NOMBRE: Gerard Mauricio Pacheco Sanchez

CORREO: [gerardpacheco06@gmail.com](mailto:gerardpacheco06@gmail.com)

GRUPO: T2

1. INVESTIGACIÓN: DIFERENCIA ENTRE UN COMPILADOR Y UN INTÉRPRETE

2. INVESTIGACIÓN: SOBRE LENGUAJES DE TIPADO FUERTE Y TIPADO DEBIL

## 

## **¿Qué es el compilador?**

Un compilador es un programa de computadora que transforma el código escrito en un lenguaje de programación de alto nivel en código de máquina. Es un programa que traduce el ser humanoreadable código a un lenguaje que entiende el procesador de una computadora (binario 1 y 0 bits). La computadora procesa el código de máquina para realizar las tareas correspondientes.

El compilador debe cumplir con la regla de sintaxis del lenguaje de programación en el que está escrito. Sin embargo, el compilador es sólo un programa y no puede corregir los errores encontrados en ese programa. Entonces, si comete un error, deberá realizar cambios en la sintaxis de su programa. Otrowise, no se compilará.

## **¿Qué es el intérprete?**

Un intérprete es un programa de computadora que convierte cada declaración de programa de alto nivel en código de máquina. Esto incluye código fuente, código precompilado y scripts. Tanto el compilador como los intérpretes hacen el mismo trabajo que es convertir un lenguaje de programación de nivel superior en código de máquina. Sin embargo, un compilador convertirá el código en código de máquina (creará un archivo ejecutable) antes de ejecutar el programa. Los intérpretes convierten el código en código de máquina cuando se ejecuta el programa.

## 

## 

## 

## 

## **Diferencia entre compilador e intérprete**

Aquí hay una diferencia importante entre compilador e intérprete:

| **base de diferencia** | **Compilador** | **Intérprete** |
| --- | --- | --- |
| Pasos de programación | * Crea el programa. * Compile analizará todas las declaraciones del lenguaje para determinar si son correctas. Si es incorrecto, arroja un error. * Si no hay ningún error, el compilador convertirá el código fuente a código de máquina. * Vincula diferentes archivos de código en un programa ejecutable (conocido como exe) * Ejecuta el programa | * Crear el programa * Sin vinculación de archivos ni generación de código de máquina * Declaraciones fuente ejecutadas línea por línea DURANTE la ejecución |
| Advantage | El código del programa ya está traducido a código de máquina. Por tanto, el tiempo de ejecución del código es menor. | Los intérpretes son más fáciles de usar, especialmente para los principiantes. |
| Desventaja | No puedes cambiar el programa sin volver al código fuente. | Los programas interpretados pueden ejecutarse en computadoras que tengan el intérprete correspondiente. |
| Codigo de maquina | Almacene el lenguaje de máquina como código de máquina en el disco | No guardar el código de máquina en absoluto. |
| Tiempo de duración | El código compilado se ejecuta más rápido | El código interpretado se ejecuta más lento |
| Modelo | Se basa en el modelo de carga y enlace de traducción de idiomas. | Se basa en el método de interpretación. |
| Generación de programas | Genera un programa de salida (en forma de exe) que se puede ejecutar independientemente del programa original. | No generar programa de salida. Por eso evalúan el programa fuente en cada momento durante la ejecución. |
| Ejecución | La ejecución del programa es independiente de la compilación. Se realizó solo después de compilar todo el programa de salida. | La ejecución del programa es parte del proceso de interpretación, por lo que se realiza línea por línea. |
| Requisito de memoria | El programa de destino se ejecuta de forma independiente y no requiere el compilador en la memoria. | El intérprete existe en la memoria durante la interpretación. |
| Más adecuado para | Está vinculado a la máquina de destino específica y no se puede transferir. C y C++ son los lenguajes de programación más populares que utilizan un modelo de compilación. | Para entornos web, donde los tiempos de carga son importantes. Debido a que se realiza todo el análisis exhaustivo, las compilaciones toman un tiempo relativamente mayor para compilar incluso el código pequeño que no se puede ejecutar varias veces. En tales casos, los intérpretes son mejores. |
| Optimización de código | El compilador ve el código completo por adelantado. Por lo tanto, realizan muchas optimizaciones que hacen que el código se ejecute más rápido. | Los intérpretes ven el código línea por línea y, por lo tanto, las optimizaciones no son tan sólidas como los compiladores. |
| Escritura dinámica | Es difícil de implementar ya que los compiladores no pueden predecir lo que sucede en el momento del turno. | Los idiomas interpretados admiten escritura dinámica |
| Uso | Es más adecuado para el entorno de producción. | Es el más adecuado para el entorno de programa y desarrollo. |
| Ejecución de errores | El compilador muestra todos los errores y advertencias en el momento de la compilación. Por lo tanto, no puedes ejecutar el programa sin corregir errores. | El intérprete lee una sola declaración y muestra el error, si lo hay. Debe corregir el error para interpretar la siguiente línea. |
| Entrada | Se necesita un programa completo | Se necesita una sola línea de código. |
| Salida | Los cumplidores generan código de máquina intermedio. | El intérprete nunca genera ningún código de máquina intermedio. |
| Errores | Muestra todos los errores después de la compilación, todos al mismo tiempo. | Muestra todos los errores de cada línea uno por uno. |
| Lenguajes de programación pertinentes | C, C++, C#, Scala y Java utilizan Complier. | [PHP](https://www.guru99.com/es/php-tutorials.html), [Perl](https://www.guru99.com/es/perl-tutorials.html), [Rubí](https://www.guru99.com/es/ruby-on-rails-tutorial.html) utiliza un intérprete. |

## **Papel del compilador**

* Los cumplidores leen el código fuente y generan código ejecutable
* Traduce software escrito en un lenguaje de nivel superior en instrucciones que la computadora puede entender. Convierte el texto que escribe un programador a un formato que la CPU pueda entender.
* El proceso de compilación es relativamente complicado. Dedica mucho tiempo a analizar y procesar el programa.
* El resultado ejecutable es algún tipo de código binario específico de la máquina.

## **Papel del intérprete**

* El intérprete convierte el código fuente línea por línea durante el tiempo de ejecución.
* Interpret traduce completamente un programa escrito en un lenguaje de alto nivel a un lenguaje de nivel de máquina.
* El intérprete permite la evaluación y modificación del programa mientras se ejecuta.
* Se dedica relativamente menos tiempo a analizar y procesar el programa.
* La ejecución del programa es relativamente lenta en comparación con el compilador.

### **IDIOMAS DE ALTO NIVEL**

Lenguajes de alto nivel, como [C](https://www.guru99.com/es/c-programming-language.html), C++, JAVA, etc., están muy cerca del inglés. Facilita el proceso de programación. Sin embargo, debe traducirse al lenguaje de máquina antes de su ejecución. Este proceso de traducción lo realiza un compilador o un intérprete. También conocido como código fuente.

### **CODIGO DE MAQUINA**

Los lenguajes de máquina están muy cerca del hardware. Cada computadora tiene su lenguaje de máquina. Los programas en lenguaje de máquina se componen de series de patrones binarios. (Ej. 110110) Representa el simple operaoperaciones que deben ser realizadas por el ordenador. Los programas en lenguaje de máquina son ejecutables, por lo que pueden ejecutarse directamente.

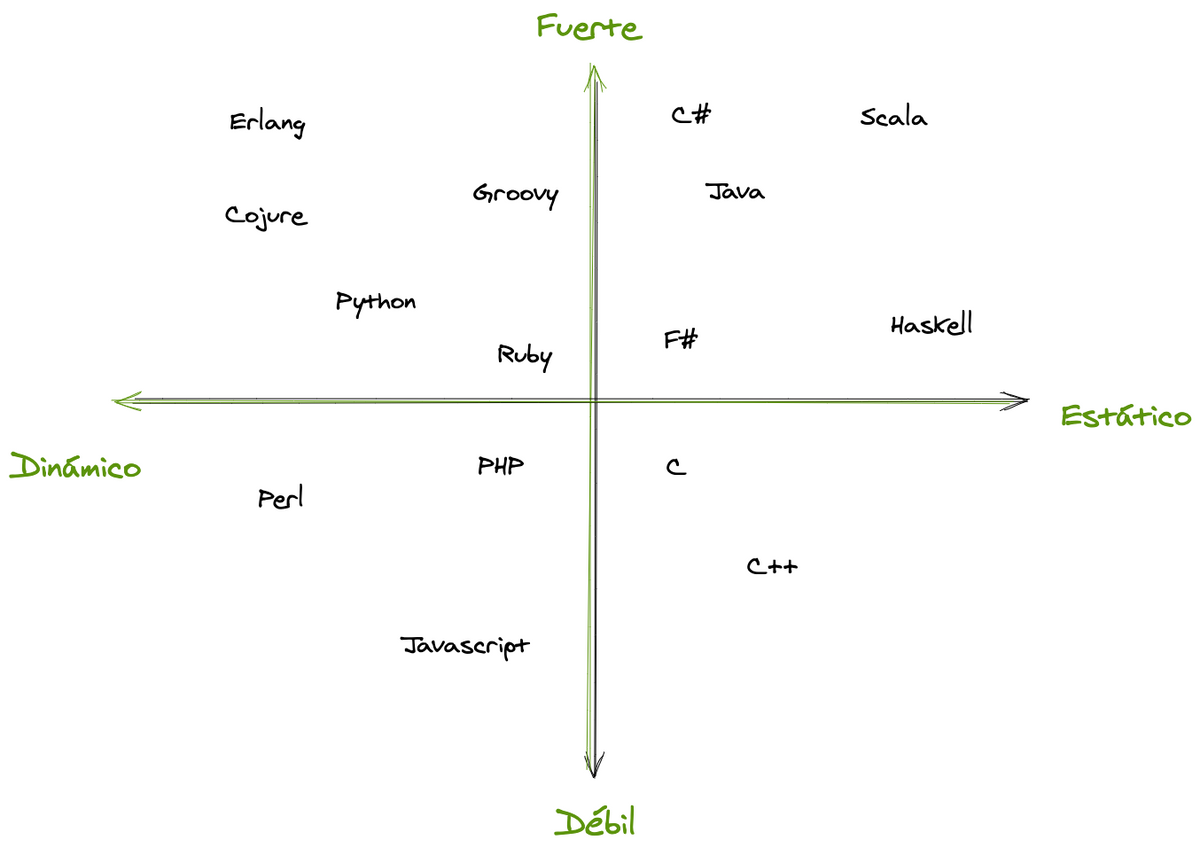
### **CÓDIGO DE OBJETO**

Al compilar el código fuente, el código de máquina generado para diferentes procesadores como Intel, AMD y ARM es diferente. Para que el código sea portátil, el código fuente primero se convierte a código objeto. Es un código intermediario (similar al código de máquina) que ningún procesador entenderá. En tiempo de ejecución, el código objeto se convierte al código de máquina de la plataforma subyacente.

### **Java está compilado e interpretado.**

Para explotar las ventajas relativas de los compiladores, los intérpretes de algunos lenguajes de programación como Java se compilan e interpretan. El código Java en sí se compila en código objeto. En tiempo de ejecución, el [JVM](https://www.guru99.com/es/java-virtual-machine-jvm.html) interpreta el código objeto en código máquina de la computadora de destino.

## **Tipo de tipados**



### **Tipado estático**

## En tipado estático el tipo está ligado a la variable. Los tipos se chequean en tiempo de compilación.

## Esto significa que en un lenguaje de tipado estático, una vez que la variable se declara de un tipo, no se puede cambiar el tipo más adelante; obviamente sí se le puede asignar otros valores del mismo tipo, porque el tipo queda ligado a la variable y no a los valores que ésta toma.

### **Tipado dinámico**

En tipado dinámico, el tipo está ligado al valor. Los chequeos son en tiempo de ejecución.

Como el tipo está ligado al valor, el tipo de la variable puede cambiar a medida que cambian el tipo de los valores que se le asigna.

## **Tipado fuerte**

Es el utilizado para que el código incluya el tipo de dato al declarar la variable. Solemos decir que esta forma de programar es más expresiva al saber qué sucederá exactamente con las órdenes programadas. La ejecución es más veloz, ya que la inferencia de los tipos es anterior y el lenguaje no necesita verificarlos a la hora de ejecutarlos. Se usa en lenguajes de **programación** como **Python, C, C#, Java y Ruby**. Sus desventajas son:

* A la hora de **programar es mucho más estricto** y necesitas escribir más código, lo que alarga el tiempo de trabajo y aumenta la necesidad de verificar lo redactado más exhaustivamente.
* El **compilador debe ejecutar** una disciplina de tipos concreta. Es decir, se siguen unas normas concretas de asignación de variables, argumentos de funciones o valores de retorno.

## **Tipado débil**

## Se usa para indicar el tipo de variable a la hora de declararla. Permite asignarle un valor entero a una variable que con anterioridad tenía una cadena. En programación informática se usa en lenguajes como el JavaScript o PHP. Presenta desventajas como:

## Es muy rápido de desarrollar, pero favorece la aparición de errores.

## Su flexibilidad ayuda a que los programas funcionen sin necesidad de utilizar tanto código. No obstante, presenta errores en el tiempo de ejecución, como el que se pierde al comparar números mezclando algunos tipados como los *strings*, que no son detectables anteriormente.

## Es necesario castear de manera periódica el programa para conseguir el resultado esperado. Ello provoca que tengas que redactar más código para aumentar su seguridad.

## **Diferencias entre ambos tipados**

Tras leer la información anterior, quizá tengas que ejecutar un proyecto en **IT** y no sepas muy bien qué **tecnología** elegir. Repasa las diferencias entre ambas opciones:

* El **tipado débil** te permite trabajar en menos tiempo y comprobar cuáles son los resultados del programa en tiempo real. Para la fase de ideación del proyecto esta opción es ideal. Se adapta muy bien a programas de menor capacidad.
* El **tipado fuerte** admite realizar operaciones con distintos tipos de variable. La consecuencia directa es que no vas a cometer tantos errores como con el lenguaje del punto anterior. Ahora bien, necesitas escribir más código. Este lenguaje es ideal para proyectos de mayor alcance y con un número mayor de especificaciones.

**¿Son incompatibles ambas opciones?** No. De hecho, lo habitual es que comiences cualquier proyecto con un **tipado** débil. Cuando el cliente lo acepta es el momento de pasarlo a un lenguaje de **tipado** fuerte, ya que ofrece un código más fácil de mantener y aumenta su escalabilidad. Igualmente, dependiendo de cada proyecto tendrás que elegir el que prefieras.

Si nos preguntas, te diríamos que el más recomendable es el **tipado** fuerte. Te exige castear o comprobar el tipo de cada variable. Al menos, tienes funciones diversas que aceleran este proceso. **Lenguajes como PHP** están incorporando progresivamente el **tipado** fuerte para la declaración de funciones.

Pero cada programador tiene sus gustos y nadie sabe mejor que un profesional qué alternativa se adapta mejor a sus necesidades. Esperamos haberte aclarado las diferencias y las características de cada **tipado.** Seguro que ya tienes más claro que nunca cuál debes elegir cuando tengas que ponerte a programar a partir de ahora. Repasa la información anterior, haz las pruebas pertinentes y compara los resultados obtenidos o a cuál de las dos opciones te adaptas mejor.