**Informe POO (Programación Orientada a Objetos) JavaScript**

**Julian Dario Triana Mosquera**

**Análisis y Desarrollo de Software (ADSO)**

**Ficha:2558104 G1\_G2**

**Instructora Isaura Maria Suarez Novoa**

**Servicio Nacional de Aprendizaje (SENA), CEET**

**Bogotá D.C.**

**23 de agosto de 2023**

Tabla de Contenido:

Introducción

Que son Prototipos y Herencia de Prototipos en JavaScript.

POO

Expresiones Regulares

Módulos: Import y Export

SetTimeOut y SetInterval

Operaciones Sincrónicas y Asincrónicas

Callbacks y Promesas

Sets y Maps

Respuesta Ejercicios JavaScript

Referencias

**Introducción**

En el presente informe, se abordarán varios temas esenciales en el ámbito de la programación en JavaScript. Se explorarán conceptos relacionados con la naturaleza de prototipos y herencia de prototipos en JavaScript, se discutirá la Programación Orientada a Objetos (POO), expresiones regulares, módulos (import y export), funciones asincrónicas, manejo de tiempo mediante setTimeout y setInterval, operaciones sincrónicas y asincrónicas, así como conceptos de callbacks y promesas, y la utilidad de los sets y maps en JavaScript.

**Prototipos y Herencia de Prototipos en JavaScript**

JavaScript es un lenguaje de programación que utiliza un sistema de prototipos para implementar la herencia entre objetos. Según (*Prototipos de objetos - Aprende Desarrollo Web | MDN*, 2023), los prototipos son objetos que sirven como modelo para la creación de otros objetos. Cada objeto tiene un enlace interno a otro objeto (su prototipo) desde el cual hereda propiedades y métodos.

Krall (s. f.) explica que, en JavaScript, la concepción de la herencia mediante prototipos consiste en establecer un objeto central, identificado como el "padre" o prototipo, el cual alberga la totalidad de los datos compartidos por todos los objetos del mismo tipo, denominados "hijos". Este enfoque tiene como objetivo prevenir la duplicación de propiedades y métodos comunes en cada objeto individual, resultando en una optimización del uso de memoria y una mayor agilidad en la ejecución del código.

**Programación Orientada a Objetos (POO)**

La programación orientada a objetos, como indica Vanegas (s. f.), se basa en que es posible representar el mundo como entidades relacionadas entre sí; por lo tanto, la programación se simplifica al identificarlas y reproducir sus atributos y acciones apropiadamente. El propósito esencial del diseño es lograr que cada componente del mundo real se asocie directamente con una entidad en el mundo computacional, que la modela.

Siguiendo la línea de ideas de Canelo (2023) y Vanegas (s. F.), queda claro que su estructura se compone de una serie de interconexiones entre los objetos, que permiten la comunicación y la colaboración entre ellos. Cada objeto posee atributos que describen su estado y métodos que determinan su comportamiento. Estos métodos pueden ser invocados por otros objetos, lo que facilita la interacción y el intercambio de información entre las distintas partes del sistema.

La idea fundamental detrás de esta estructura es lograr un diseño modular y reutilizable. Al dividir el sistema en objetos individuales, se puede abordar cada componente de manera independiente, lo que facilita la gestión y el mantenimiento del código. Además, la capacidad de heredar propiedades y comportamientos de objetos preexistentes a través de la herencia de prototipos añade flexibilidad al diseño y promueve la eficiencia en el desarrollo.

En JavaScript, las clases son funciones constructoras que se utilizan para crear objetos. Se pueden definir propiedades y métodos en las clases y luego crear instancias de esas clases utilizando la palabra clave new.

**Expresiones Regulares**

Según "¿Qué es una expresión regular? - Javascript en español" (s. f.), las expresiones regulares, también conocidas como RegExp o RegEx, constituyen un sistema que facilita la búsqueda, captura y reemplazo de texto mediante patrones. Estos patrones simplifican las búsquedas de texto al permitir una abstracción efectiva y amplia, que abarca múltiples posibilidades que, de otra manera, serían extensas y complicadas.

Los patrones se expresan como cadenas de texto en las cuales ciertos símbolos poseen significados especiales. Para comprender el significado de estos símbolos especiales, se ilustrará a través de un ejercicio práctico.

El objetivo es determinar si un nombre inicia con las letras "p" o "s" y finaliza con las letras "o" o "a". Es decir, se busca verificar si se cumplen dos condiciones, cada una con dos opciones posibles.

**Módulos: Import y Export**

En el contexto de JavaScript, es posible realizar la acción contraria a la exportación mediante la utilización del comando "import" ("Importar módulos - Javascript en español," s. f.). Si previamente se destacó que mediante "export" se ponen a disposición de otros datos o elementos provenientes de un archivo .js, el uso de "import" permite cargar y emplear estos elementos en el código de nuestro archivo actual.

Según "Exportar módulos - Javascript en español" (s. f.), en un archivo JavaScript, se carece de un módulo de exportación por defecto si no se utiliza al menos una vez la instrucción "export" en su código. Sin embargo, cuando se emplea dicho "export" al menos una vez, se genera un objeto denominado "módulo de exportación", el cual puede contener uno o varios conjuntos de datos. Existen diversas maneras de exportar datos a través de la palabra clave "export" en JavaScript.

**Manejo de Tiempo: setTimeout y setInterval**

Las funciones setTimeout y setInterval son métodos proporcionados por el entorno de ejecución de JavaScript para controlar el tiempo. setTimeout permite ejecutar una función después de un cierto período de tiempo, mientras que setInterval permite ejecutar una función en intervalos regulares.

**Operaciones Sincrónicas y Asincrónicas**

En su obra, Padilla (2022) explora el proceso de ejecución de funciones en JavaScript y el concepto de asincronía en este contexto.

El motor de JavaScript gestiona la ejecución de funciones a través de una pila de ejecución denominada "function execution stack". Cuando se invoca una función, se agrega a esta pila, y si la función invocada llama a otra función, esta última se añade y ejecuta en la pila. Una vez finalizada su ejecución, se retira de la pila, y se continúa con la ejecución de la función previa desde donde se detuvo. Este proceso se repite hasta que la pila queda vacía.

La asincronía en JavaScript se refiere a la ejecución que no ocurre simultáneamente. Se presenta cuando se necesita obtener datos del servidor o retardar la ejecución de una función, situaciones que no necesitan ocurrir inmediatamente. En estos casos, el motor de JavaScript debe manejar la ejecución de manera eficiente para que otras tareas secuenciales no se detengan.

Las operaciones asíncronas en JavaScript se pueden clasificar en dos categorías principales: eventos o funciones Browser API/Web API, como setTimeout o eventos de usuario, y promesas, un concepto único que permite realizar operaciones asíncronas de manera estructurada y controlada.

**Callbacks y Promesas**

Según Ortega (2020), los callbacks representan una de las técnicas más habituales para gestionar la asincronía en JavaScript. Esta técnica se basa en los siguientes pasos:

1. Se crea una función A que toma como uno de sus argumentos otra función B, conocida comúnmente como callback.

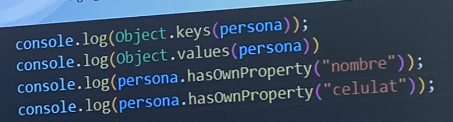
2. El callback recibe dos argumentos: el primero se refiere a un error, que puede ser personalizado o generado por la propia aplicación, y el segundo contiene el resultado proveniente de la función A que hace uso del callback.

Esta aproximación permite manejar eficazmente la ejecución asíncrona en JavaScript. Además, las promesas también son una herramienta importante para gestionar la asincronía en JavaScript. Estas ofrecen una estructura más estructurada y legible para tratar operaciones asíncronas, permitiendo manejar tanto el éxito como el fracaso de manera más organizada.

**Sets y Maps**

Los sets son colecciones de valores únicos en JavaScript, lo que significa que no pueden contener duplicados. Los maps son colecciones que permiten almacenar pares clave-valor, donde cada clave es única y se utiliza para acceder a su valor correspondiente de manera eficiente.

**Ejercicios JavaScript:**



1. `console.log (Object.keys(persona)); `

Enseña en la consola las propiedades que tiene el objeto `persona`. Es como si estuviéramos mostrando los nombres de las cosas que el objeto `persona` tiene guardadas.

2. `console.log (Object.values(persona)); `

Muestra en la consola los valores que están almacenados en las propiedades del objeto `persona`. Esto es como mostrar lo que hay dentro de cada compartimento del objeto.

3. `console.log(persona.hasOwnProperty("nombre")); `

Comprueba si el objeto `persona` tiene una propiedad llamada "nombre". Luego, nos dice si encontró o no esa propiedad. Es como si estuviéramos buscando en una lista de características de la persona para ver si hay algo llamado "nombre".

4. `console.log(persona.hasOwnProperty("celulat")); `

Intenta averiguar si el objeto `persona` tiene una propiedad que se llama "celulat". Después, muestra si el objeto tiene o no esa propiedad. Es como revisar si la persona tiene una característica que suena rara, como "celulat" en lugar de "celular".

**Conclusión**

En este informe, se ha explorado una serie de conceptos fundamentales en el lenguaje de programación JavaScript. Desde la implementación de prototipos y herencia hasta la utilización de promesas para gestionar operaciones asincrónicas, estos temas son esenciales para cualquier desarrollador que desee aprovechar al máximo la versatilidad y potencia de JavaScript en sus proyectos. La comprensión de estos conceptos sienta las bases para escribir código eficiente y mantenible en este lenguaje de programación.

**Referencias:**

*Prototipos de objetos - Aprende Desarrollo Web | MDN*. (2023, 22 julio). <https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/Object_prototypes>

Krall, C. (s. f.). *Prototype JavaScript. Ejemplos de prototipos y herencia. Cómo usarlos con propiedades y métodos (CU01147E)*. <https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=828:prototype-javascript-ejemplos-de-prototipos-y-herencia-como-usarlos-con-propiedades-y-metodos-cu01147e&catid=78&Itemid=206>

Vanegas, C. A. (s. f.). *Vista de la programación orientada a objetos frente a la programación estructurada*. <https://revistas.udistrital.edu.co/index.php/Tecnura/article/view/6096/7620>

Canelo, M. M. (2023). ¿Qué es la programación orientada a objetos? *Profile Software Services*. <https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/>

*¿Qué es una expresión regular? - Javascript en español*. (s. f.). Lenguaje JS. <https://lenguajejs.com/javascript/regexp/expresiones-regulares/>

*Importar módulos - Javascript en español*. (s. f.). Lenguaje JS. <https://lenguajejs.com/javascript/modulos/import/>

*Exportar módulos - Javascript en español*. (s. f.). Lenguaje JS. <https://lenguajejs.com/javascript/modulos/export/>

Padilla, C. (2022). JavaScript sincronía vs asincronía - Call stack, promesas y más. *freeCodeCamp.org*. <https://www.freecodecamp.org/espanol/news/sincrono-vs-asincrono-en-javascript/>

Ortega, J. M. (2020, 22 abril). Callbacks, promesas y Async/Await, ¡Que alguien me explique! *Medium*. <https://medium.com/@jmz12/callbacks-promesas-y-async-await-que-alguien-me-explique-514137cb57e2>