***Aplicando lo Aprendido III***

***Alumno: Gimenez Thomas Valentin***

1. Generalización simbólica: ¿Cuáles son las reglas escritas del lenguaje?

**Prototipos y Herencia**:

En JavaScript, los objetos pueden heredar propiedades y métodos directamente de otros objetos. Cada objeto tiene un enlace interno a otro objeto llamado su "prototipo". Cuando intentas acceder a una propiedad de un objeto que no existe en ese objeto, JavaScript busca la propiedad en el prototipo del objeto. Este proceso se repite a lo largo de una cadena de prototipos, creando una estructura de herencia basada en prototipos.

**Prototipado Dinámico:**

En JavaScript, se puede agregar, modificar o eliminar propiedades y métodos de los prototipos en tiempo de ejecución. Esto proporciona una flexibilidad dinámica que no es tan común en lenguajes con sistemas de clases estáticas.

1. Creencias de los profesionales: ¿Qué características particulares del lenguaje se cree que sean "mejores" que en otros lenguajes?

**Prototipado Dinámico y Flexibilidad:**

Los profesionales aprecian la capacidad de modificar objetos en tiempo de ejecución, lo que facilita la creación y adaptación dinámica de objetos en aplicaciones web. Esta flexibilidad es especialmente útil en contextos donde las estructuras de datos pueden cambiar dinámicamente.

**Herencia Flexible:**

A diferencia de la rigidez de la herencia basada en clases, el prototipado en JavaScript permite una herencia más flexible. Los objetos pueden heredar directamente de otros objetos, y no hay una distinción estricta entre clases y objetos. Esto permite un diseño más adaptable a las necesidades cambiantes del código.

**Programación del Lado del Cliente:**

JavaScript es el lenguaje principal para la programación del lado del cliente en el desarrollo web. Su capacidad para manipular el DOM (Document Object Model) y responder a eventos del usuario ha llevado a su adopción generalizada en el desarrollo de interfaces de usuario interactivas.

**Desafíos y Críticas**:

Aunque el prototipado dinámico y la flexibilidad son ventajas, también pueden llevar a desafíos en la gestión de código a medida que los proyectos crecen. La falta de estructuras más rígidas, como las clases en otros lenguajes, ha llevado a algunas críticas sobre la legibilidad y mantenibilidad del código en proyectos grandes y complejos.

4. En mi código utilice los siguientes Principios de los Paradigmas Orientados a Objetos basados en Prototipos

**Encapsulamiento:**

Fundamentación:

El encapsulamiento permite ocultar detalles de implementación y proteger ciertas partes del código, proporcionando así una interfaz más limpia y segura. En este caso, se asegura de que la función Tarea sus propiedades internas no sean modificadas accidentalmente desde fuera.

Implementación:

El encapsulamiento aparece a través de la definición de variables privadas dentro del constructor de Tarea. Las propiedades como nombrePrivado, estadoPrivado, etc., solo son accesibles mediante métodos getters y setters, lo que protege la información y proporciona control sobre cómo se puede acceder y modificar.



Ejemplo de encapsulamiento donde obtenemos las variables privadas del nombre y estado de la tarea a través de getters y setters.

**Modularidad:**

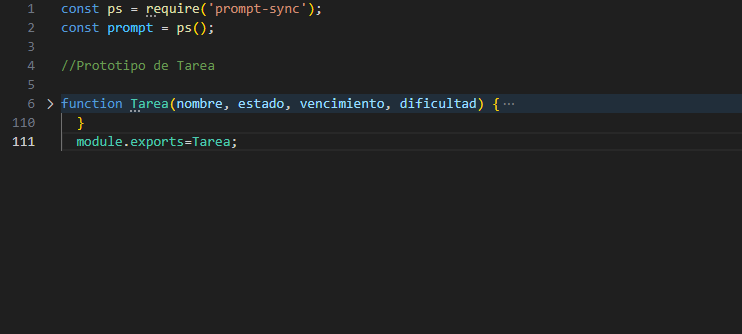
El modularidad ayuda a estructurar y organizar el código, facilitando el mantenimiento y la escalabilidad. Cada módulo tiene una responsabilidad específica lo que mejora la legibilidad y la reutilización del código.

Implementación:

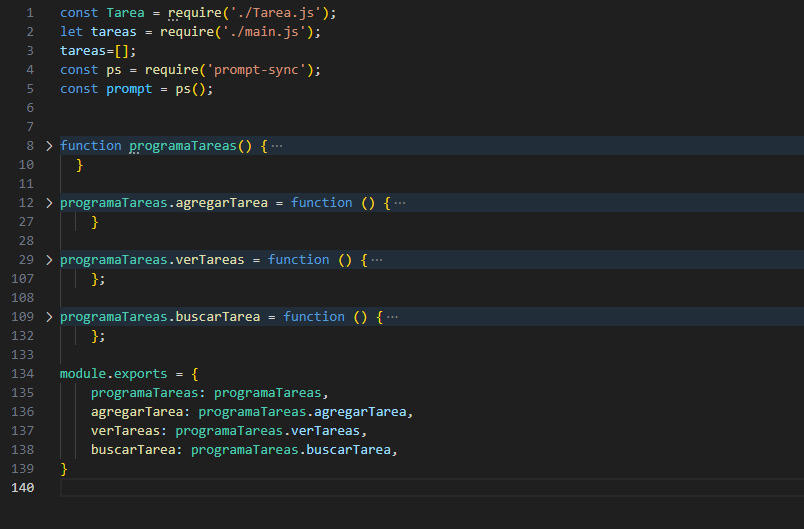
Organice el código en tres archivos, main.js donde se ejecuta el código, Tarea.js donde generamos el prototipo “padre” de las demás tareas a generar, y programaTareas.js donde se ven todos los comportamientos que van a haber en las tareas generadas en base al prototipo de Tarea.js



Main.js



Tarea.js



programaTareas.js

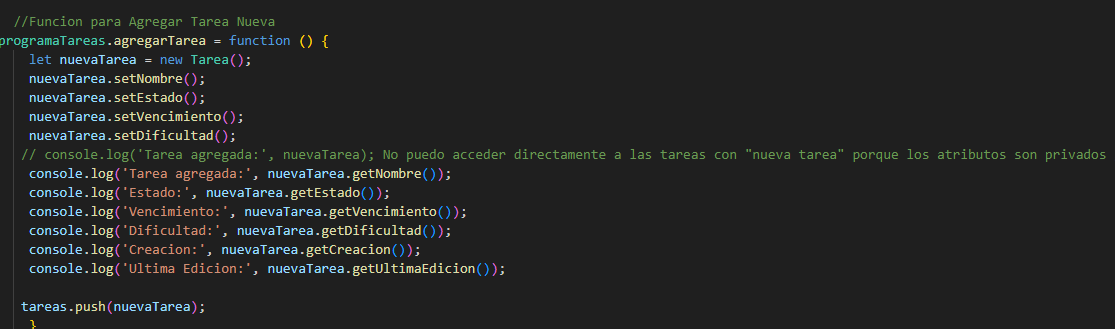
**Abstracción:**

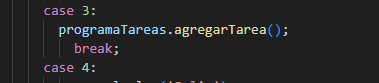
Fundamentación:

La abstracción se refiere a la simplificación de la realidad compleja a través de modelos. En este caso, la abstracción de una tarea como objeto y la manipulación de tareas a través de funciones abstractas proporciona una forma más clara de interactuar con el sistema de gestión de tareas.

Implementación:

Los detalles específicos de cómo se implementa la adición de tareas están ocultos detrás de una interfaz simple y fácil de usar. Los usuarios del código solo necesitan saber que agregarTarea agrega una nueva tarea, sin preocuparse por los pasos internos.





No pongo Herencia y Polimorfismo, si bien la herencia se aplica de manera natural en el programa ya que se trabaja con prototipos, no la considero como un principio del todo aplicable en este caso, ya que no utilicé herencia para una relación clara de "es un/a" entre diferentes tipos de tareas que justificara la creación de una jerarquía de prototipos extensa.

El polimorfismo implica en la habilidad de un método, variable u objeto de poseer varias formas distintas, En este caso, aunque hay diferentes tipos de tareas, no hay necesidad inmediata de tratarlas de la misma manera en todos los contextos.