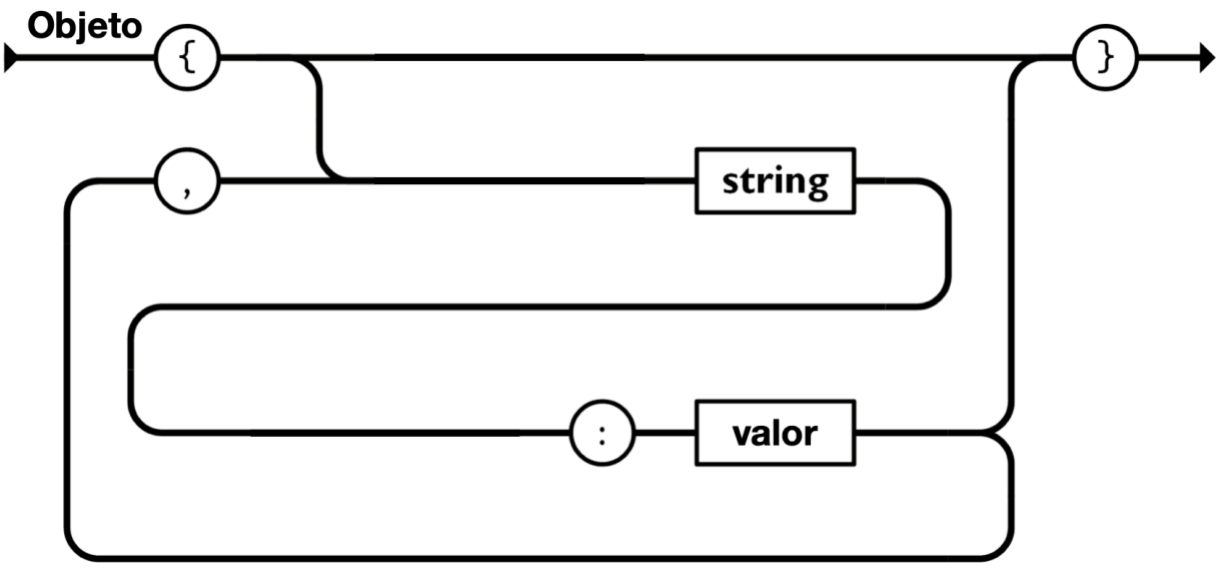
HINTS

OBJETOS JSON

JSON proviene de la sigla “JavaScript Object Notation”, o “Notación de objetos de JavaScript” en castellano. Es un formato ligero de intercambio de datos. Según json.org, “se basan en un subconjunto del estándar de lenguaje de programación JavaScript ECMA-262, de la 3ra edición (diciembre de 1999). Este formato de texto es completamente independiente del lenguaje, pero utiliza convenciones que son familiares para los programadores de la familia de lenguajes C, incluidos: C, C++, C#, Java, JavaScript, Perl, Python y muchos otros. Estas propiedades hacen de JSON un lenguaje de intercambio de datos ideal.”

Deben ser creados entre llaves, y en su interior pueden contener las siguientes dos estructuras:

- Una colección de pares de nombre/valor. En varios idiomas, esto se realiza como un objeto, registro, estructura, diccionario, tabla hash, lista con clave o matriz asociativa.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se realiza como una matriz, un vector, una lista o una secuencia.



Para ayudarnos a crear y validar nuestros objetos JSON, podemos usar varias herramientas, incluidas las siguientes:

https://jsoncompare.com

https://jsonformatter.org

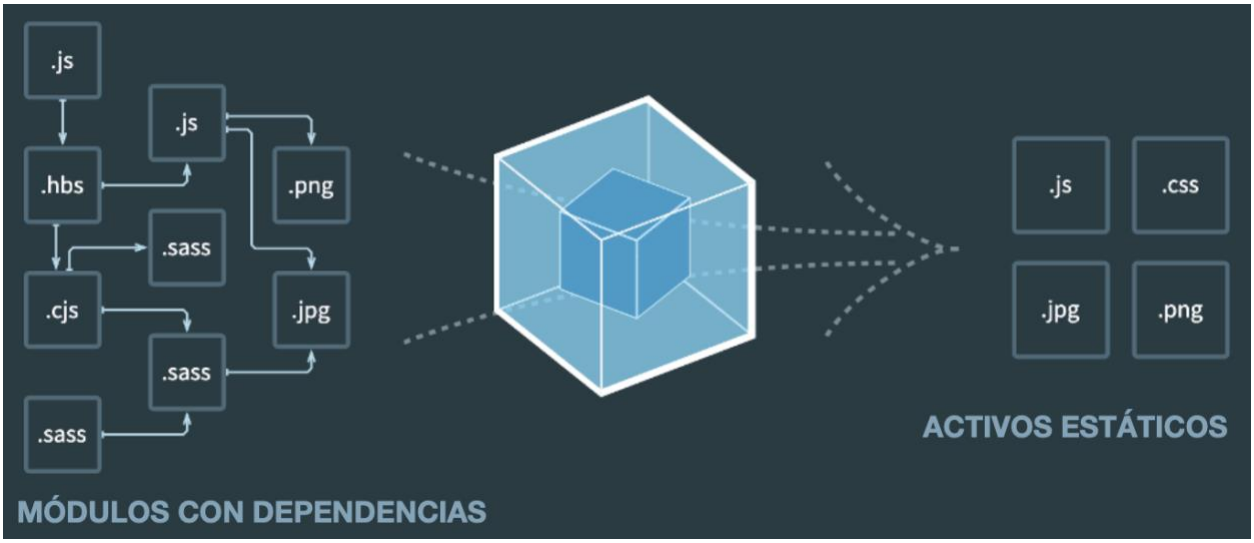
https://onlinejsontools.com

Aunque éstas son diferentes en su diseño y disposición, todas realizan las mismas operaciones, creando, validando y formateando objetos JSON.

WEBPACK

Cuando se trabaja con módulos, un proyecto puede crecer desproporcionadamente, especialmente si todo un equipo está trabajando en el mismo. Para ayudar a organizar todos los distintos documentos y módulos, se puede usar un empaquetador de módulos como Webpack.

En esencia, este es un empaquetador de módulos estáticos para aplicaciones JavaScript modernas. Cuando procesa una aplicación, crea internamente un gráfico de dependencia a partir de uno o más puntos de entrada, y luego combina todos los módulos que el proyecto necesita en uno o más paquetes, que son activos estáticos desde los que sirve su contenido.

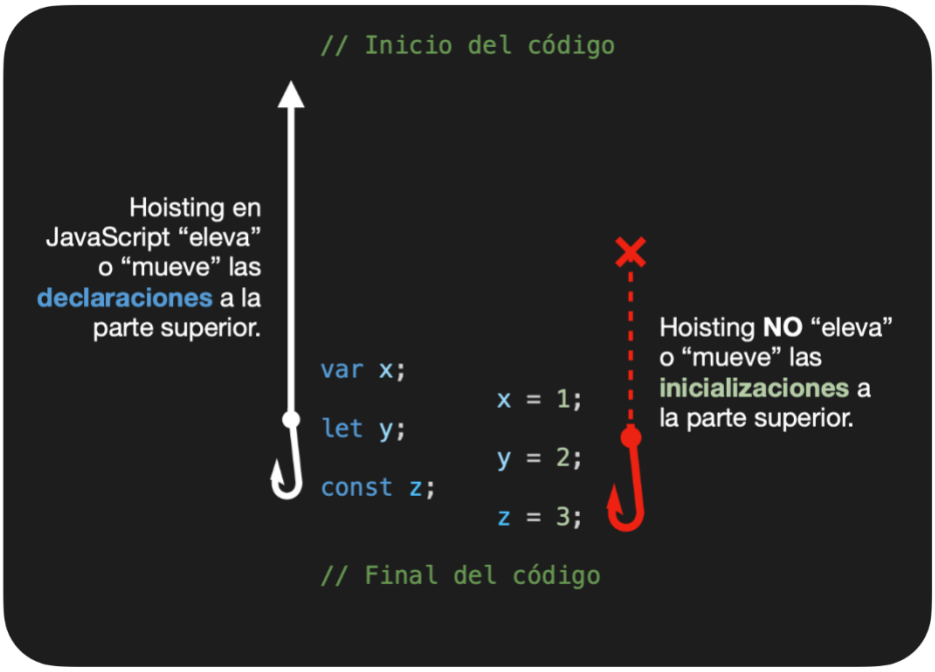


Si se desea obtener más información sobre el paquete web, y leer su documentación, puede hacerlo en su página web oficial: https://webpack.js.org/

HOISTING

En ES6 el comportamiento de las variables cambia, pero también es importante destacar que en JavaScript sucede lo mismo. A este cambio de comportamiento se le llama ***Hoisting,*** que en castellano alude a la idea de "engancharse a algo y elevarlo".

Se sabe que, en JavaScript, una variable se puede utilizar antes de que se declare. En base a este contexto, `hoisting` "mueve" o "eleva" las declaraciones a la parte superior o inicial del código. Esto significa que las declaraciones de variables `var`, `const` y `let` se interpretan como si estuvieran en la parte superior de su bloque. Se detalla más en el siguiente diagrama:



Independiente de la posición de las declaraciones, éstas son elevadas a la parte inicial del código. Hoisting no es así para las ***inicializaciones,*** como `x = 1`, `y = 2` o `x = 3`, por lo que no son elevadas en la parte inicial del código.

En el caso de las variables definidas con `let` y `const`, se elevan a la parte *superior del bloque*, no de todo el código. Entonces, el bloque de código es consciente de la variable, pero no se puede utilizar hasta que se haya declarado.

Revisemos el siguiente ejemplo:

```
var a = 'hey';

function miFuncion() {
        console.log(a); //undefined
        var a = 'Hola';
        console.log(a); //Hola
}
miFuncion();
```

Con `hoisting`, lo que podemos esperar que suceda es que la variable `a` se elevará a la parte superior de su alcance local, con un valor inicial de indefinido (`undefined`). Lo siguiente es cómo JavaScript interpretará el ejemplo anterior después de la elevación:

```
var a = 'hey';
function miFuncion(){
    var a // La "a" en el ámbito local se eleva aquí y es 'indefinida'
    console.log(a); //undefined
    a = 'Hola';
    console.log(a); //Hola
}
miFuncion();
```

Como podemos ver, `hoisting` permite que se produzcan declaraciones de `var` después de su llamada, pues JavaScript la interpreta e inicializa como una variable *indefinida* en la parte superior de su alcance.

Aunque JavaScript elevará las declaraciones de las variables `var`, `const` y `let` hasta la parte superior de su alcance, `const` y `let` no se comportarán igual que `var`, pues éste levantará una declaración ***e inicializará la*** `variable` como indefinida si no se proporciona la definición. Sin embargo, `const` y `let` elevarán la declaración de la variable, y ***no la inicializarán*** si no se proporciona la definición.

Esto se puede apreciar en el siguiente ejemplo:

```
function miFuncion() {
        a;
        console.log(a);
        let a = 'Hola';
}
miFuncion();
```

La variable `a` se elevará a la parte superior de su alcance. Sin embargo, debido a que `a` se declaró usando `let`, ésta permanecerá sin inicializar. Por lo tanto, cuando intente ejecutar `a`, JavaScript arrojará un Error de referencia que indica que `a` no está inicializado.

Para muchos desarrolladores, `hoisting` es un comportamiento desconocido o pasado por alto de JavaScript. Si nosotros, como desarrolladores, no entendemos cómo funciona el concepto, nuestros desarrollos pueden contener errores. **Para evitarlos, siempre debemos declarar todas las variables al comienzo de cada ámbito**.

CONOCER TECNOLOGIAS PARA FACILITAR LA INTEGRACIÓN CON NAVEGADORES

Uno de los problemas más comunes a la hora de desarrollar, es la interpretación del DOM que tiene cada navegador. Como respuesta a esto, un enfoque común para atacar la compatibilidad entre navegadores es el uso de bibliotecas de JavaScript, que resumen las funciones del DOM y funcionan de igual manera en cualquier navegador.

Algunos framework más utilizados son: JQuery, Prototype y YUI.