como parámetros, el lenguaje no está tan diseñado para facilitar este tipo de programación. El modularidad, por lo tanto, depende en gran medida de la orientación a objetos y la organización en clases. En cambio, **C#** ofrece un soporte mucho más fuerte para la programación funcional, especialmente a través de **LINQ**, **expresiones lambda** y **delegados**, lo que permite trabajar con colecciones y datos de manera más declarativa y modular. Estas características permiten a los desarrolladores escribir código más conciso, reutilizable y expresivo, sin perder la capacidad de utilizar estructuras orientadas a objetos. La integración de estos paradigmas funcionales dentro del sistema orientado a objetos de **C#** fomenta una modularidad superior, ya que los datos pueden ser manipulados de manera más flexible y eficiente, lo que aumenta la reutilización y reduce la complejidad. | **3.5** | **4.6** |
| 1. **Facilidades de E/S** | 1. **Manejo de Archivos**: En Objective-C, la manipulación de archivos se realiza principalmente a través de las clases de **Foundation** como NSFileManager y NSData, lo cual ofrece un enfoque algo manual y específico de cada plataforma, lo que puede hacer que trabajar con archivos sea más tedioso. Por otro lado, C# ofrece un conjunto de clases más accesibles y modernas dentro de **System.IO**, como FileStream y StreamReader, que permiten un manejo más simple y directo de archivos, lo que facilita tanto el trabajo con archivos secuenciales como con archivos directos. 2. **Acceso a Bases de Datos**: Objective-C maneja el acceso a bases de datos principalmente a través de **Core Data**, que abstrae el uso de SQLite pero requiere una configuración más detallada y compleja. En contraste, C# ofrece **Entity Framework (EF)**, un ORM que permite manejar bases de datos de manera mucho más eficiente y sencilla, ocultando gran parte de la complejidad SQL y ofreciendo herramientas de alto nivel como LINQ para realizar consultas, lo que simplifica el acceso y manipulación de bases de datos relacionales. 3. **Soporte para Bases de Datos No Relacionales**: Mientras que Objective-C no tiene una solución nativa integrada para bases de datos NoSQL, los desarrolladores pueden recurrir a bibliotecas de terceros como **MongoDB** o **Firebase**, lo que puede ser más complejo en términos de configuración e integración. C#, en cambio, tiene un soporte mucho más robusto y directo para bases de datos NoSQL como **MongoDB** o **CosmosDB**, con bibliotecas bien documentadas que facilitan la interacción con bases de datos no estructuradas y semi-estructuradas. 4. **Conectividad de Red y Servicios Web**: En el caso de Objective-C, la conectividad de red se maneja mediante bibliotecas como **NSURLSession** y **NSURLConnection**, que pueden ser menos intuitivas y requieren un manejo más explícito de las conexiones. C# destaca por su facilidad en este campo gracias a herramientas como **HttpClient**, **WebClient**, y **WCF (Windows Communication Foundation)**, que no solo simplifican las solicitudes HTTP, sino que también ofrecen una integración moderna con servicios web, especialmente en el contexto de aplicaciones empresariales y en la nube. 5. **Facilidad de Uso y Productividad**: Mientras que en Objective-C trabajar con E/S y bases de datos puede resultar más laborioso debido a su enfoque más bajo nivel y menos abstraído, C# ofrece un entorno mucho más productivo gracias a su sintaxis moderna y herramientas de desarrollo como Visual Studio, que permiten a los desarrolladores trabajar de manera más ágil. La integración de C# con bibliotecas y marcos de trabajo como **Entity Framework** y **System.IO** hace que las tareas de E/S sean más sencillas y rápidas de implementar, mejorando significativamente la productividad. | **3.4** | **4.7** |
| 1. **Transportabilidad** | 1. **Estandarización del Lenguaje**: **Objective-C** no es completamente estandarizado y depende de compiladores y bibliotecas específicas de Apple como **Cocoa** y **Cocoa Touch**, lo que limita su portabilidad fuera del ecosistema Apple. En cambio, **C#** está estandarizado a través del **.NET Framework** y **CLI**, permitiendo que las aplicaciones se ejecuten en plataformas como **Windows**, **Linux** y **macOS**, lo que le otorga mayor portabilidad. 2. **Compatibilidad Multiplataforma**: **Objective-C** está diseñado para los sistemas operativos de Apple, como **macOS** y **iOS**, y su portabilidad a otras plataformas es limitada, a pesar de esfuerzos como **GNUstep**. **C#**, por otro lado, es compatible con múltiples plataformas gracias a **.NET Core** y **Mono**, permitiendo su ejecución en **Linux**, **macOS**, **Android**, y más, lo que lo hace más flexible y transportable. 3. **Dependencia de Herramientas y Entornos Específicos**: **Objective-C** depende de **Xcode** y las bibliotecas de **Apple** para su desarrollo, lo que restringe su portabilidad. En contraste, **C#** es menos dependiente de plataformas específicas gracias a **.NET Core** y **Mono**, lo que permite desarrollar en una amplia variedad de entornos, aumentando su flexibilidad y portabilidad. 4. **Soporte para Sistemas Operativos No Relacionados**: **Objective-C** está limitado a los sistemas operativos de Apple debido a su dependencia de bibliotecas propietarias, lo que dificulta su portabilidad. **C#** ofrece soporte para una variedad de sistemas operativos no relacionados, como **Linux** y **macOS**, gracias a **.NET Core** y **Xamarin**, lo que lo hace más adaptable y portátil a diferentes entornos. 5. **Estandarización de Librerías y Frameworks**: Las bibliotecas en **Objective-C**, como **Cocoa** y **Cocoa Touch**, están diseñadas específicamente para plataformas de Apple, lo que limita su uso fuera de este ecosistema. En comparación, **C#** se beneficia de las bibliotecas estandarizadas de **.NET**, que son consistentes y disponibles en diversas plataformas, mejorando la portabilidad y consistencia del código. | **3.1** | **4.7** |
| 1. **Eficiencia** | 1. **Tiempo de Compilación:** El tiempo de compilación en **Objective-C** puede ser relativamente más lento en comparación con otros lenguajes debido a la combinación de compilación estática y dinámica, así como su estrecha dependencia de herramientas como **Xcode** y bibliotecas específicas de Apple. En proyectos grandes, esto puede aumentar el tiempo necesario para generar la aplicación. Por otro lado, **C#** tiene tiempos de compilación más rápidos, principalmente gracias a su integración con el **.NET Core** y la optimización de sus herramientas de compilación, lo que lo hace más eficiente para desarrollos en múltiples plataformas. 2. **Optimización del Código de Ejecución**: En términos de ejecución, **Objective-C** es en general eficiente, especialmente en dispositivos Apple, pero la sobrecarga del despacho dinámico de mensajes y el modelo de objetos dinámico pueden afectar el rendimiento en ciertos escenarios. La dependencia de **Cocoa** y **Cocoa Touch** también introduce cierta latencia en aplicaciones que requieren alta eficiencia en tiempo de ejecución. En cambio, **C#** es muy eficiente debido a la optimización del código intermedio que se compila a código nativo mediante el compilador JIT en la **Common Language Runtime (CLR)**. Esto permite que **C#** realice optimizaciones dinámicas durante la ejecución, manteniendo un buen rendimiento en diversas plataformas. 3. **Gestión de Memoria**: La gestión de memoria en **Objective-C** es realizada mediante **Automatic Reference Counting (ARC)**, que ayuda a evitar errores comunes de memoria y optimiza la eficiencia al liberar objetos automáticamente. Sin embargo, en aplicaciones complejas con muchas interacciones entre objetos, la recolección de memoria puede generar algo de sobrecarga. En cambio, **C#** utiliza un sistema de **garbage collection (GC)**, lo que facilita la gestión automática de memoria, aunque puede haber pausas ocasionales cuando se realiza la recolección de basura. A pesar de esto, el recolector de basura de **C#** está optimizado y sigue mejorando en cada versión, reduciendo significativamente su impacto en el rendimiento. 4. **Rendimiento en Entornos Multihilo**: En el caso de **Objective-C**, el rendimiento en entornos multihilo se ve favorecido por herramientas como **Grand Central Dispatch (GCD)** y **NSOperationQueue**, que permiten gestionar tareas en paralelo de manera eficiente dentro del ecosistema de Apple. Sin embargo, la gestión de hilos y la sincronización de recursos pueden ser complejas y afectadas por la naturaleza del lenguaje. En cambio, **C#** destaca en este aspecto con su poderoso **Task Parallel Library (TPL)** y el modelo **async/await**, que permiten gestionar operaciones asincrónicas y paralelismo de manera más sencilla y eficiente, optimizando el rendimiento en aplicaciones que requieren alto rendimiento multihilo, especialmente en servidores y aplicaciones con alta concurrencia. 5. **Optimización para Dispositivos Específicos**: **Objective-C** es extremadamente eficiente en dispositivos Apple debido a su estrecha integración con el hardware y el sistema operativo de Apple. Esto le permite aprovechar al máximo las optimizaciones específicas del ecosistema, resultando en un rendimiento sobresaliente en plataformas como **macOS** e **iOS**. En comparación, **C#** es más flexible, pero su rendimiento en dispositivos Apple no es tan optimizado como en **Objective-C**, ya que no se integra de la misma forma con el hardware y el sistema operativo. Sin embargo, con herramientas como **Xamarin**, **C#** permite crear aplicaciones multiplataforma, aunque siempre existe una ligera sobrecarga cuando se ejecutan aplicaciones en plataformas no nativas. | **4.0** | **4.4** |
| 1. **Pedagogía** | 1. **Facilidad de Aprendizaje y Sintaxis**: La sintaxis de **Objective-C** es compleja y puede resultar difícil de entender para los principiantes. Su mezcla de código basado en **C** y programación orientada a objetos, combinada con el uso de punteros y el despacho dinámico de mensajes, hace que aprender este lenguaje sea un desafío. Además, la necesidad de comprender el manejo manual de memoria (aunque **ARC** facilita este proceso) puede aumentar la curva de aprendizaje. En cambio, **C#** ofrece una sintaxis moderna, clara y estructurada, alineada con lenguajes populares como **Java** y **C++**. Su enfoque orientado a objetos es intuitivo, y la falta de complejidad innecesaria en la definición de clases y objetos lo hace más accesible para los nuevos programadores. 2. **Documentación y Recursos Educativos**: La disponibilidad de recursos educativos para **Objective-C** ha disminuido en los últimos años, ya que muchos desarrolladores ahora optan por **Swift**. Aunque hay documentación en línea y libros, estos recursos tienden a estar enfocados en desarrolladores experimentados que ya están inmersos en el ecosistema Apple. En contraste, **C#** cuenta con una vasta cantidad de recursos educativos, tutoriales y documentación oficial, en gran parte debido a su amplia adopción en la industria del software. Con plataformas como **Microsoft Learn** y una comunidad activa en foros y redes sociales, **C#** es mucho más accesible para los principiantes, lo que facilita su aprendizaje y enseñanza. 3. **Entornos de Desarrollo Integrado (IDE) y Herramientas de Soporte**: **Objective-C** depende en gran medida de **Xcode**, un entorno de desarrollo potente pero exclusivo de los sistemas operativos de Apple. Aunque **Xcode** ofrece muchas herramientas útiles, su curva de aprendizaje es más empinada para quienes no están familiarizados con el ecosistema de Apple. En cambio, **C#** es compatible con **Visual Studio**, uno de los IDE más populares y completos. **Visual Studio** no solo ofrece herramientas de depuración, análisis de código y sugerencias contextuales, sino que también es multiplataforma a través de **Visual Studio Code**. La accesibilidad de **Visual Studio** y su diseño amigable facilitan la experiencia de aprendizaje, tanto para estudiantes como para educadores. 4. **Popularidad y Comunidad de Aprendizaje**: Aunque **Objective-C** fue muy popular en su época, su relevancia ha disminuido con el crecimiento de **Swift**. Esto ha llevado a una reducción en la base de usuarios novatos y la disponibilidad de recursos educativos. Además, la comunidad de **Objective-C** es más limitada en comparación con otros lenguajes. Por otro lado, **C#** sigue siendo uno de los lenguajes más populares y cuenta con una comunidad global activa. Con un gran número de desarrolladores, foros, grupos de usuarios y tutoriales en línea, los nuevos programadores pueden fácilmente encontrar apoyo y recursos educativos. Esto hace que **C#** sea una opción más atractiva para aquellos que comienzan a aprender a programar. 5. **Enfoque en la Programación Orientada a Objetos (OOP)**: **Objective-C** implementa programación orientada a objetos, pero su enfoque se ve condicionado por la influencia de **C**. Esto puede complicar la enseñanza de los conceptos fundamentales de OOP, ya que los principiantes deben comprender cómo funcionan los punteros y el despacho dinámico de mensajes, lo que agrega complejidad al proceso. En cambio, **C#** fue diseñado desde el principio para ser un lenguaje orientado a objetos puro, lo que hace que conceptos como clases, objetos, herencia, y polimorfismo sean más fáciles de entender. Su estructura limpia y moderna en cuanto a OOP permite que los estudiantes aprendan estos principios de manera más directa, lo que facilita su enseñanza. | **3.4** | **4.5** |
| 1. **Generalidad** | 1. **Ecosistema de Bibliotecas y Frameworks:** **Objective-C** depende en gran medida de bibliotecas propietarias específicas de Apple, como Cocoa y Cocoa Touch, que son fundamentales para el desarrollo de aplicaciones en plataformas iOS y macOS. Esto limita su capacidad de ser utilizado en otros ecosistemas, ya que estas bibliotecas solo funcionan en plataformas Apple. Aunque existen algunas alternativas de código abierto como GNUstep, estas no tienen el mismo nivel de soporte ni son tan ampliamente adoptadas. En cambio, **C#** se beneficia de un conjunto de bibliotecas y frameworks estandarizados a través de .NET, que son consistentes en diversas plataformas, incluidas Windows, Linux y macOS. Gracias a la estandarización de las bibliotecas, las aplicaciones escritas en C# pueden acceder a una vasta cantidad de recursos que son útiles en una variedad de dominios, desde desarrollo web hasta aplicaciones móviles y empresariales. 2. **Integración con Tecnologías Emergentes:** **Objective-C** es muy eficaz para desarrollar aplicaciones dentro del ecosistema de Apple, pero su capacidad de integración con tecnologías emergentes fuera de este ecosistema es limitada. Por ejemplo, aunque se puede usar para crear aplicaciones móviles, su flexibilidad para integrar nuevas tecnologías como la inteligencia artificial (IA), Big Data, cloud computing, y blockchain es menor en comparación con otros lenguajes. **C#**, en cambio, tiene un fuerte apoyo para la integración con tecnologías emergentes. Gracias a la integración con Azure (plataforma en la nube de Microsoft) y su compatibilidad con herramientas de machine learning (como ML.NET), C# está bien posicionado para adaptarse a los desarrollos más recientes en campos como la IA, análisis de datos y servicios en la nube, lo que lo convierte en una opción muy flexible para diversas aplicaciones tecnológicas. 3. **Apoyo en la Comunidad y Recursos de Aprendizaje**: **Objective-C** tiene una comunidad activa, pero está algo restringida debido a su enfoque limitado en plataformas Apple. Esto puede hacer que encontrar recursos, tutoriales y soporte fuera del contexto de iOS o macOS sea más complicado, limitando su alcance en el aprendizaje y la enseñanza. Además, el hecho de que Objective-C esté cada vez más en desuso, con Apple favoreciendo Swift, puede disminuir aún más la disponibilidad de materiales y recursos de aprendizaje a largo plazo. **C#,** por otro lado, es uno de los lenguajes más populares a nivel mundial y cuenta con una enorme comunidad de desarrolladores. Además, su vinculación con Microsoft asegura una gran cantidad de documentación, tutoriales, foros y soporte a través de plataformas como Stack Overflow, GitHub, y Microsoft Learn, lo que facilita su aprendizaje y uso en una amplia gama de contextos. 4. **Desarrollo de Aplicaciones para la Web:** **Objective-C** está optimizado para el desarrollo de aplicaciones en dispositivos Apple y no se utiliza de manera eficiente para crear aplicaciones web, lo que limita su versatilidad en este tipo de proyectos. Para aplicaciones web, los desarrolladores deben recurrir a otros lenguajes como JavaScript o frameworks basados en Objective-C para adaptarse a este contexto. **C#**, sin embargo, tiene un fuerte enfoque en el desarrollo web gracias a su integración con ASP.NET, un potente framework que permite crear aplicaciones web y servicios backend de manera eficiente. Además, con la expansión de Blazor, C# también ha ganado relevancia en el desarrollo web del lado del cliente, lo que le proporciona una mayor capacidad de adaptación para crear aplicaciones web interactivas y de alto rendimiento. 5. **Flexibilidad en la Programación Móvil:** Si bien **Objective-C** es ideal para el desarrollo de aplicaciones nativas en iOS, su uso fuera del ecosistema de Apple es muy limitado, lo que lo hace menos adecuado para desarrolladores que buscan crear aplicaciones móviles multiplataforma. Aunque Objective-C puede ejecutarse en dispositivos Apple, los desarrolladores que buscan aplicaciones para Android deben aprender otro lenguaje y framework, como Java o Kotlin. Por otro lado, **C#** ofrece una gran ventaja con Xamarin, un framework que permite a los desarrolladores escribir código compartido para aplicaciones móviles en iOS, Android y Windows. Esto hace que C# sea más adecuado para proyectos que requieren desarrollo multiplataforma, lo que mejora la eficiencia y reduce el esfuerzo necesario para crear aplicaciones en diferentes dispositivos móviles. | **3.3** | **4.6** |
|  | **Calificación Promedio** |  |  |

**Evaluaciones de**

**Conclusión**