Uso de librerías

JavaScript es uno de los lenguajes más populares a nivel mundial, y por tanto, hay una gran comunidad de usuarios que lo utiliza. Esto facilita el hecho de que algunos de esos usuarios o grupos desarrolle **librerías**. Una librería básicamente es un conjunto de funcionalidades implementadas que podemos incorporar y utilizar directamente en nuestras aplicaciones. Por lo general proporcionan una serie de funciones y objetos definidos, que evitan que tengamos que "reinventar la rueda" si queremos hacer algo parecido en nuestros programas.

En este documento veremos algunos ejemplos de librerías JavaScript y cómo utilizarlas en nuestras webs.

1. Chart.js

Chart.js es una librería que proporciona un conjunto de elementos que nos permiten incorporar gráficas en nuestras páginas de forma sencilla. La vamos a utilizar para, por un lado, ver cómo incluir librerías de terceros en nuestras aplicaciones web, y por otro, ver concretamente cómo usar las funcionalidades proporcionadas por esta librería. Podemos consultar información actualizada en su web oficial.

1.1. Descarga e instalación

El procedimiento para utilizar cualquier librería JavaScript es muy similar, aunque en algunos casos se complica más porque debemos incluir varios ficheros, e incluso manipular archivos CSS. En el caso de *Char.js*, en la propia web nos dan diferentes alternativas:

- Por ejemplo, podemos usar el gestor de paquetes *npm* si tenemos instalado *Node.js* en el sistema.
- También podemos descargar la librería desde su repositorio GitHub. En este caso descargamos un archivo comprimido. Sólo necesitamos enlazar con el archivo `chart.min.js` o, en su defecto, el archivo `chart.js`, que está menos optimizado para su carga. Ambos se encuentran dentro de la subcarpeta *dist* (es el único archivo que necesitamos copiar a nuestra web).
- Alternativamente, también podemos enlazar con la versión online de la librería en CDN:

```
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
```

1.2. Uso básico

Una vez incorporado el *script* correspondiente en la(s) página(s) que lo necesite(n), ya podemos utilizar los elementos que incorpora la librería. Por ejemplo, vamos a representar en un gráfico de barras la temperatura media (en rojo) y lluvias acumuladas (en azul) en los meses de Enero a Marzo. Para ello, definimos un objeto JavaScript con los siguientes campos:

- `labels` : etiquetas que se pondrán en el eje X. En este caso, los meses del año
- `datasets` : un array con cada conjunto de datos que queramos representar. En este caso habrá dos grupos: uno en azul con los datos de lluvias y otro en rojo con los datos de temperaturas medias. Los colores de fondo se especifican con la propiedad *backgroundColor*, y también podemos especificar un color de contorno con *borderColor*. El objeto JavaScript puede quedar así:

```
let datos = {
    labels : ["Enero","Febrero","Marzo"],
    datasets : [
    {
        label: "Lluvias acumuladas",
        backgroundColor : "blue",
        borderColor : "black",
        data : [100, 80, 125]
    },
    {
        label: "Temperaturas medias",
        backgroundColor : "red",
        borderColor : "black",
        data : [8, 10, 16]
    }]
};
```

Después, construimos un objeto con la configuración del gráfico. En él indicamos el tipo de gráfico (campo `type`), los datos que va a cargar (campo `data`, donde asociaremos el objeto *datos* anterior) y opciones adicionales de visualización, si son necesarias.

```
let config = {
    type: 'bar',
    data: datos,
    options: {}
};
```

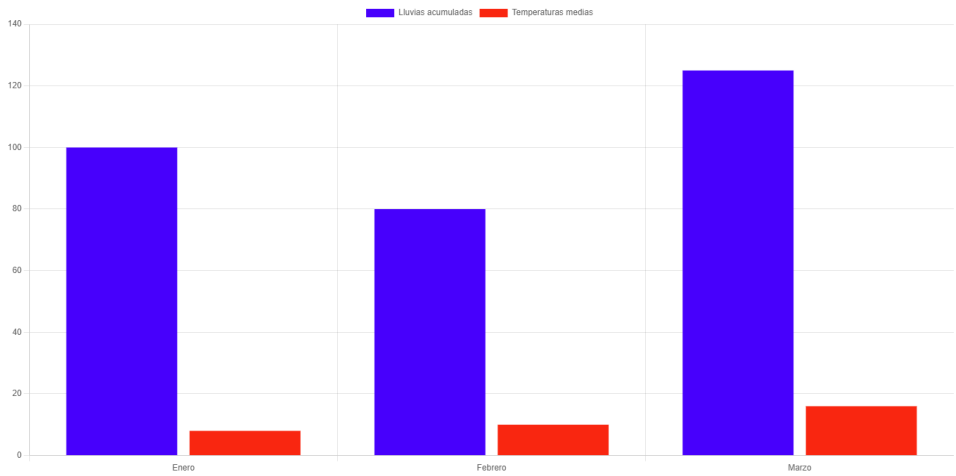
En cuanto al código HTML, básicamente deberemos definir un canvas donde mostrar el dibujo (normalmente los gráficos se dibujan sobre un objeto *canvas*).

```
<body>
    ...
    <canvas id="grafico" height="450" width="600"></canvas>
    ...
```

Y después, crear un objeto de tipo *Chart* e indicarle que cargue la configuración anterior en el canvas (esto deberemos hacerlo una vez esté cargado el canvas en la página, evidentemente)

```
const chart = new Chart(document.getElementById("grafico"), config);
```

Al final quedará algo como esto:



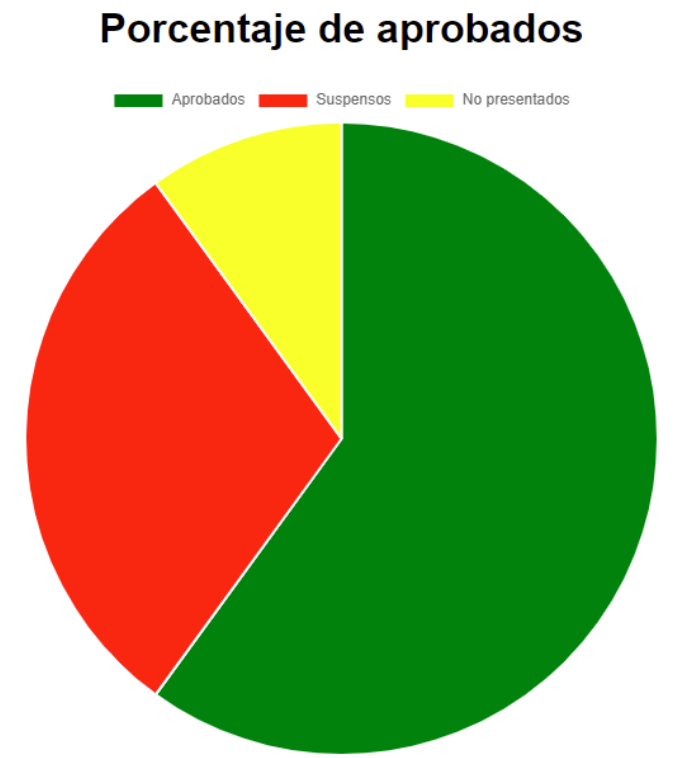
NOTA: el gráfico por defecto va a tomar todo el ancho disponible, y en función de eso ajusta el alto según las proporciones del *canvas* que hayamos definido. Si queremos acotar más en qué zona de la página desplegar el gráfico, debemos incluirlo en algún tipo de contenedor (*div*, *section*), y establecer el tamaño de dicho contenedor: anchura, altura, *grid*...

Ejercicio 1:

Crea una página llamada **graficos.html**. Usando la librería *Chart.js*, investiga en su documentación cómo hacer:

- Un gráfico de líneas para representar la evolución de las notas de una asignatura (con las notas 4, 7, 5, 8 y 7),
- Un gráfico circular para representar el porcentaje de aprobados (60%), suspensos (30%) y no presentados (10%) de una asignatura.

Al final deberán quedarte estos dos gráficos:



2. Moment.js

Moment.js es una librería que nos permite gestionar de forma cómoda fechas, para mostrarlas en distintos formatos y para hacer operaciones con ellas: calcular diferencias entre fechas, ver cuántos días faltan para una fecha determinada, etc.

Podemos descargar la librería desde la web oficial. Necesitamos el archivo `moment.min.js`, que copiaremos a nuestra web y enlazaremos con la correspondiente etiqueta *script*.

2.1. Uso básico

Una vez incorporada la librería, podemos hacer algunas operaciones básicas y útiles, como por ejemplo:

- Obtener la fecha actual creando un nuevo objeto *moment*:

```
let ahora = moment();
```

- Crear una fecha a partir de una cadena con un patrón determinado (más información sobre los patrones aquí)

```
let fechaString = "07/03/2013";
let fecha = moment(fechaString, "dd/MM/YYYY");
```

- Existen otras formas de crear fechas. Por ejemplo, pasando un array con el año, mes y día (en ese orden, y numerando el mes desde el 0 = enero):

```
// 7 de abril de 2022
let fecha = moment([2022, 3, 7]);
```

- Podemos obtener las partes de una fecha por separado (día, mes, año...) con las instrucciones `date()` (día), `month()` (mes numerado desde el 0) y `year()`.

```
// 7 de abril de 2022
let fecha = moment([2022, 3, 7]);
let dia = fecha.date();     // 7
let mes = fecha.month();    // 3 (abril)
```

- Calcular la diferencia entre dos fechas con *diff*:

```
let dias = fecha1.diff(fecha2, 'days');
let meses = fecha1.diff(fecha2, 'months');
```

- Sumar o restar a una fecha una cantidad (en días, meses, años, etc), con *add* y *substract*

```
let fechaNueva1 = fecha1.add(1, 'months');
let fechaNueva2 = fecha2.substract(2, 'years');
```

- Comprobar si una fecha es válida o no con `isValid`, o ver si es anterior o posterior a otra con `isBefore` o `isAfter`:

```
let fechaErronea = moment([2022, 3, 40]);
alert(fechaErronea.isValid());      // false
...
if (fecha1.isBefore(fecha2))
    alert("La fecha 1 es anterior");
```

- También podemos trabajar con horas, minutos y segundos. Basta con que los añadamos a continuación de los datos referentes al día, mes y año.

```
let fechaCompleta = moment('24/12/2019 09:15:00', "DD/MM/YYYY hh:mm:ss");
let fechayHora = fechaCompleta.add(20, 'minutes');
let minutos = fechaYHora.minute();
```

Aquí podéis consultar más información sobre las opciones que ofrece *Moment* para procesar fechas y horas.

> **Ejercicio 2:**
>
> Crea una página llamada **moment.html** y carga en ella la librería *Moment*. Después, sigue estos pasos:
>
> - Pide al usuario su fecha de nacimiento con *prompt*, en formato *DD/MM/YYYY*
> - Dile cuántos años tiene
> - Dile cuántos días faltan para su próximo cumpleaños
> - Dile "Buenos días", "Buenas tardes" o "Buenas noches" dependiendo de la hora actual:
>   - "Buenas noches" si estamos entre las 21:00 y las 07:00
>   - "Buenos días" si estamos entre las 07:00 y las 14:00
>   - "Buenas tardes" si estamos entre las 14:00 y las 21:00