CONCEPTOS BÁSICOS DE POO

VARIABLES

¿Qué es una variable?

- Una variable es un elemento que **contiene información** (datos) que puede ser ejecutada por el código.
- Las variables sirven para **guardar** información que será utilizada posteriormente.
- Cada variable tiene un **tipo de datos** (número, cadena, fecha…)
- Javascript es un **lenguaje implícito** por lo que no necesita especificar previamente el tipo de dato que contiene una variable.
- Las variables pueden cambiar de tipo de dato, pero no es recomendable.

Para declarar una variable se indica **var** antes del nombre:

- `var name;`

Por ejemplo, para **declarar** una variable llamada “periodismo”:

- `var periodismo;`

Se pueden declarar múltiples variables en una única declaración

- `var name, address;`

Una variable se **inicializa** cuando se le otorga por primera vez un valor:

- `var x = 100, message = 'Hello';`

Una **variable** es un **contenedor de información**.

Mediante las **asignaciones** podemos guardar datos en una variable.

- ✓ El signo = significa asignación.
- ✓ Por ejemplo:

Colocar el valor 3 en la variable num:

```
num = 3;
```

En una fila se define la variable y se asigna un valor:

```
var num = 1;
```

En este ejemplo (JsFidle) o Codepen x se define como una variable. Luego, asignamos a x el valor de 6:

Ejercicio

Sobre el ejemplo de creación de variable anterior, modifícalo para:

1. Declarar e inicializar la variable x en el mismo paso
2. Declara otra variable, y, e inicialízala con una cadena (string)
3. Crea un encabezado de segundo nivel (h2) debajo del párrafo y haz que se muestre la nueva variable en él

Solución JSFidle – *Codepen*

Explicación: 1. Creamos un h2 vacío con un id 2. Declaramos la variable y y le damos un valor como string 3. Sobre el objeto document, aplicamos el método getElementByID() para capturar el id del h2, y le pasamos con innerHTML el valor de la variable y

Ejercicio

Abre el **siguiente ejercicio** (JSFidle) o Codepen que usa variables con datos numéricos y modifícalo para incluir el IVA en el precio total:

1. Crea un nuevo <p> con id subtotal
2. Declarar una tercera variable llamada IVA e inicializarla con el valor 1.21
3. A la variable total llámala subtotal
4. Crea una nueva variable total que multiplique subtotal por iva
5. Utiliza el método getElementById para escribir el resultado

Solución JSFidle – Codepen

Explicación: 1.Creamos el párrafo con el nuevo id 2.Creamos la variable IVA y le damos el valor 1.21 (para que luego haga el múltiplo sumatorio del 21%) 3.Cambiamos la variable total original a subtotal 4.Creamos la nueva variable

total como la multiplicación de las otras dos variables, subtotal * iva
Escribimos con getElementById().innerHTML = el valor con IVA

ALCANCE O ÁMBITO DE UNA VARIABLE

El **alcance o ámbito** de una **variable**. Puede ser:

- ✓ **Global**

```
<script>
    var x = 10;
</script>
```

- ✓ **Local**
  - ✓ Afecta sólo dentro de una función [como en este ejemplo, sólo para la función "hazesto"], o en bucles:

```
<script>
    function hazesto() {
        var y = 99;
    }
    alert(y);
</script>
```

Una variable es local normalmente en cualquier lugar en la que aparece acotada por llaves.

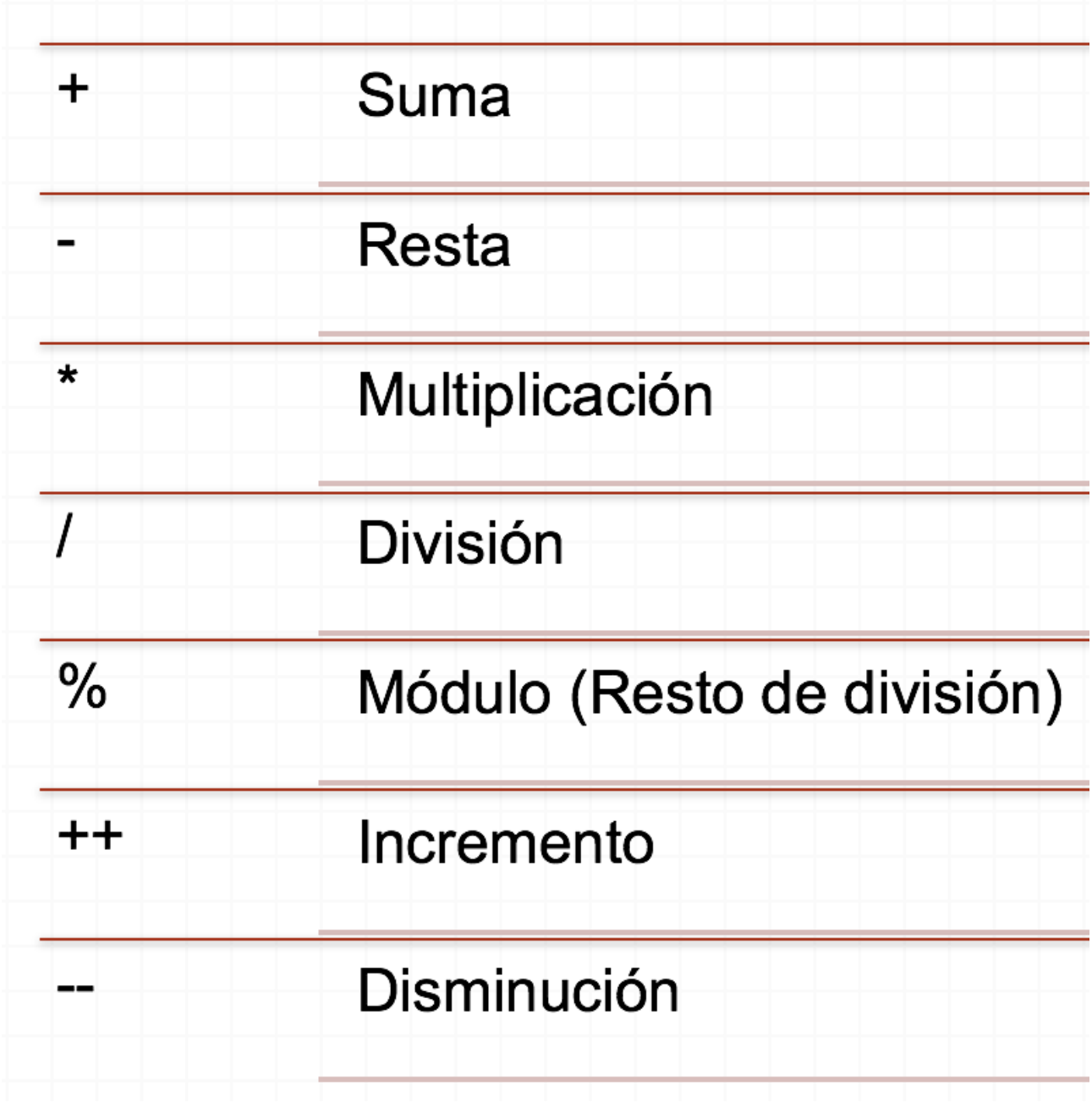
Se pueden tener una variable global y local que se llamen igual, pero no es recomendable, porque induce a confusión y hace más complejo trabajar con ellas.

OPERADORES

Los operadores son instrumentos básicos para realizar operaciones en cualquier lenguaje de programación.

> Recuerda que en Javascript el operador **=** se utiliza para la **asignación** de valor a una variable.

Operadores aritméticos



Los operadores aritméticos sirven para realizar **manipulaciones matemáticas** sobre el valor de las variables numéricas.

Un operador aritmético siempre dará como **resultado un número.**

Vamos a realizar algunos **ejercicios** para conocer su funcionamiento:

- **Suma** (JSFidle) – Codepen. A partir de este código, realiza la suma de dos variables. [Solución JSFidle – Codepen]
- **Multiplicación y división**(JsFidle) – Codepen. A partir de este ejemplo: a) Inicializa las dos variables dándoles un valor numérico. b) Crea una tercera variable y asígnale un valor. c) Escribe sobre este texto el resultado de multiplicar x * y, y dividir el resultado por la tercera variable. [Solución JsFidle – Codepen]
- **Incremento y disminución** (JsFidle) – Codepen. El incremento y disminución sirven para aumentar o disminuir un número dado. Sobre el ejemplo, vamos a restar de nuevo, con el operador de disminución para que se vuelva a quedar la variable en 5. a) Crea un nuevo párrafo con id “disminución”; b) utiiza el operado de dismunición para x; c) crea una nueva variable que asigne el

valor a x; d) escribe el resultado en el párrafo disminución. [Solución JSFidle – Codepen]

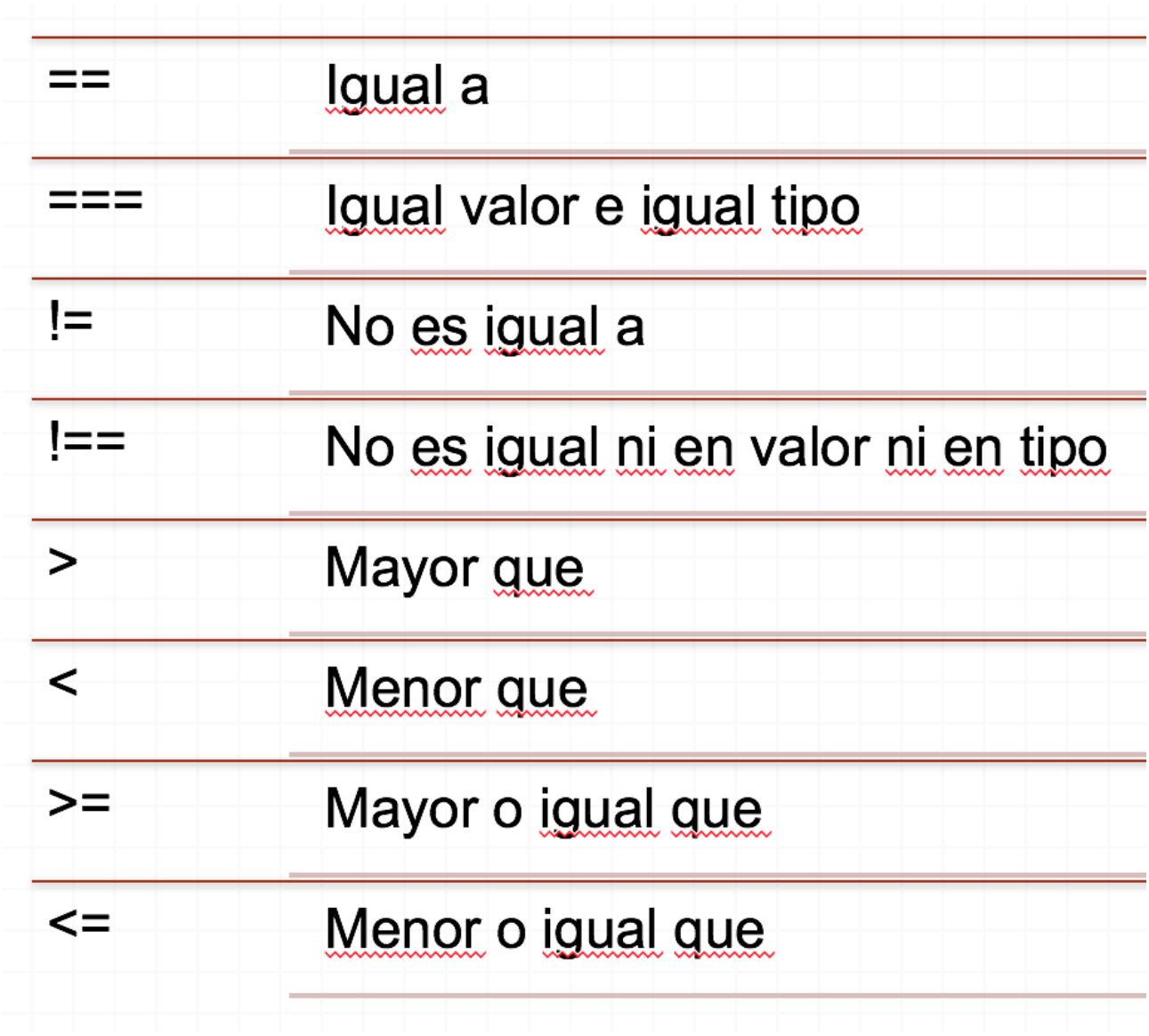
```
var numero1 = 5;
numero1 += 3;  // numero1 = numero1 + 3 = 8
numero1 -= 1;  // numero1 = numero1 - 1 = 4
numero1 *= 2;   // numero1 = numero1 * 2 = 10
numero1 /= 5;   // numero1 = numero1 / 5 = 1
numero1 %= 4;   // numero1 = numero1 % 4 = 1
```

Operadores aritméticos combinados con operador de asignación.

Los operadores aritméticos se pueden combinar con el operador de asignación.

En este enlace del tutorial JS del W3School puedes profundizar en cómo funcionan el reto de operadores aritméticos.

Operadores relacionales y condicionales



Los operadores condicionales o relacionales permiten **comparar números** y son esenciales para realizar aplicaciones complejas que se basen en el cumplimiento de condiciones

Un operador relacional siempre dará como resultado un **valor booleano**.

Veamos algunos ejemplos de operadores relacionales:

- **Igual a** ==. Hace una comparativa entre una sentencia y el valor esperado. Si el valor es cierto, dará como resultado «true»; en caso contrario, indicará «false».
- **Igual valor e igual tipo** ===. Como en el anterior, hace una comparativa entre una sentencia y el valor esperado, pero en este caso solo dará «true» si es igual el valor y el tipo de dato; en caso contrario dará «false». Por ejemplo, el valor 12 (number) y «12» (string) darían false porque son igual valor pero distinto tipo.

Operadores lógicos

Los operadores relacionales sirven para realizar **operaciones lógicas de comparación** y se utilizan para tomar decisiones en los scripts.

Se utilizan valores booleanos, por lo que un operador relacional siempre dará como r**esultado un valor booleano**.

Los operadores lógicos son:

- Negación lógica
- AND (&&)
- OR (||)

0 Negación lógica → ! (delante del nombre de la variable que se niega)

```
var visible = true;
console.log(!visible);  // Muestra "false" y no "true"
```

variable	!variable
true	false
false	true

p	q	p ^ q
V	V	V
V	F	F
F	V	F
F	F	F

Se pone el ! Delante del nombre de la variable. Retorna falso para sentencias verdaderas y verdadero para sentencias falsas. Para qué se usa: Cuando se necesitan evaluar expresiones complejas, y se usan tablas de verdad, con el fin de evitar incluir muchas sentencias con if anidados (permiten agrupar sentencias)

0 AND → && (entre dos valores o variables)

```
var valor1 = true;
var valor2 = false;
resultado = valor1 && valor2; // resultado = false

valor1 = true;
valor2 = true;
resultado = valor1 && valor2; // resultado = true
```

variable1	variable2	variable1 && variable2
true	true	true
true	false	false
false	true	false
false	false	false

Operación lógica AND: Combina dos valores boolenaos que compara. Es verdad si los dos valores son verdad



Operación lógica OR: Combina dos valores boolenaos que compara. Es verdad si uno de los valores es verdad

CONCEPTOS DE POO

La **POO (Programación Orientada a Objetos)** es un paradigma de programación que permite optimizar los procesos de programación y los resultados que se obtienen con ellos. Permite pre-diseñar objetos que son almacenados en librerías o bibliotecas para que puedan ser reutilizados por los programas sin necesidad de tener que volver a escribir las funciones necesarias cada vez que se quiere hacer uso de ellas.

Javascript es uno de los muchos lenguajes que funcionan bajo este paradigma. Dentro de las dos corrientes existentes (basarse en clases o basarse en prototipos), **JS corresponde al modelo basado en prototipos** en el que solo hay objetos. Otros lenguajes similares en este sentido, muy utilizados actualmente, son Python y Ruby.

Elementos fundamentales



Los elementos fundamentales de la POO incorporan un número amplio de componentes (clase, herencia, objeto, método..), pero nos centraremos solo en los que resultan fundamentales para manejarnos en el entorno de trabajo de un proyecto periodístico

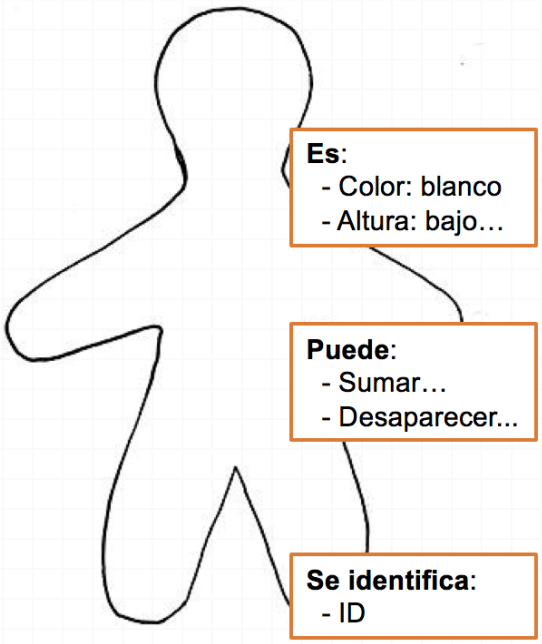
- Objetos
- Eventos
- Funciones
- Métodos

Objetos

Un objeto es una entidad definida por un estado, caracterizado por un conjunto de atributos que toman valores (datos) concretos, un comportamiento definido por los métodos, operaciones o mecanismos de interacción que pueden realizarse sobre él, y una identidad que le diferencia del resto (un identificador).

Una **entidad** definida por:

- un **estado**, caracterizado por
  - un conjunto de atributos que
    - toman valores (datos) concretos,
- un **comportamiento** definido por
  - los métodos, operaciones o mecanismos de interacción que pueden realizarse sobre él,
- una **identidad** que
  - le diferencia del resto (un identificador)



Object	Properties	Methods
entidad 	car.name = Fiat car.model = 500 car.weight = 850kg car.color = white estado	car.start() car.drive() car.brake() comportamiento

Identidad= coche45

Eventos

Los eventos son **sucesos**, producidos normalmente por una acción del usuario, que **producen algún efecto**. Por ejemplo, cuando un usuario pulsa un botón o hace clic sobre un enlace.

Algunos de los eventos DOM más habituales son:

Evento	Descripción	Elementos para los que está definido
onclick	Pinchar y soltar el ratón	Todos los elementos
onload	La página se ha cargado completamente	<body>
onmouseout	El ratón "sale" del elemento (pasa por encima de otro elemento)	Todos los elementos
onmouseover	El ratón "entra" en el elemento (pasa por encima del elemento)	Todos los elementos
onsubmit	Enviar el formulario	<form>

Veamos un par de ejemplos para entender cómo funcionan:

– **Onload**. Lanza un comportamiento cuando se ha cargado la página completamente.

– **Onmouseover – (ver en Codepen)**. Cuando pasamos por encima de un elemento, se lanza el comportamiento.

Hay muchos otros eventos como el *blur* que es cuando se dispara el evento al perder el foco un elemento, o *On Keyup* que ejecuta el evento cuando el usuario suelta una tecla.

Evento	Descripción	Elementos para los que está definido
onblur	Deseleccionar el elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onchange	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>
ondblclick	Pinchar dos veces seguidas con el ratón	Todos los elementos
onfocus	Seleccionar un elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onkeydown	Pulsar una tecla (sin soltar)	Elementos de formulario y <body>
onkeypress	Pulsar una tecla	Elementos de formulario y <body>
onkeyup	Soltar una tecla pulsada	Elementos de formulario y <body>
onmousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
onmousemove	Mover el ratón	Todos los elementos
onmouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
onreset	Inicializar el formulario (borrar todos sus datos)	<form>
onresize	Se ha modificado el tamaño de la ventana del navegador	<body>
onselect	Seleccionar un texto	<input>, <textarea>
onunload	Se abandona la página (por ejemplo al cerrar el navegador)	<body>

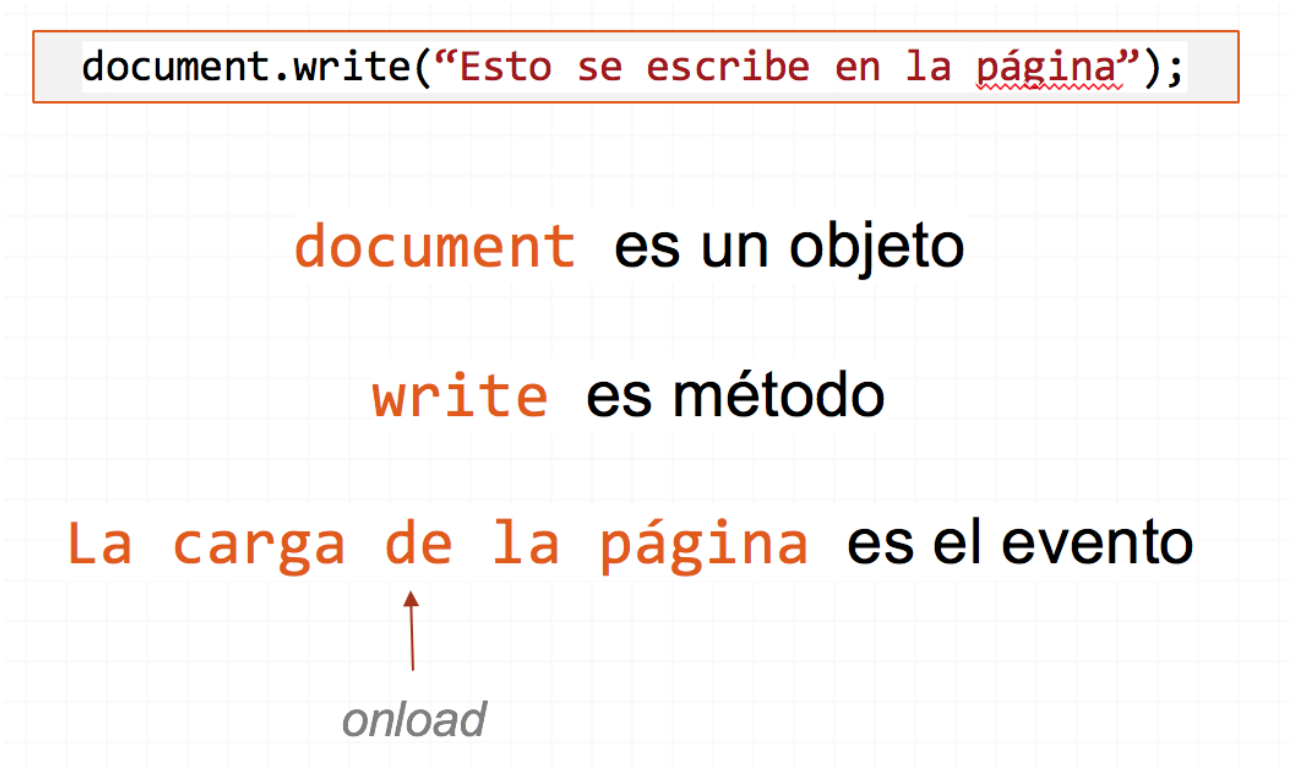
Aquí puedes ver un ejemplo del funcionamiento de onkeyup, (Ver en Codepen) que se utiliza en muchas ocasiones para modificar los datos de entrada en un formulario. En este caso, transforma las letras minúsculas en mayúsculas.
En este ejemplo, usamos también onkeyup pero para contar el número de caracteres que introducimos en un input.

Todo evento lleva asociado, normalmente, una **función**.

Las funciones son fragmentos de código que realizan alguna acción cuando son

invocadas por un evento. Por ejemplo, cuando el usuario pulsa el botón, y sucede algo.

Visto en código, y explicado, sería esto:



LIBRERÍAS Y APIS

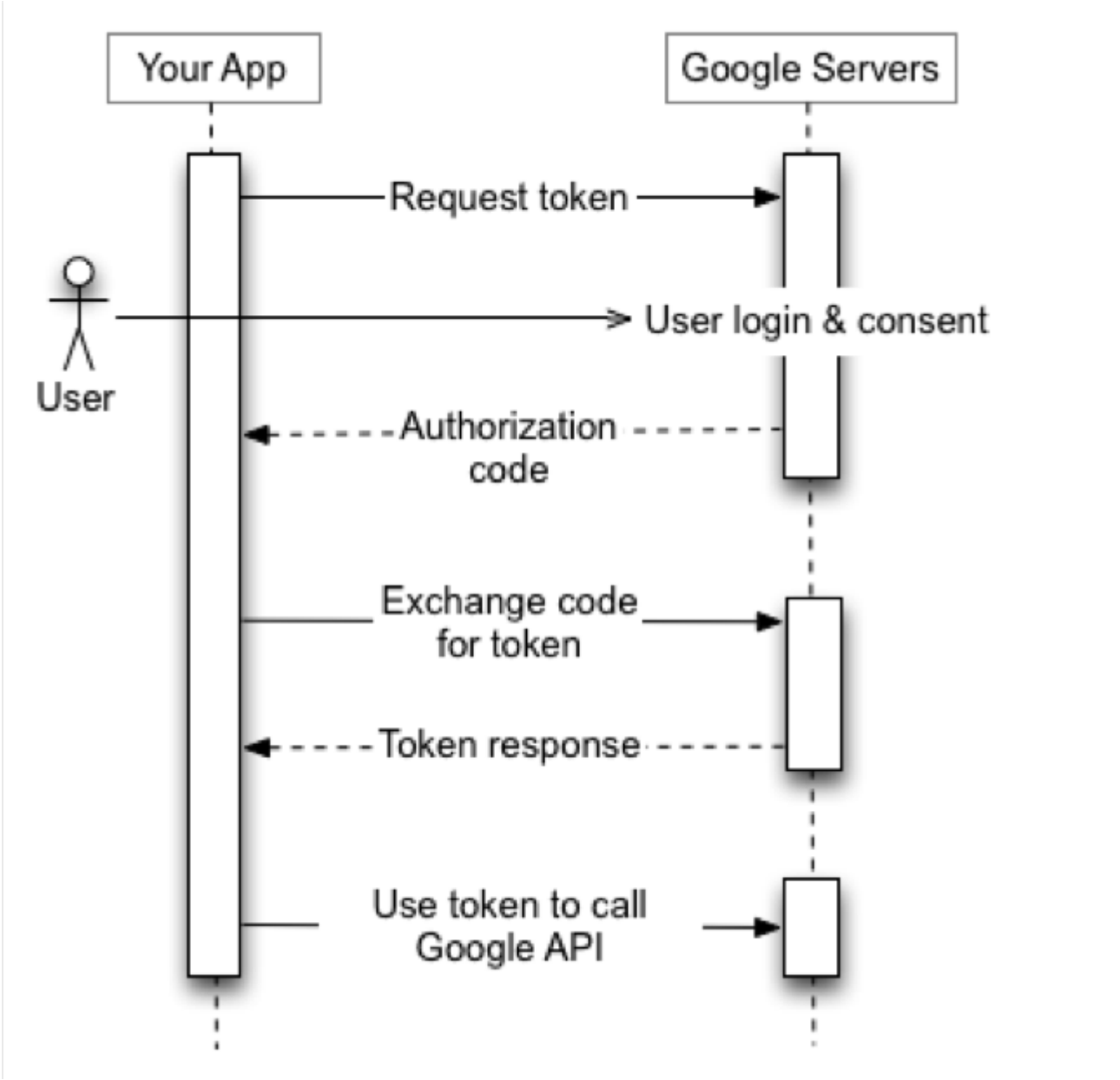
LIBRERÍAS Y APIS: ¿QUÉ SON Y CÓMO SE RELACIONAN?

Estos dos conceptos van de la mano. **Una librería** es un conjunto de funciones y comportamientos listos para usar, y **una API** son las instrucciones que nos indican *cómo* usar esas funciones dentro de nuestro propio código.

¿Qué es una librería?

Una librería o biblioteca es un conjunto de código que alguien ya ha escrito y organizado para resolver tareas comunes. En otras palabras: es una **colección de funciones** que podemos "importar" a nuestro programa para no tener que escribirlas desde cero.

Por ejemplo, hay librerías para crear gráficos, manipular fechas o mostrar mapas. En lugar de programar todo eso tú mismo, usas la librería.



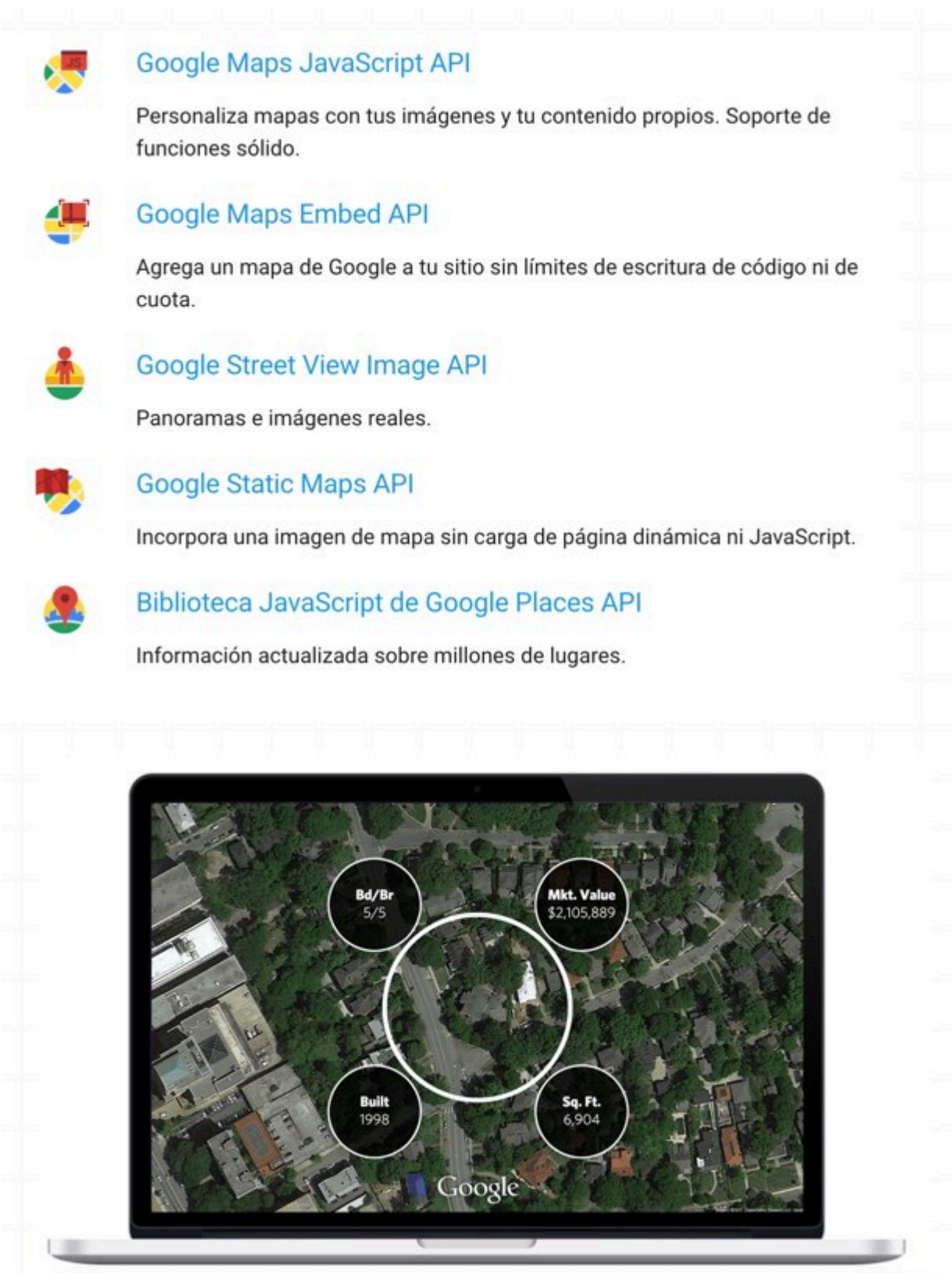
¿Qué es una API?

Una API (*Application Programming Interface*) es el **manual de uso** de una librería o servicio. Te explica qué funciones existen, qué datos necesitan y qué resultados devuelven.

Podemos pensar en una **cafetera Nespresso** como una librería (el sistema para hacer café) y en su **libro de instrucciones** como la API: te indica cómo preparar cada tipo de café y qué botones pulsar.

Las APIs permiten que distintos programas "hablen entre sí". Por eso son tan útiles cuando queremos conectar, por ejemplo, nuestro sitio web con Google Maps o Twitter.

Ventajas de usar Librerías y APIs



Trabajar con librerías y APIs **ahorra tiempo y esfuerzo**. En lugar de escribir miles de líneas de código, aprovechamos soluciones ya creadas y probadas.

Por ejemplo, con la **librería de Google Maps** podemos dibujar mapas, añadir marcadores o calcular rutas sin necesidad de programar la lógica desde cero. Solo seguimos las instrucciones de su API.

- Las librerías contienen las funciones.
- Las APIs explican cómo usarlas.

- Nosotros solo tenemos que “llamarlas” desde nuestro programa.

API Keys

Para acceder a la mayoría de las APIs necesitamos una **API Key**, una clave de autenticación que identifica nuestra aplicación y limita su uso.

Esto permite:

- Controlar cuántas veces se usa el servicio.
- Evitar abusos o accesos no autorizados.

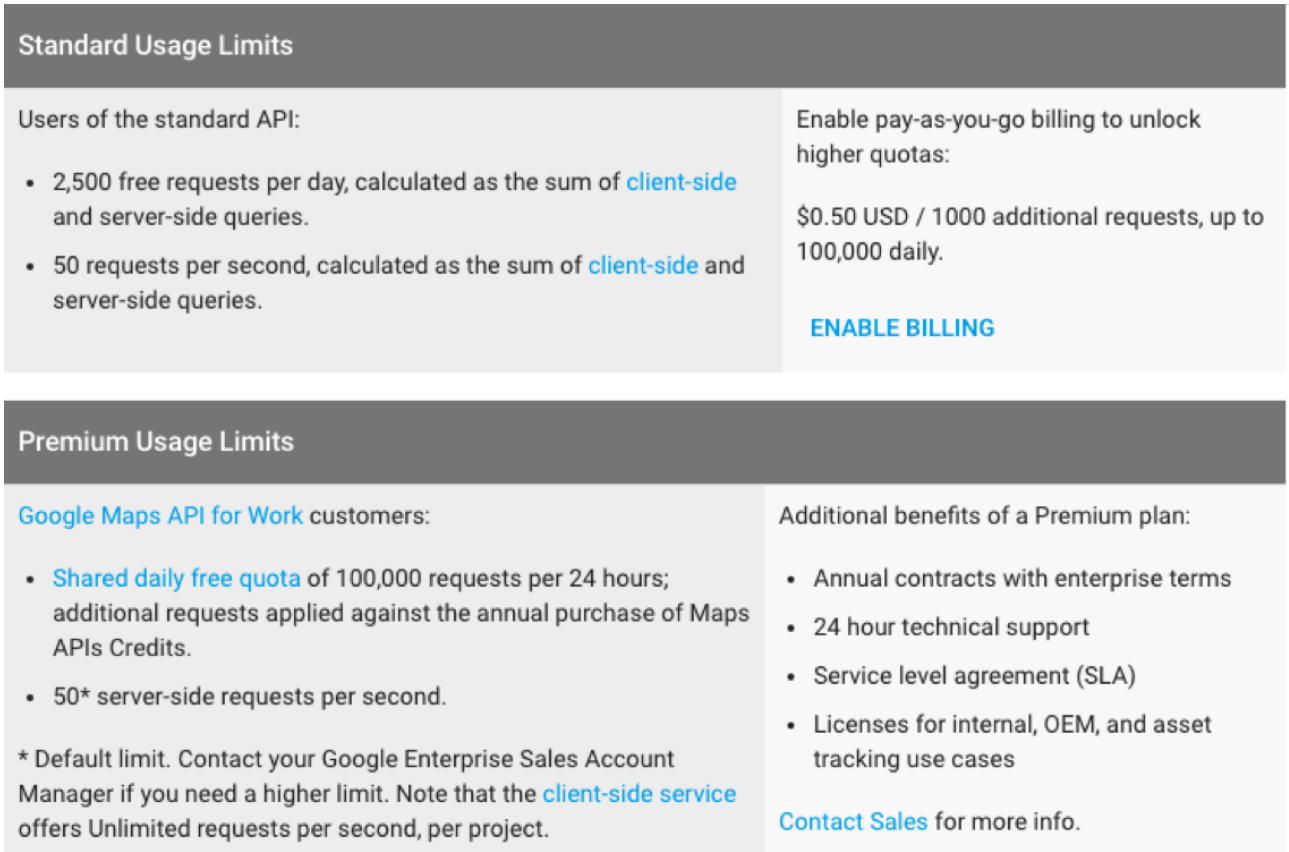
Cómo obtener la API Key de Google Maps

1. Entra en la Google API Console.
2. Crea un nuevo proyecto.
3. Asigna un nombre a la clave (no hace falta restringirla para la práctica).
4. Copia la **clave de API** que te genera Google.

En un proyecto real deberías restringir la clave por seguridad (por dominio, IP o aplicación).

Límites de uso

Casi todas las APIs tienen **límites de uso gratuito**. Por ejemplo, la Google Maps API permite un número máximo de llamadas sin coste cada día.



Standard Usage Limits

Users of the standard API:

- 2,500 free requests per day, calculated as the sum of client-side and server-side queries.
- 50 requests per second, calculated as the sum of client-side and server-side queries.

Enable pay-as-you-go billing to unlock higher quotas:

$0.50 USD / 1000 additional requests, up to 100,000 daily.

ENABLE BILLING

Premium Usage Limits

Google Maps API for Work customers:

- Shared daily free quota of 100,000 requests per 24 hours; additional requests applied against the annual purchase of Maps APIs Credits.
- 50* server-side requests per second.

* Default limit. Contact your Google Enterprise Sales Account Manager if you need a higher limit. Note that the client-side service offers Unlimited requests per second, per project.

Additional benefits of a Premium plan:

- Annual contracts with enterprise terms
- 24 hour technical support
- Service level agreement (SLA)
- Licenses for internal, OEM, and asset tracking use cases

Contact Sales for more info.

TIPOS DE LIBRERÍAS Y APIS

APIs y librerías útiles para proyectos periodísticos

Estas son algunas de las librerías y APIs más usadas en proyectos de periodismo de datos e interactivos. Incluyen tanto opciones abiertas como servicios de Google muy populares.

APIs y librerías para proyectos periodísticos

Categoría	Librerías / APIs	Uso típico
Mapas y geodatos	Leaflet, MapLibre GL JS, Google Maps JS API, Google Geocoding API	*Mapas interactivos, rutas, marcadores y geocodificación.*
Animación	GSAP, Lottie, Web Animations API	*Scrollytelling, transiciones y efectos visuales.* Ejemplo de uso de GSAP. Ejemplo de uso de Lottie.
Visualización de datos	Google Charts, D3.js, Chart.js, Vega-Lite, Observable Plot	*Gráficos interactivos y dashboards periodísticos.*
DOM y Web APIs	Fetch API, querySelector, IntersectionObserver	*Cargar datos y manipular elementos sin dependencias.*
Aplicaciones y UI	React, Vue, Svelte	*Componentes y módulos interactivos reutilizables.*
Imágenes y gráficos base	Canvas API, WebGL, three.js	*Efectos personalizados y visualización 3D ligera.*
Datos y feeds	Papa Parse, Google Knowledge Graph API, fetch en Node	*Cargar CSV/JSON, enriquecer entidades y procesar datasets.*
Contenido multimedia	YouTube Data API, Google Photos Library API	*Integrar vídeos o galerías en reportajes interactivos.*
Verificación y noticias	Google Fact Check Tools API, NewsAPI, The Guardian Open Platform	*Fact-checking, titulares y acceso a artículos.*

Consejo: para proyectos abiertos y ligeros, usa Leaflet + Chart.js; si ya trabajas con Google, combina Maps/Geocoding con Google Charts o D3.js.

ANYCHART

AnyChart

Para trabajar con un ejemplo de librería de gráficos o DataViz, para usar el Playground de AnyChart.

Nota: Recordad que tenéis que copiar el código o embeber el ejemplo en vuestro porfolio de JSFiddle o de Codepen para que quede recogido en este.

See the Pen

AnyChart – Donut Chart by gertrudix (@gertrudix)

on CodePen.

GOOGLE CHART

Google Charts

Google Charts es otro ejemplo de librería completa para realizar visualizaciones de datos.

Vamos a ver cómo trabajar un ejemplo:

1. Accede a la Galería de gráficos

2. Selecciona el diagrama de Sankey.

3. Copia el código y pégalo en tu JsFiddle. El resultado debería ser similar a este en JsFidle o en Codepen.

4. Modifica los parámetros para ajustarlo a un modelo de visualización que te pueda ser útil.

Puedes ver en este **ejemplo en Codepen**, cómo usar otro set de gráficos de la librería, con un caso sobre energías renovables.

FUNCIONES. INTRODUCCIÓN

Utilidad

Una función es un **bloque de código que puede ser ejecutado** cuando es llamada por un evento

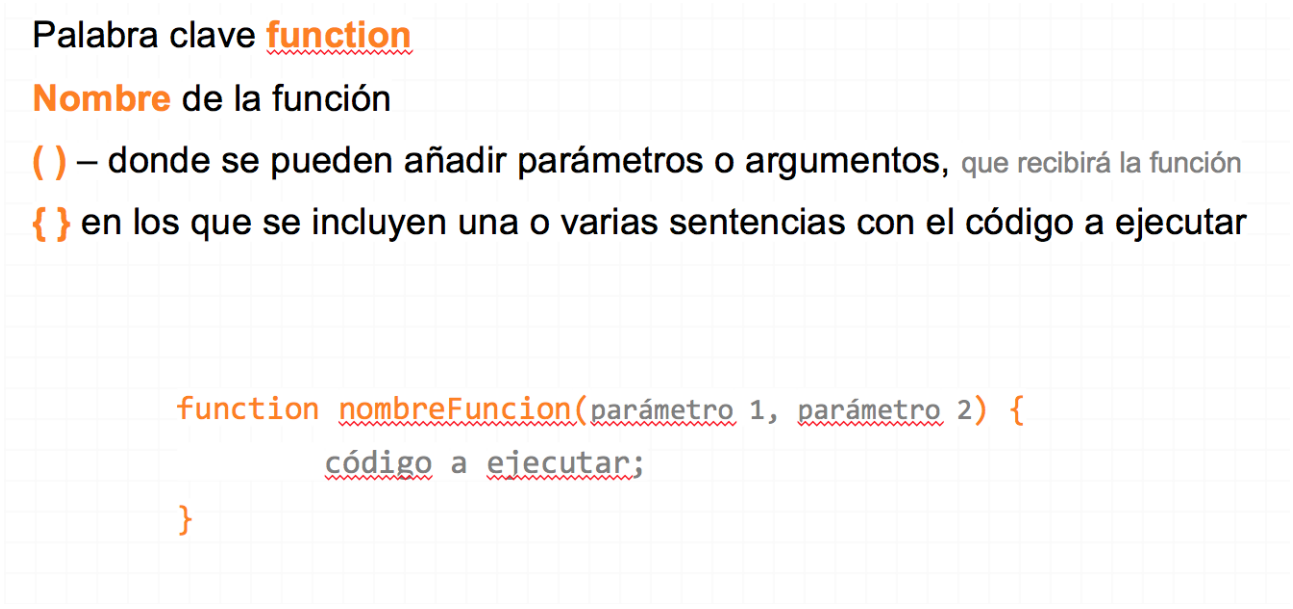
Son muy útiles porque es muy habitual reutilizar código con diferentes argumentos a lo largo de un programa, por lo que podemos utilizar la función sin tener que escribir de nuevo el código, sólo **invocándola**

```
<!DOCTYPE html>
<html>
 <head>
   <script>
    function myFunction()
    {
     alert("Cerrad la puerta cuando vayáis a salir");
    }
   </script>
 </head>
 <body>
  <button onclick="myFunction()">Pulsa aquí</button>
  <p>Cuando hagas clic en el botón se cargará una función. La función mostrará una
alerta</p>
 </body>
</html>
```

Nota: Como veremos más adelante, los paréntesis que van detrás del nombre de la función sirven para pasar parámetros (información) a la función.

Sintaxis

Una función se define del modo siguiente:

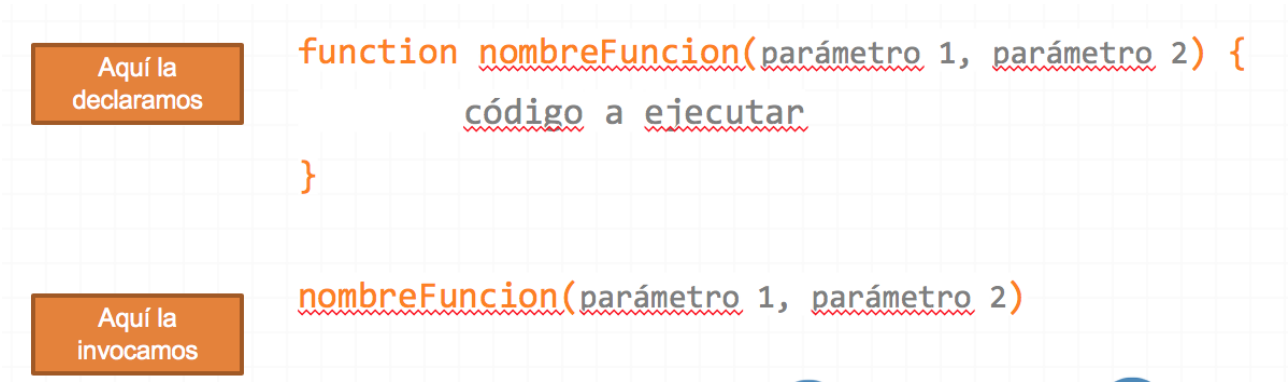


¿Cómo se llama a una función?

Para **llamar a una función** hay que invocarla en cualquier parte de la página web.

Cuando se invoca una función, todo el código que contenga entre **llaves {}** se ejecuta

Para invocarla, basta con escribir su nombre seguido de paréntesis



Ejemplos

Veamos un ejemplo explicado para el uso de las funciones.

Ejemplo sencillo en JSFidle o en Codepen. En este caso, la función, llamada cambiaTexto, escribe en el objeto documento el texto que se pasa como argumento en el método write.

Ahora podemos hacer una variante escribiendo dentro del script, directamente la función para invocarla: cambiaTexto(); y se escribirá el texto directamente sin llamar al botón.

Ejercicio

Vamos a realizar ahora un ejercicio para comprobar cómo opera una función.

1. Abre https://jsfiddle.net/ o en Codepen
2. Escribe una función llamada escribeConsola
3. Haz que registre, en la consola, el siguiente string o cadena de texto: "Esta es mi primera función que aparece en la consola
4. Abre la consola para comprobar el resultado

Solución en JSFidle o en Codepen

Pasos: 1. Abrimos la consola JS del navegador

2. Escribimos la función

```
function escribreConsola()      {        console.log("Esta es mi primera función que aparece en la consola");      } 3. La invocamos escribeConsola()
```

TRABAJANDO CON FUNCIONES

Vamos a adentrarnos en el uso de funciones. Para ello, vamos a trabajar con la librería Leaflet, un framework open source, ligero y muy extendido para crear aplicaciones de mapas web.

Crear un mapa simple incluyendo un marcador

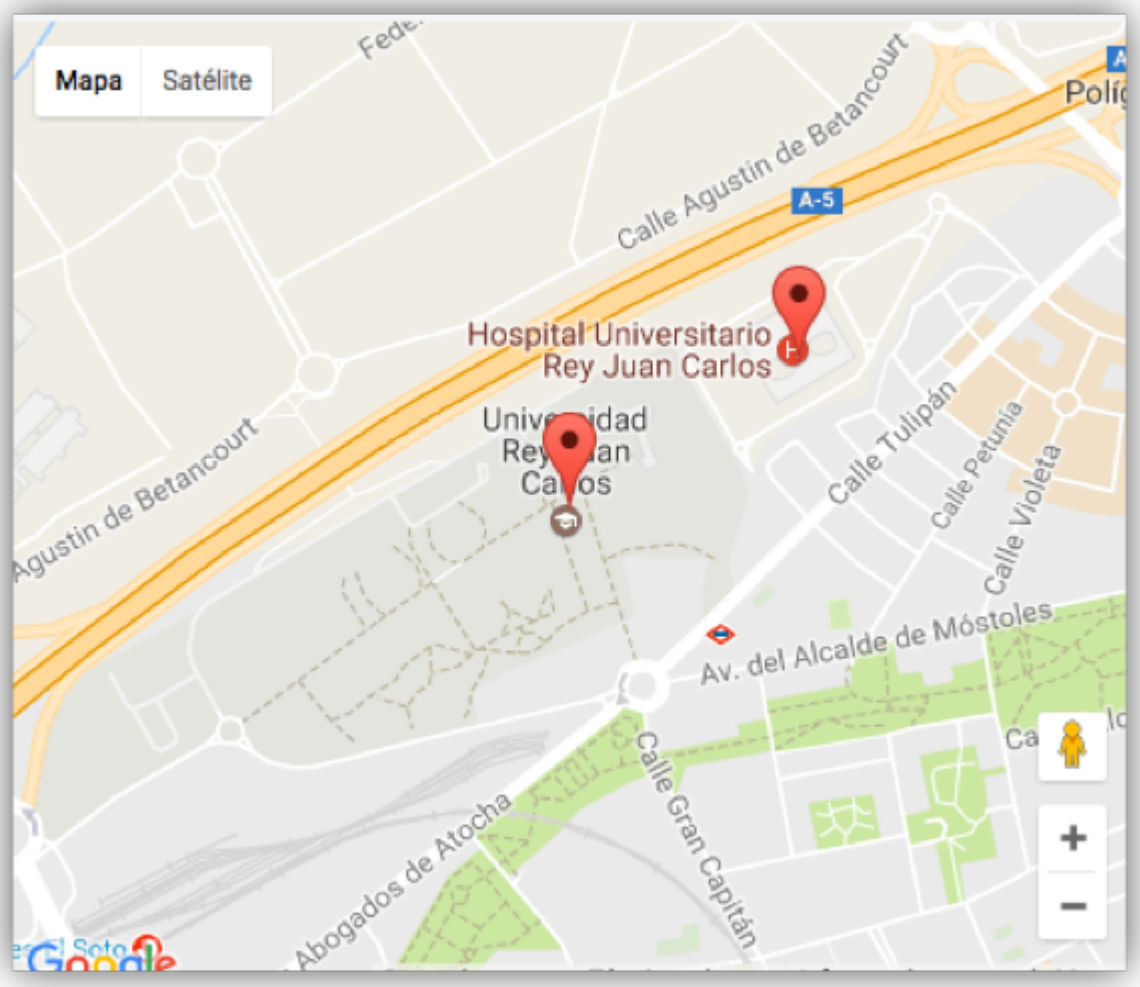
La elaboración de este mapa nos permitirá comprender mejor cómo operan las funciones con Leaflet.

Vamos a elaborar un **mapa sencillo en el que incluiremos un marcador**:

1. **Prepara el entorno**. En CodePen (o tu HTML), añade las dependencias de Leaflet (CSS y JS). En el HTML crea un contenedor para el mapa, por ejemplo . En el CSS dale altura (p. ej. #map { height: 420px; }).
2. **Inicializa el mapa y el marcador**. En JavaScript:
    a. Declara una variable con las coordenadas que centrarán el mapa (lat, lng).
    b. Crea el mapa con L.map('map').setView([lat, lng], zoom), y añade una capa base OSM con L.tileLayer(...) (recuerda la *attribution*).
    c. Crea un marcador con L.marker([lat, lng]).addTo(map) y añade un .bindPopup() si quieres contenido emergente.

En la documentación oficial de Leaflet tienes un Quick Start muy claro con estos pasos.

Ejercicio



Sobre el ejemplo anterior, cambia el mapa para que:

- Se centre en el **Campus de Móstoles de la URJC** (zoom 15) (lat: 40.3367965478043, lng: -3.874826431274414)
- Incluya un **marcador** en dicho Campus
- Incluya un **segundo marcador** en el Hospital Rey Juan Carlos (lat: 40.338759259710955, lng: -3.8707923889160156)

Solución: Como suele ser habitual, podemos resolverlo de varias maneras:

a) **Solución 1**. Creando una nueva variable para la segunda localización y pasándola al constructor de L.marker().

b) **Solución 2**. Pasando directamente la latitud y longitud en el L.marker([lat, lng]) del segundo marcador.

Parámetros en funciones

Los parámetros son los **valores de entrada que recibe una función**. Se indican entre paréntesis detrás del nombre de la función. Adaptaremos los ejemplos al contexto de Leaflet.

Parámetros simples

En este ejemplo con un parámetro simple:

1. Escribimos una función que recibe un **texto** y devuelve un popup listo para usar: por ejemplo, function popupTexto(t) { return L.popup().setContent(t); }
2. Al crear el marcador, encadenamos .bindPopup(popupTexto('Hola, Móstoles')).
3. Invocamos la función pasando el valor (el mensaje) que deseamos mostrar.

Múltiples parámetros

Habitualmente las funciones llevan varios parámetros. En Leaflet, una función puede construir un elemento del mapa con propiedades distintas. Por ejemplo, en este ejemplo se crea un **circleMarker** con múltiples parámetros:

1. Creamos la función creaPunto(lat, lng, texto, color) que devuelve un L.circleMarker([lat,lng], { color }) con .bindPopup(texto).
2. Declaramos las variables con los valores a pasar.
3. Invocamos la función con cada conjunto de parámetros que queramos (coordenadas, texto del popup y color del punto).

Ejercicio con parámetros múltiples 1 Sobre la base del ejemplo anterior, pasa ahora **tres parámetros**: además del texto y el color, pasa el **radio** del circleMarker para controlar su tamaño.

Solución

1. Añade el parámetro radio en la función ·
2. En las opciones del circleMarker utiliza { color, radius: radio } ·
3. Declara una variable miRadio con un número ·
4. En la invocación pasa el nuevo parámetro.

Grupo Ciberimaginario | Manuel Gertrudix - Alejandro Carbonell | 2025/2026 | Esta obra está bajo una Licencia Creative Commons Atribución 4.0 Internacional. Los contenidos citados se ajustan a lo regulado en el art. 32 del TRLPI de España