**SERVICIO NACIONAL DE APRENDIZAJE SENA**

****

**PROGRAMA:**

**Análisis y Desarrollo de Software**

**FICHA:**

**3172879**

**INSTRUCTOR:**

**Fernando Betancour**

**APRENDIZ:**

**MARIA CAMILA PALECHOR**

**TITULO:**

**JavaScript**

**SERVICIO NACIONAL DE APRENDIZAJE SENA**

**POPAYAN – CAUCA**

**2025**

**Taller**

**1. Temas de consulta para afianzar conocimientos**

1. ¿Qué significa que JavaScript sea interpretado y tipado dinámico?
2. Diferencias entre var, let y const.
3. Operadores aritméticos y de comparación.
4. Tipos de datos primitivos y objetos.
5. Funciones: declaración, parámetros, retorno y scope.
6. Arrays y métodos básicos (push(), pop(), length, etc.).
7. Objetos literales y la palabra clave this.
8. ¿Por qué es mejor usar let y const en lugar de var?
9. ¿Qué diferencia hay entre == y ===?
10. ¿Qué sucede si intentas reasignar una variable declarada con const?
11. ¿Qué ventajas ofrece usar funciones en tu código?
12. ¿Qué tipo de dato es un arreglo en JavaScript?

Respuestas

* 1. Que **JavaScript sea interpretado** significa que el código fuente no se traduce completamente a código máquina antes de la ejecución. En su lugar, el código es **leído y ejecutado línea por línea** por un programa llamado intérprete (como el motor V8 en Chrome). Esto hace que el ciclo de desarrollo sea rápido, ya que puedes ejecutar el código inmediatamente. Que sea de **tipado dinámico** significa que el tipo de una variable **se verifica en tiempo de ejecución**, no en tiempo de compilación. Lo más importante es que una variable puede **cambiar su tipo** a lo largo del programa. Por ejemplo, la misma variable puede almacenar primero un número (let x = 10;) y luego una cadena de texto (x = "hola";).
  2. **Var:** Ámbito de función o global. No tiene ámbito de bloque ({}). Se puede reasignar, se puede redeclarar en el mismo ámbito, opcional. Se inicializa a undefined si no se asigna.

**Let:** Ámbito de **bloque** ({}), se puede reasignar, no se puede redeclarar, Opcional. Se inicializa a undefined si no se asigna.

**Const:** Ámbito de **bloque** ({}), **no** se puede reasignar, **no** se puede redeclarar y obligatoria debe ser inicializada al declararse.

* 1. **Aritméticos**+, -, \*, /, % (módulo o resto), \*\* (exponenciación), ++ (incremento), -- (decremento). Realizan operaciones matemáticas.

==, ===, !=, !==, >, <, >=, <=. Comparan dos valores y devuelven un valor booleano (true o false).

* 1. Los **tipos de datos primitivos** representan valores simples y **son inmutables** (no se pueden modificar, solo reasignar la variable). JavaScript tiene siete:

1. **string** (cadenas de texto).
2. **number** (números, incluidos enteros y flotantes).
3. **boolean** (true o false).
4. **null** (ausencia intencional de valor).
5. **undefined** (variable declarada pero sin valor asignado).
6. **symbol** y **bigint**.

Los **objetos** son colecciones de propiedades (pares clave-valor) y son **mutables**. A diferencia de los primitivos, los objetos se pasan por **referencia**. Ejemplos de objetos son los **objetos literales**, las **funciones** y los **Arrays**.

* 1. Una función es un bloque de código diseñado para realizar una tarea particular.

**Declaración:** Es la forma estándar de definir una función (ej: function nombre() {}).

**Parámetros:** Son las variables que la función espera recibir al ser invocada.

**Retorno:** La palabra clave **return** se utiliza para devolver un valor de la función. Si no se usa o se usa sin un valor, la función devuelve implícitamente **undefined**.

**Scope (Ámbito):** Cada función crea su propio **ámbito local**. Las variables declaradas dentro de una función solo son accesibles dentro de ella, proporcionando aislamiento y evitando conflictos.

* 1. Un **Array (arreglo)** es un tipo de **objeto** utilizado para almacenar una **colección ordenada** de elementos. Los elementos pueden ser de cualquier tipo de dato.

Algunos **métodos básicos** y propiedades son:

**length:** Propiedad que indica cuántos elementos tiene el array.

**push():** Agrega un elemento **al final** del array.

**pop():** **Elimina el último** elemento del array y lo devuelve.

**shift():** Elimina el **primer** elemento.

**unshift():** Agrega un elemento **al inicio**.

* 1. Un **objeto literal** es la forma más sencilla de crear un objeto en JavaScript utilizando llaves ({}) para definir propiedades y métodos (ej: const persona = { nombre: "Juan", edad: 30 };). La palabra clave **this** es fundamental y su valor **depende del contexto** en el que se ejecuta. En la mayoría de los casos dentro de una función que es un **método de un objeto**, this hace referencia a **ese objeto** que contiene el método. Ayuda a acceder a las propiedades del objeto desde dentro de él mismo.
  2. Es mejor usar let y const porque ofrecen **ámbito de bloque**. Esto previene problemas comunes asociados con var, como la reescritura involuntaria de variables fuera de un bucle o condicional. Usar **const** para variables que no cambian y **let** para las que sí, hace que el código sea más **predecible**, **seguro** y fácil de mantener.
  3. == (Igualdad abstracta): Compara solo los **valores**, aplicando **coerción de tipo** (convierte los operandos al mismo tipo antes de comparar).

=== (Igualdad estricta): Compara los **valores** y los **tipos de datos** sin realizar ninguna coerción. Es **la forma recomendada** de comparación en JavaScript para evitar resultados inesperados.

* 1. Se produce un **error de tipo** (TypeError) en tiempo de ejecución. La variable declarada con const no permite que se cambie su **referencia** (el puntero a su espacio de memoria).Si la const almacena un objeto o un array, puedes modificar sus propiedades internas, pero no puedes reasignar la variable completa a un nuevo objeto o array.
  2. Las funciones ofrecen **modularidad** y **reutilización**. Permiten dividir tareas complejas en bloques lógicos y pequeños, lo que hace que el código sea más fácil de entender (**legibilidad**), depurar y probar. Al reutilizar el mismo bloque de código con diferentes entradas, reduces la repetición (**DRY - Don't Repeat Yourself**).
  3. Un arreglo (Array) en JavaScript es un tipo de **objeto**. Aunque se comporta como una lista ordenada, internamente, JavaScript lo trata como un objeto especial con propiedades y métodos diseñados para la manipulación de listas.