JS tipos de variables

Innovación en Formación Profesional

1. Tipos Primitivos

String: Representa secuencias de caracteres y se utiliza para trabajar con texto.
Ejemplo: `let_nombre = “Juan";

Number: Representa tanto números enteros como de punto flotante.
Ejemplo: `let_edad = 25;

BigInt: Se utiliza para números enteros muy grandes que superan el límite de `Number`.
Ejemplo: `let_gigante = 123456789012345678901234567890123456789012345678901234567890n;

Boolean: Tiene dos valores posibles: **`true`** o **`false`**. Se utiliza para operaciones lógicas.
Ejemplo: `let_es_Mayor = true;

Undefined: Indica que una variable ha sido declarada pero aún no se le ha asignado un valor.
Ejemplo: `let_color;

Null: Es un valor especial que indica la ausencia de valor.
Ejemplo: `let_salario = null;

Symbol: Introduce un identificador único que es inmutable. Se utiliza para propiedades de objeto que necesitan ser únicas.
Ejemplo: `let_simbolo = Symbol('miSimbolo');

JS tipos de variables

Innovación en Formación Profesional

1. Objetos

Representan colecciones de datos y/o funcionalidades más complejas.
Los objetos pueden contener múltiples valores en forma de propiedades y métodos.
Ejemplo: let_persona = { nombre: "Juan", edad: 25 };

Array: Es un tipo especial de objeto utilizado para almacenar listas ordenadas de valores.
Ejemplo: let_colores = ["Rojo", "Verde", “Azul”];

Array compuesto: Es un tipo especial de objeto utilizado para almacenar listas ordenadas de valores que a su vez tiene otros array dentro.
Ejemplo: let_colores = ["Rojo", "Verde", “Azul”,[“jamón”,”chorizo”,”lomo”,”chuleta”],”rosa”,”violeta”];

Function: También son objetos en **JavaScript** y pueden ser almacenados en variables.
Ejemplo: let_suma = function(a, b) { return a + b; };.

Date, RegExp, Map, Set, etc., son otros tipos de objetos incorporados que proporcionan funcionalidades específicas.

tipos de variables

Variables “let” y “var”...

Innovación en Formación Profesional

*La diferencia entre **`var`** y **`let`** en JavaScript radica en el ámbito de la variable (scope), el hoisting y el comportamiento en bloques de código:*

1. Ámbito (Scope):

- **`var`**: Declara una variable con ámbito de función o global si se declara fuera de una función. Esto significa que si `var` se utiliza dentro de una función, la variable solo es accesible dentro de esa función.

  Si se declara fuera de cualquier función, se convierte en una variable global.

- **`let`**: Introduce el ámbito de bloque (`{}`), lo que significa que la variable solo es accesible dentro del bloque en el que se declara (por ejemplo, dentro de un bucle o un bloque `if`).

2. Hoisting:

- Las variables declaradas con `var` son "elevadas" o "hoisted" al principio de la función o del script global, independientemente de dónde se haya realizado la declaración, y se inicializan con `undefined`.

- Las variables declaradas con `let` también son hoisted al comienzo de su ámbito de bloque, pero no se inicializan.

- El intento de acceder a ellas a las variables ‘let’ antes de su declaración resulta en un `ReferenceError`. Este fenómeno se conoce como "Temporal Dead Zone" (Zona Muerta Temporal).

3. Creación de propiedades en el objeto global:

- Las variables declaradas con **`var`** en el ámbito global se añaden como propiedades al objeto global (en el caso de los navegadores, este objeto global es `window`).

- Las declaradas con **`let`** no añaden sus nombres al objeto global.

4. Re-declaración:

- **`var`** permite re-declarar la misma variable en el mismo ámbito sin errores.

- **`let`** no permite la re-declaración de la misma variable en el mismo ámbito o bloque, resultando en un error si se intenta hacerlo.

*“Debido a estas diferencias, **`let`** es generalmente preferido por su predictibilidad y por ofrecer un control más fino sobre el ámbito de las variables, lo que puede ayudar a prevenir errores difíciles de detectar y hacer el código más legible y seguro.”*

impresión de valores

En **JavaScript**, existen varias maneras de imprimir o mostrar datos, dependiendo del entorno en el que estés trabajando (navegador, Node.js, etc.) y del propósito.

Innovación en Formación Profesional

- **console.log()**: Esta es probablemente la forma más común de imprimir datos para fines de depuración. Muestra el mensaje en la consola del desarrollador del navegador o en la terminal si estás usando Node.js.
- **console.error()**: Similar a console.log(), pero se utiliza para imprimir mensajes de error. En la consola del desarrollador, estos mensajes suelen aparecer con un ícono de error y en color rojo para destacarlos.
- **console.warn()**: Se utiliza para imprimir advertencias. Los mensajes aparecen con un ícono de advertencia en la consola del desarrollador.
- **console.info()**: Para imprimir mensajes informativos. En algunas consolas, se muestra con un ícono de información.
- **alert()**: En un entorno de navegador, puedes usar alert() para mostrar un cuadro de diálogo emergente con un mensaje. Es una forma de imprimir datos, pero es intrusiva y bloquea la ejecución del script hasta que el usuario cierra el cuadro de diálogo.

Cada una de estas opciones tiene sus propios casos de uso y peculiaridades. Por ejemplo, console.log() y sus variantes son excelentes para la depuración, mientras que alert(), document.write(), y la manipulación de innerHTML o textContent son más apropiados para interactuar con el usuario o mostrar datos en la página web.

- **document.write()**: Esta función escribe directamente en el documento HTML. Es menos utilizada para la depuración y más para la manipulación directa del contenido de una página web, pero puede servir para imprimir datos en el cuerpo del documento.
- **innerHTML o textContent**: Estas propiedades de los elementos del DOM se pueden utilizar para insertar datos en un elemento específico del documento HTML, lo que efectivamente "imprime" los datos en la página web.
- **console.table()**: Es útil para imprimir objetos o arrays en forma de tabla, lo que puede facilitar la visualización de datos complejos.
- **console.dir()**: Proporciona una lista interactiva de las propiedades del objeto JavaScript especificado, lo cual es útil para inspeccionar objetos.
- **process.stdout.write()**: En un entorno Node.js, esta función se puede utilizar para escribir en la salida estándar del proceso. Es similar a console.log() pero no añade automáticamente un carácter de nueva línea al final del mensaje.