Variables en JavaScript

En JavaScript, los nombres de las variables pueden estar compuestos por cualquier secuencia de letras (mayúsculas y minúsculas), dígitos, caracteres de subrayado y signos de dólar, pero no deben comenzar con un dígito. Existe una lista de palabras reservadas que no se pueden usar como nombres de variables.

Lo importante también es que el intérprete de JavaScript distinga entre letras minúsculas y mayúsculas, también en los nombres de variables, por lo que nombres como prueba, Prueba, o PRUEBA serán tratados como diferentes.

Declaración de Variables

La declaración, al igual que otras instrucciones de JavaScript, debe terminar con un punto y coma.
Var peso;
Console.log(peso);
Console.log(altura);

En la segunda línea, intentamos escribir el valor de esta variable (es decir, lo que hay en el contenedor) en la consola. Como todavía no hemos puesto nada allí, el resultado no está definido (el intérprete conoce esta variable, pero aún no tiene valor; el valor no está definido). En la siguiente línea, intentamos imprimir el contenido de la variable de altura... que olvidamos declarar. Esta vez, veremos **Error de referencia.** El intérprete de JavaScript, que ejecuta nuestro programa, nos ha informado que no conoce una variable con este nombre (por lo que la variable en sí no está definida).

La alternativa a esto es la palabra clave **let**; Usamos ambas palabras clave de la misma manera. Ambas están pensadas para declarar variables y ambas se pueden encontrar en diferentes ejemplos en Internet o en libros.

La palabra clave **var** proviene de la sintaxis original de JavaScript y la palabra clave **let** se introdujo mucho más tarde. Por lo tanto, se podrá ver **var** en programas más antiguos. Actualmente, se recomienda utilizar la palabra let.
Ejemplo

```
    <script>
        let peso;
        console.log(peso);
    </script>
```

Una de las diferencias básicas en el uso de **var** y **let** es que let nos impide declarar otra variable con el mismo nombre (se genera un error). El uso de var permite volver a declarar una variable, lo que potencialmente puede provocar errores en la ejecución del programa.
En la siguiente imagen el intérprete no marcará error.

```
<script>
    var altura;
```

```
    var altura;
    console.log(altura);
</script>
```

Mientras que en la siguiente imagen el intérprete marcará un error.

```
<script>
    let altura;
    let altura;
     console.log(altura);// Uncaught SyntaxError: Identifier 'altura' has already been declared
</script>
```

Inicialización de variables

La inicialización se puede realizar junto con la declaración o por separado como un comando independiente. Es importante introducir el primer valor en la variable antes de intentar leerla, modificarla o visualizarla.

```
<script>
    let altura = 180;
    let otraAltura = altura;
    let peso;
    console.log(altura);
    console.log(otraAltura);
    peso = 70;
    console.log(peso);
</script>
```

Si transcribimos el código anterior; vamos a ver que las declaraciones de las variables altura y otraAltura se combinan con su inicialización, mientras que la variable peso se declara e inicializa por separado. Las variables peso y altura se inicializan proporcionando valores específicos (más precisamente, un número), mientras que la variable otraAltura recibe un valor leído de la variable altura.

Declaraciones y modo estricto

En 2009 y 2015 se introdujeron algunos cambios importantes en JavaScript. La mayoría de estos cambios ampliaron la sintaxis del lenguaje con nuevos elementos, pero algunos de ellos solo afectaban al funcionamiento de los intérpretes de JavaScript. A menudo se trataba de aclarar el comportamiento de los intérpretes en situaciones potencialmente erróneas, como en los casos de inicialización de variables sin ninguna declaración previa.
Lo vemos en el siguiente ejemplo, se creó una variable y se asignó un valor, sin declarar.

```
<script>
    altura = 180;
    console.log(altura);
</script>
```

La sintaxis original de JavaScript permitía tal negligencia y, en el momento de la inicialización, realizó esta declaración por nosotros. Parece una buena solución, pero, por desgracia, a veces puede dar lugar a situaciones ambiguas y potencialmente erróneas.

Sin embargo como lo vamos a ver en la siguiente imagen:
Al comienzo de nuestro código, agregamos `"use strict";` Esta declaración ha cambiado radicalmente el comportamiento del intérprete. ¿Por qué? La usamos cuando queremos obligar al intérprete a comportarse de acuerdo con los estándares modernos de JavaScript.
Esto hará que el intérprete trabaje con el resto del código utilizando el modo estricto, que es el estándar moderno de JavaScript.

```
<script>
    "use strict";
    altura = 180;
    console.log(altura); //ejemplos.html:12 Uncaught ReferenceError: altura is not defined
</script>
```

Cambio de valores de variables

En el siguiente ejemplo, vamos a declarar una variable llamada **pasos**. Inicialmente, contiene el número 100, que luego se cambia a 120. Luego, añadimos 200 al contenido actual de la variable, por lo que la variable contiene 320.

```
<script>
    let pasos = 100;
    console.log(pasos); // 100
    pasos = 120;
    console.log(pasos); // 120
    pasos = pasos + 200;
    console.log(pasos); // 320
</script>
```

Las variables en el lenguaje JavaScript no tienen tipos (o, para ser más precisos, tienen tipos débiles y dinámicos). Esto significa que JavaScript no controlará qué tipo de valor almacenamos en la variable. En JavaScript, los tipos principales son los números y las cadenas de caracteres.

```
<script>
    let saludo = "Hola como estas";
    let contador = 10;
</script>
```

Como se puede ver en la siguiente imagen, la variable saludo se inicia con un valor de tipo cadena, mientras que la variable contador se inicia con un valor de tipo número. Siguiendo con el ejemplo, haremos un pequeño cambio en el contenido de la variable saludo.

```
<script>
```

```
    let saludo = "Hola como estas";
    console.log(saludo);
    saludo = 1;
    console.log(saludo);
    let contador = 10;
</script>
```

JavaScript nos permite reemplazar fácilmente la variable saludo por un valor cuyo tipo sea diferente al que se almacenaba originalmente allí. JavaScript va un paso más allá y no solo nos permite cambiar los tipos de valores guardados en una variable, sino que también realiza su conversión implícita si es necesario

```
<script>
    let saludo = "Hola como estas";
    let contador = 1;
    saludo = saludo + contador;
    console.log(saludo);
</script>
```

El intérprete comprobará el tipo de valor que almacena la variable saludo y convertirá el valor de contador al mismo tipo antes de hacer la operación suma. Como resultado el valor de contador pasa a ser una cadena, para almacenarse en la varialble saludo.

Constantes

El objetivo principal de una constante es eliminar la posibilidad de cambiar accidentalmente un valor almacenado en ella. Esto es importante cuando tenemos algunos valores que realmente nunca deberían cambiar. Ejemplos típicos de constantes son las rutas a recursos, tokens y otros datos que nunca cambian durante la vida útil del script.
Pero las constantes también se pueden utilizar como subresultados en cálculos o en otros lugares donde la información que se recopiló o calculó no cambiará más. El uso de una constante, además de evitar que un valor se modifique por error, permite que el motor de JavaScript optimice el código, lo que puede afectar su rendimiento.

```
<script>
    const saludo = "Hola como estas";
    saludo = "hola";
    console.log(saludo); // TypeError: Assignment to constant variable.
</script>
```

Flujo de control en JS

La declaración if

La sentencia if es la primera y más simple instrucción de flujo de control disponible en JavaScript. Tiene varias formas, pero en su forma básica, verifica una condición dada y, según su valor booleano, ejecuta un bloque de código o lo omite. La sintaxis es la siguiente:

if (condicion) {
  codigo del bloque
  }

```
<script>

    let siUsuarioListo = confirm("¿Se encuentra listo?");
    console.log(siUsuarioListo);
    if (siUsuarioListo) {
        alert("Usuario listo!");
    }

</script>
```

En el ejemplo anterior, una alerta con el mensaje "`El usuario se encuentra listo para iniciar`!" se mostrará únicamente cuando el usuario cierre el cuadro de diálogo de confirmación haciendo clic en el botón Aceptar. Pero si el usuario cierra el cuadro de confirmación `"¿Se encuentra listo para iniciar?"` sin hacer clic en Aceptar, el mensaje "`El usuario se encuentra listo para iniciar`!" nunca se mostrará.

Comenzamos con un ejemplo sencillo, en el que, además de la instrucción condicional, utilizaremos los cuadros de diálogo recientemente aprendidos:

```
<script>

    let precioUnitario = 10;
    let productos = prompt("Cuantos productos ha solicitado?", 0);
    if (productos > 0) {
        let total = precioUnitario * productos;
        alert('El calculo total a pagar será: ' + total);
    }

</script>
```

Como podemos ver en el código anterior, usamos el operado concatenación, para el mensaje al usuario.

Pero podemos mejorar el código utilizando **template literals** introducidas en ES6. Esta sintaxis te permite incrustar expresiones directamente dentro de una cadena utilizando comillas invertidas (`).

```
<script>

    let precioUnitario = 10;
    let productos = prompt("Cuantos productos ha solicitado?", 0);
    if (productos > 0) {
        let total = precioUnitario * productos;
        alert(`El calculo total a pagar será: ${total}`);
    }
```

Notemos lo siguiente, en la imagen anterior: La variable total se declara dentro del **bloque**. Podemos asignarle un valor, podemos leerla, pero todo esto solo dentro del bloque. Intentar hacer referencia a ella fuera del bloque terminará en un error. El intérprete nos informará que no conoce dicha variable en este caso no podría mostrar su valor. Trata, de imprimir la variable total, fuera del bloque if, para confirmarlo.

El siguiente código simula una lógica de cálculo de costos de envío en función de la edad del usuario, el valor de su carrito y la cantidad de puntos que tiene. Si el usuario cumple con las condiciones establecidas, el costo de envío se reduce a cero.

```
let usuarioEdad = 22;
let esMujer = false;
let puntos = 703;
let valorCarrito = 299;
let costoEnvio = 10000;
if (usuarioEdad > 21) {
    if (valorCarrito >= 300 || puntos >= 500) {
        costoEnvio = 0;
    }
}
alert(`El costo de envio es de ${costoEnvio}`);
```

En JavaScript, las variables declaradas dentro de un bloque (como un if) tienen un alcance local a ese bloque. Esto significa que solo pueden ser utilizadas dentro de ese bloque. Sin embargo, las variables declaradas fuera de cualquier bloque tienen un alcance global y pueden ser utilizadas en cualquier parte del código. Como podemos ver en la imagen anterior, es posible acceder al valor “costoEnvio”, ya que la variable se definió fuera del bloque.

Otra forma de escribir lo mismo es utilizar el operador lógico AND. Ahora podemos eliminar una instrucción if y describir todo como una condición más compleja. Tengamos en cuenta que utilizamos paréntesis adicionales para agrupar las operaciones lógicas seleccionadas. Esto nos permitirá forzar su precedencia de ejecución.

```
let usuarioEdad = 20;
let esMujer = false;
let puntos = 703;
let valorCarrito = 299;
let costoEnvio = 10000;
if (usuarioEdad > 21 && (valorCarrito >= 300 || puntos >= 500)) {
    costoEnvio = 0;
}
alert(`El costo de envio es de ${costoEnvio}`);
```

la instrucción if...else

La sentencia if es muy útil, pero ¿qué pasa si también queremos ejecutar algún código cuando no se cumple una condición determinada? Por supuesto, podríamos utilizar otra sentencia if y cambiar la condición, como se muestra en el ejemplo:

```
let usuarioListo = confirm("¿Esta listo?");
if (usuarioListo) {
    alert("Usuario Listo");
}
if (usuarioListo == false) {
    alert("Usuario no Listo");
}
```

Esto funcionará como se espera, pero no se ve muy bien. ¿Qué pasa si en el futuro tenemos que cambiar esta condición? ¿Recordaremos cambiarla en ambos lugares? Este es un posible error de lógica futuro. Entonces, nos convendría usar un else.

```
let usuarioListo = confirm("¿Esta listo?");
if (usuarioListo) {
    alert("Usuario Listo");
} else {
    alert("Usuario no Listo");
}
```

la instrucción if…else…if

Ramificar el flujo de ejecución del código solo en dos ramas a veces no es suficiente. Hay una solución sencilla para esto en JavaScript: podemos anidar “if…else if”. El formato es el siguiente:

```
if (conditions_1) {
code
} else if (condition_2) {
code
} else if (condition_3) {
code
} else {
code
}
```

Ejemplo de uso:

```
<script>
    let numero = prompt("Ingrese un número", 0);
    if (numero < 10) {
        alert("<10");
    } else if (numero < 30) {
        alert("<30");
    } else if (numero < 60) {
        alert("<60");
    } else if (numero < 90) {
        alert("<90");
    } else if (numero < 100) {
        alert("<100");
    } else if (numero == 100) {
        alert("100")
    } else {
        alert(">100")
    }
</script>
```

Vamos a complejizar nuestro análisis sobre las declaraciones. En la siguiente imagen vamos a poder observar lo siguiente:

- uso de **Prompts** para obtener datos
- Conversión de tipos: Se utilizan **parseInt** y **parseFloat** para convertir las respuestas de los **prompts** a números, ya que los **prompts** siempre devuelven cadenas de texto.
- Confirmaciones: Se utilizan **confirm** para preguntas que tienen respuestas de sí o no.

```
<script>
let SEGURO_ENVIO = 4000;
let costoEnvio = 6000;
let esOrdenValida = true;
// Se piden al usuario los datos
let usuarioEdad = parseInt(prompt('Ingrese su edad:'));
let puntos = parseInt(prompt('Ingrese la cantidad de puntos que tiene',0));
let valorCompra = parseFloat(prompt('Ingresar valor de comprar:'));
let tienePromo = confirm('Posee un cupon de descuento?:( OK  para Si y  Cancelar   para no)');
let tieneAprobacionDePadres = confirm('¿posee aprobación de sus padres? ( OK   para Si y  Cancelar   para no)');
let agregarSeguro = confirm('desea agregar un seguro de envio? ( OK  para Si y  Cancelar   para no)')

//Se calcula el costo de envio
if((usuarioEdad > 65) || (usuarioEdad >= 18 && (tienePromo || valorCompra > 50000 || puntos > 500))){
    constoEnvio = 0;
}else if (usuarioEdad < 18 && tieneAprobacionDePadres){
    costoEnvio = costoEnvio - 100;
}else if(usuarioEdad < 18){
    esOrdenValida = false;
}
//se agrega el costo de seguro si corresponde
if (esOrdenValida && agregarSeguro && !tienePromo){
    costoEnvio += SEGURO_ENVIO;
}

//se muestra el mensaje
if(esOrdenValida){
    alert(`El costo de envio es: $ ${costoEnvio}`);
}else{
    alert(`No puede realizar la compra. Debe ser mayor a 18 años.`);
}
</script>
```

Operador condicional

Este nos permite realizar una de dos acciones en función del valor del primer operando. La mayoría de las veces se utiliza como alternativa al operador condicional **if.**
Veamos un ejemplo sencillo:

```
<script>
    let cantidadProductos = parseInt(prompt("Ingrese cantidad de productos:"));

    // Calculamos el precio total (asumiendo que cada producto cuesta 150)
    let precioTotal = cantidadProductos * 15000;
    let costoEnvio;
    if (precioTotal > 50000) {
        costoEnvio = 0;
    } else {
        costoEnvio = 5000;
    }
    alert(`Precio total: $${precioTotal}\nCosto de envío: $${costoEnvio}`);
</script>
```

Ahora usamos el operador ternario:

```
<script>
    let cantidadProductos = parseInt(prompt("Ingrese cantidad de productos:"));
    // Calculamos el precio total (asumiendo que cada producto cuesta 150)
    let precioTotal = cantidadProductos * 15000;
    let costoEnvio;
    costoEnvio= precioTotal > 50000 ? 0 : 5000
    alert(`Precio total: $${precioTotal}\nCosto de envío: $${costoEnvio}`);
</script>
```

El mismo código reescrito utilizando el operador **“?”**condicional parece un poco más fácil. Como recordatorio: el primer operando (antes del signo de interrogación) es la condición a verificar, el segundo operando (entre el signo de interrogación y los dos puntos) se devolverá si la condición es verdadera, y el tercer operando (después de los dos puntos) si la condición es falsa.

Ejemplo:

```
<script>

    let start = confirm("Iniciar?");
    start ? alert("Iniciado :) ") : alert("Desestimado :( ");

</script>
```

Para tener en cuenta: *Es posible tener código más largo con* **?***, pero es mejor usar declaraciones if, ya que queda mucho más claro qué intención tenía el desarrollador al escribir el código.*

Switch case

Otra estructura usada para la ejecución de código condicional es la sentencia switch. Es algo similar a las sentencias if…else anidadas, pero en lugar de evaluar diferentes expresiones, switch evalúa una expresión condicional y luego intenta hacer coincidir su valor con uno de los casos dados. Esta es la sintaxis de la sentencia switch:

```
switch (expression) {
    case first_match:
     code
     break;
    case second_match:
     code
     break;
    default:
     code
}
```

La sentencia switch es una estructura de control que nos permite evaluar una expresión y ejecutar diferentes bloques de código en función del valor resultante de esa expresión. Es una alternativa más concisa a una serie anidada de if...else cuando tenemos múltiples condiciones posibles.
Donde cada case representa un valor posible para la expresión.
El break, termina la ejecución del bloque switch una vez que se encuentra un case coincidente. Si se omite, el programa continuará ejecutando los siguientes casos.
Mientras que default: Se ejecuta si ninguno de los case coincide con la expresión.

- La comparación en switch es siempre estricta (===). Esto significa que tanto el valor como el tipo de dato deben coincidir para que se ejecute el case.
- El orden de los case no importa, ya que la comparación se realiza de forma independiente para cada uno.
- Podemos agrupar varios case para ejecutar el mismo bloque de código:

```
switch (expresión) {
    case valor1:
    case valor2:
    // Código a ejecutar si expresión === valor1 o expresión === valor2
    break;
    // ... más casos
    default:
        // Código a ejecutar si ningún caso coincide
    }
```

Ejemplo de uso,

```
<script>
    let puerta = prompt("Elige una puerta: a, b, or c");
    let win = false;
    switch (puerta) {
        case "a":
        alert("La puerta A esta vacia");
        break;
        case "b":
        alert("puerta B: Ganaste crack!!!");
        win = true;
        break;
        case "c":
        alert("puerta C: vacia");
        break;
        default:
        alert("Perdiste " + String(puerta));
    }
    if (win) {
        alert("Ganaste!");
    }
</script>
```

Ejercicios:

Primer Enunciado:

Crea un programa que simule una tienda en línea que vende productos electrónicos. El programa debe solicitar al usuario los siguientes datos:

Nombre del producto: Un string.
Precio del producto: Un número decimal.
Cantidad de productos: Un número entero.
Método de pago: Un string (puede ser "efectivo", "tarjeta" o "transferencia").
Código promocional: Un string (opcional).
Calcular:

Subtotal: Precio del producto * cantidad de productos.
Descuento: Si el código promocional es "DESCUENTO10", aplicar un descuento del 10% sobre el subtotal.
IVA (19%): Calcular el IVA sobre el subtotal (después de aplicar el descuento, si corresponde).
Total a pagar: Subtotal + IVA.
Mostrar en pantalla:

Un mensaje que indique el nombre del producto, la cantidad, el subtotal, el descuento (si aplica), el IVA y el total a pagar.
Si el método de pago es "efectivo" y el total es mayor a $1000, ofrecer un descuento adicional del 5%.
Consideraciones adicionales:

Validación de datos: Asegurarse de que los datos ingresados por el usuario sean válidos (por ejemplo, que el precio y la cantidad sean números positivos).
Formato de moneda: Mostrar los valores monetarios con el símbolo de moneda y dos decimales.
Mensajes personalizados: Mostrar mensajes claros y concisos al usuario.
Ejemplo de salida:

Producto: Teléfono celular
Cantidad: 2
Subtotal: $1500.00
Descuento: $150.00
IVA: $256.50
Total a pagar: $1606.50

Segundo Enunciado

Crea un programa que simule un menú de un restaurante. El programa debe pedir al usuario que ingrese un número del 1 al 4 para seleccionar un plato. Según la opción elegida, se mostrará un mensaje indicando el plato seleccionado y su precio.

- Opción 1: Hamburguesa ($5000)
- Opción 2: Pizza ($8000)
- Opción 3: Ensalada ($3000)
- Opción 4: Sopa ($4800)

Si el usuario ingresa un número fuera de este rango, se mostrará un mensaje indicando que la opción no es válida.

Tercer enunciado:

Crea una calculadora básica en JavaScript. La calculadora debe solicitar al usuario que ingrese los siguientes datos:

- Número 1: Un número decimal.
- Número 2: Otro número decimal.
- Operación: Un carácter que represente la operación matemática a realizar (+, -, *, /).

Validación de la entrada:
Comprobar si los valores ingresados son números válidos: usar isNaN() para verificar que los valores convertidos a números no sean NaN (Not a Number).
Valida el operador: corroborar de que el operador ingresado sea uno de los cuatro permitidos (+, -, *, /).