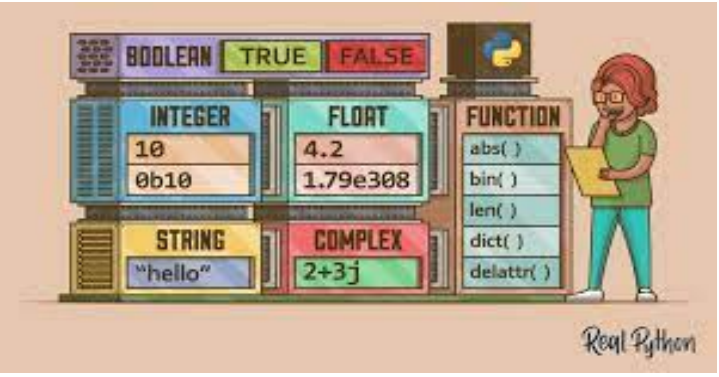
# SISTEMA DE TIPOS DE DATOS EN LENGUAJES DE PROGRAMACIÓN




La tipificación en programación se refiere a cómo se manejan los tipos de datos en un lenguaje de programación. A grandes rasgos existen 4 tipos de datos:

1) Número entero (1, 2, 5, -1, 100) - Tipo INT
2) Número real (1.434, 55.4333, 3.1453, etc) - Tipo FLOAT
3) Caracteres ("Hola", "123", "1.323", "$%%@", etc) (notese que van entre comillas) Tipo STRING
4) Booleanos (verdadero/falso) Tipo Bool

En un ordenador, cualquier valor simplemente consiste en un conjunto de bits, el hardware no hace distinción entre caracteres, números enteros, números en coma flotante y datos booleanos. Asignar tipos de datos (tipificar) da significado a las colecciones de bits. Los tipos de datos informan a los programas y programadores cómo deben ser tratados esos bits. Es decir que el procesador (cpu) necesita saber que tipo de dato es para saber cómo tratar esa información almacenada en bits. Si es un número entero lo procesa distinto a si es un carácter o un dato booleano.

**Los sistemas de tipificación varían significativamente entre lenguajes. Existen 2 grandes divisiones:**

1) **lenguajes estáticos o dinámicos.**

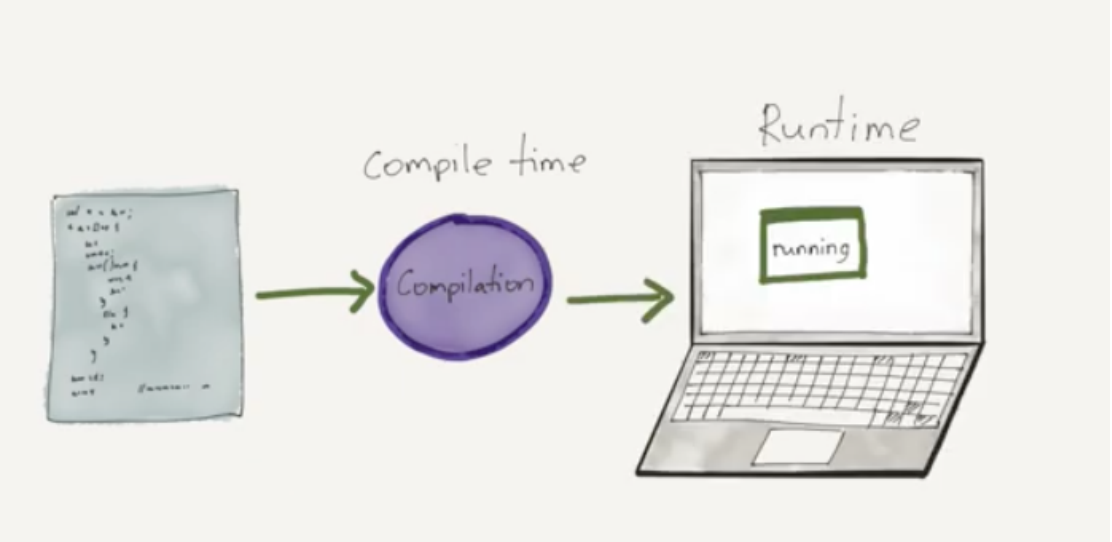
2) **Lenguajes fuertemente tipados o débilmente tipados**

## LENGUAJES ESTÁTICOS O DINÁMICOS:

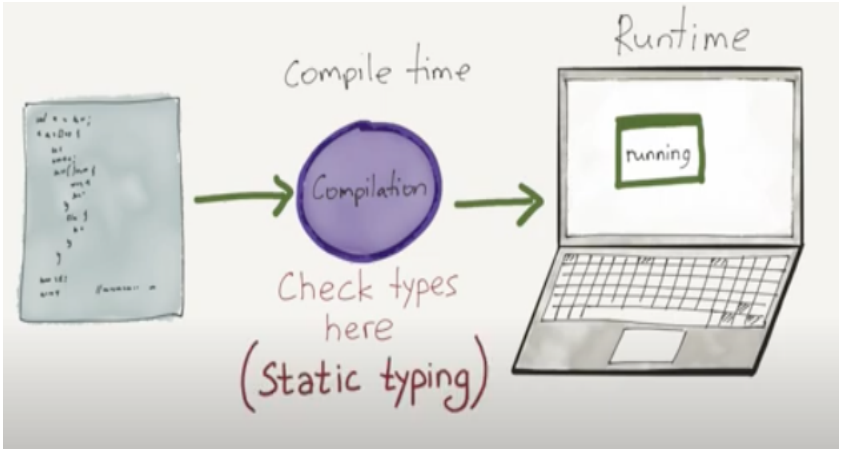
En la clase pasada vimos como los lenguajes pasan de alto a bajo nivel mediante un proceso de compilación o interpretación, para que los ordenadores puedan comprender las instrucciones que le brindan los programadores.

Mediante el proceso de compilación el código fuente que entiende el programador es traducido a lenguaje máquina (que entiende el ordenador). El tiempo de compilación se denomina Compile-time. Una vez que el proceso de compilación terminó el programa se ejecuta; el tiempo en el que el programa se está ejecutando se denomina Runtime.

Según cuando se verifique la corrección de los tipos de datos, ya sea en tiempo de compilación (Compile-time), o posteriormente, en tiempo de ejecución (Runtime) el lenguaje será estáticamente tipado o dinámicamente tipado.




### Tipado estático

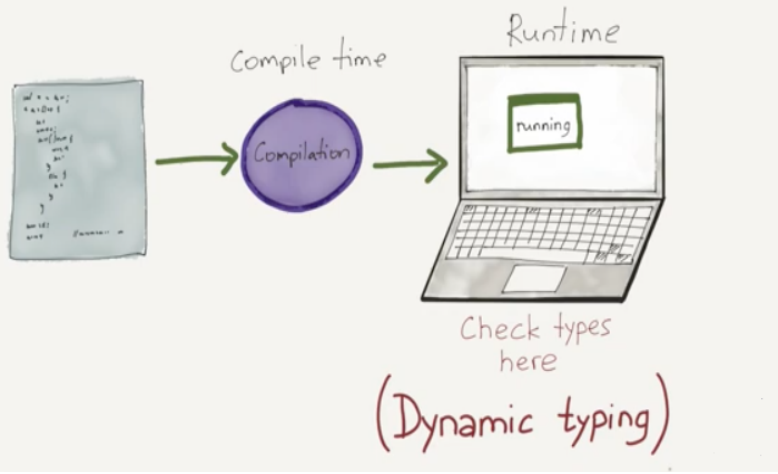



Los lenguajes que chequean los tipos de datos y se fijan en errores de tipificación en el tiempo de compilación son de tipo estático. El compilador analiza el código fuente y comprueba que todas las operaciones y asignaciones de variables se realicen con tipos de datos compatibles. Por lo tanto la verificación de tipos se realiza antes de que el programa sea ejecutado (Runtime).

Si se detecta algún error de tipo durante la verificación de tipos, el compilador genera un mensaje de error y el programa no se compila. En los lenguajes estáticamente tipados se debe definir el tipo de variable antes de utilizarla, y en algunos casos es necesario realizar un casting explícito si se quiere cambiar el tipo de variable. Esto se debe a que es necesario informarle al compilador el tipo de dato para que este pueda realizar la verificación por tipos. Como veremos más adelante, esto trae algunas ventajas y/o desventajas según como se lo mire.

Ejemplos de lenguajes que usan tipado estático son C, C + +, Java y Haskell. Comparado con el tipado dinámico, el estático permite que los errores de tipificación sean detectados antes, y que la ejecución del programa sea más eficiente y segura.

## Tipado dinámico

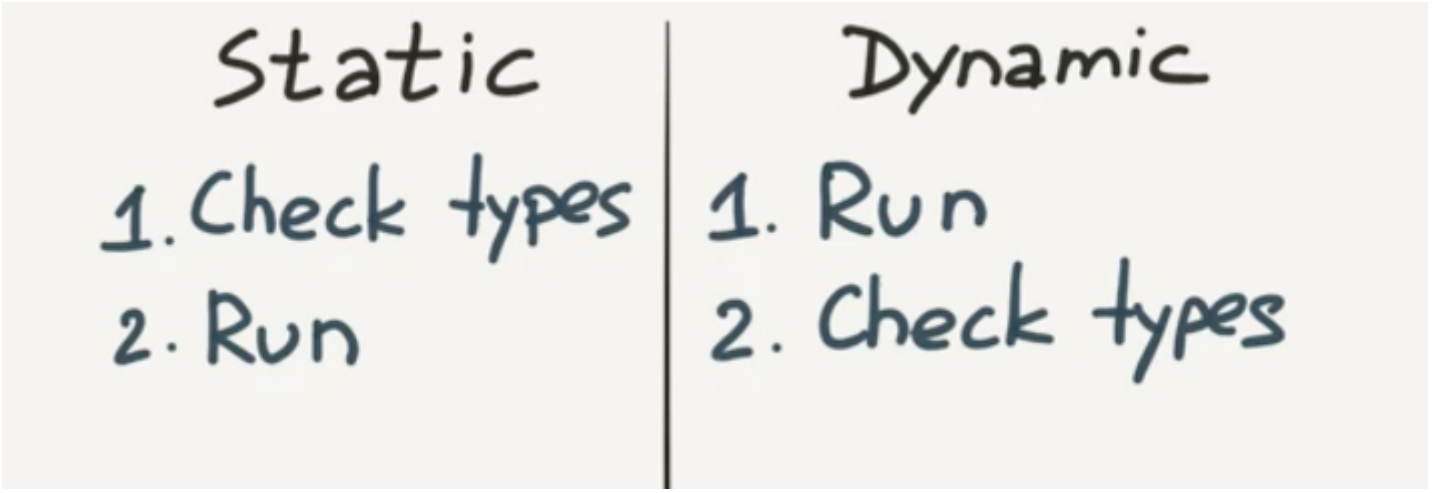



**En los lenguajes de tipado dinámico**, el chequeo de tipos se realiza en tiempo de ejecución (Runtime). Esto significa que el tipo de una variable se determina en el momento en que se le asigna un valor y se verifica su tipo durante la ejecución del programa.

Si una operación se realiza con dos valores de tipos incompatibles, el programa puede generar un error en tiempo de ejecución, lo que puede llevar a una excepción o un comportamiento no deseado. Por ejemplo, si intenta sumar una cadena de caracteres con un número, Python, lenguaje dinámicamente tipado, generará un error en el tiempo de ejecución.

En un lenguaje de programación dinámicamente tipado el tipo de una variable se determina automáticamente en tiempo de ejecución en función del valor que se le asigna a la variable, y no es necesario definir el tipo de la variable explícitamente. El ordenador analiza la variable y en función al dato que tiene le asigna implícitamente un tipo. Esto significa que se puede cambiar el tipo de una variable en cualquier momento simplemente asignándole un valor de un tipo diferente.

Ejemplos de lenguajes que usan tipado dinámico son Perl, Python y Lisp.




**Algunas ventajas y desventajas:**

El tipado estático busca errores en los tipos de datos durante la compilación. Esto debería incrementar la fiabilidad de los programas procesados. Sin embargo, los programadores, normalmente, están en desacuerdo en cómo los errores de tipos de datos más comunes ocurren, y en qué proporción de estos errores que se han escrito podrían haberse cazado con un tipado estático. El tipado estático aboga por la creencia de que los programas son más fiables cuando son chequeados sus tipos de datos, mientras que el tipado dinámico apunta al código distribuido que se ha probado que es fiable y es menos verboso (menor cantidad de código ya que no es necesario definir el tipo de cada variable).

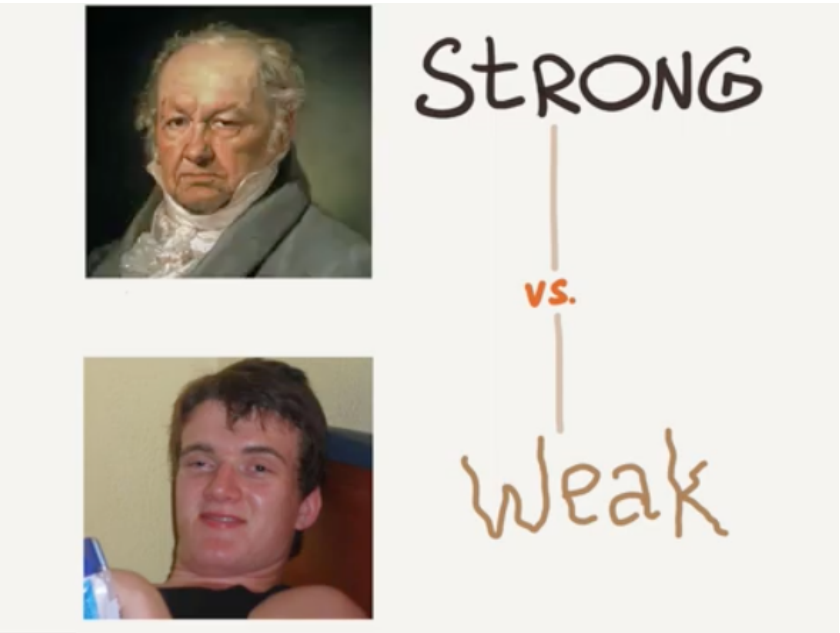
El tipado estático resulta, normalmente, en un código compilado que se ejecuta más rápidamente. Cuando el compilador conoce los tipos de datos exactos que están en uso, puede producir código máquina optimizado. Además, los compiladores en los lenguajes de tipo estático pueden encontrar atajos más fácilmente. Algunos lenguajes de tipificación dinámica como el lisp permiten declaraciones de tipos de datos opcionales para la optimización por esta misma razón. El tipado estático generaliza este uso.

En contraste, el tipado dinámico es más flexible y permite a los compiladores ejecutarse más rápidamente, debido a que los cambios en el código fuente en los lenguajes dinámicamente tipados puede resultar en menores comprobaciones y menos código que revisar. Esto también reduce el ciclo editar - compilar - comprobar - depurar.

Hay frecuentemente conflictos entre aquellos que prefieren la tipificación fuerte y/o estática y aquellos que se inclinan por la tipificación dinámica, libre o débil. El

primer grupo aboga por la detección temprana de errores durante la compilación y el aumento de rendimiento en tiempo de ejecución, mientras que el segundo grupo aboga por los prototipos rápidos que son posibles con un sistema de tipificación dinámica.

## FUERTEMENTE TIPADO O DÉBILMENTE TIPADO:




Los tipos estáticos o dinámicos hacen referencia a CUANDO se chequea congruencia de datos, mientras que fuerte o débilmente tipado hace referencia a cuán riguroso será ese chequeo en la congruencia de datos.

Un **lenguaje fuertemente tipado** no permite operaciones o asignaciones entre tipos de datos incompatibles sin que se realice una conversión explícita de tipos. En otras palabras, en un lenguaje fuertemente tipado, el compilador o intérprete hace cumplir las reglas de tipos de manera estricta y no permite que los tipos de datos sean convertidos de forma implícita o automática. Esto ayuda a prevenir errores comunes en tiempo de ejecución y mejora la seguridad y robustez del código.
Algunos lenguajes fuertemente tipados son: Java, C#, Python, Ruby, Swift, Kotlin, TypeScript, Rust

Un l**enguaje débilmente tipado** es aquel en el que la conversión de tipos puede ocurrir de manera implícita, es decir, el lenguaje puede convertir automáticamente un tipo de dato a otro sin que se requiera una conversión explícita por parte del programador. Esto puede llevar a resultados impredecibles y a errores de programación difíciles de detectar. Por lo tanto, en un lenguaje débilmente tipado,

el programador debe tener cuidado de asegurarse de que los tipos de datos sean compatibles en todas las operaciones que realice en su programa.
Algunos lenguajes débilmente tipados son: JavaScript, PHP, Perl, Lua, VBScript, Tcl

Ejemplo en javascript: