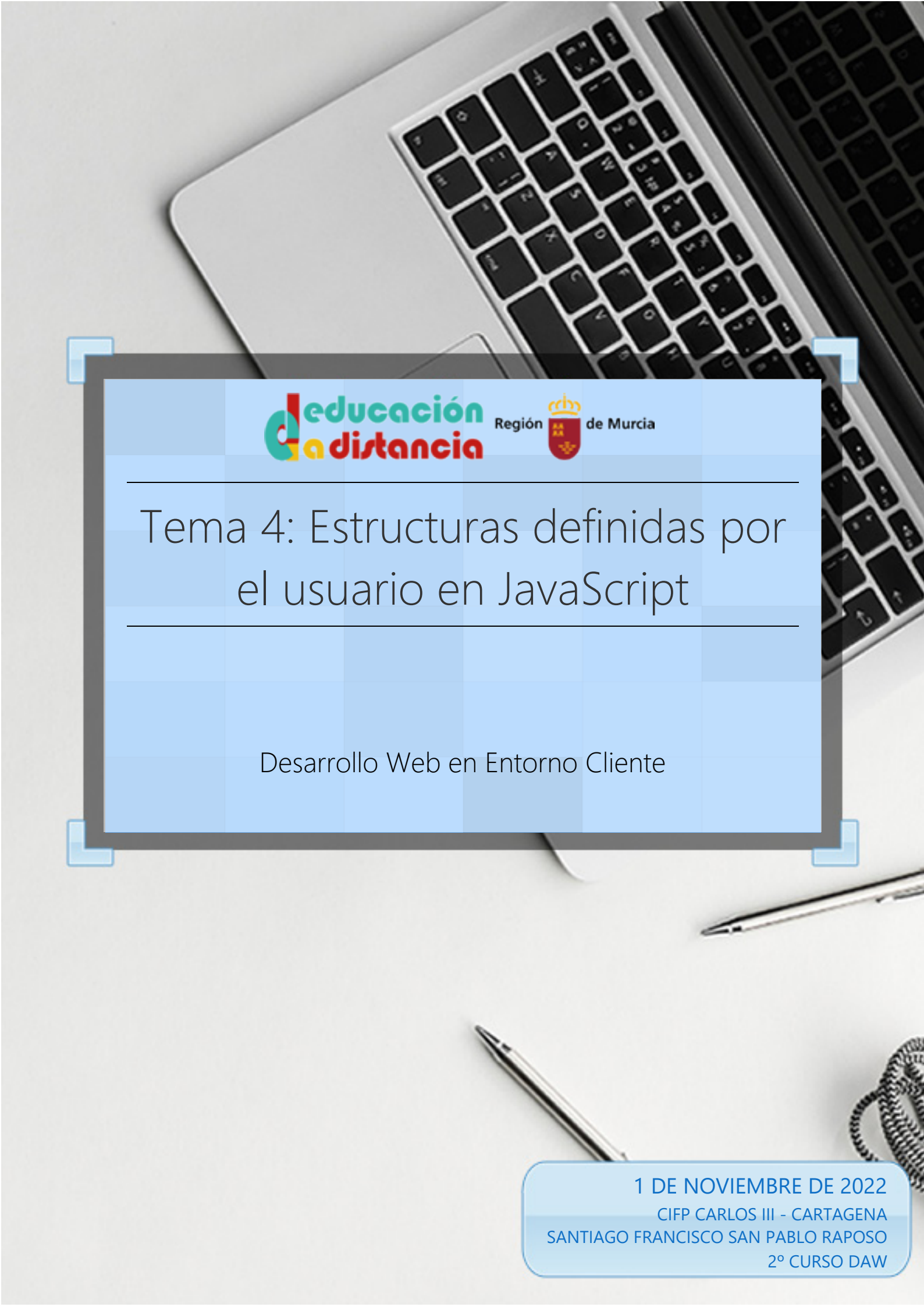

educación a distancia Región de Murcia

Tema 4: Estructuras definidas por el usuario en JavaScript

Desarrollo Web en Entorno Cliente

1 DE NOVIEMBRE DE 2022
CIFP CARLOS III - CARTAGENA
SANTIAGO FRANCISCO SAN PABLO RAPOSO
2º CURSO DAW

Contenido

Índice de ilustraciones.

No se encuentran elementos de tabla de ilustraciones.

Índice de tablas.

No se encuentran elementos de tabla de ilustraciones.

Resumen tema 4.

1.- Estructuras de datos.

Caso práctico: Veremos cómo crear los Arrays, recorrerlos, borrar sus elementos, veremos sus propiedades y métodos, y terminaremos por explicar lo que son los arrays paralelos y multidimensionales.

En los lenguajes de programación existen estructuras de datos especiales, que nos sirven para guardar información más compleja que una variable simple.

Hay un montón de tipos de estructuras de datos (listas, pilas, colas, árboles, conjuntos,…) pero una estructura de las más utilizadas en todos los lenguajes es el array. **El array, es como zona de almacenamiento continuo**, donde podemos introducir varios valores en lugar de solamente uno.

Los arrays se suelen denominar **matrices** o **vectores**. Una matriz se puede ver como **un conjunto de elementos ordenados en fila** (o filas y columnas si tuviera dos o más dimensiones).

Se puede considerar que todos los arrays son de una dimensión, la dimensión principal, **pero** los elementos de dicha fila, **pueden contener a su vez otros arrays** o matrices. **Esto abre la puerta a los arrays multidimensionales** (los más fáciles de imaginar son los de una, dos y tres dimensiones).

¿Cúando usar un array y cuando una lista?

- Acceso de forma aleatoria e impredecible = array.
- Elementos ordenados y acceso secuencial = lista. Ya que la lista puede cambiar de tamaño fácilmente durante la ejecución de un programa.

Posiciones del Array:

Cada elemento es referenciado por la posición que ocupa dentro del array. Dichas posiciones se llaman **índices** y siempre son correlativos. Existen **tres formas de indexar los elementos de un array**:

- Indexación base-cero (0): el primer elemento del array será la componente 0.
- Indexación base-uno (1): el primer elemento tiene el índice 1.
- Indexación base-n (n): el índice del primer elemento puede ser elegido libremente.

En JavaScript cuando trabajamos con índices numéricos utilizaremos la indexación base-cero(0).

Para saber más: sobre los tipos de estructuras de datos (generales, no para JavaScript en concreto).

1.1.- Objeto Array.

Los arrays en JavaScript son muy versátiles, ya que podemos almacenar diferentes tipos de datos en cada posición del Array. **Un array se define como una colección ordenada de datos**. Lo mejor es que pienses en un array **como si fuera una tabla que contiene datos**, o también como si fuera una hoja de cálculo.

Ya vimos que JavaScript usa internamente algunos arrays para gestionar los objetos HTML en el documento, propiedades del navegador, etc. **Las colecciones anchors[], forms[], links[] e images[] son arrays internos de JavaScript** en el navegador que te permiten acceder a todas las anclas (hiperenlaces), formularios, enlaces e imágenes del documento.

Tendrás que identificar las pistas que te permitan utilizar arrays para almacenar datos. Por ejemplo, imagínate que tienes que almacenar un montón de coordenadas geográficas de una ruta a caballo. Este caso es un buen candidato para almacenar los datos en un array.

Autoevaluación

A la hora de referenciar las posiciones de un array en JavaScript, que estén definidas de forma numérica, la primera posición de ese array comenzará con el índice 0.

- (•) Verdadero.
- ( ) Falso.

Es correcto, la primera posición en un array será la posición 0.

Creación de un Array

```
var misCoches=new Array();
misCoches[0]="Seat";
misCoches[1]="BMW";
misCoches[2]="Audi";
misCoches[3]="Toyota";
```

```
var datos = [
          ["Cristina","Seguridad",24],
          ["Catalina","Bases de Datos",17],
          ["Vieites","Sistemas Informáticos",28],
          ["Benjamin","Redes",26]
      ];
```

Recorrido de elementos

Utilizando un bucle for:

```
for (i=0;i<sistemaSolar.length;i++)
     document.write(sistemaSolar[i] + "<br/>");
```

Utilizando un bucle while:

```
var i=0;
while (i < sistemaSolar.length)
{
     document.write(sistemaSolar[i] + "<br/>");
     i++;
}
```

Empleando for each in (disponible en JavaScript 1.6 o superiores):

```
for each (var planeta in sistemaSolar)
     document.write(planeta+"<br/>");
```

Otro ejemplo:

	`[document.write(planeta +" ") for each (planeta in sistemaSolar)];`
Borrado de elementos	

Autoevaluación

Si accedemos a una posición de un array que no está definida, obtendremos el valor:

- ○ Error en tiempo de ejecución y se detiene la aplicación.
- ◉ Undefined.
- ○ No devuelve ningún valor.

Correcto. Muy bien, ya que esa posición no está definida y ese es el mensaje que devolverá el intérprete de JavaScript al intentar acceder a esa posición.

1.1.1.- Propiedades y métodos

Propiedades del objeto Array

Propiedad	Descripción
constructor	Devuelve la función que creó el prototipo del objeto array.
length	Ajusta o devuelve el nº de elementos en un array.
prototype	Te permite añadir propiedades y métodos a un objeto.

Métodos del objeto Array

Métodos	Descripción
concat(array1, array2...)	Une dos o más arrays, y devuelve una copia de los arrays unidos.
join()	Une todos los elementos de un array en una cadena de texto.
shift()	Elimina el primer elemento de un array, y devuelve ese elemento.
pop()	Elimina el último elemento de un array y devuelve ese elemento.
push(item)	Añade nuevos elementos al final de un array, y devuelve la nueva longitud.

reverse()	Invierte el orden de los elementos en un array.
	Por ejemplo:
	`<script type="text/javascript">`
	`var frutas = ["Plátano", "Naranja", "Manzana", "Melocotón"];`
	`document.write(frutas.reverse());  // Imprimirá: Melocotón,Manzana,Naranja,Plátano`

	`</script>`
slice(posicion1, posicion2…)	Selecciona una parte de un array y devuelve el nuevo array. **Por ejemplo**: `<script type="text/javascript">` `var frutas = ["Plátano", "Naranja", "Manzana", "Melocotón"];` `document.write(frutas .slice(0,1) + " ");  // imprimirá: Plátano` `document.write(frutas .slice(1) + " ");  // imprimirá: Naranja,Manzana,Melocotón` `document.write(frutas .slice(-2) + " ");  // imprimirá: Manzana, Melocotón` `document.write(frutas + " ");  // imprimirá: Plátano,Naranja,Manzana,Melocotón` `</script>`
sort()	Ordena los elementos de un array.
splice(index, howManyRemove, item1, …, itemX)	Añade/elimina elementos a un array.
toString()	Convierte un array a una cadena y devuelve el resultado.
unshift(item1, …, itemX)	Añade nuevos elementos al comienzo de un array, y devuelve la nueva longitud.

valueOf()	Devuelve el valor primitivo de un array.
	Array.valueOf() es equivalente a array.
	Por ejemplo:
	`const fruits = ["Banana", "Orange", "Apple", "Mango"];`
	`const myArray = fruits; //= myArray = fruits.valueOf();`

Debes conocer: JavaScript Array Reference (w3schools.com)

1.2.- Arrays paralelos.

En algunos casos, podría ser muy útil hacer las búsquedas en varios arrays a la vez, de tal forma que podamos **almacenar diferentes tipos de información, en arrays que estén sincronizados**. Por ejemplo:

```
var profesores = ["Cristina","Catalina","Vieites","Benjamin"];
var asignaturas=["Seguridad","Bases de Datos","Sistemas Informáticos","Redes"];
var alumnos=[24,17,28,26];
```

Usando estos tres arrays de forma sincronizada, podemos decir que el índice = 3 es Benjamín, profesor de Redes, y que tiene 26 alumnos en clase.

Para que los arrays paralelos sean homogéneos, **han de tener la misma longitud** para mantener la consistencia de la estructura lógica creada. **Ventajas del uso de arrays paralelos**:

- **Se pueden usar en lenguajes que soporten solamente arrays**, como tipos primitivos y no registros (como puede ser JavaScript).
- Son **fáciles de entender y utilizar**.
- Pueden **ahorrar una gran cantidad de espacio**, en algunos casos **evitando complicaciones de sincronización**.

- El **recorrido secuencial** de cada posición del array, es **extremadamente rápido** en las máquinas actuales.

1.3.- Arrays multidimensionales.

Si bien es cierto que en JavaScript los arrays son uni-dimensionales, **podemos crear arrays que en sus posiciones contengan otros arrays u otros objetos**. Podemos crear de esta forma arrays bidimensionales, tridimensionales, etc.

Por ejemplo, un array bidimensional:

```
var datos = new Array();
datos[0] = new Array("Cristina","Seguridad",24);
datos[1] = new Array("Catalina","Bases de Datos",17);
datos[2] = new Array("Vieites","Sistemas Informáticos",28);
datos[3] = new Array("Benjamin","Redes",26);
```

O bien usando una definición más breve y literal del Array:

```
var datos = [
          ["Cristina","Seguridad",24],
          ["Catalina","Bases de Datos",17],
          ["Vieites","Sistemas Informáticos",28],
          ["Benjamin","Redes",26]
      ];
```

Para acceder a un dato particular, se requiere hacer una **doble referencia**. Ejemplo:

```
document.write("<br/>Quien imparte Bases de Datos? "+datos[1][0]);          // Catalina
document.write("<br/>Asignatura de Vieites: "+datos[2][1]);               // Sistemas
Informaticos
document.write("<br/>Alumnos de Benjamin: "+datos[3][2]);          // 26
```

Si queremos imprimir toda la información de un array multidimensional:

```
document.write("<table border=1>");
for (i=0;i<datos.length;i++)
{
     document.write("<tr>");
     for (j=0;j<datos[i].length;j++)
     {
          document.write("<td>"+datos[i][j]+"</td>");
     }
     document.write("</tr>");
}
document.write("</table>");
```

2.- Creación de funciones.

Caso práctico: vamos a ver cómo crear funciones, cómo pasar parámetros, el ámbito de las variables dentro de las funciones, cómo anidar funciones y las funciones predefinidas de JavaScript.

Una función es la definición de un conjunto de acciones pre-programadas. Las funciones se llaman a través de eventos o bien mediante comandos desde nuestro script.

En JavaScript no vamos a distinguir entre **procedimientos** (que ejecutan acciones), o **funciones** (que ejecutan acciones y devuelven valores). **En JavaScript siempre se llamarán funciones**.

Una función es capaz de **devolver un valor a la instrucción que la invocó**, pero esto no es un requisito obligatorio en JavaScript. **Cuando una función devuelve un valor**, la instrucción que llamó a esa función, **la tratará como si fuera una expresión**.

```
function nombreFunción ( [parámetro1]....[parámetroN] )
{
     // instrucciones
     return valor; //Solo si deseamos que devuelva algún valor.
}
```

Buenas prácticas

- Siempre que sea posible, tienes que diseñar funciones que puedas reutilizar en otras aplicaciones.
- Los nombres de las funciones deben ser nombres que realmente identifiquen al tipo de acción que realiza.
  - No debe empezar por mayúscula.
  - Se prefiere *chequearMail* a *chequear_mail*.
- Las funciones deben ser muy específicas, no realizar tareas adicionales a las inicialmente propuestas en esa función.

Ejemplos:

```
nombreFuncion( );  // Esta llamada ejecutaría las instrucciones programadas dentro de la función.

variable=nombreFuncion( );  // En este caso la función devolvería un valor que se asigna a la variable.
```

Las funciones en JavaScript también **son objetos**, **y como tal tienen métodos y propiedades**. **Por ejemplo**: Un método, aplicable a cualquier función puede ser **toString**(), el cuál **nos devolverá el código fuente de esa función**.

2.1.- Parámetros

Ejemplo 1:

Declaración de la función

```
function saludar(a,b)
{
     alert("Hola " + a + " y "+ b +".");
}
```

Uso de la función

```
saludar("Martin","Silvia");      //Mostraría una alerta con el texto: Hola Martin y Silvia.
```

Ejemplo 2:

Declaración de la función	`function devolverMayor(a,b)`
	`{`
	`  if (a > b)`
	`  return a;`
	`  else`
	`  return b;`
	`}`

Uso de la función	`document.write ("El número mayor entre 35 y 21 es el: " + devolverMayor(35,21) + ".");`

Reglas:

- **En la definición de parámetros no se usa la palabra reservada var para inicializar las variables**. Esos parámetros a y b serán variables locales a la función.
- También podemos acceder a los argumentos a través del **array arguments**.
- **En el estándar ES6, se admiten los parámetros con valor por defecto**, para que en caso de que no se le pase a la función ningún valor para dichos parámetros, tenga al menos el valor por defecto. **Ejemplo**

```
function devolverMayor(a=3,b=2)
        {
             if (a > b)
                  return a;
             else
                  return b;
        }
    document.write(devolverMayor())
```

Citas para pensar: "La inteligencia es la función que adapta los medios a los fines."HARTMANN, N

Para saber más: Funciones - JavaScript | MDN (mozilla.org)

2.2.- Ámbito de las variables.

Las variables que se definen fuera de las funciones se llaman **variables globales.** Las **variables** que se definen dentro de las funciones, con la palabra reservada **var**, se llaman variables locales.

Para un script de JavaScript, el alcance de una variable global, se limita al documento actual que está cargado en la ventana del navegador o en un frame.

En el momento que una página se cierra, todas las variables definidas en esa página se eliminarán de la memoria para siempre. Si necesitas que el valor de una variable persista de una página a otra, tendrás que utilizar técnicas que te permitan almacenar esa variable (como las cookies, o bien poner esa variable en el documento frameset, etc.).

Una **variable local será definida dentro de una función**. Antes viste que podemos definir variables en los parámetros de una función (sin usar **var**), pero también podrás definir nuevas variables dentro del código de la función. En este caso, **sí que se requiere el uso de la palabra**

reservada var (o let) cuando definimos una variable local, ya que de otro modo, esta variable será reconocida como una variable global.

El alcance de una variable local está solamente dentro del ámbito de la función. Ninguna otra función o instrucciones fuera de la función podrán acceder al valor de esa variable.

Buenas prácticas

Reutilizar el nombre de una variable global como local es uno de los errores más sutiles y por consiguiente, más difíciles de encontrar en el código de JavaScript. **La variable local en momentos puntuales ocultará el valor de la variable global, sin avisarnos de ello**.

Como recomendación, **no reutilices un nombre de variable global como local en una función**, y tampoco declares una variable global dentro de una función, ya que podrás crear fallos que te resultarán difíciles de solucionar.

2.3.- Funciones anidadas.

Los navegadores más modernos nos proporcionan la opción de anidar unas funciones dentro de otras. Es decir, podemos programar una función dentro de otra función.

Con las funciones anidadas, podemos encapsular la accesibilidad de una función dentro de otra y hacer que esa función sea privada o local a la función principal.

Buenas prácticas

- **Tampoco es recomendable** reutilizar nombres de funciones con esta técnica, para evitar problemas o confusiones posteriores.
- Una buena opción para aplicar las funciones anidadas, es **cuando tenemos una secuencia de instrucciones que necesitan ser llamadas desde múltiples sitios dentro de una función**, y esas instrucciones sólo tienen significado dentro del contexto de esa función principal

```
function  principalA()
{
// instrucciones
     function  internaA1()
     {
          // instrucciones
     }
// instrucciones
}
function  principalB()
{
     // instrucciones
     function  internaB1()
     {
          // instrucciones
     }
     function  internaB2()
     {
          // instrucciones
     }
     // instrucciones
}
```

2.4.- Funciones predefinidas del lenguaje.

En JavaScript, disponemos de algunos elementos que necesitan ser tratados a escala global y que no pertenecen a ningún objeto en particular (o que se pueden aplicar a cualquier objeto).

Propiedades globales en JavaScript

Propiedad	Descripción
Infinity	Valor numérico que representa el infinito positivo/negativo.
NaN	Indica que el valor no es numérico "Not a Number".
undefined	Indica que a esa variable no le ha sido asignado un valor.

Funciones globales o predefinidas en JavaScript

Función	Descripción
decodeURI()	Decodifica los caracteres especiales de una URL excepto: **, / ? : @ & = + $ #**
decodeURIComponent()	Decodifica todos los caracteres especiales de una URL.
encodeURI()	Codifica los caracteres especiales de una URL excepto: **, / ? : @ & = + $ #**
encodeURIComponent()	Codifica todos los caracteres especiales de una URL.
escape()	Codifica caracteres especiales en una cadena, excepto: *** @ - _ + . /**
eval()	Evalúa una cadena y la ejecuta si contiene código u operaciones.
isFinite()	Determina si un valor es un número finito válido.
isNaN()	Determina cuando un valor no es un número.
Number()	Convierte el valor de un objeto a un número.
parseFloat()	Convierte una cadena a un número real.
parseInt()	Convierte una cadena a un entero.
unescape()	Decodifica caracteres especiales en una cadena, excepto: *** @ - _ + . /**

Ejemplo de uso de encode y decode URI:

```
const url = 'https://www.twitter.com'

console.log(encodeURI(url))             //https://www.twitter.com
```

```
console.log(encodeURIComponent(url))    //https%3A%2F%2Fwww.twitter.com


const paramComponent = '?q=search'
console.log(encodeURIComponent(paramComponent)) //"%3Fq%3Dsearch"
console.log(url + encodeURIComponent(paramComponent))
//https://www.twitter.com%3Fq%3Dsearch
```

Ejemplo de la función eval():

```
<script type="text/javascript">
     eval("x=50;y=30;document.write(x*y)");     // Imprime 1500
     document.write("<br />" + eval("8+6"));      // Imprime 14
     document.write("<br />" + eval(x+30));       // Imprime 80
</script>
```

Debes conocer: más información sobre las funciones predefinidas de JavaScript.

3.- Creación de objetos a medida.

JavaScript te da la oportunidad de crear tus propios objetos en memoria, con propiedades y métodos que tú puedes definir a tu antojo.

También hay que dejar claro que, **JavaScript no es un lenguaje orientad a objetos de verdad en sentido estricto**. Se considera un **lenguaje basado en objetos**. La diferencia entre orientado a objetos y basado en objetos es significante, y tiene que ver sobre todo en cómo los objetos se pueden extender.

- **JavaScript soporta clases, interfaces, herencia, etc**.
- **Un objeto en JavaScript es realmente una colección de propiedades**. Las propiedades pueden tener forma de datos, tipos, funciones (métodos) o incluso otros objetos.
- Una función contenida en un objeto se conoce como un método.
- La conexión entre propiedades y métodos es uno de los ejes centrales de la orientación a objetos.
- Los objetos se crean empleando la sintaxis **new Constructor()**.

Ejemplo de un constructor:

```
function Coche( )
{
     // propiedades y métodos
}
```

Los nombres de los constructores se ponen generalmente con las iniciales de cada palabra, y cuando creamos un objeto con ese constructor (**instancia de ese objeto**), lo haremos empleando minúsculas al principio. **Por ejemplo**: **var unCoche = new Coche( );**

Para saber más: sobre creación de objetos,

3.1.- Definición de propiedades.

- **Las propiedades para nuestro objeto se crearán dentro del constructor** empleando para ello la palabra reservada **this**.

```
function Coche( )
{
     // Propiedades
     this.marca =  "Audi A6";
     this.combustible =  "diesel";
     this.cantidad = 0;          // Cantidad de combustible en el depósito.
}
```

- La palabra reservada **this**, se utiliza **para hacer referencia al objeto actual**, que en este caso será el objeto que está siendo creado por el constructor.
- Podemos pasarle argumentos al constructor si lo definimos así:

```
function Coche(marca, combustible)
{
     // Propiedades
     this.marca =  marca;
     this.combustible =  combustible;
     this.cantidad = 0;          // Cantidad de combustible inicial por defecto en el
depósito.
}
```

- De esta forma, **podemos acceder a las propiedades de esos objetos**:

```
document.write("<br/>El coche de Martin es un: "+cocheDeMartin.marca+" a
"+cocheDeMartin.combustible);
document.write("<br/>El coche de Silvia es un: "+cocheDeSilvia.marca+" a "+
cocheDeSilvia.combustible);
// Imprimirá:
// El coche de Martin es un: Volkswagen Golf a gasolina
// El coche de Silvia es un: Mercedes SLK a diesel
// Ahora modificamos la marca y el combustible del coche de Martin:
cocheDeMartin.marca = "BMW X5";
cocheDeMartin.combustible = "diesel";
document.write("<br/>El coche de Martin es un: " + cocheDeMartin.marca + " a " +
cocheDeMartin.combustible);
// Imprimirá: El coche de Martin es un: BMW X5 a diesel
```

3.2.- Definición de métodos.

Los **métodos son funciones que se enlazan a los objetos**, para que dichas funciones puedan acceder a las propiedades de los mismos.

Primera forma de incluir métodos en objetos (no es la mejor práctica)

---

1. **Definimos la función de manera global**, usando this para referirnos a una propiedad del objeto al que pertenecerá:

```
function rellenarDeposito (litros)
{
     // Modificamos el valor de la propiedad cantidad de combustible
     this.cantidad = litros;
```

```
}
```

2. Conectamos la función al objeto de la siguiente manera (Aquí se puede ver de forma ilustrada que los métodos de los objetos son en realidad propiedades):

```
function Coche(marca, combustible)
{
     // Propiedades
     this.marca =  marca;
     this.combustible =  combustible;
     this.cantidad = 0;          // Cantidad de combustible inicial por defecto en el
depósito.
     // Métodos
     this.rellenarDeposito = rellenarDeposito; // Creamos una propiedad rellenarDepostio a
la que se le asocia el método rellenarDeposito
}
```

2ª forma: la mejor práctica de programación

1. **Definir el contenido de la función rellenarDeposito dentro del constructor**, ya que de esta forma los métodos al estar programados a nivel local aportan mayor privacidad y seguridad al objeto en general.

```
function Coche(marca, combustible)
{
     // Propiedades
     this.marca =  marca;
     this.combustible =  combustible;
     this.cantidad = 0;
     // Métodos
     this.rellenarDeposito = function (litros)
     {
          this.cantidad=litros;
     };
}
```

3.3.- Definición de objetos literales.

Otra forma de definir objetos es hacerlo de forma literal.

Un **literal** es un valor fijo que se especifica en JavaScript. Un objeto literal será un conjunto, de cero o más parejas del tipo **nombre:valor**. **Dos formas de declararlo**:

```
avion = { marca:"Boeing" , modelo:"747" , pasajeros:"450" };
```

Es equivalente a:

```
var avion = new Object();
avion.marca = "Boeing";
avion.modelo = "747";
avion.pasajeros = "450";
```

Para referirnos desde JavaScript a una propiedad del objeto *avión*, **podríamos hacerlo con las siguientes sintaxis** (ambas son válidas):

```
document.write(avion.marca);  // o también se podría hacer con:
document.write(avion["modelo"]);
```

Podríamos tener un **conjunto de objetos literales simplemente creando un array** que contenga en cada posición una definición de objeto literal:

```
var datos=[
{"id":"2","nombrecentro":"IES A Piringalla" ,"localidad":"Lugo","provincia":"Lugo"},
{"id":"10","nombrecentro":"IES As Fontiñas","localidad":"Santiago","provincia":"A
Coruña"},
{"id":"9","nombrecentro":"IES As Lagoas","localidad":"Ourense","provincia":"Ourense"},
{"id":"8","nombrecentro":"IES Cruceiro Baleares","localidad":"Culleredo","provincia":"A
Coruña"},
{"id":"6","nombrecentro":"IES Cruceiro Baleares","localidad":"Culleredo","provincia":"A
Coruña"},{"id":"4","nombrecentro":"IES de
Teis","localidad":"Vigo","provincia":"Pontevedra"},
{"id":"5","nombrecentro":"IES Leliadoura","localidad":"Ribeira","provincia":"A Coruña"},
{"id":"7","nombrecentro":"IES Leliadoura","localidad":"Ribeira","provincia":"A Coruña"},
{"id":"1","nombrecentro":"IES Ramon Aller
Ulloa","localidad":"Lalin","provincia":"Pontevedra"},
{"id":"3","nombrecentro":"IES San Clemente","localidad":"Santiago de
Compostela","provincia":"A Coruña"}
];
```

De la siguiente forma se podría recorrer el array de datos para mostrar su contenido:

```
for (var i=0; i< datos.length; i++)
{
     document.write("Centro ID: "+datos[i].id+" - ");
     document.write("Nombre: "+datos[i].nombrecentro+" - ");
     document.write("Localidad: "+datos[i].localidad+" - ");
     document.write("Provincia: "+datos[i].provincia+"<br/>");
}
```

3.4.- Definición de clases con ES6.

A partir la versión ES6 podemos crear clases utilizando la palabra reservada class.

- La creación de atributos sigue siendo necesario hacerla dentro del método **constructor**:

```
class Coche {
  constructor(marca,combustible){
    this.marca=marca;
    this.combustible=combustible;
    this.cantidad=0;
  }
}

var micoche = new Coche("Ford","Diesel");
```

- **Para definir los métodos no se utiliza la palabra function**:

```
class Coche {
  constructor(marca,combustible){
    this.marca=marca;
    this.combustible=combustible;
    this.cantidad=0;
  }
  rellenarDeposito(litros){
    this.cantidad=litros;
```

```
  }
}

var micoche = new Coche("Ford","Diesel");
micoche.rellenarDeposito(12);
console.log(micoche.cantidad);//muestra 12
```

JavaScript también soporta herencia con la palabra clave **extends**.

Para saber más: sobre clases y subclases.

Las **declaraciones de funciones** son alojadas y las **declaraciones de clases** no lo son. En primer lugar necesitas declarar tu clase y luego acceder a ella, de otro modo el ejemplo de código siguiente arrojará un ReferenceError:

```
const p = new Rectangle(); // ReferenceError

class Rectangle {}
```

Bibliografía.

A continuación, presento la relación bibliográfica que he consultado para la realización de este trabajo.

No hay ninguna fuente en el documento actual.