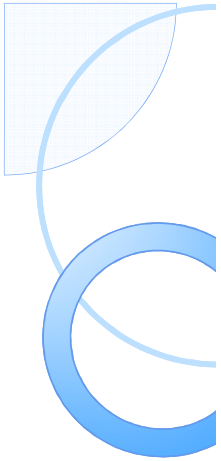
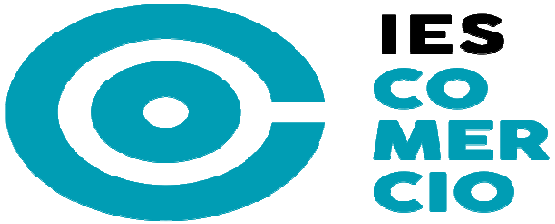




Tema 5:
Programación con estructuras definidas por el usuario

Desarrollo Web en Entorno Cliente

Objetivos

- Conocer las principales funciones predefinidas del lenguaje JavaScript.
- Crear funciones personalizadas para realizar tareas específicas.
- Comprender el objeto *Array* y familiarizarse con sus propiedades y métodos.
- Crear objetos personalizados.
- Definir propiedades y métodos de los objetos personalizados.

Contenidos

1. Funciones predefinidas del lenguaje.
2. Funciones del usuario.
3. Objeto Array.
4. Objetos definidos por el usuario.

1. Funciones predefinidas del lenguaje.

- JavaScript cuenta con una serie de funciones integradas en el lenguaje.
- Dichas funciones se pueden utilizar sin tener que declararlas previamente y sin conocer todas las instrucciones que ejecuta.
- Simplemente se debe conocer el nombre de la función y el resultado que se obtiene al utilizarla.

1. Funciones predefinidas del lenguaje.

- Algunas de las principales funciones predefinidas de JavaScript son:

Función predefinida	Descripción
escape(string)	Recibe como argumento una cadena de caracteres y devuelve esa misma cadena sustituida con su codificación ASCII.
eval(string)	Convierte una cadena que pasamos como argumento en código JavaScript ejecutable.
isFinite(numero)	Verifica si un número que pasamos como argumento es o no un número finito.
isNaN()	Comprueba si el valor que le pasamos como argumento es un tipo numérico.
Number()	Convierte el objeto pasado como argumento en un tipo numérico.
parseInt(string)	Convierte la cadena que pasamos como argumento en un valor numérico de tipo entero.
parseFloat(string)	Convierte la cadena que pasamos como argumento en un valor numérico de tipo flotante.
String(objeto)	Convierte el objeto pasado como argumento en una cadena de texto.
unescape(string)	Hace la función inversa e escape(string), es decir, descodifica la cadena.

- Las vemos a continuación:

1. Funciones predefinidas del lenguaje.

- `escape`(): recibe como argumento una cadena de caracteres y devuelve esa misma cadena sustituida con su codificación en ASCII.
  Ejemplo: el carácter (`?`) devolvería la codificación `%3F`.

```
<script type="text/javascript">
  var input = prompt("Introduce una cadena");
  var inputCodificado = escape(input);
  alert("Cadena codificada: " + inputCodificado);
</script>
```

- `unescape`(): es la función opuesta a `escape`(), es decir decodifica los caracteres que estén codificados.

1. Funciones predefinidas del lenguaje.

- `eval()`: convierte una cadena que pasamos como argumento en código JavaScript ejecutable.

  Tiene como argumento una expresión y devuelve el valor de la misma para poder ser ejecutada como código JavaScript.

```
<script type="text/javascript">
  var input = prompt("Introduce una operación numérica");
  var resultado = eval(input);
  alert ("El resultado de la operación es: " + resultado);
</script>
```

1. Funciones predefinidas del lenguaje.

- `isFinite()`: verifica si el número que pasamos como argumento es o no un número finito.

Un valor se define como finito si se encuentra en el rango $\pm 1.79769313486231570^{308}$. Si el argumento no se encuentra en este rango la función devuelve `false`, en caso contrario devuelve `true`.

```
if(isFinite(argumento)) {
  //instrucciones si el argumento es un número finito
}else{
  //instrucciones si el argumento no es un número finito
}
```

1. Funciones predefinidas del lenguaje.

- `isNan()`: comprueba si el valor que pasamos como argumento no es de tipo numérico. (*Is Not a Number*).

```
<script type="text/javascript">
  var input = prompt("Introduce un valor numérico: ");
  if (isNaN(input)){
    alert("El dato ingresado no es numérico.");
  }else{
    alert("El dato ingresado es numérico.");
  }
</script>
```

1. Funciones predefinidas del lenguaje.

- `String()`: convierte el objeto pasado como argumento en una cadena de texto que represente el valor de dicho objeto.

```
<script type="text/javascript">
  var fecha = new Date()
  var fechaString = String(fecha)
  alert("La fecha actual es: "+fechaString);
</script>
```

1. Funciones predefinidas del lenguaje.

- `Number()`: convierte el objeto pasado como argumento en un número que represente el valor de dicho objeto.

  Si la conversión falla, la función devuelve `NaN` (Not a Number).

```
1  Number("123")     // 123
2  Number("")        // 0
3  Number("0x11")    // 17
4  Number("0b11")    // 3
5  Number("0o11")    // 9
6  Number("foo")     // NaN
7  Number("100a")    // NaN
```

1. Funciones predefinidas del lenguaje.

- `parseInt()`: convierte la cadena que pasamos como argumento en un valor numérico de tipo entero con una base especificada: decimal (10) si no especificamos la base, binaria (2), octal(8) o hexadecimal (16).

```
parseInt(string, base);
```

Si la función encuentra algún carácter no numérico, devuelve el valor encontrado hasta ese punto. Si el primer valor es no numérico, devuelve `NaN`.

```
<script type="text/javascript">
  var input = prompt("Introduce un valor: ");
  var inputParsed = parseInt(input);
  alert("parseInt("+input+"): "+inputParsed);
</script>
```

1. Funciones predefinidas del lenguaje.

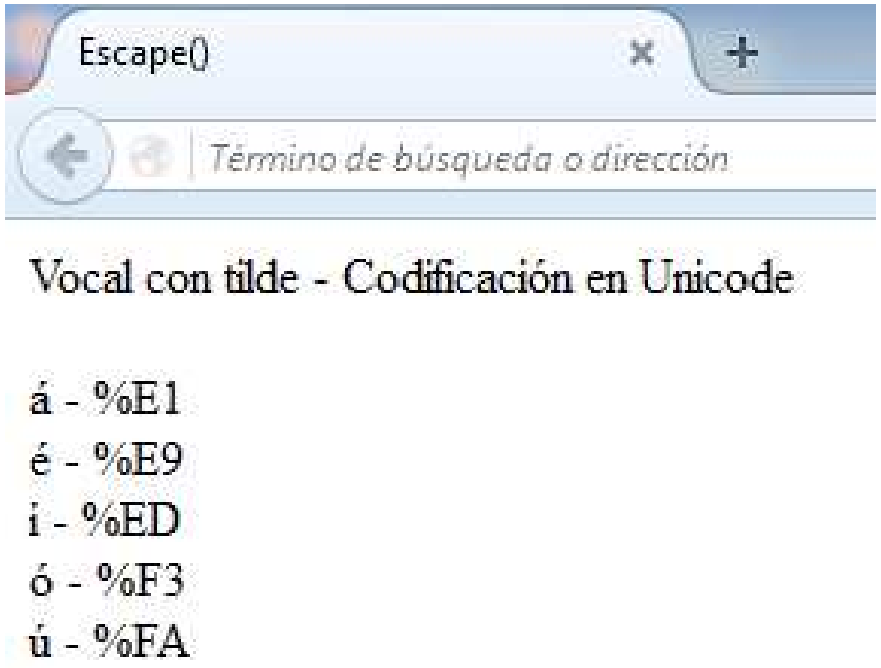
- `parseFloat()`: convierte la cadena que pasamos como argumento en un valor numérico de tipo flotante.

  Si la función encuentra algún carácter que no corresponda al formato decimal, devuelve el valor encontrado hasta ese punto. Si el primer valor es no numérico, devuelve `NaN`.

```
<script type="text/javascript">
  var input = prompt("Introduce un valor: ");
  var inputParsed = parseFloat(input);
  alert("parseFloat("+input+"): " + inputParsed);
</script>
```

Actividad 1

Crea un script que muestre por pantalla la codificación de todas las vocales con tilde.



2. Funciones del usuario

- Es posible crear funciones personalizadas, diferentes a las funciones predefinidas por el lenguaje.
- Con estas funciones se pueden realizar las tareas que queramos.
- Esa tarea se realiza mediante un grupo de instrucciones relacionadas a las cuales debemos dar un nombre, el nombre de la función.

2. Funciones del usuario

- Definición de funciones:
  - El mejor lugar para definir las funciones es dentro de las etiquetas HTML `<head>` y `</head>`.
  - El motivo es que el navegador carga siempre todo lo que se encuentra entre estas etiquetas.
  - La definición de una función consta de cinco partes:
    - La palabra clave `function`.
    - El nombre de la función.
    - Los argumentos utilizados.
    - El grupo de instrucciones.
    - La palabra clave `return`.

2. Funciones del usuario

- Definición de funciones – Sintaxis:

```
function nombre_función ([argumentos])
{
  grupo_de_instrucciones;
  [return valor;]
}
```

- `Function`:
  - Es la palabra clave que se debe utilizar antes de definir cualquier función.

2. Funciones del usuario

- Definición de funciones – Nombre:
  - El nombre de la función se sitúa al inicio de la definición y antes del paréntesis que contiene los posibles argumentos.
    - Deben usarse sólo letras, números o el carácter de subrayado.
    - Debe ser único en el código JavaScript de la página web.
    - No pueden empezar por un número.
    - No puede ser una de las palabras reservadas del lenguaje.

2. Funciones del usuario

- Definición de funciones – Argumento:
  - Los argumentos se definen dentro del paréntesis situado después del nombre de la función y separados por comas.
  - No todas las funciones requieren argumentos, con lo cual el paréntesis se deja vacío.

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

  - No usan la palabra reservada var y serán variables locales a la función.
  - JavaScript no muestra ningún error si se pasan más o menos argumentos de los necesarios. Si se pasan más, los argumentos sobrantes se ignoran y si se pasan menos, al resto se les asigna valor indefinido.

2. Funciones del usuario

- Definición de funciones – Argumentos:
  - Como el número de argumentos que se pasa a una función de JavaScript puede ser variable e independiente del número de argumentos incluidos en su definición, JavaScript proporciona una variable especial que contiene todos los parámetros con los que se ha invocado la función.
  - Se trata de un array `arguments` y solamente está definido dentro de cualquier función.

```
function suma (a,b){
  alert (arguments.length);
  alert (arguments[2]);
  return a + b;
}
suma (3,5);
```

2. Funciones del usuario

- Ámbito de las variables:
  - Las variables que se definen fuera de las funciones serán **variables globales**.
  - Las variables que se definen dentro de las funciones, con la palabra reservada `var` son **variables locales**.
  - Si dentro de una función definimos una variable sin la palabra reservada `var`, estamos definiendo una variable global, no una variable local.

2. Funciones del usuario

- Ámbito de las variables:
  - El alcance de una **variable global** se limita al documento actual que está cargado en la ventana del navegador.
  - Cuando se inicializa una variable como variable global, quiere decir que todas las instrucciones de tu script, tendrán acceso directo al valor de esa variable. Todas las instrucciones podrán leer y modificar el valor de esa variable global.

2. Funciones del usuario

- Ámbito de las variables:
  - En contraste, una **variable local** será definida dentro de una función. Y aunque hemos visto que podemos definir variables sin usar la palabra reservada `var`, en este caso, si que se requiere para definir una variable local. Ya que de otro modo, sería reconocida como una variable global.
  - El alcance de una variable local está solamente dentro del ámbito de la función.

2. Funciones del usuario

- Ejemplo de variables locales y globales:

```
//variables globales
var chica = “Maria”;
var perros = “Bat, Samba y Black”;
function demo(){
  var chica = “Sonia”; //vble. Local
  document.write(“<br>”+perros+ “ no
pertenecen a  ”+chica+ “.”);
}
//llamamos a la función para usar las variables locales
demo();
document.write(“<br>”+perros+
“pertenecen a  ”+chica+ “.”);
```

2. Funciones del usuario

- Definición de funciones – Grupo de instrucciones:
  - El grupo de instrucciones es el bloque de código JavaScript que se ejecuta cuando invocamos a la función desde otra parte de la aplicación.
  - Las llaves ({}) delimitan el inicio y el fin de las instrucciones.

2. Funciones del usuario

- Definición de funciones – `Return`:
  - La palabra clave `return` es opcional en la definición de una función.
  - Se utilizará para indicar que la función declarada devuelve un valor.

```
function Mayor(a,b){
  if (a>b) then
    return a;
  else
    return b;
}
var elMayor = Mayor(14,21);
```

  - Si no se indica el nombre de ninguna variable, JavaScript no muestra ningún error pero el valor devuelto se pierde.

2. Funciones del usuario

- Ejemplo – Función que calcula el importe de un producto después de haberle aplicado el IVA:

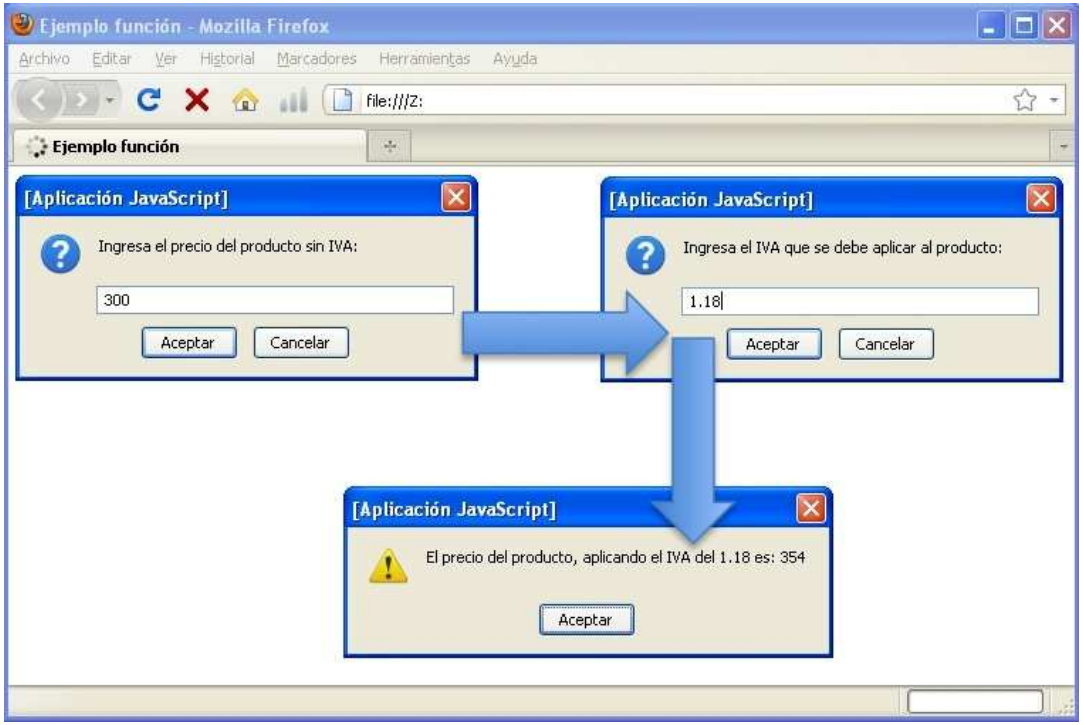
```
function aplicar_IVA(valorProducto, IVA){
  var productoConIVA = valorProducto * IVA;
  alert("El precio del producto, aplicando
    el IVA del  " + IVA + " es: " +
    productoConIVA);
}
```

2. Funciones del usuario

- Invocación de funciones:
  - Una vez definida la función, es necesaria llamarla para que el navegador ejecute el grupo de instrucciones.
  - Se invoca usando su nombre seguido del paréntesis.
  - Si tiene argumentos, se deben especificar en el mismo orden en el que se han definido en la función.

2. Funciones del usuario

- Ejemplo: `aplicar_IVA(300, 1.18).`



2. Funciones del usuario

- Invocar una función desde JavaScript:

```
<html>
<head>
    <title>Invocar función desde JavaScript</title>
    <script type="text/javascript">
      function mi_funcion([args]){
        //instrucciones
      }
    </script>
</head>
<body>
    <script type="text/javascript">
      mi_funcion([args]);
    </script>
</body>
</html>
```

2. Funciones del usuario

- Invocar una función desde HTML:

```
<html>
  <head>
    <title>Invocar función desde JavaScript</title>
    <script type="text/javascript">
      function mi_funcion([args]){
        //instrucciones
      }
    </script>
  </head>

  <body onload="mi_funcion([args])">
  </body>
</html>
```

Actividad 2

Realiza una página que muestre un formulario para la conversión de Euros a Libras y viceversa.

Utiliza la creación de funciones para realizar la tarea.

2. Funciones del usuario

- Funciones anónimas:
  - JavaScript permite definir funciones anónimas, es decir permite realizar la declaración de una función sin darle un nombre y además también es posible asignarla a una variable, a un método, o incluso pasar la función con un parámetro más de otrs función.
  - Ejemplos:

```
var saluda = function (mensaje){
  console.log("Hola " + mensaje);
}
saluda("mundo");

typeof(saluda);   //function
```

2. Funciones del usuario

- Funciones anónimas:

```
function sumar(num1, num2)
{
  return num1 + num2;
}

var SumaDosNums = function (num1, num2)
{
  return num1 + num2;
}
```

2. Funciones del usuario

- Pila de funciones:
  - Cuando se invoca a una función en una expresión, ésta debe esperar a que la función finalice para poder completar la expresión.
  - Cuando dentro de una función se llama a otra, se debe esperar a que finalice la función mas interna para finalizar la que le llamo. Para esto las funciones utilizan la **pila de llamadas**.

2. Funciones del usuario

- Ejemplo:

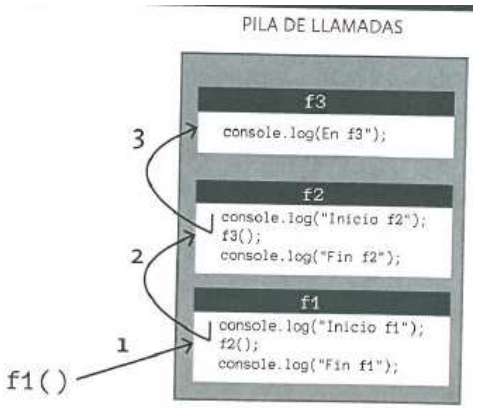
```
function f1(){
      console.log("Inicio f1");
      f2;
      console.log("Fin f1");
}
function f2(){
      console.log("Inicio f2");
      f3;
      console.log("Fin f2");
}
function f3(){
      console.log("En f3");
}
```

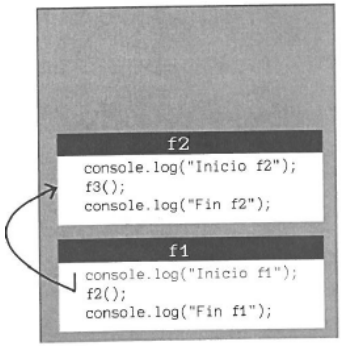
2. Funciones del usuario

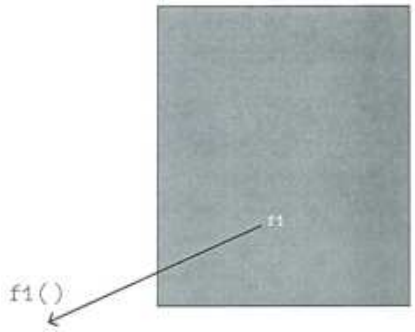
- Ejemplo:

```
function f1(){
          console.log("Inicio f1");
          f2;
          console.log("Fin f1");
}
function f2(){
          console.log("Inicio f2");
          f3;
          console.log("Fin f2");
}
function f3(){
          console.log("En f3");
}
```

Inicio f1
Inicio f2
En f3
Fin f2
Fin f1







2. Funciones del usuario

- Funciones recursivas:
  - Una función recursiva se declara igual que cualquier función, pero en el cuerpo de la misma se realizan llamadas sobre sí misma, donde se modifica alguno de los parámetros, y existe un caso base donde termina la recursión.
  - Ejemplo: Calcular el sumatorio de un número ∑x

```
function sumatorio(x){
  if (x==0) return 0;
  else return x + sumatorio(x-1);
}
```

Actividad 3

Crea una función recursiva que calcule el factorial de un número.

$$n! = n * (n-1)!$$

Crea una función recursiva que muestre la serie de Fibonacci de 15 números.

$$F(n) = \begin{cases} 0 & \text{si } n = 0; \\ 1 & \text{si } n = 1; \\ F(n-1) + F(n-2) & \text{si } n > 1; \end{cases}$$

3. Arrays

- Un array es un conjunto ordenado de valores, homogéneos o no, que están relacionados.
- Cada uno de estos valores se denomina elemento y cada elemento tiene un índice que indica su posición numérica en el array.
- Un array es un objeto más de JavaScript con una serie de funcionalidades.

3. Arrays

- Declaración de arrays:
  - Al igual que ocurre con las variables, es necesario declarar un array antes de poder usarlo.
  - La declaración de un array consta de seis partes:
    - La palabra clave `var`.
    - El nombre del array.
    - El operador de asignación.
    - La palabra clave para la creación de objetos `new`.
    - El constructor `Array`.
    - El paréntesis.

3. Arrays

- Declaración de arrays – Sintaxis:
  - *(1)*:

```
var nombre_del_array = new
Array();
```

  - *(2)*:

```
var nombre_del_array = new
Array(numero_de_elementos);
```

3. Arrays

- Inicialización de arrays:
  - Una vez declarado el array se puede comenzar el proceso de inicializar el array con los elementos que contendrá.
  - La sintaxis es la siguiente:

```
nombre_del_array[indice] = valor_del_elemento;
```

3. Arrays

- Es posible declarar e inicializar al mismo tiempo mediante la escritura de los elementos dentro del paréntesis del constructor, separados por comas (array denso).

```
var marcas_Coche = new Array('Audi', 'Ford',
'Seat', 'Toyota', 'BMW');
```

- También se puede crear sin el constructor, se definen de forma literal:

```
var marcas_Coche = ["Audi", "Ford", "Seat",
"Toyota", "BMW"];


miCoche = marcas_Coche[2]  => // Seat
```

3. Arrays

- **Arrays asociativos o mixtos**: es posible crear arrays donde las posiciones son referenciadas con índices de tipo texto o números.
- Para acceder a sus posiciones utilizaremos el nombre del índice, no su número.
- Para definir un array de tipo asociativo podemos utilizar la notación tradicional o también mediante llaves { }

```
var array_asoc = new Array();
array_asoc['cero'] = "nombre";
array_asoc['uno'] = "email";
alert(array_asoc[cero]);

var datos = {"numero":42, "mes":"Junio", 10:"diez" }
datos[mes]  => // Junio
```

3. Arrays

- Recorrido de un array:
  - Empleando un bucle **for**:

```
var planetas = new Array("Mercurio",
"Venus", "Tierra", "Marte", "Jupiter",
"Saturno", "Urano", "Neptuno", "Pluton");

for (i=0; i<planetas.length; i++){
 document.write(planetas[i] + "<br>");
}
```

  - Empleando un bucle for…in :

```
for(i in planetas) {
    document.write(planetas[i]);
}
```

3. Arrays

- Recorrido de un array:
  - Empleando un bucle **while**:
  - Por ejemplo:

```
var i=0;

while (i < planetas.length){
 document.write(planetas[i] + "<br>");
 i++;
}
```

3. Arrays

- Propiedades de los arrays:
  - El objeto array tiene dos propiedades:
    1. **lenght**: devuelve el número de elementos que contiene el array.

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

    2. **prototype**: para agregar nuevas propiedades y métodos al objeto nativo Array.

```
Array.prototype.nueva_propiedad = valor;
Array.prototype.nuevo_metodo = nombre_de_la_funcion;
```

3. Arrays

- Métodos del objeto array:

Métodos	Descripción	Sintaxis
push()	Añade nuevos elementos al array y devuelve la nueva longitud del array.	`nombre_array.push(valor1, valor2,…)`
concat()	Selecciona un array y lo concatena con otros elementos en un nuevo array.	`nombre_array.concat(valor1, valor2,…)`
join()	Concatena los elementos de un array en una sola cadena separada por un carácter opcional.	`nombre_array.join([separador])`
reverse()	Invierte el orden de los elementos de un array.	`nombre_array.reverse()`
unshift()	Elimina el primer elemento de un array, y devuelve ese elemento.	`nombre_array.shift()`
pop()	Elimina el último elemento de un array.	`nombre_array.pop()`
slice()	Devuelve un nuevo array con un subconjunto de los elementos del array que ha usado el método.	`nombre_array.slice(índice_inicio, [índice_final])`
sort()	Ordena alfabéticamente los elementos de un array podemos definir una nueva función para ordenarlos con otro criterio.	`nombre_array.sort([funcion])`
splice()	Elimina, sustituye o añade elementos del array dependiendo de los argumentos del método.	`nombre_array.splice(inicio,[num_elem_a_borrar],[valor1, valor2,…])`
toString()	Convierte un array a una cadena y devuelve el resultado.	`nombre_array.toString()`
unShift()	Añade nuevos elementos al inicio de un array y devuelve el número de elementos del nuevo array modificado.	`nombre_array.unshift()`

3. Arrays. Métodos

- `push()`:
  - Añade nuevos elementos al final del array y devuelve la nueva longitud del array.

```
<script type="text/javascript">
  var pizzas = new Array("Carbonara", "Quattro_Stagioni",
    "Diavola");
  var nuevo_numero_de_pizzas = pizzas.push("Margherita",
    "Boscaiola"); //añade 2 elementos y devuelve 5
  document.write("Número de pizzas disponibles: " +
    nuevo_numero_de_pizzas + "<br />"); //imprime 5
  document.write(pizzas); //imprime todos los elementos
separados por comas
</script>
```

3. Arrays. Métodos

- `concat()`:
  - Selecciona un array y lo concatena con otros elementos en un nuevo array.

```
<script type="text/javascript">
  var equipos_a = new Array("Real Madrid", "Barcelona",
    "Valencia");
  var equipos_b = new Array("Hércules", "Elche",
    "Valladolid");
  var equipos_copa_del_rey = equipos_a.concat(equipos_b);
  document.write("Equipos que juegan la copa: " +
    equipos_copa_del_rey);
</script>
```

3. Arrays. Métodos

- `join()`:
  - Une todos lo elementos del vector en una sola cadena de texto separada por un carácter opcional, si lo omitimos, lo separa mediante comas (,).

```
<script type="text/javascript">
  var pizzas = new Array("Carbonara", "Quattro_Stagioni",
    "Diavola");
  document.write(pizzas.join(" - "));
</script>
```

Carbonara - Quattro_Stagioni - Diavola

3. Arrays. Métodos

- `reverse()`:
  - Invierte el orden de los elementos de un array.

```
<script type="text/javascript">
  var numeros = new Array(1,2,3,4,5,6,7,8,9,10);
  numeros.reverse();
  document.write(numeros);
</script>
```

  - Resultado: `10,9,8,7,6,5,4,3,2,1`

3. Arrays. Métodos

- `unshift()`:
  - Añade nuevos elementos al inicio de un array y devuelve el número de elementos del nuevo array modificado.
  - La diferencia entre `push()` es que éste los añade al final del array, mientras que `unshift()` los añade al principio.

```
<script type="text/javascript">
  var sedes_JJOO = new Array("Atenas", "Sydney",
    "Atlanta");
  var numero_sedes = sedes_JJOO.unshift("Pekín");
  document.write("Últimas " + numero_sedes + " sedes
    olímpicas: " + sedes_JJOO);
</script>
```

Últimas 4 sedes olímpicas: Pekín,Atenas,Sydney,Atlanta

3. Arrays. Métodos

- `shift()`:
  - Elimina el primer elemento de un array y devuelve dicho elemento.

```
<script type="text/javascript">
  var pizzas = new Array("Carbonara", "Quattro_Stagioni",
    "Diavola");
  var pizza_removida = pizzas.shift();
  document.write("Pizza eliminada de la lista: " +
    pizza_removida + "<br />");
  document.write("Nueva lista de pizzas: " + pizzas);
</script>
```

- Resultado:

Pizza eliminada de la lista: Carbonara
Nueva lista de pizzas: Quattro_Stagioni,Diavola

3. Arrays. Métodos

- `pop()`:
  - Elimina el <u>último</u> elemento de un array y devuelve ese elemento.
  - La diferencia entre `shift()` es que éste elimina el primer elemento del array, mientras que `pop()` elimina y devuelve el último.

```
<script type="text/javascript">
  var premios = new Array("Coche", "Viaje", "1000 Euros");
  var tercer_premio = premios.pop();
  document.write("El tercer premio es: " + tercer_premio +
    "<br />");
  document.write("Quedan los siguientes premios: " +
    premios);
</script>
```

El tercer premio es: 1000 Euros
Quedan los siguientes premios: Coche,Viaje

3. Arrays. Métodos

- `slice()`:
  - Devuelve un nuevo array con un subconjunto de los elementos del array que ha usado el método. `miarray.slice(posicion, elementos);`
  - El número de elementos es un argumento opcional.

```
<script type="text/javascript">
  var numeros = new Array(1,2,3,4,5,6,7,8,9,10);
  var primeros_cinco = numeros.slice(0,5);
  var desde_tercer = numeros.slice(3);
  var ultimos_cuatro = numeros.slice(-4);
  document.write(primeros_cinco + "<br>");
  document.write(desde_tercer + "<br>");
  document.write(ultimos_cuatro);
</script>
```

1,2,3,4,5
4,5,6,7,8,9,10
7,8,9,10

3. Arrays. Métodos

- `sort()`:
  - Ordena alfabéticamente los elementos de un array.

```
<script type="text/javascript">
  var apellidos = new Array("Pérez", "Guijarro", "Arias",
    "González");
  apellidos.sort();
  document.write(apellidos);
</script>
```

  - Resultado: `Arias, González, Guijarro, Pérez`

3. Arrays. Métodos

- `splice()`:
  - Elimina, sustituye o añade elementos del array dependiendo de los argumentos del método.
    - Primer argumento: posición de inicio (índice);
    - Segundo argumento: nº de elemento a eliminar.
    - Tercer argumento (opcional): nuevos elementos para añadir.

```
<script type="text/javascript">
  var coches = new Array("Ferrari", "BMW", "Fiat");
  coches.splice(2,0,"Seat");
  document.write(coches); //Ferrary,BMW,Seat,Fiat
</script>
```

3. Arrays. Métodos

- `splice()`:
  - Otro ejemplo:

```
<script type="text/javascript">
  var oceanos = new Array("Atlantico", "Pacifico",
"Artico", "Mediterraneo", "Indico");
  oceanos.splice(3,1); //eliminamos el tercer elemento
  document.write(oceanos);
  //imprime Atlantico,Pacifico,Artico,Indico
</script>
```

  - La instrucción:

```
delete oceanos[3];  //vers. modernas navegadores
```

Borra el elemento del vector, pero no libera la memoria ocupada ni actualiza la longitud del vector.

```
oceanos[3]    // devolvería undefined.
```

3. Arrays. Métodos

- `splice()`:
  - El método `splice()` está soportado en la mayoría de los navegadores y además dependiendo de la operación realizada, actualiza la longitud del array al nuevo ajuste.
  - El operador `delete` (solo en versiones modernas de navegadores):

```
delete oceanos[3];
```

Borra el elemento del vector, pero no libera la memoria ocupada ni actualiza la longitud del vector.

```
oceanos[3]     // devolvería undefined.
```

3. Arrays.

- **Arrays paralelos**: cuando dos o más arrays, utilizan el mismo índice para referirse a términos homólogos.
  - Ejemplo:

```
<script type="text/javascript">
var profesores = ["Gregorio","Javier","David","Mari
Luz"];
var asignaturas = ["Desarrollo Web en Servidor",
"Despliegue de Aplicaciones", "Diseño de Interfaces",
"Desarrollo Web en Cliente"];
var alumnos = [24,17,28,26];

function imprimeDatos(indice){
   document.write("<br/>"+profesores[indice]+" del
     módulo de "+asignaturas[indice]+",
     tiene"+alumnos[indice]+" alumnos en clase.");
}
for (i=0;i<profesores.length;i++) {
   imprimeDatos(i);
}
</script>
```

3. Arrays.

- Arrays paralelos:
  - Para que los arrays paralelos sean homogéneos, éstos tendrán que tener *la misma longitud*, ya que de esta forma se mantendrá la consistencia de la estructura lógica creada.
  - Resultado:

I.E.S. COMERCIO

Gregorio del módulo de Desarrollo Web en Servidor, tiene 18 alumnos en clase.

David del módulo de Despliegue de aplicaciones, tiene 18 alumnos en clase.

Javier del módulo de Diseño de Interfaces, tiene 14 alumnos en clase.

Mari Luz del módulo de Desarrolo Web en Cliente, tiene 16 alumnos en clase.

3. Arrays.

- **Arrays multidimensionales**: Arrays que en sus posiciones contienen otros arrays. Podemos crear arrays bidimensionales, tridimensionales, etc.
  - Ejemplo: Creando un array bidimensional de dos formas:

```
var datos = new Array();
datos[0] = new Array("Gregorio","Desarrollo Web en
Servidor",18);
datos[1] = new Array("David","Despliegue de
Aplicaciones",18);
datos[2] = new Array("Javier","Diseño de
Interfaces",14);
datos[3] = new Array("Mari Luz","Desarrollo Web en
Cliente",16);
```

```
var datos = [
    ["Gregorio"," Desarrollo Web en Servidor",18],
    ["David"," Despliegue de Aplicaciones",18],
    ["Javier"," Diseño de Interfaces",14],
    ["Mari Luz"," Desarrollo Web en Cliente",16]
];
```

3. Arrays.

- **Arrays multidimensionales**:
  - Para acceder a un dato se necesita hacer una doble referencia para arrays bidimensionales, triple referencia para arrays tridimensionales, etc.

    `nombreArray[indice1][indice2]`
  - Por ejemplo:

```
document.write("<br>Quien imparte Despliegue de
Aplicaciones? " +datos[1][0]); //David
document.write("<br>Asignatura de Nuria: "
+datos[2][1]); // Diseño de Interfaces
document.write("<br>Alumnos de Mari Luz: " +
datos[3][2]); //16
```

3. Arrays.

- **Arrays multidimensionales**:
  - Si quisiéramos imprimir toda la información del array multidimensional, podríamos hacerlo con un bucle for.
  - Por ejemplo:

```
for (i=0; i<datos.length; i++){
document.write("<br/>"+datos[i][0]+" del módulo
de "+datos[i][1]+", tiene "+datos[i][2]+"
alumnos en clase.");
}
```

Obtendríamos como resultado:

Gregorio del módulo de Desarrollo Web en Servidor, tiene 18 alumnos en clase.

David del módulo de Despliegue de aplicaciones, tiene 18 alumnos en clase.

Javier del módulo de Diseño de Interfaces, tiene 14 alumnos en clase.

Mari Luz del módulo de Desarrolo Web en Cliente, tiene 16 alumnos en clase.

Actividad 4

Crea un script que tome una serie de palabras introducidas por el usuario y almacene esas palabras en un array. Posteriormente, manipule ese array para mostrar en una ventana nueva los siguientes datos:

- La primera palabra introducida por el usuario.
- La última palabra introducida por el usuario.
- El número de palabras presentes en el array.
- Todas las palabras ordenadas alfabéticamente.

Actividad 4


Arrays
Término de búsqueda o dirección
Manipular Arrays
Introduce algunas palabras:
perro casa setas pan leche
Aceptar
Manipular arrays - Mozilla Firefox
file:///D:/Users/ESPE/Documents/_COMERCIO/DWEC/ra-ma/Capít
Primera palabra: perro
Última palabra: leche
Número de palabras: 5
Ordenadas alfabéticamente:
casa - leche - pan - perro - setas

4. Objetos definidos por el usuario

- JavaScript proporciona una serie objetos predefinidos, sin embargo es posible crear nuevos objetos definidos por el usuario.
- Cada uno de estos objetos puede tener sus propios métodos y propiedades.
- Todos los objetos de JavaScript pertenecen a la 'clase' `Object`.
- JavaScript es un lenguaje basado en objetos: hay objetos pero no clases (Class).
- Los objetos predefinidos de JavaScript (Date, Number, String, Array, Function, …) derivan de la 'clase' Object.

4. Objetos definidos por el usuario

- La forma más sencilla de crear un objeto es mediante la palabra reservada new seguida del nombre de la clase que se quiere instanciar:

```
var elObjeto = new Object();
var laCadena = new String();
```

- El objeto `laCadena` creado mediante el objeto nativo String permite almacenar una cadena de texto y la variable `elObjeto` almacena un objeto genérico de JavaScript, al que se pueden añadir propiedades y métodos propios para definir su comportamiento.

4. Objetos definidos por el usuario

- Definición de un objeto:
  - Técnicamente, un objeto de JavaScript es un array asociativo formado por las propiedades y métodos.
  - La forma más directa para definir las propiedades y métodos de un objeto es mediante la notación de puntos de los arrays asociativos.

```
//NOTACION PUNTOS
var elArray = new Array();

elArray.primero = 1;
elArray.segundo = 2;
```

```
//NOTACION TRADICIONAL
var elArray = new Array();

elArray['primero'] = 1;
elArray['segundo'] = 2;
```

4. Objetos definidos por el usuario

- Declaración e inicialización de los objetos:
  - Un objeto es una entidad que posee unas propiedades que lo caracterizan y unos métodos que actúan sobre estas propiedades.
  - En JavaScript la base para crear objetos es el “constructor”: se trata de una función que da nombre a la “clase” (simulación de la clase) y permite inicializar el objeto. Por lo que, no existe la necesidad de definir explícitamente un método constructor.

4. Objetos definidos por el usuario

Constructor: es una función especial para crear un objeto. Empieza por letra mayúscula:

```
function Coche()

{ //Propiedades y métodos }
```

Creación de un objeto:

```
var unCoche = New Coche();
```

4. Objetos definidos por el usuario

- Sintaxis:

```
function mi_objeto (valor_1, valor_2, valor_x){
  this.propiedad_1 = valor_1;
  this.propiedad_2 = valor_2;
  this.propiedad_x = valor_x;
}
```

- Las **propiedades** de nuestro objeto que se creen dentro del constructor, se definirán empleando la palabra reservada `this`, que se utiliza para hacer referencia al objeto actual.

4. Objetos definidos por el usuario

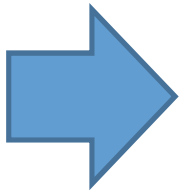
- Ejemplo: Creación del objeto Coche.

"Constructor" sin parámetros:

```
function Coche()
  {this.marca = "" //Vacío
   this.modelo = "" //Vacío
   this.combustible = "Diesel"
//Inicializado
   this.cantidad = 0//Inicializado
}
// Cantidad indica la cantidad de
// combustible inicial.
```

"Constructor" con parámetros:

```
function Coche(marca,modelo,
combustible,cantidad)
  {this.marca = marca;
   this.modelo = modelo;
   this.combustible = combustible;
   this.cantidad = cantidad;
  }
//Cada propiedad toma los valores
// recibidos como parámetros.
```

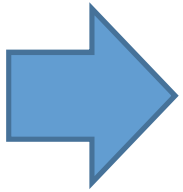


Crear un objeto vacío (sin propiedades):

```
var cocheVacio = New Coche();
```

Cambiar valores en las propiedades:

```
cocheVacio.marca = "Seat";
cocheVacio.maodelo = "Ibiza";
cocheVacio.combustible =
"Diesel";
cocheVacio.cantidad = 40;
```



Crear un objeto inicializado(con propiedades):

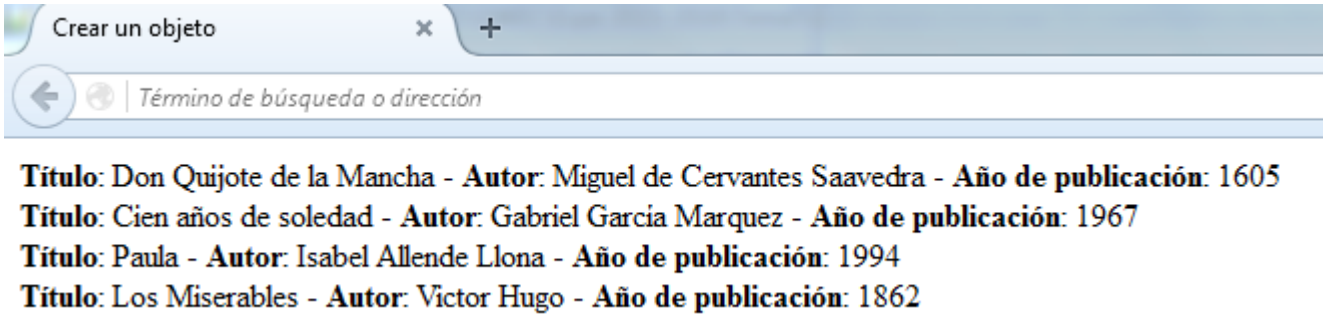
```
var miCoche = New
Coche("Kia","Ceed","Diesel",1);
```

Acceder a las propiedades:

```
Document.write("Mi coche es un:
" + miCoche.marca +
miCiche.modelo);
```

Actividad 5

- Declara un nuevo tipo de objeto llamado Libro que posea solamente tres propiedades (*título, autor y año de publicación*).
- Crea cinco instancias del mismo y almacena cada una de ellas en un array.
- Posteriormente, recorre el array con un bucle para mostrar los datos de cada uno de los libros por pantalla.



4. Objetos definidos por el usuario

- Definir métodos del objeto:
  - Permiten acceder y modificar las propiedades de los objetos. Empiezan por minúscula..

Método que modifica la cantidad de combustible:

```
function rellenardeposito(litros)
  { this.cantidad = litros
  }
```

Referencia al método fuera del objeto (¡no recomendado!):

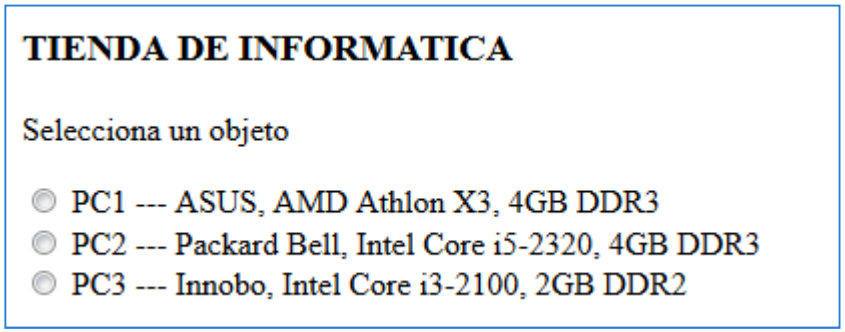
```
function Coche(marca,modelo,
combustible,cantidad)
  { this.marca = marca;
    this.modelo = modelo;
    this.combustible = combustible;

   //Métodos

    this.rellenarDeposito =
          rellenardeposito;
  }
```

Referencia al método dentro del objeto:

```
function Coche(marca,modelo,
combustible,cantidad)
  { this.marca = marca;
    this.modelo = modelo;
    this.combustible = combustible;

   //Métodos

    this.rellenarDeposito = function
             (litros);
       {
          this.cantidad = litros;
       }
  }
```

Actividad 6

- Crea un objeto llamado PC con tres propiedades (marca, cpu y memoria).
- Haz una página donde mostremos tres ordenadores de características diferentes y podamos elegir uno de ellos mediante un radio button.
- Una vez hecha la elección crearemos una instancia del objeto seleccionado y mostraremos sus características mediante un mensaje.



4. Objetos definidos por el usuario

- Los objetos también pueden definirse con el literal, el cual es un valor fijo, formado por parejas de tipo: nombre : valor

```
var coche= {marca:"Kia", modelo:"Ceed", combustible: "Diesel",
cantidad: 40};
```

Ejemplo equivalente:

```
var coche = new Object();
coche.marca = "Kia";
coche.modelo = "Ceed";
coche.combustible = "Diesel";
coche.cantidad = 40;
```

Acceso:

```
coche.marca;
coche[marca];
```

- Aplicar el operador punto sobre undefined o null provoca un error de ejecución y aborta la ejecución del programa.

4. Objetos definidos por el usuario

- La notación array permite acceder también a propiedades cuyo nombre está en una variable en forma de string. (Esto no es posible con la notación punto).

```
var x = {titulo:`Avatar´, director:`James
Cameron´};


x.titulo;       //”Avatar”
x[‘titulo’];    //”Avatar”

var p = ‘titulo’; //inicializada con string ‘titulo’
x[p];           //”Avatar”
x.p;           //undefined
//el objeto x no tiene ninguna propiedad de nombre p
```

4. Objetos definidos por el usuario

- Un objeto independientemente del tipo que sea, posee las propiedades `length`, `constructor`, `prototype` ... y un conjunto de métodos entre los que se encuentran `call()` y `apply()`.
- Ambos métodos permiten ejecutar una función como si fuera un método de otro objeto.

```
function miFuncion(x){
        return this.numero + x;
}

var elObjeto = new Object();
elObjeto.numero = 5;

var resultado = miFuncion.call(elObjeto, 4);
alert(resultado); //9
```

4. Objetos definidos por el usuario

- Método **`call()`:**

```
function miFuncion(x){
        return this.numero + x;
}

var elObjeto = new Object();
elObjeto.numero = 5;

var resultado = miFuncion.call(elObjeto, 4);
alert(resultado); //9
```

- El primer parámetro del método `call()` es el objeto sobre el que se va a ejecutar la función.
- La función externa está implementada como si fuera un método del objeto, ya que la palabra reservada `this` hace referencia al objeto indicado en la llamada a `call()`.
- El resto de parámetros del método `call()` son los parámetros que se pasan a la función, en este ejemplo sólo un parámetro.

4. Objetos definidos por el usuario

- Método **`apply()`:**
- Es idéntico al método `call`(), salvo que en este caso los parámetros se pasan como un array.

```
function miFuncion(x){
        return this.numero + x;
}

var elObjeto = new Object();
elObjeto.numero = 5;

var resultado = miFuncion.apply(elObjeto, [4]);
alert(resultado); //9
```

4. Objetos definidos por el usuario

- **HERENCIA:**
- Un objeto hereda métodos de su clase.
- Por ejemplo: los objetos de la clase Date heredan métodos como toString(), getDay(), getHours(), …

```
var fecha = new Date();

fecha.toString() //Fri Aug 08 2014 12:3436 GTM+0200(CEST)
fecha.getHours() //12
```

- Solo se puede invocar métodos heredados o definidos en un objeto. Invocar un método no heredado ni definido en un objeto provoca error de ejecución.

4. Objetos definidos por el usuario

- **Herencia**:
- La herencia en JavaScript se basa en la utilización de la propiedad `prototype`.
- Por ejemplo: Dada la "pseudoclase" Factura,

```
function Factura(IdFactura, IdCliente){
        this.idFactura = IdFactura;
        this.idCliente = IdCliente;

        this.muestraCliente = function(){
        alert(this.idCliente);}

        this.muestraId = function(){
        alert(this.idFactura);}
}
var laFactura1 = new Factura (1, 8);
laFactura1.muestraCliente();

var laFactura2 = new Factura (2, 5);
laFactura2.muestraId();
```

4. Objetos definidos por el usuario

- En el ejemplo anterior, las funciones `muestraCliente()` y `muestraId()` se crean de nuevo por cada objeto creado. Es decir, cada vez que se instancia un objeto, se definen tantas nuevas funciones como métodos incluya la función constructora.
- Si un método se declara con `this.NombreMetodo` en el constructor, este método no se hereda, es necesario definirlo con el atributo `prototype` para que se pueda heredar:

```
FuncionConstructora.prototype.NombreMetodo
```

(FuncionConstructora equivale a la pseudoclase)

4. Objetos definidos por el usuario

- Ejemplo: ObjetosHerencia.html

```
<script type="text/javascript">
function Persona(){
  //define la pseudoclase Persona
      //metodo camina
      Persona.prototype.camina = function(){
         alert("Persona caminando");}
      //metodo saluda
      Persona.prototype.saluda = function(){
         alert("Persona saludando");}
}
function Alumno(){
  //define la pseudoclase Alumno
}
//se define la herencia
Alumno.prototype = new Persona(); //Alumno hereda de Persona
//reemplazamos el método saluda
Alumno.prototype.saluda = function(){ alert("Hola, soy alumno");}
//Añadimos el método seDespide
Alumno.prototype.seDespide = function(){alert("Alumno dice adiós");}
//Creamos instancia del objeto Alumno
var alumno1 = new Alumno();
alumno1.saluda(); //metodo reemplazado
alumno1.camina(); //metodo heredado de Persona
alumno1.seDespide(); //metodo agregado a la pseudoclase Alumno
//comprueba la herencia
alert(alumno1 instanceof Persona); //true
alert(alumno1 instanceof Alumno); //true
</script>
```

Notación JSON

- JSON (JavaScript Object Notation) es un formato ligero para el intercambio de información. El formato JSON permite representar estructuras de datos (arrays) y objetos (arrays asociativos) en forma de texto.
- La especificación completa de JSON se puede consultar en RFC 4627 (http://tools.ietf.org/html/rfc4627) y su tipo MIME oficial es **application/json**
- Como ya se sabe, la notación tradicional de los arrays es tediosa cuando existen muchos elementos:

```
var modulos = new Array();
modulos[0] = "Lector RSS";
modulos[1] = "Gestor email";
modulos[2] = "Agenda";
modulos[3] = "Buscador";
modulos[4] = "Enlaces";
```

Notación JSON

- El ejemplo anterior se puede reescribir de la siguiente manera utilizando JSON:

```
var modulos = ["Lector RSS", "Gestor email",
"Agenda", "Buscador", "Enlaces"];
```

- Para los **arrays asociativos** se puede utilizar la notación de puntos, pero la notación JSON permite definirlos de una forma mucho más concisa.

```
//NOTACION TRADICIONAL
var modulos = new Array();
modulos.titulos = new Array();
modulos.titulos.['rss'] = "Lector RSS";
modulos.titulos.['email'] = "Gestor email";
modulos.titulos.['agenda'] = "Agenda";

//NOTACION JSON
var modulos = {titulos:{rss:"Lector RSS",
email:"Gestor email", agenda:"Agenda"}}
```

Notación JSON

- Notación JSON genérica para crear arrays y objetos:
- Arrays: (corchetes [])

```
var array = [valor1, valor2, valor3,..., valorN];
```

- Objetos (arrays asociativos): (llaves {})

```
var objeto = {clave1:valor1, clave2:valor2,
clave3:valor3,..., claveN:valorN};
```

- Manera habitual de definir objetos en JavaScript mediante JSON:

```
var objeto = {
  propiedad1: valor_simple_1,
  propiedad2: [array_valor1, array_valor2],  //array
  propiedad3: {propiedad_anidada: valor}, //objeto
  metodo1: nombre_funcion_externa,
  metodo2: function(){ ... }, //función anónima
};
```

Notación JSON

- Ejemplo: Objeto Libro
- Notación tradicional:

```
var Libro = new Object();
  Libro.numeroPaginas = 150;
  Libro.autores = new Array();
  Libro.autores[0] = new Object();
  Libro.autores[0].id = 50;
  Libro.autores[1] = new Object();
  Libro.autores[1].id = 67;
```

- Notación JSON:

```
var Libro = {numeroPaginas:150, autores:[{id:50},
{id:67}]};
```

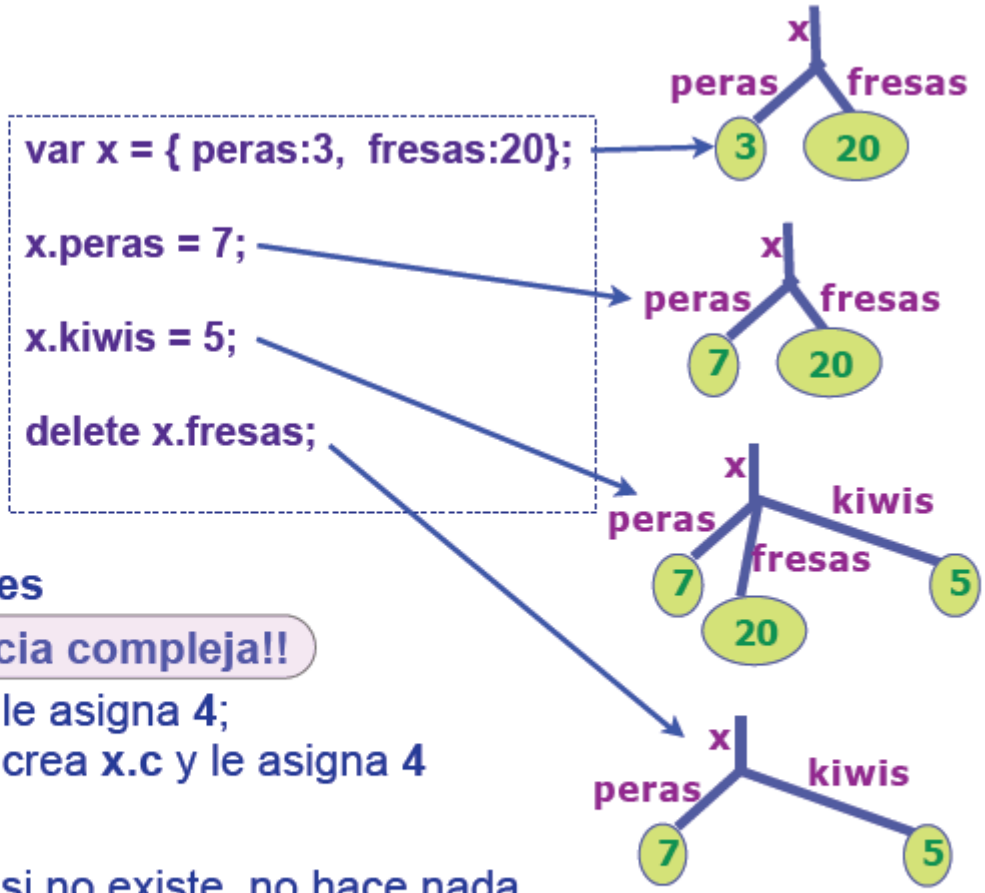
Actividad 7

- Utilizando JSON, definir la estructura de un objeto que almacena una factura. Las facturas están formadas por la información de la empresa (nombre de la empresa, dirección, teléfono, NIF), la información del cliente (similar a la de la empresa), una lista de elementos (cada uno de los cuales dispone de descripción, precio, cantidad) y otra información básica de la factura (importe total, tipo de IVA, forma de pago).
- Una vez definidas las propiedades del objeto, añadir un método que calcule el importe total de la factura y actualice el valor de la propiedad correspondiente. Por último añadir otro método que muestre por pantalla el importe total de la factura.

Objetos: características avanzadas

Propiedades dinámicas

- Las propiedades de objetos
  - Pueden **crearse**
  - Pueden **destruirse**
- Operaciones sobre **propiedades**
  - **x.c = 4** **¡¡OJO: sentencia compleja!!**
    - si propiedad **x.c** existe, le asigna **4**;
      si **x.c** no existe, crea **x.c** y le asigna **4**
  - **delete x.c**
    - si existe **x.c**, la elimina; si no existe, no hace nada
  - **"c" in x**
    - si **x.c** existe, devuelve **true**, sino devuelve, **false**

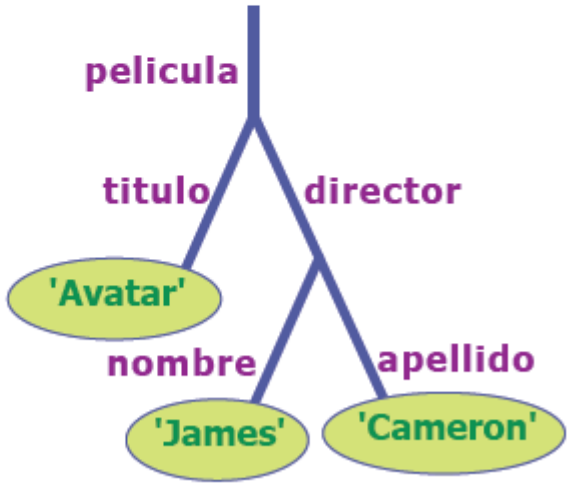


Objetos: características avanzadas

Objetos anidados: árboles

```
var pelicula = {
    titulo: 'Avatar',
    director:{
        nombre:'James',
        apellido: 'Cameron'
    }
};
```

- Los objetos pueden **anidarse** entre si
  - Los objetos anidados representan **arboles**



- La notación punto o array puede **encadenarse**
  - Representando un **camino en el árbol**
    - **Las siguientes expresiones se evaluan así:**
      - pelicula.director.nombre => 'James'
      - pelicula['director']['nombre'] => 'James'
      - pelicula['director'].apellido => 'Cameron'
      - pelicula.estreno => undefined
      - pelicula.estreno.año => Error_de_ejecución

Objetos: características avanzadas

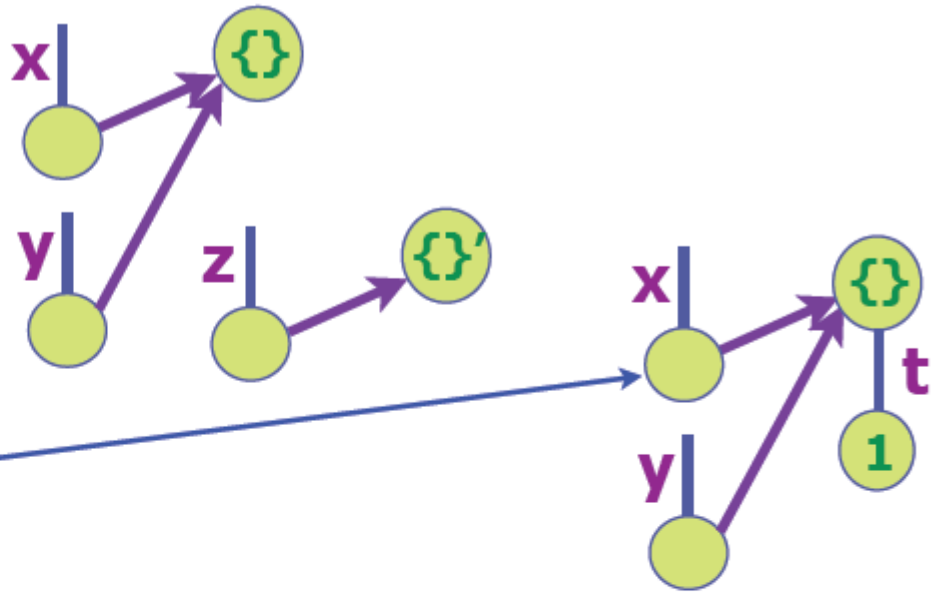
Usar propiedades dinámicas

- Las propiedades dinámicas de JavaScript
  - son muy útiles si se utilizan bien
- Un objeto solo debe definir las propiedades
  - que contengan información conocida
    - añadirá mas solo si son necesarias
- La información se puede consultar con
  - **prop1 && prop1.prop2**
    - para evitar errores de ejecución
      - si las propiedades no existen

```
// Dado un objeto pel definido con

var pel = {
      titulo: 'Avatar',
      director:'James Cameron'
};

// se puede añadir pel.estreno con

pel.estreno = {
    año:  '2009',
   cine: 'Tivoli'
}

// La expresión

pel.estreno && pel.estreno.año

// devuelve pel.estreno o undefined,
// evitando ErrorDeEjecución, si
// pel.estreno no se hubiese creado
```

Objetos: características avanzadas

```
var x = {};  // x e y tienen la
var y = x;   // misma referencia

var z = {};  // la referencia a z
             // es diferente de
             // la de x e y

x.t = 1;

x.t    => 1   // x accede al mismo
y.t    => 1   // objeto que y
z.t    => undefined
```



Referencias a objetos

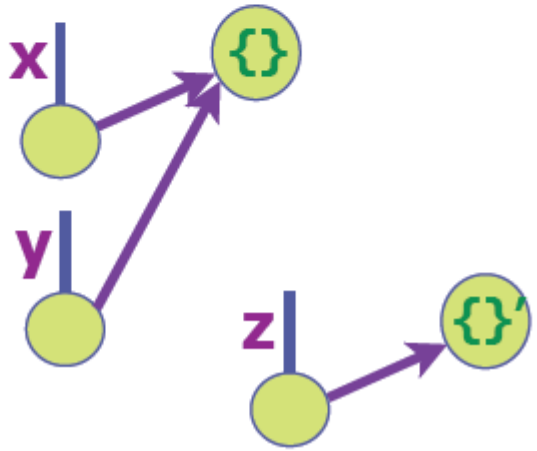
- Las variables que contienen objetos
  - solo contienen la referencia al objeto
- El objeto esta en otro lugar en memoria
  - indicado por la referencia
- Esto produce efectos laterales
  - como ilustra el ejemplo

Objetos: características avanzadas

Identidad de objetos

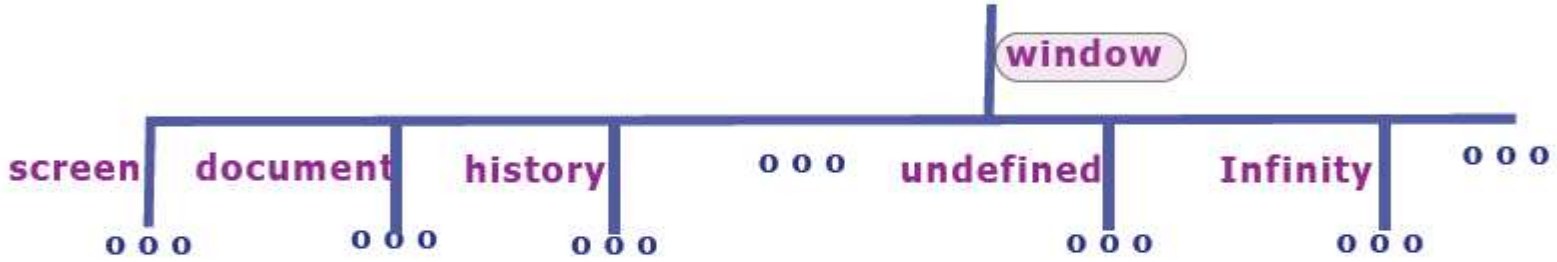
- Las referencias a objetos afectan a la identidad
  - porque identidad de objetos
    - es identidad de referencias
  - los objetos no se comparan
    - se comparan solo las referencias
  - es poco util si no se redefine
- Igualdad (debil) de objetos **==** y **!=**
  - no tiene utilidad tampoco con objetos
    - no se debe utilizar

```
var x = {};  // x e y contienen la
var y = x;   // misma referencia

var z = {}   // la referencia a z
             // es diferente de x e y

x === y          => true
x === {}         => false
x === z          => false
```



Objetos: características avanzadas

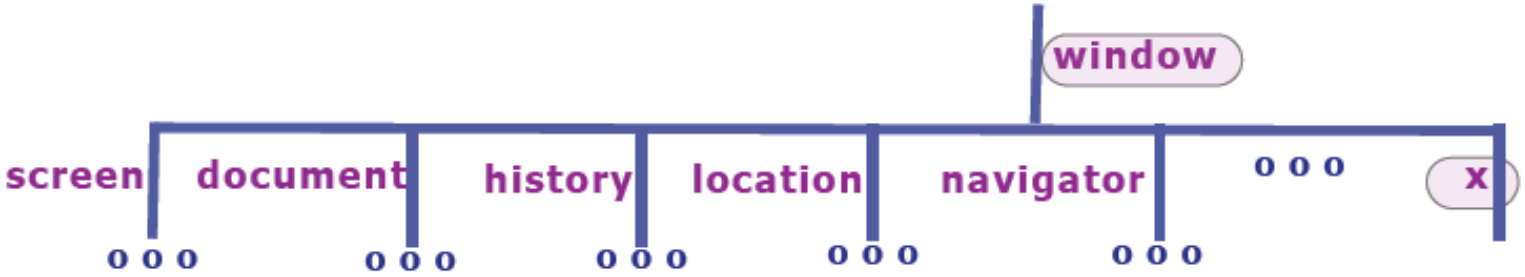
Objeto window o this



- El entorno de ejecución de JavaScript es el **objeto global window**
  - El **objeto global window** tiene propiedades con información sobre
    - **Objetos predefinidos** de JavaScript, el **navegador**, el **documento HTML**, ...
- **window** se referencia también como **this** en el entorno global
  - La **propiedad document** de **window** se referencia como
    - **window.document**, **this.document** o **document**
- Documentación: https://developer.mozilla.org/en-US/docs/Web/API/Window

Objetos: características avanzadas

Variables globales y el entorno de ejecución



- Un programa JavaScript se ejecuta con el objeto **window** como entorno
  - una asignación a una variable no definida como **x = 1;**
    - Crea una nueva **propiedad de window** de nombre **x**, porque
      - **x = 1;** es equivalente a **this.x = 1;** y a **window.x = 1;**
- Olvidar definir una variable, es un error muy habitual
  - y al asignar un valor a la variable no definida, JavaScript no da error
    - **sino que crea una nueva propiedad de window**
  - Es un error de diseño de JavaScript y hay que tratar de evitarlo

ANEXO sobre arrays

Métodos que emplean funciones

Hay métodos de arrays que pueden tener como parámetro una función para modificar los valores que formarán el resultado.

Los parámetros que se pueden pasar a la función, aunque no es obligatorio son:

1. `valor`: corresponde al valor de la posición actual del array.
2. `indice`: indica la posición del elemento dentro del array.
3. `arrayOrigen`: es el array completo con el que hemos comenzado a operar.

La función que declaremos no necesita tener definidos todos los parámetros, pero si deben estar en ese orden.

Función Callback

Es una función que se pasa como parámetro de otras funciones y esta función parámetro se ejecuta cuando la función principal ha terminado de ejecutarse, o cuando a nosotros nos interese.

En JavaScript son muy habituales, se utilizan en métodos, a los cuales se les pasa una función como parámetro.

Ejemplo:

```
function funcionPrincipal(callback)
{
 alert('hago algo y llamo al callback avisando que terminé');
 callback();
}

funcionPrincipal(function()
{
 alert('terminó de hacer algo');
});
```

Método forEach

Es un método que no devuelve nada y espera que se le pase una función como parámetro, que se ejecutará por cada elemento del array.

Sintaxis:

```
nomArray.forEach(fnTransformar)
```

A la función se le pueden pasar parámetros.

Este método es una forma de recorrer un array, se podría hacer lo mismo con un bucle for o while, pero de esta manera es mas elegante.

Método map

Es un método que recorre cada elemento del array y a través de una función callback que se le pasa como único parámetro, establece el calculo para cada elemento.

Este método NO modifica el array origen, sino que nos DEVUELVE otro array con los mismos elementos, al que se le ha aplicado la función.

Sintaxis:

```
nomArray.map(fnTransformar)
```

Método map

Ejemplo:

```
const notas = [5,6,,,9,,8,7,8,9,,10];
const notasConPeso= notas.map(x=> x*1,2);
console.log(notasConPeso);
```

El nuevo array tendría las notas actualizadas al 20%.

Lo hemos hecho con la función flecha, pero se podría hacer declarando la función:

```
function SumarPeso(valor){
    return valor*1,2;
}
const notasConPeso= notas.map(sumarPeso);
console.log(notasConPeso);
```

Función flecha

Es una función anónima, relacionada con métodos y no pueden ser usada como constructor.

Se definen utilizando una flecha =>

Sintaxis:

```
(param1, param2, …rest) => {expresión;}
```

Ejemplo:

var suma = (x, y) => x+y;

Método reduce

Es un método aplica la función callback a cada elemento del array para ir formando o acumulando un resultado, reduciendo así el array a un único valor de salida que puede ser de cualquier tipo.

Este método devuelve un valor.

Sintaxis:

```
nomArray.reduce (fnAcumular, valor)
```

Ejemplo:

```
const array=[1,2,3,4,5];
let suma=array.reduce((acu,valor)=>acu+valor,0);
console.log(suma);
```

Método filter

Genera un array con los elementos que cumplen con la función callback fnComprobar. Si ninguno cuadra con las condiciones de la función, devuelve el array vacio..

Este método devuelve un array con los elementos que cumplen la condición de la función.

Sintaxis:

```
nomArray.filter(fnComprobar)
```

Ejemplo:

```
const array=[4,9,2,6,5,7,8,1,10,3];
const arrayFiltrado=array.filter(x=>x>5);
console.log(arrrayFiltrado)
```