MIGUEL RODRÍGUEZ POLO |2º DAW | 1º HITO INDIVIDUAL

CUESTIÓN 1 A) Todos los lenguajes de programación necesitan compilarse para funcionar. JavaScript no es una excepción.

Sin embargo, cuando JavaScript comienza a compilar, realiza las necesarias conversiones de forma interna para permitir un tratamiento ambiguo de las variables según se requieran en tiempo de ejecución. Eso significa que entra dentro del espectro de lenguajes débilmente tipados (o no-tipados)

B) ## Tipos de variables

Javascript pertenece al grupo de lenguajes imperativos pero, sin embargo, se caracteriza por estructuras de tipado débil donde la declaración de variables no exige la asociación con un tipo de datos de forma implícita y unívoca.

Javascript tiene un tipo de datos dinámico. Esto significa que una variable puede declararse conteniendo un tipo de dato y, más adelante, asignarle otro tipo mediante una simple asignación. Ejemplo:

```
<script>
var cambioTipo = "hola";  //es un string
cambioTipo = 89;  //Ahora es un número
</script>
```

Numéros (Number)

Números enteros y decimales, positivos y negativos.

En otros lenguajes de programación se distingue entre distintos tipos de números (int: entero, float: decimal…), por lo que hay que indicarlo al declarar la variable, pero en JavaScript no.

```
var numero = 5;
var numero_decimal = 4.1;
var numero_negativo = -12;
```

¿Que ocurre al intentar sumar decimales con enteros? (2.1 + (-12))

```
numero_decimal + numero_negativo
```

Resultado = -7.9

El resultado es el esperado. Los números en JavaScript funcionan y se convierten de forma automática.

Cadenas (String)

Una o varias cadenas de caracteres alfanuméricos (letras, números, signos puntuación…).

Las denominamos tipos de datos o variables de texto porque las empleamos para almacenar información de esa naturaleza o porque sus valores los vamos a tratar como texto. Ejemplo:

```
var nif = "12332112W",
ciudad= "Santiago de Compostela",
codigoPostal = "28001",
telefono = "003423145627";
ocupacion = "Estudiante de Biología";
```

Como vemos en los ejemplos, el código postal o el teléfono, aunque contengan solo números, no nos interesa tratarlos de esa forma (en el sentido de aplicarles operaciones matemáticas: sumarlos, restarlos, comparar si son mayores o menores…). Tiene más sentido tratarlos como cadenas de texto, por lo que asignamos su valor entrecomillado.

Pruebas cruzadas con concatenaciones de String y Number

```
var numTexto1= "3";
//variable tipo string-texto al asignar el valor entrecomillado
var numTexto2= "5";
//igual tipo que la anterior
document.write("Al usar el operador + con las variables string 3
y 5 obtenemos: ");
document.write(numTexto1 + numTexto2);
//te aparecerá la cadena "35”, al haber sido tratadas como texto
var num1 = 3 ;
//variable tipo number-número al asignar el valor sin comillas
var num2 = 5;
//igual tipo que la anterior
document.write("<br>");
document.write("Al usar el operador + con las variables number 3
y 5 obtenemos: ");
document.write(num1 + num2);
//te aparecerá el resultado de la suma: 8, al haber sido tratado
como números
```

Booleanos (Banderas,condiciones...etc)

El nombre viene de la lógica de Boole. Pueden tomar solo dos valores: ‘true‘ o ‘false‘ (verdadero/falso).

Se utilizan mucho en las sentencias condicionales, a modo de interruptor, pudiendo equiparar los valores true/false, en ese símil, a 'encendido' /apagado', es decir, que hace que el programa se siga ejecutando por una instrucción o por otra diferente:

"Si es verdad, sigue ejecutando X, si es falso, haz Y".

```
    var boolean = false
    if(boolean ==true){
        //esto no ocurre
    }

    if(boolean ==false){
        //esto si ocurre
    }
```

Estas variables no deberían combinarse con el resto, ya que están mayormente diseñadas para ser usadas en condiciones.

```
    var bool1 = true;               //variable tipo number-número
al asignar el valor sin comillas
    var num2 = 5;                 //igual tipo que la anterior
    document.write("<br>");
    document.write("Al usar el operador + con las variables
boolean en true y number 5 obtenemos: ");
    document.write(bool1 + num2);  //te aparecerá el resultado de
la suma: 8, al haber sido tratado como números
```

¿Cómo es posible este resultado? Esto se debe a que las variables booleanas tienen 2 valores. 0(false) y 1(true).

Al forzar al programa a sumar un número con un boolean, este ha obtenido su valor 1(true) y lo ha convertido automáticamente en numérico para sumarlo.

Normalmente otros lenguajes de programación más fuertemente tipados no permitirían esto bajo ningún concepto, pero Javascript lo hace.

List (Listas)

Los Arrays ó matrices pueden contener múltiples valores, como un buzón con diversos compartimentos numerados en su interior. Son del tipo objetos:

```
var granPoblacion = ["Madrid", "Barcelona", "Valencia",
"Sevilla"];
```

Hay dos principales elementos en su estructura:

- **corchetes [ ]:** para incluir los valores
- **comas (,):** para separar los valores dentro del array

En el array, los valores (que pueden ser numéricos, de texto o una combinación de ellos) se encuentran en posiciones numeradas, empezando a contar desde el cero, no desde el uno. Es decir, los valores de cada elemento del array (a los que se accede con nombreArray[n], siendo n= 0,1,2…) serían:

```
granPoblacion[0] = Madrid
granPoblacion[1] = Barcelona
granPoblacion[2] = Valencia
granPoblacion[3]= Sevilla
```

Podrías añadir o cambiar valores dentro del array. Por ejemplo, la sentencia: granPoblacion[4] = "Zaragoza"; añadiría este valor detrás del valor 'Sevilla'. Sin embargo es preferible el método push()

Ahora voy a construir esta lista leyendo el array uno a uno

```
granPoblacion.forEach(function(ciudad){
  document.write("<li> " + ciudad + "</li>");
})
```

Las listas no se pueden sumar o concatenar con otras variables de forma primitiva. Ahora voy a intentar sumarle 3 a la lista

Resultado: Madrid,Barcelona,Valencia,Sevilla3

Lo que ocurre es que el 3 simplemente se concatena. La lista se ha impreso ordenada con todos sus elementos.

C)

Diferencias var, let y const

JavaScript fue diseñado en 10 días en 1994 para hacer scripts simples en Netscape (el navegador más popular de ese momento que hoy conocemos como Firefox). Nunca se creyó que se convertiría en el lenguaje más popular 20 años después.

Por el afán de lanzar el lenguaje rápidamente se cometieron muchos errores de diseño. Uno de ellos fue la forma en que se declaran las variables.

Por un lado, es posible declarar una variable sin utilizar var, let o const, lo que crea una variable global en la aplicación:

```
nombre = "hola"
```

Var contra Let

El problema de las variables declaradas con var es que se comportan diferente que en la mayoría de lenguajes de programación, que limitan el alcance(o contexto) por bloque, es decir, entre llaves ({ y }). Esto no ocurre en JavaScript cuando utilizamos var. Por ejemplo:

```
     if (true) {
  var color = "negro"
}
console.log(color) // "negro"
```

Con var, la única forma de limitar el alcance es dentro de una función:

```
     function miFuncion() {
  var color = "negro"
}
console.log(color) // error
```

Esto puede llegar a ocasionar una gran cantidad de problemas, entre ellos la sobreescritura

```
if (orange === 'orange') {
  var orange = 'blue'; // el ambito es global
  let apple = 'green'; // el ambito está dentro del bloque IF

  console.log(orange);  // azul
  console.log(apple);  // verde
}

console.log(orange); // azul
console.log(apple); // rojo
```

En este ejemplo, el valor original de la variable orange se pierde completamente en la función debido a la sobreescritura. JavaScript no sabe distinguir de contexto local o global cuando se trabajo con var.

A diferencia de var, let trabaja siempre en su bloque, que puede ser un if, un for, un while...etc. Esto impide que se declaren variables let completamente globales, y por lo tanto impide los riesgos derivados.

Hoy en dia, se recomienda encarecidamente utilizar let lo máximo posible por estas razones.

Const

Const es una variable let, pero con 2 peculiaridades

Debe inicializarse. No puede declararse sin valor

```
const direccion = "calle paco"
```

Una vez se le asigna un valor inicial, no puede ser modificado

```
const color_manzana = "rojo"
color_manzana="verde" //error
```

Const es un tipo de variable intransigente, cuyo contenido no puede ser modificado. Su utilidad reside en la memoria que se ahorra al utilizarla.

Cuando declaras una variable como let o var, estas reservando una cantidad X de memoria, para los posibles valores que pueda albergar esa variable (Que pueden llegar a ser muy elevados si el programador lo desea). Sin embargo, al usar const, el programa asigna automáticamente solo la memoria necesaria para ese valor.

CUESTIÓN 2

Diseño en HTML (Haz click aquí)

Se trata de un formulario visualmente bonito gracias a una plantilla de CSS. El formulario en el texto superior es capaz de detectar cuál es tu navegador, además de decirle al usuario las dimensiones de su ventana. (BOM)

Además, si el usuario no desea hacer el formulario, se le dará la opción de abandonar la página. Si pulsa el botón de abandonar, se le obligará a realizar el formulario haciendo un scroll hacia él y deshabilitando el botón. (BOM)

El formulario no aceptará números en la caja de nombre o apellido. Además, el formulario incluirá al usuario que se registre en él en una tabla que se encuentra más abajo y que crecerá con cada solicitud (Nótese que el color favorito se aplica en la celda correspondiente de cada usuario de la tabla), aunque volverá a la normalidad tras recargar la página (mostrando solo la única persona de prueba que había). (DOM)

Por último, el formulario imprimirá una imagen aleatoria para felicitar al usuario por realizarlo correctamente.

CUESTIÓN 3

Diseño en html: Cuestion3\index.html

En el archivo html podrás introducir y cargar datos para hacer pruebas con los distintos métodos de almacenamiento local. Además de introducir datos

en texto, hay dos botones para introducir datos poco convencionales, como una lista en LocalStorage, y un Date en sessionStorage.

DIFERENCIAS ENTRE COOKIES, LOCALSTORAGE Y SESSIONSTORAGE

- **cookie:** Un archivo de texto guardado en la computadora del usuario para almacenar y recuperar datos. Pueden ser leídas también por el servidor.
- **sessionStorage:** Es el espacio de memoria en un navegador para guardar datos temporales hasta que se cierre la ventana o pestaña. No pueden ser leído correctamente por el servidor
- **localStorage:** Como una cookie, donde los datos se pueden guardar y recuperar después de las sesiones del navegador, pero se almacenan en la memoria como sessionStorage. Los datos se almacenan como pares clave-valor sin formato y se pueden almacenar como objetos JSON. No pueden ser leído correctamente por el servidor

Esta imagen permite visualizar mejor sus diferencias. El servidor en realidad puede leer LocalStorage y SessionStorage, pero es un proceso engorroso y poco eficiente, normalmente.

	cookie	localStorage	sessionStorage
Initiator	Client or server. Server can use `Set-Cookie` header	Client	Client
Expiry	Manually set	Forever	On tab close
Persistent across browser sessions	Depends on whether expiration is set	Yes	No
Sent to server with every HTTP request	Cookies are automatically being sent via `Cookie` header	No	No
Capacity (per domain)	4kb	5MB	5MB
Accessibility	Any window	Any window	Same tab

¿EN QUÉ CARPETA DE WINDOWS SE ALMACENA CADA DATO?

Depende del navegador.

- **Firefox** guarda el almacenamiento web en un archivo SQLite llamado webappsstore.sqlite, dentro de la carpeta del perfil del usuario.
- **Google Chrome** guarda el almacenamiento web en un archive SQLite en el perfil del usuuario. La subcarpeta es: "\AppData\Local\Google\Chrome\User Data\Default\Local Storage" en Windows.

- **Opera** El almacenamiento web de Opera está guardado en "\AppData\Roaming\Opera\Opera\sessions\autosave.win" o "\AppData\Local\Opera\Opera\pstorage\" En función de la version de Opera.
- **Internet Explorer** El almacenamiento es en: "\AppData\LocalLow\Microsoft\Internet Explorer\DOMStorage".

En la página HTML, puedes usar los distintos botones para cargar los datos en un párrafo, donde se mostrarán los resultados. Si no rellenas los campos, crearás almacenamiento web vacío.