Desarrollo web en entorno cliente

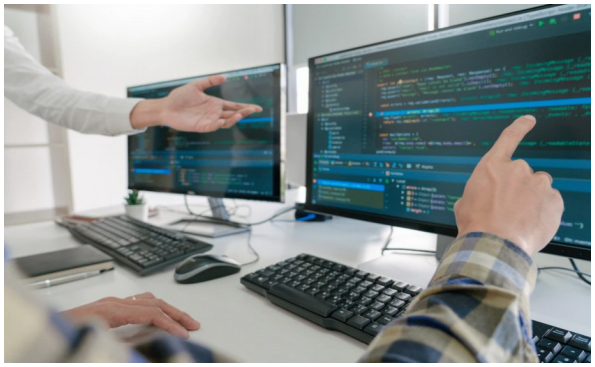
Unidad 4: Programación con arrays, funciones y objetos definidos por el usuario

ÍNDICE

INTRODUCCIÓN....3
OBJETIVOS / CAPACIDADES....4
PROYECTO DE LA UNIDAD....5
1. FUNCIONES PREDEFINIDAS DEL LENGUAJE....7
2. LLAMADAS A FUNCIONES. PARÁMETROS. DEFINICIÓN DE FUNCIONES Y PROCEDIMIENTOS. ARGUMENTOS Y PASO DE VALORES. VALORES DE RETORNO....9
3. ARRAYS. DECLARACIÓN, ARRAYS MULTIDIMENSIONALES. CREACIÓN DE ARRAYS Y TIPOS DE DATOS EN ARRAYS. DEFINICIÓN, CREACIÓN Y USO....11
Cuestionario....13
4. INICIALIZACIÓN DE ARRAYS....14
5. RECORRIDO DE ARRAYS. BÚSQUEDA....16
5.1. Búsqueda....18
Cuestionario....20
6. CREACIÓN Y UTILIZACIÓN DE OBJETOS....21
7. DEFINICIÓN DE MÉTODOS Y PROPIEDADES....23
8. LIBRERÍAS DE FUNCIONES. USO DE LIBRERÍAS DE TERCEROS. PRINCIPALES LIBRERÍAS PARA FINES ESPECÍFICOS....25
Cuestionario....27
9. CREACIÓN Y USO DE PROTOTIPOS....28
10. GESTIÓN DE CLASES....31
Cuestionario....33
RESUMEN....34
RECURSOS PARA AMPLIAR....35
BIBLIOGRAFÍA....
GLOSARIO....37

INTRODUCCIÓN

Una vez conocidas las especificaciones básicas de **JavaScript** y el uso de **objetos**, en esta unidad nos vamos a centrar en cosas más específicas.



Será importante conocer las **funciones predefinidas** del lenguaje, que nos permitirán **agilizar** el trabajo de manera considerable.

Además de eso, veremos cómo trabajar con nuestras **propias funciones**, incluyendo el paso de **parámetros** y los **valores de retorno**.

Una vez tengamos claros estos conceptos profundizaremos sobre los **arrays**, que hemos visto vagamente en unidades anteriores.

Finalmente veremos las diferencias entre **objetos**, **clases** y **prototipos** y cómo podemos trabajar con ellos. Todos estos nuevos conceptos nos darán nuevas posibilidades y mejorarán la manera que tenemos de trabajar.

Es importante remarcar, que, como en el resto de unidades, ésta se apoya en los conocimientos obtenidos en otros módulos, como por ejemplo en relación a la **programación orientada a objetos**, o la programación web.

OBJETIVOS / CAPACIDADES

En esta unidad de aprendizaje, las capacidades que más se van a trabajar son:

- ✔ **Utilizar lenguajes de marcas y estándares Web, asumiendo el manual de estilo, para desarrollar interfaces en aplicaciones Web.**

PROYECTO DE LA UNIDAD

Antes de empezar a trabajar el contenido, te presentamos la **actividad** que está relacionada con esta unidad de aprendizaje. Se trata de un **caso práctico** basado en una **situación real** con la que te puedes encontrar en tu puesto de trabajo. Con esta actividad se evaluará la puesta en práctica de los **criterios de evaluación** vinculados al resultado de aprendizaje que se trabaja en esta unidad. Para realizarla deberás hacer lo siguiente: lee el enunciado que te presentamos a continuación, dirígete al área general del módulo profesional, concretamente a la actividad de evaluación que se encuentra dentro de esta unidad, allí encontrarás todos los detalles sobre fecha y forma de entrega, objetivos... A lo largo de la unidad irás adquiriendo los conocimientos necesarios para ir elaborando este proyecto.

Enunciado:

Organizando al personal

Una vez cerrado el proyecto de las encuestas, y cambiando radicalmente de tema, desde nuestra empresa están empezando a crear un sistema de gestión de personal propio Nos piden ayuda en los siguientes apartados:

- Habrá que crear un objeto que represente al empleado, y que guarde el nombre, el departamento, la antigüedad y la fecha de nacimiento.
- Dentro del objeto tendremos un método que nos genere la descripción del empleado. La fecha habrá de mostrarse en formato “dd/mm/aaaa”.
- Tendremos un array donde guardar los empleados, que serán pedidos por pantalla mediante prompt.
- La idea será recorrer el array de empleados, mostrando las descripciones de todos.

- Finalmente habrá que convertir ese objeto empleado en prototipo, para ver cuál de las dos opciones nos resulta más útil, y usarlo de la misma forma que el objeto.
- Tendrás que crear las funciones que consideres necesarias.

El programa tiene que estar comentado y probado en al menos 2 navegadores.

1. FUNCIONES PREDEFINIDAS DEL LENGUAJE

Hemos visto por encima, en unidades anteriores, como podemos crear nuestras **propias funciones** en JavaScript. Esto nos permite hacer nuestro código más modular, con las consiguientes ventajas de **estructuración** y no **repetición de código**.

A parte esto, JavaScript, como la mayoría de lenguajes de programación pone al alcance del desarrollador una serie de funciones predefinidas, o integradas, en el propio lenguaje, que facilitan el trabajo del desarrollador, ahorrando mucho tiempo y recursos en la mayoría de ocasiones.

Muchas de estas funciones están **asociadas** a librería y **objetos**, como ya hemos visto, por ejemplo, con el objeto String y todas sus funciones, charAt(), indexOf(), concat()… Además de éstas, existen una serie de funciones que **no están asociadas** a ningún objeto en concreto, como son algunas de las que veremos a continuación:

- **eval(string)**: recibe une cadena de caracteres y la ejecuta como una sentencia de JavaScript.
- **parseInt(cadena, base)**: convierte la cadena en número en función de la base indicada.
- **parseFloat(cadena, base)**: convierte la cadena en número en función de la base indicada.
- **escape(carácter)**: devuelve un carácter que recibe por parámetro en codificación ISO.
- **isNaN(número)**: indica si el valor pasado es o no un número válido.
- **isFinite(número)**: indica si el valor pasado es o no un número finito.

Ejemplo:

Todas estas funciones se usan de forma normal en el código JavaScript, como, por ejemplo:

```
var num = 3
var a = parseInt(num, 10)
eval("document.write(" + a +")")
```

Vídeo: JavaScript para principiantes

Visualiza este vídeo en el que se ven los métodos Number, parseInt y parseFloat para transformar strings en números.
https://www.youtube.com/embed/ONtYza8g1N4

2. LLAMADAS A FUNCIONES. PARÁMETROS. DEFINICIÓN DE FUNCIONES Y PROCEDIMIENTOS. ARGUMENTOS Y PASO DE VALORES. VALORES DE RETORNO

Sabemos ya que una **función** no es más que un trozo de código, más o menos independiente que es capaz de realizar una **tarea** en concreto y que nos ayuda con la **estructuración** y **reutilización** del código de nuestro programa.

Estas funciones, ya sean definidas por el desarrollador, o las predefinidas del lenguaje que acabamos de ver, necesitan ser llamadas dentro de nuestro código para poder ser utilizadas.

Para ello necesitamos conocer con precisión dos **cosas básicas dentro de una función**:

- Argumentos o parámetros.
- Valor de retorno.

➔ **Argumentos**: el verdadero poder de una función reside en sus argumentos o parámetros. Dicho de otra manera, mediante los argumentos podemos hacer que **varias llamadas** a una misma función obtengan **diferentes resultados**. Es decir, parametrizamos las llamadas.

En JavaScript no es necesario **especificar** el tipo de datos de un parámetro a la hora de declarar la función, lo que puede hacer que nos encontremos con más problemas a la hora de utilizarlas.

De todas maneras, siempre tendremos que hacer coincidir los **parámetros** que pasemos a una función, con el tipo esperado, dentro de su cabecera. Para esto es muy importante **documentar** correctamente el código.

Algunos de estos parámetros, como ya vimos, pueden tener valores por defecto, que nos evita tener, dependiendo de nuestras necesidades, pasarle valor en algunas ocasiones.

```
function writeMsg(name="John", surname="Doe") {
    alert("Hello " + name + " " + surname);
}
writeMsg();
writeMsg("James");
writeMsg("James", "Bond");
```

➔ **Valor de retorno**: como valor de retorno nos referimos a aquel **resultado** que puede o no, devolver una función. Hay funciones que retornas algún resultado y otras que no.

Al igual que con los parámetros, es muy importante conocer el **valor de retorno** de una función, para poder utilizarlo correctamente en la **llamada**.

Si una función devuelve algún parámetro, lo normal es **recogerlo** en la llamada y hacer algo con él. Ya sea asignarlo a una **variable**, o usarlo directamente en una **expresión**.

```
function suma(a, b) {
    return (a + b);
}
alert(suma(5,6));
```

Vídeo: llamadas a funciones

Visualiza este vídeo en el que hablamos sobre las llamadas a funciones.

3. ARRAYS. DECLARACIÓN, ARRAYS MULTIDIMENSIONALES. CREACIÓN DE ARRAYS Y TIPOS DE DATOS EN ARRAYS. DEFINICIÓN, CREACIÓN Y USO

Durante el curso hemos trabajado en algunas ocasiones con **arrays**, pero no hemos profundizado demasiado sobre ellas. Ahora llega el momento de hacerlo. Veamos pues **que son**, **como se utilizan** los arrays en JavaScript y conoceremos que son los a**rrays multidimensionales**:

➔ **¿Qué es un array?**:

Un array en JavaScript, como en la mayoría de lenguajes de programación no es más que un tipo especial de variables que puede contener más de un valor al mismo tiempo. Algo así como un conjunto de valores en uno solo.

➔ **¿Cómo se utiliza?**: es muy útil para poder **agrupar** elementos (usuarios, vehículos, animales…).

Técnicamente un array en JavaScript no es un tipo de datos por sí mismo, porque se incluye dentro de los tipos de datos complejos (no primitivos), es decir **Object**.

Por lo tanto, podemos declarar un array **vacío** (sin datos), de las siguientes maneras:

```
var array1 = [];
var array2 = new Array(5);
```

La principal diferencia es que con el segundo método podemos definir el tamaño del array (en este caso de 5 posiciones), por lo que estará guardando espacio para ese número de elementos.

➔ **Arrays multidimensionales**: además de los arrays vistos hasta ahora, también podemos trabajar con **arrays multidimensionales**. Lo normal es utilizar como máximo dos o tres dimensiones.

¿Qué es un array multidimensional?

Podemos definirlo como un array de arrays. Dicho de otra manera, un array en que, en cada una de sus posiciones, en lugar de contener, por ejemplo, números o Strings, contiene otro array.

Cuantas más **dimensiones** tengamos, más arrays contendrán otros arrays. Así, por ejemplo, un **array bidimensional**, que es el más utilizado es únicamente un array de arrays. A este tipo de arrays, también se les conoce como **matrices**.

En este caso podemos definirlos como:

```
var array1 = [[],[]];
var array2 = new Array([])
```

Cuestionario

Lee el enunciado e indica la opción correcta:

¿Cuál es la función principal de un array en JavaScript?

a. Almacenar varios valores en una sola variable.

b. Realizar operaciones matemáticas complejas.

c. Definir el tamaño de un objeto.

Lee el enunciado e indica la opción correcta:

¿Cuál es la diferencia entre declarar un array vacío utilizando [] y utilizando new Array()?

a. No hay diferencia, ambas formas son equivalentes.

b. La forma [] permite definir el tamaño del array.

c. La forma new Array() es más eficiente en cuanto a rendimiento.

Lee el enunciado e indica la opción correcta:

¿Cómo se define un array multidimensional en JavaScript?

a. Mediante la palabra clave "multidimensional" seguida de corchetes [].

b. Utilizando la función "multidimensionalArray()" con el número de dimensiones como argumento.

c. Creando un array que contenga otros arrays dentro.

4. INICIALIZACIÓN DE ARRAYS

Acabamos de ver que es un array, los diferentes tipos que podemos encontrar y como declararlos. Veamos ahora como podemos **inicializarlos**, o darles valor.

Hay dos maneras básicas de dotar de valore a un array:

- Una vez creados.
- En el momento de la creación.

➔ **Una vez creados**: para dar valores al array una vez creado, lo primero, obviamente, será crear ese array, por lo que podemos hacer como antes para crear un **array vacío**.

```
var coches = [];
```

Una vez tenemos el array creado, en este caso en la variable "coches", podemos proceder a dar valores al array. Podemos ir haciéndolo **posición a posición**, o mediante algún tipo de repetición o bucle.

Si optamos por hacerlo uno a uno, sería algo así:

```
coches[0] = "Mazda"
coches[1] = "BMW"
coches[2] = "Peugeot"
```

Si por el contrario queremos hacerlo mediante un **bucle**:

```
var numeros = []
for (let i = 0; i < 10; i++) {
    numeros[i] = i+1
}
```

En este segundo caso tenemos un array que contendrá números del 1 al 10, y por tanto lo hemos podido hacer de manera más **automática**.

➔ **En el momento de la creación**: si optamos por dar valores al array justo en el momento de la creación conseguiremos **ahorrarnos** algunos pasos, pudiendo hacer, por ejemplo:

```
var coches2 = ["Mazda", "BMW", "Peugeot"]
```

Vemos que aquí necesitamos saber los **valores del array** justo en el momento de crearlo, lo cual puede hacer que no siempre sea posible hacerlo de esta manera.

Además, en el caso de querer hacerlo de manera más o menos automática, en ocasiones esto nos perjudicaría:

```
var numeros2 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Imaginemos que en el caso anterior en lugar de guardar del 1 al 10 queremos hacerlo del 1 al 1.000. Vemos que sería mucho mejor mediante el bucle que intentando inicializar desde el principio.

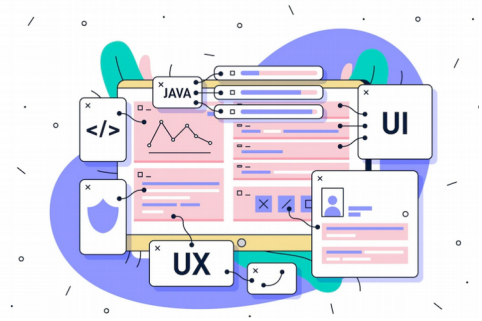
5. RECORRIDO DE ARRAYS. BÚSQUEDA

En un array se guarda información normalmente **relacionada** entre sí, datos del mismo tipo. Por ese motivo una de las cosas que se hace con mayor frecuencia en los arrays es recorrerlos: bien para dar valor a sus posiciones o bien para leer los valores.

DESTACADO

Denominamos recorrer un array al proceso por el cual, mediante algún tipo de bucle, pasamos por todas las posiciones de éste. También existen los recorridos parciales, en los que únicamente pasamos por algunas posiciones del array, normalmente correlativas.

Veamos ahora algunos **métodos para el recorrido de arrays**:



- **Bucle for**: el método más común de recorrer un array es mediante el **bucle for**, accediendo a todos los elementos del array a través de su **índice**. Para eso necesitamos saber la **longitud** del array, cosa que podemos conocer gracias a la propiedad **length**.

```
var coches = ["BMW", "Mazda", "Saab", "Peugeot"]
for (x = 0; x < coches.length; x++) {
    console.log(coches [x])
}
```

- **Bucle for -in**: otra forma de recorrerlo sería mediante el bucle llamado **for-in**:

```
for (x in coches) {
    console.log(coches [x]);
}
```

Igual que en el método anterior accedemos al elemento mediante su **posición numérica**.

➔ **Otros métodos**: además de estos dos métodos principales que nos sirven para pasar por cada una de estas posiciones, existen otros métodos en el **objeto array** que permiten realizar acciones más complejas de manera simple. Por ejemplo:

- ✔ **forEach()** – Ejecuta una acción para cada elemento.
- ✔ **map()** – Crea un nuevo array con el resultado de una acción.
- ✔ **filter()** – Crea un nuevo array con los elementos filtrados.
- ✔ **reduce()** – Reduce todo el array a un valor.

Puedes encontrar información detallada sobre estos y otros métodos de array en los recursos para ampliar.

5.1. Búsqueda

Otro de los usos más comunes de los arrays se basa en **buscar ciertos datos** en ellos. Lo que se conoce comúnmente como búsquedas.

Para ello tenemos dos **opciones**:

- **Recorrer el array**, haciendo comparaciones, hasta encontrar el dato requerido.

```
for (i in coches) {
    if (coches[i] == "Mazda") {
        document.write("Mazda está en la posición " + i)
    }
}
```

- **Hacer uso de métodos del array**, como por ejemplo indexOf(), lastindexOf(), find() o findIndex().

```
document.write(coches.indexOf("Mazda"))
```

La segunda opción es más sencilla, pero un inconveniente podría ser que sólo nos indicaría la posición del primer elemento encontrado, en caso de que hubiera más. Esto no pasaría con la primera opción, ya que, al recorrer el array completo, encontraría todos los elementos repetidos.

Visualiza este vídeo en el que se habla sobre el método foreach.
https://www.youtube.com/embed/2SESIT4Qe3Y

Actividad de aprendizaje 1: arrays

Una vez realizada la encuesta sobre el color corporativo ideal de la empresa a todos los empleados, nos encontramos en que las votaciones han provocado muchos

duplicados. Mucha gente ha votado lo mismo y queremos contar cuántos votos hay de cada uno.

Teniendo en cuenta que las opciones finales eran Rojo, Azul y Verde, y que las votaciones están recogidas en el siguiente array, se pide realizar un programa que cuente las votaciones de cada color y muestre el color más votado por pantalla.

```
var votos = ["Rojo", "Verde", "Rojo", "Verde", "Azul", "Azul",
"Rojo",  "Rojo",  "Verde",  "Azul",  "Rojo",  "Verde",  "Azul",
"Azul", "Azul", "Rojo", "Verde", "Rojo", "Rojo", "Verde"]
```

Entrega el archivo .html en el espacio habilitado para ello.

Cuestionario

Lee el enunciado e indica la opción correcta:

¿Qué método usamos para ejecutar una sentencia recibida como parámetro?

a. parseInt()

b. eval()

c. isNaN()

Lee el enunciado e indica la opción correcta:

¿Qué otro nombre reciben los argumentos de una función?

a. Parámetros.

b. Atributos.

c. Number.

Lee el enunciado e indica la opción correcta:

¿Para que sirve el método reduce en un array?

a. Ejecuta una acción para cada elemento.

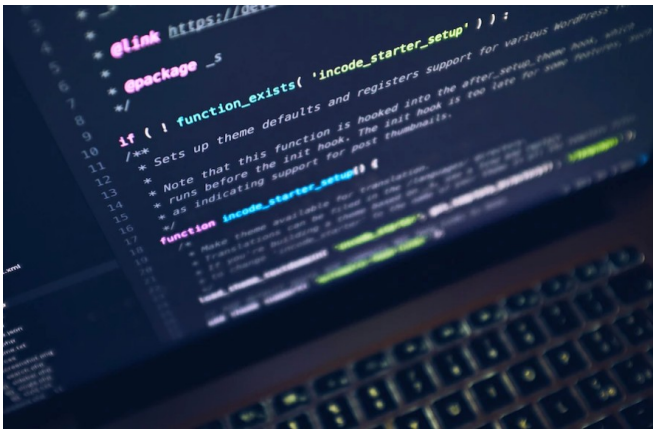
b. Busca un elemento.

c. Reduce todo el array a un valor.

6. CREACIÓN Y UTILIZACIÓN DE OBJETOS

Un objeto en JavaScript, como en otros lenguajes, es un elemento manipulable, es decir que le ocurren cosas y que se le pueden aplicar acciones. Esto se conoce como los métodos del objeto.

La principal **característica** de JavaScript es que casi todo en este lenguaje son **objetos**: una función, una variable, un array… Todos ellos son objetos.



Pese a eso, hay algunos objetos a los que denominamos y tratamos de una forma especial, como son los **tipos primitivos** de datos (string, number, boolean, null, undefined y symbol).

- **Creación de objetos**: si queremos crear objetos fuera de los predefinidos de JavaScript, lo que haremos es crear un **variable**, con un cierto nombre, en la que dentro de ella podemos indicar las **propiedades** y **métodos** que tiene ese objeto.

  Las sintaxis sería algo así:

```
var persona = {
    // Propiedades y métodos separados por comas,
}
```

  El cómo indicamos cuáles son esas propiedades y esos métodos lo veremos en el siguiente capítulo. Veamos ahora cómo podemos utilizar los **objetos** una vez creados.

- **Utilización de objetos**: un objeto no dejará de ser una variable, a la cual podremos acceder de forma normal, y al igual que con los objetos predefinidos

del sistema, si queremos acceder a sus **métodos** y **propiedades** lo podemos hacer mediante la **notación por punto**.

```
console.log(persona); // Muestra que hay dentro del objeto
alert(persona.nombre); // Accederíamos al nombre de la persona.
persona.nombre = "José"; // Cambiamos el nombre de la persona.
```

Otra forma en la que podemos crear un objeto es mediante la cláusula new, seguido de las propiedades y métodos, de la siguiente manera.

```
var persona = new Object();
// Aquí indicaríamos las propiedades y métodos.
```

7. DEFINICIÓN DE MÉTODOS Y PROPIEDADES

Ya hemos visto como crear un objeto, de diferentes maneras. Siguiendo con el ejemplo anterior, veamos ahora como definiremos sus métodos y propiedades. En esta ocasión veremos dos formas distintas de **declarar métodos y propiedades**:

- **Parejas nombre-valor**:

```
var persona = {
    nombre: "Pepito",
    apellido: "Pérez",
    edad: 24,
    getEdad: function() {
        return this. edad;
    }
}
```

En este caso hemos declarado el objeto en una variable llamada persona, con una serie de **propiedades** (nombre, apellidos y edad) y **métodos** (getEdad()).

Todo esto se define como parejas **nombre-valo**r, con el ":" como separación. El nombre de antes de los ":" será tanto el nombre de propiedades como de métodos.

En cambio, en el caso del valor, este cambia radicalmente entre **propiedades** y **métodos**, ya que éstos últimos, es ahí donde declaramos el contenido de la función.

- **Cláusula new**: la otra manera de poder declarar métodos y propiedades mediante la **cláusula new** sería:

```
var persona = new Object()
persona.nombre = "Pepito",
persona.apellido = "Pérez",
persona.edad = 24,
persona.getEdad = function() {
   return this. edad;
}
```

Como vemos, el funcionamiento es bastante similar, aunque es bastante más **elegante** la primera opción. De todos modos, en las dos opciones podemos añadir **propiedades** en cualquier momento al objeto:

```
persona.dni = "123456789A";
```

Cabe destacar también, que las propiedades de los objetos no sólo pueden ser de tipos primitivos, si no que también podemos usar tipos más complejos.

```
persona.hobbies = ["Fútbol", "Música", "Videojuegos"]
persona.estudiss = {ESO: true, CFGM: false, CFGS:
true, UNIVERSIDAD: false}
```

Vídeo: objeto

Visualiza el siguiente vídeo en el que hablamos sobre los objetos.

8. LIBRERÍAS DE FUNCIONES. USO DE LIBRERÍAS DE TERCEROS. PRINCIPALES LIBRERÍAS PARA FINES ESPECÍFICOS

Hablamos de librerías como un conjunto de funciones, normalmente relacionadas entre sí, que nos permiten agregar utilidades de forma sencilla al lenguaje.

Debido a la popularidad de JavaScript en los últimos años, ha crecido mucho el uso de librerías de terceros.

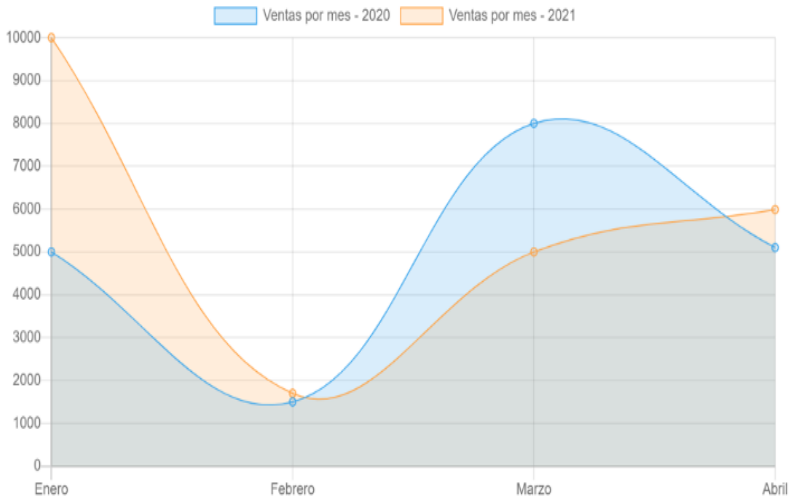
Estas librerías estás creadas por desarrolladores, fuera del contenido nativo de JavaScript, y compartidas con la **comunidad de desarrolladores**, para facilitar el trabajo de éstos. Pueden ser **librerías** de carácter general o bien creadas para fines más específicos.

Veamos algunas de las más **populares**:

- **jQuery**: es una librería que permite, de manera fácil, agregar interactividad y efectos visuales en una web.
- **Moment.js**: es una librería basada en el uso y trabajo de fechas. Permite la manipulación de éstas de forma sencilla, de manera que dar formato a una fecha es mucho más rápido y con más posibilidades que los métodos de Date propios de JavaScript.

```
moment().format ("MMMM Do YYYY, h:mm:ss a');
moment().format('dddd');
moment().format ("MMM Do YY");
moment().format('YYYY [escaped] yyyy);
```

- **anime.js**: librería creada para animar de forma sencilla diferentes propiedades CSS o SVG en una página web.
- **Chart.js**: es una librería que nos permite de forma fácil incluir gráficos animados e interactivos en un sitio web.



- **Math.JS**: es una librería matemática, que permite trabajar de forma extensa con todo tipo de expresiones, funciones y constantes.

Cuestionario

Lee el enunciado e indica la opción correcta:

¿Cuál es el propósito principal de la librería Moment.js?

a. Agregar interactividad y efectos visuales en una web.

b. Manipular fechas de forma sencilla.

c. Animar propiedades CSS o SVG en una página web.

Lee el enunciado e indica la opción correcta:

¿Cuál de las siguientes librerías se utiliza para agregar gráficos animados e interactivos en un sitio web?

a. jQuery.

b. Chart.js.

c. Math.js.

Lee el enunciado e indica la opción correcta:

¿Cuál de las siguientes librerías proporciona funcionalidad matemática extensa?

a. jQuery.

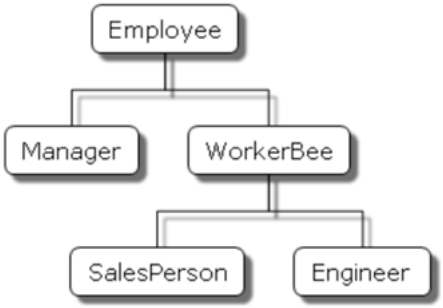
b. Math.js.

c. Moment.js.

9. CREACIÓN Y USO DE PROTOTIPOS

JavaScript es un lenguaje orientado a objetos basado en **prototipos**, en lugar de en clases, como suele ser habitual en otros lenguajes. Así como en los lenguajes tradicionales una **clase** puede servir para **instanciar objetos**, en JavaScript un prototipo nos permite crear nuevos objetos a partir de él.

Para trabajar con el concepto de **prototipo** usaremos la siguiente **jerarquía**, de empleados.



A continuación, veremos como crear un **prototipo** y como **usarlos**:

- **Prototipo de empleado**: podemos crear un prototipo de Empleado, con dos propiedades (nombre y departamento) de la siguiente manera:

```
function Employee() {
  this.name = ""
  this.dept = "General"
}
```

La partícula "**this**" permite hacer visible la propiedad desde fuera, algo así como el "**public**" en otros lenguajes.

Ahora podríamos crear un empleado:

```
var e1 = new Employee()
e1.name = "John"
```

➔ **Prototipo manager y trabajador**: ahora podemos crear dos nuevos prototipos, para representar el manager y el trabajador, añadiendo propiedades sobre informes y proyectos respectivamente:

```
function Manager() {
    Employee.call(this)
    this.reports = []
}
Manager.prototype = Object.create(Employee.prototype)

function WorkerBee() {
    Employee.call(this)
    this.projects = []
}
WorkerBee.prototype = Object.create(Employee.prototype)
```

➔ **Crear instancias**: para crear ahora instancias de estos objetos lo haremos como sigue:

```
var juan = new Employee()
juan.name = "Juan Manzano Martínez"

var ana = new Manager()
ana.name = "Ana Valiente Delgado"
ana.dept = "Dirección"
ana.reports = ["Finanzas", "Organización", "Recursos humanos"]
```

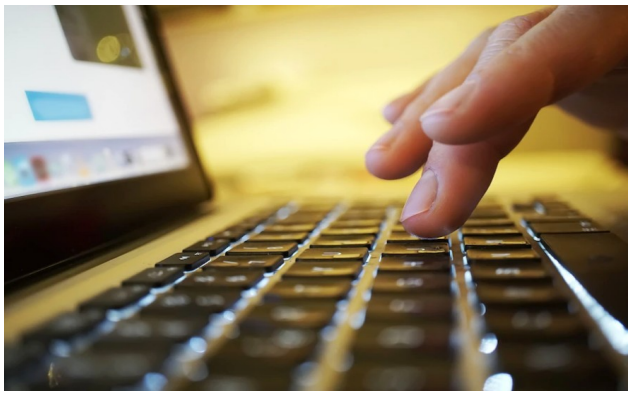
➔ **Dar valor a las propiedades**: si quisiéramos dar valor a las propiedades en el momento de crear el objeto deberíamos realizar unas modificaciones en la declaración:

```
function Employee (name, dept) {
    this.name = name || ""
    this.dept = dept || "General"
}
```

Lo cual nos permitiría instanciar los objetos de manera más sencilla:

```
var juan = new Employee("Juan Manzano Martínez")
```

10. GESTIÓN DE CLASES



Como hemos dicho, JavaScript utiliza el modelo de **prototipos**. La ida del prototipo es buena, pero la **sintaxis** es un poco complicada.

Debido a eso, y aprovechando las nuevas especificaciones de **ECMAScript 2015 (ES6)**, podemos continuar utilizando los prototipos, pero los disfrazamos de clases.

Crear una clase

Vamos un ejemplo de cómo crear una clase que nos permita representar un polígono:

```
class Polygon {
    constructor(height, width) {
        this.height = height
        this.width = width
    }

    get h() {
        return this.height
    }
    set h(value) {
        this.height = value
    }
    get w() {
        return this.width
    }
    set w(value) {
        this.width = value
    }
```

```
    get area() {
        return this.calcArea()
    }
    calcArea() {
        return this.height * this.width;
    }
}
```

En primer lugar, vemos un **método** que nos servirá de **constructor de la clase**. Seguidamente tenemos métodos que representan los **getters** y **setters** de las distintas propiedades. Por último, tenemos un método que nos ayudará en el cálculo del área.

Una vez declarada la clase, veamos cómo podemos **instanciar objetos** de esa clase, y acceder a sus valores.

```
var rectangulo = new Polygon(10, 10)
document.write(rectangulo.area)
rectangulo.h = 25
document.write(rectangulo.area)
```

En el ejemplo, creamos un rectángulo de 10 x 10. Mostramos el área por pantalla. Posteriormente cambiamos la altura, a 25 y volvemos a mostrar el área.

Actividad de aprendizaje 2: codificar prototipo de objeto

Para acabar con el tema de las encuestas nos proponen guardar la información de cada voto, de manera anónima, pero eso sí, sabiendo la fecha en que se votó.

Para eso nos proponen codificar un prototipo de objeto que guarde el color escogido y la fecha. Además, a la hora de mostrarlo por pantalla tendréis que hacer uso de la librería Moment.js, para formatear la fecha de manera adecuada.

Entrega el archivo .html en el espacio habilitado para ello.

Cuestionario

Lee el enunciado e indica la opción correcta:

¿Cuál de las siguientes no es una librería de terceros?

a. Moment.js

b. Math

c. Chart.js

Lee el enunciado e indica la opción correcta:

En la definición de un prototipo indicamos…

a. Setters y getters.

b. Constructores.

c. Propiedades y métodos.

Lee el enunciado e indica la opción correcta:

¿Con que palabra se define un método?

a. function.

b. func.

c. method.

RESUMEN

A través de conceptos más complicados hemos ido avanzando en nuestros conocimientos del lenguaje **JavaScript**.

Gracias a esto, hemos sido capaces de crear cada vez programas más complicados, utilizando las **ventajas** que nos aportan, por ejemplo, las funciones predefinidas del lenguaje.

Éstas las hemos podido complementar con las **creadas por el propio desarrollador** y las llamadas **librerías de terceros**, que permiten compartir código entre diferentes usuarios. Con este conjunto de funciones, se simplifica mucho la programación, consiguiendo, además, muchos mejores resultados.

Hemos incidido en el concepto de **array**, viendo cómo declararlos, inicializarlos, y recorrerlos, para, por ejemplo, hacer búsquedas. Además, hemos visto **nuevos métodos** que amplían las posibilidades de los arrays.

Por último, hemos trabajado con el concepto de **prototipo**, comparándolo, de alguna manera con la tradicional programación orientada a objetos, y viendo las principales diferencias entre éstos y las clases.

Todo esto nos sirve para seguir avanzando en nuestro objetivo de ser un buen programador web.

RECURSOS PARA AMPLIAR

PÁGINAS WEB

- anime.js: *https://animejs.com/* [Consulta junio 2022].
- Chart.js: *https://www.chartjs.org/* [Consulta junio 2022].
- JavaScript Array Reference: *https://www.w3schools.com/jsref/jsref_obj_array.asp* [Consulta junio 2022].
- jQuery: *https://jquery.com/* [Consulta junio 2022].
- math.js: *https://mathjs.org/* [Consulta junio 2022].
- Moment.js | Home: *https://momentjs.com/* [Consulta junio 2022].

BIBLIOGRAFÍA

PÁGINAS WEB

- JSON - JavaScript | MDN: *https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON* [Consulta junio 2022].
- W3C: *https://www.w3.org/* [Consulta junio 2022].

GLOSARIO

- **Inicializar un array**: dar valor a cada de sus posiciones, usualmente en la declaración.
- **Instanciar**: crear un objeto a partir de una clase, con unos valores propios.
- **Métodos**: funciones que permiten a un objeto realizar acciones o cálculos.
- **Propiedades**: atributos de un objeto que definen su identidad.