Clase 1: Material Complementario

Sitio:	Centro de E-Learning - UTN.BA	Imprimido por:	Nelson Brian Avila Solano
Curso:	Curso de Backend Developer - Turno Noche	Día:	Tuesday, 4 de November de 2025, 20:00
Libro:	Clase 1: Material Complementario		

Tabla de contenidos

1. JAVASCRIPT

Temario:

- ¿Qué es?
- Características básicas.
- Variables.
- Estructuras de control.
- Bucles.

1.1. JAVASCRIPT

¿Qué es?

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas. Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario. Técnicamente,

JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

Cómo incluir JavaScript en documentos HTML

La integración de JavaScript y HTML es muy flexible, ya que existen al menos tres formas para incluir código JavaScript en las páginas web.

Incluir JavaScript en el mismo documento HTML

El código JavaScript se encierra entre etiquetas <script> y se incluye en cualquier parte del documento. Aunque es correcto incluir cualquier bloque de código en cualquier zona de la página, se recomienda definir el código JavaScript dentro de la cabecera del documento (la etiqueta <head>):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script type="text/javascript">
        alert("Un mensaje de prueba");
    </script>
</head>
<body>

</body>
</html>
```

Para que la página HTML resultante sea válida, es necesario añadir el atributo type a la etiqueta <script>. Los valores que se incluyen en el atributo type están estandarizados y para el caso de JavaScript, el valor correcto es text/javascript.

Este método se emplea cuando se define un bloque pequeño de código o cuando se quieren incluir instrucciones específicas en un determinado documento HTML que completen las instrucciones y funciones que se incluyen por defecto en todos los documentos del sitio web.

El principal inconveniente es que si se quiere hacer una modificación en el bloque de código, es necesario modificar todas las páginas que incluyen ese mismo bloque de código JavaScript.

Definir JavaScript en un archivo externo

Las instrucciones JavaScript se pueden incluir en un archivo externo de tipo JavaScript que los documentos HTML enlazan mediante la etiqueta <script>. Se pueden crear todos los archivos JavaScript que sean necesarios y cada documento HTML puede enlazar tantos archivos JavaScript como necesite.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script type="text/javascript" src="js/codigo.js">

    </script>
</head>
<body>

</body>
</html>
```

Además del atributo type, este método requiere definir el atributo src, que es el que indica la URL correspondiente al archivo JavaScript que se quiere enlazar. Cada etiqueta <script> solamente puede enlazar un único archivo, pero en una misma página se pueden incluir tantas etiquetas <script> como sean necesarias.

Los archivos de tipo JavaScript son documentos normales de texto con la extensión .js, que se pueden crear con cualquier editor de texto como Notepad, Wordpad, EmEditor, UltraEdit, Vi, etc.

La principal ventaja de enlazar un archivo JavaScript externo es que se simplifica el código HTML de la página, que se puede reutilizar el mismo código JavaScript en todas las páginas del sitio web y que cualquier modificación realizada en el archivo JavaScript se ve reflejada inmediatamente en todas las páginas HTML que lo enlazan.

Características básicas

La sintaxis de un lenguaje de programación se define como el conjunto de reglas que deben seguirse al escribir el código fuente de los programas para considerarse como correctos para ese lenguaje de programación. La sintaxis de JavaScript es muy similar a la de otros lenguajes de programación como Java y C.

Las normas básicas que definen la sintaxis de JavaScript son las siguientes:

● No se tienen en cuenta los espacios en blanco y las nuevas líneas: como sucede con HTML, el intérprete de JavaScript ignora cualquier espacio en blanco sobrante, por lo que el código se puede ordenar de forma adecuada para entenderlo mejor (tabulando las líneas, añadiendo espacios, creando nuevas líneas, etc.)

● Se distinguen las mayúsculas y minúsculas : al igual que sucede con la sintaxis de las etiquetas y elementos HTML. Sin embargo, si en una página HTML se utilizan indistintamente mayúsculas y minúsculas, la página se visualiza correctamente, siendo el único problema la no validación de la página. En cambio, si en JavaScript se intercambian mayúsculas y minúsculas el script no funciona.

● No se define el tipo de las variables: al crear una variable, no es necesario indicar el tipo de dato que almacenará. De esta forma, una misma variable puede almacenar diferentes tipos de datos durante la ejecución del script.

● No es necesario terminar cada sentencia con el carácter de punto y coma (;) : en la mayoría de lenguajes de programación, es obligatorio terminar cada sentencia con el carácter ;. Aunque JavaScript no obliga a hacerlo, es conveniente seguir la tradición de terminar cada sentencia con el carácter del punto y coma (;).

● Se pueden incluir comentarios : los comentarios se utilizan para añadir información en el código fuente del programa. Aunque el contenido de los comentarios no se visualiza por pantalla, si que se envía al navegador del usuario junto con el resto del script, por lo que es necesario extremar las precauciones sobre la información incluida en los comentarios.

Variables

Las variables en los lenguajes de programación siguen una lógica similar a las variables utilizadas en otros ámbitos como las matemáticas. Una variable es un elemento que se emplea para almacenar y hacer referencia a otro valor. Gracias a las variables es posible crear "programas genéricos", es decir, programas que funcionan siempre igual independientemente de los valores concretos utilizados.

De la misma forma que si en matemáticas no existieran las variables no se podrían definir las ecuaciones y fórmulas, en programación no se podrían hacer programas realmente útiles sin las variables.

Si no existieran variables, un programa que suma dos números podría escribirse como:

```
resultado = 3 + 1
```

El programa anterior es tan poco útil que sólo sirve para el caso en el que el primer número de la suma sea el 3 y el segundo número sea el 1. En cualquier otro caso, el programa obtiene un resultado incorrecto.

Sin embargo, el programa se puede rehacer de la siguiente manera utilizando variables para almacenar y referirse a cada número:

```
<script type="text/javascript">
        numero_1 = 3;
        numero_2 = 1;
        resultado = numero_1 + numero_2;
</script>
```

Los elementos numero1 y numero2 son variables que almacenan los valores que utiliza el programa. El resultado se calcula siempre en función del valor almacenado por las variables, por lo que este programa funciona correctamente para cualquier par de números indicados. Si se modifica el valor de las variables numero1 y numero2, el programa sigue funcionando correctamente.

Tipos de variables

Aunque todas las variables de JavaScript se crean de la misma forma (mediante la palabra reservada var), la forma en la que se les asigna un valor depende del tipo de valor que se quiere almacenar (números, textos, etc.)

Numéricas

Se utilizan para almacenar valores numéricos enteros (llamados integer en inglés) o decimales (llamados float en inglés). En este caso, el valor se asigna indicando directamente el número entero o decimal. Los números decimales utilizan el carácter . (punto) en vez de , (coma) para separar la parte entera y la parte decimal:

```
var iva = 16; // variable tipo entero
var total = 234.65; // variable tipo decimal
```

Cadenas de texto

Se utilizan para almacenar caracteres, palabras y/o frases de texto . Para asignar el valor a la variable, se encierra el valor entre comillas dobles o simples, para delimitar su comienzo y su final:

```
var mensaje = "Bienvenido al curso";
var nombreProducto = 'Compu X';
var letraSeleccionada = 'F';
```

Arrays

En ocasiones, a los arrays se les llama vectores, matrices e incluso arreglos. No obstante, el término array es el más utilizado y es una palabra comúnmente aceptada en el entorno de la programación.

Un array es una colección de variables, que pueden ser todas del mismo tipo o cada una de un tipo diferente. Su utilidad se comprende mejor con un ejemplo sencillo: si una aplicación necesita manejar los días de la semana, se podrían crear siete variables de tipo texto:

```
var dia1 = "Lunes";
var dia2 = "Martes";
var dia3 = "Miércoles";
var dia4 = "Juevess";
var dia5 = "Viernes";
var dia6 = "Sábado";
var dia7 = "Domingo";
```

Aunque el código anterior no es incorrecto, sí que es poco eficiente y complica en exceso la programación. Si en vez de los días de la semana se tuviera que guardar el nombre de los meses del año, el nombre de todos los países del mundo o las mediciones diarias de temperatura de los últimos 100 años, se tendrían que crear decenas o cientos de variables.

En este tipo de casos, se pueden agrupar todas las variables relacionadas en una colección de variables o array. El ejemplo anterior se puede rehacer de la siguiente forma:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado",
"Domingo"];
```

Ahora, una única variable llamada días almacena todos los valores relacionados entre sí, en este caso los días de la semana. Para definir un array, se utilizan los caracteres [y] para delimitar su comienzo y su final y se utiliza el carácter , (coma) para separar sus elementos:

```
var nombre_array = [valor1, valor2, ..., valorN];
```

Una vez definido un array, es muy sencillo acceder a cada uno de sus elementos. Cada elemento se accede indicando su posición dentro del array. La única complicación, que es responsable de muchos errores cuando se empieza a programar, es que las posiciones de los elementos empiezan a contarse en el 0 y no en el 1:

```
var diaSeleccionado = dias[0]; // diaSeleccionado = "Lunes"
var otroDia = dias[5]; // otroDia = "Sábado"
```

En el ejemplo anterior, la primera instrucción quiere obtener el primer elemento del array. Para ello, se indica el nombre del array y entre corchetes la posición del elemento dentro del array. Como se ha comentado, las posiciones se empiezan a contar en el 0, por lo que el primer elemento ocupa la posición 0 y se accede a él mediante dias[0].

El valor días[5] hace referencia al elemento que ocupa la sexta posición dentro del array días. Como las posiciones empiezan a contarse en 0, la posición 5 hace referencia al sexto elemento, en este caso, el valor Sábado.

Estructuras de control

Los programas que se pueden realizar utilizando solamente variables y operadores son una simple sucesión lineal de instrucciones básicas.

Sin embargo, no se pueden realizar programas que muestren un mensaje si el valor de una variable es igual a un valor determinado y no muestren el mensaje en el resto de casos. Tampoco se puede repetir de forma eficiente una misma instrucción, como por ejemplo sumar un determinado valor a todos los elementos de un array.

Para realizar este tipo de programas son necesarias las estructuras de control de flujo, que son instrucciones del tipo "si se cumple esta condición, hazlo; si no se cumple, haz esto otro". También existen instrucciones del tipo "repite esto mientras se cumpla esta condición".

Si se utilizan estructuras de control de flujo, los programas dejan de ser una sucesión lineal de instrucciones para convertirse en programas inteligentes que pueden tomar decisiones en función del valor de las variables.

Estructura if

La estructura más utilizada en JavaScript y en la mayoría de lenguajes de programación es la estructura if. Se emplea para tomar decisiones en función de una condición. Su definición formal es:

```
if(condición){
//bloque a ejecutar ...
}
```

Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro de {...}. Si la condición no se cumple (es decir, si su valor es false) no se ejecuta ninguna instrucción contenida en {...} y el programa continúa ejecutando el resto de instrucciones del script. Ejemplo:

```
var mostrarMensaje = true;

 if(mostrarMensaje) {
         alert("Hola Curso!");
  }
```

Estructura if...else

En ocasiones, las decisiones que se deben realizar no son del tipo "si se cumple la condición, hazlo; si no se cumple, no hagas nada". Normalmente las condiciones suelen ser del tipo "si se cumple esta condición, hazlo; si no se cumple, haz esto otro".

Para este segundo tipo de decisiones, existe una variante de la estructura if llamada if...else. Su definición formal es la siguiente:

```
if(condición) {

... }

else {

... }
```

Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro del if(). Si la condición no se cumple (es decir, si su valor es false) se ejecutan todas las instrucciones contenidas en else { }. Ejemplo:

```
var edad = 18;
if(edad >= 18) {
    alert("Eres mayor de edad");
 } else {
   alert("Todavía eres menor de edad");
 }
```

Si el valor de la variable edad es mayor o igual que el valor numérico 18, la condición del if() se cumple y por tanto, se ejecutan sus instrucciones y se muestra el mensaje "Eres mayor de edad". Sin embargo, cuando el valor de la variable edad no es igual o mayor que 18, la condición del if() no se cumple, por lo

que automáticamente se ejecutan todas las instrucciones del bloque else { } . En este caso, se mostraría el mensaje "Todavía eres menor de edad".

Estructura switch

La estructura if...else se puede utilizar para realizar comprobaciones múltiples y tomar decisiones complejas. Sin embargo, si todas las condiciones dependen siempre de la misma variable , el código JavaScript resultante es demasiado redundante:

if(numero == 5) {

... }

else if(numero == 8) {

... }

else if(numero == 20) {

... }

else { ...

}

En estos casos, la estructura switch es la más eficiente, ya que está especialmente diseñada para manejar de forma sencilla múltiples condiciones sobre la misma variable. Su definición formal puede parecer compleja, aunque su uso es muy sencillo:

```
switch (valor) {
    case 1: document.write('uno');
      break;
    case 2: document.write('dos');
      break;
    case 3: document.write('tres');
      break;
    case 4: document.write('cuatro');
      break;
    case 5: document.write('cinco');
      break;
    default:document.write('debe ingresar un valor comprendido entre
            1 y 5.');
  }
```

La estructura switch se define mediante la palabra reservada switch seguida, entre paréntesis, del nombre de la variable que se va a utilizar en las comparaciones. Como es habitual, las instrucciones que forman parte del switch se encierran entre comillas { y }

Dentro del switch se definen todas las comparaciones que se quieren realizar sobre el valor de la variable. Cada comparación se indica mediante la palabra reservada case seguida del valor con el que se realiza la comparación. Si el valor de la variable utilizada por switch coincide con el valor indicado por case, se ejecutan las instrucciones definidas dentro de ese case.

Normalmente, después de las instrucciones de cada case se incluye la sentencia break para terminar la ejecución del switch, aunque no es obligatorio. Las comparaciones se realizan por orden, desde el primer case hasta el último, por lo que es muy importante el orden en el que se definen los case.

¿Qué sucede si ningún valor de la variable del switch coincide con los valores definidos en los case? En este caso, se utiliza el valor default para indicar las instrucciones que se ejecutan en el caso en el que ningún case se cumpla para la variable indicada.

Aunque default es opcional, las estructuras switch suelen incluirlo para definir al menos un valor por defecto para alguna variable o para mostrar algún mensaje por pantalla.

Bucles

Estructura for

Las estructuras if y if...else no son muy eficientes cuando se desea ejecutar de forma repetitiva una instrucción . Por ejemplo, si se quiere mostrar un mensaje cinco veces, se podría pensar en utilizar el siguiente if:

```
var veces = 0;
if(veces < 4) {
    alert("Mensaje");
    veces++;
}
```

Se comprueba si la variable veces es menor que 4. Si se cumple, se entra dentro del if(), se muestra el mensaje y se incrementa el valor de la variable veces. Así se debería seguir ejecutando hasta mostrar el mensaje las cinco veces deseadas.

Sin embargo, el funcionamiento real del script anterior es muy diferente al deseado, ya que solamente se muestra una vez el mensaje por pantalla. La razón es que la ejecución de la estructura if() no se repite y la comprobación de la condición sólo se realiza una vez, independientemente de que dentro del if() se modifique el valor de la variable utilizada en la condición. La estructura for permite realizar este tipo de repeticiones (también llamadas bucles) de una forma muy sencilla. No obstante, su definición formal no es tan sencilla como la de if():

for(inicialización; condición; actualización) {

... }

La idea del funcionamiento de un bucle for es la siguiente: "mientras la condición indicada se siga cumpliendo, repite la ejecución de las instrucciones definidas dentro del for. Además, después de cada repetición, actualiza el valor de las variables que se utilizan en la condición".

● La " inicialización " es la zona en la que se establecen los valores iniciales de las variables que controlan la repetición.

● La " condición " es el único elemento que decide si continúa o se detiene la repetición.

● La " actualización " es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

```
var mensaje = "Hola, estoy dentro de un bucle";

for(var i = 0; i < 5; i++) {
alert(mensaje);
 }
```

Estructura while

La estructura while permite crear bucles que se ejecutan una o más veces, dependiendo de la condición indicada . Su definición formal es:

while(condición) {

...

}

El funcionamiento del bucle while se resume en: " mientras se cumpla la condición indicada, repite indefinidamente las instrucciones incluidas dentro del bucle".

Si la condición no se cumple ni siquiera la primera vez, el bucle no se ejecuta. Si la condición se cumple, se ejecutan las instrucciones una vez y se vuelve a comprobar la condición. Si se sigue cumpliendo la condición, se vuelve a ejecutar el bucle y así se continúa hasta que la condición no se cumpla.

Evidentemente, las variables que controlan la condición deben modificarse dentro del propio bucle, ya que de otra forma, la condición se cumpliría siempre y el bucle while se repetirá indefinidamente. El siguiente ejemplo utiliza el bucle while para sumar todos los números menores o iguales que otro número:

```
var x;
x=1;
while(x<=100){
    document.write(x);
    document.write('<br>');
    x=x+1;
}
```

Bibliografía:

- Eguíluz Pérez, Javier. Introducción a JavaScript.España: www.librosweb.es 2008.

2. JavaScript ES 6

Bloques temáticos:

- ES6 – ECMAScript 6
- Javascript – Template Strings
- Javascript – Let && Const
- Javascript – Función Arrow
- Javascript – Deconstructor
- Javascript – Valores por defecto
- Javascript – Import && Export
- Javascript – Rest y Spread
- Javascript – Promises
- Javascript – Clases
- Instalar node js

2.1. JavaScript ES 6

ES6 – ECMAScript 6

ECMAScript es el estándar que define cómo debe de ser el lenguaje Javascript.

Javascript es interpretado y procesado por multitud de plataformas, entre las que se encuentran los navegadores web, así como NodeJS y otros ámbitos más específicos como el desarrollo de aplicaciones para Windows y otros sistemas operativos. Todos los responsables de cada una de esas tecnologías se encargan de interpretar el lenguaje tal como dice el estándar ECMAScript.

ES5 estuvo con nosotros durante muchos años y a día de hoy es la versión de Javascript más extendida en todo tipo de plataformas. Cuando alguien dice que conoce o usa Javascript es común entender que lo que usa es ES5, el Javascript con mayor índice de compatibilidad. De esto también se entiende que, cuando queremos escribir un código compatible con todos los navegadores o sistemas, lo normal es que ese código sea ES5.

Transpiladores

Conscientes de los problemas de compatibilidad o soporte a ES6 en las distintas plataformas, un desarrollador poco informado podría pensar que:

- Es poco aconsejable usar hoy ES6, debido a la falta de compatibilidad.
- Lo correcto sería esperar a que todos los navegadores se pongan al día para empezar a usar ES6 con todas las garantías.

Afortunadamente, ninguna de esas suposiciones se ajusta a la realidad. Primero porque si ES6 nos aporta diversas ventajas, lo aconsejable es usarlo ya. Luego porque es absurdo quedarse esperando a que todos los navegadores soporten Javascript en la versión ES6. Quizás nunca llegue ese momento de total compatibilidad o posiblemente para entonces hayan sacado nuevas versiones del lenguaje que también deberías usar.

Así que, para facilitar nuestra vida y poder comenzar a usar ES6 en cualquier proyecto, han surgido los transpiladores, una herramienta que ha venido a nuestro kit de desarrollo para quedarse.

Los transpiladores son programas capaces de traducir el código de un lenguaje para otro, o de una versión para otra. Por ejemplo, el código escrito en ES6, traducirlo a ES5. Dicho de otra manera, el código con posibles problemas de compatibilidad, hacerlo compatible con cualquier plataforma.

El transpilador es una herramienta que se usa durante la fase de desarrollo. En esa fase el programador escribe el código y el transpilador lo convierte en un proceso de "traducción/compilación = transpilación". El código transpilado, compatible, es el que realmente se distribuye o se despliega para llevar a producción. Por tanto, todo el trabajo de traducción del código se queda solo en la etapa de desarrollo y no supone una carga mayor para el sistema donde se va a ejecutar de cara al público.

Hoy tenemos transpiladores para traducir ES6 a ES5, pero también los hay para traducir de otros lenguajes a Javascript. Quizás hayas oído hablar de TypeScript, o de CoffeeScript o Flow. Son lenguajes

que una vez transpilados se convierten en Javascript ES5, compatible con cualquier plataforma.

Javascript – Template Strings

Con ES6 podemos interpolar Strings de una forma más sencilla que como estábamos haciendo hasta ahora. Fíjate en este ejemplo:

```
/*IN ESS*/
var userName = 'Hello World';
var message = 'Hey' + userName +',';

/*IN ES6*/
let userName = 'Hello World';
let message = 'Hey ${userName},';
```

En el ejemplo de ES6 utilizamos el **backtick (`)** en lugar de las comillas. Luego colocamos el placeholder ${...} para identificar el contenido a ser interpretado.

De esta forma evitamos tener que concatenar las variables al string definido con comillas.

Let && Const

En Javascript ES 6 podemos definir una constante con la palabra reservada **const**

```
const PI = 3.141593;
alert (PI > 3.0);
```

Las constantes no podrán ser redefinidas luego.

Var - Let

El scope (alcance) de **var** abarca toda la función en la cual está definida

```
function helloworld() {
    for (var x = 0; x < 2; x++) {
        // x should only be scoped to this block because this is where
we have defined x.
    }
    // But it turns out that x is available here as well!
    console.log(x);
    // 2
```

En el ejemplo vemos que “x” (definida en el for) tiene un alcance a la función, por lo cual al realizar un console.log(x) luego del for accedemos al último valor de dicha variable.

Con **Let** el alcance de la variable pasa a ser las llaves en la cual está definida:

```
function helloworld() {
    for (let x = 0; x < 2; x++) {
        // With the "let" keyword, now x is only accessible in this
block.
    }
    // x is out of the scope here
    console.log(x); // x is not defined
}
```

En este ejemplo vemos que al querer realizar un console.log(x) nos arroja “undefined”. Esto se debe a que let quedó declarada dentro de las llaves del for y no para toda la función.

Tanto **let** como **const** no crean una propiedad global si se utilizan en el nivel superior

Función Arrow

La función arrow es una simplificación sintáctica de la función en ES5

```
//ES5
const add = function (num) {
    return num + num;
}
//ES6
const add = (num) => {
    return num + num;
}

//en caso de ser una unica sentencia y sin {} aplica return
const add = (num) => num + num;
//Si recibe un unico parametro no hace falta los ()
const add = num => num + num;
//Si no recibe parametros se debe colocar ()
const add = () => num + num;
```

Vemos en los ejemplos que no debemos colocar la palabra reservada function, quedando su declaración de la siguiente manera:

```
const add = () => {} (el = y > forman la "flecha" o "arrow")
```

En los ejemplos podemos ver que en caso de no colocar {} y solo tener una única sentencia aplica el return de forma implícita

Deconstructor

Es una expresión que permite extraer propiedades de un objeto o ítems de un array:

```
//Objeto
const address = {
    street: 'Pallimon',
    city: 'Kollam',
    state: 'Kerala',
};
//ES5
var street = address.street;
var city = address.city;
var state = address.state;
//ES6
const { street, city, state } = address;
```

En este caso vemos que podemos acceder y crear una constante “Street”, “city”, “state” relacionada con las propiedades que tienen el mismo nombre en el objeto address

En caso de ser un array:

```
//ES5
var values = ['Hello', 'World'];
var first = values[0];
var last = values[1];
//ES6
const values = ['Hello', 'World'];
const [first, last] = values;
```

La constante “first” tendrá asignado el valor del array en el índice 0, mientras que “last” el

valor del array en el índice 1

Valores por defecto

Otra novedad es asignar valores por defecto a las variables que se pasan por parámetros en las funciones. Antes teníamos que comprobar si la variable ya tenía un valor. Ahora con ES6 se la podemos asignar según creemos la función.

```
//ES5
function getUser(name, year) {
    year = (typeof year !== 'undefined') ? year : 2018;
    // reminder of the function.
}
//ES6
function getUser(name, year = 2018) {
    // function body here.
}
```

Como vemos el parámetro “year” cuando no reciba valor tomará por default el valor 2018

Import && Export

Ahora JavaScript se empieza a parecer a lenguajes como Python o Ruby. Llamamos a las funciones desde los propios Scripts, sin tener que importarlos en el HTML, si usamos JavaScript en el navegador.

```
// MyClass.js
class MyClass {
    constructor() { }
}
export default MyClass;
// Main.js
import MyClass from 'MyClass';
```

En el archivo “MyClass.js” tenemos la declaración de la clase y el export de la misma.

En el archivo “Main.js” importamos MyClass (debe tener el export previamente)

En caso de no realizar export **default** (es decir no declarar el artefacto como default) debemos realizar el import con {}:

```
// MyClass.js
class MyClass {
    constructor() { }
}
    export MyClass;
// Main.js
import { MyClass } from 'MyClass';
```

Solo se puede declarar un solo artefacto como **default** por módulo.

Rest y Spread

Spread**:** Propaga los elementos de un array de forma individual

```
function getSum(x, y, z) {
    console.log(x + y + z);
}
let sumArray = [10, 20, 30];
getSum(...sumArray);
```

Podemos utilizarlo para concatenar 2 arrays:

```
var a = [1, 2];
var b = [3, 4];
var c = [...a, ...b]
console.log(c);
```

En este caso “c” tendrá los valores de a y b. Este operador también puede ser utilizados para objetos:

```
let alumno = {
    nombre: "Leandro",
    apellido: "Gil"
}
let cursoAlumno = { ...alumno, { curso: "php"}}
```

Promises

Una Promise (promesa en castellano) es un objeto que representa la terminación o el fracaso eventual de una operación asíncrona. Una promesa puede ser creada usando su constructor. Sin embargo, la mayoría de la gente son consumidores de promesas ya creadas devueltas desde funciones.

Esencialmente, una promesa es un objeto devuelto al cual enganchas las funciones callback, en vez de pasar funciones callback a una función.

Por ejemplo, en vez de una función del viejo estilo que espera dos funciones callback, y llama a una de ellas en caso de terminación o fallo:

```
function exitoCallback(resultado) {
    console.log("Tuvo éxito con " + resultado);
}
function falloCallback(error) {
    console.log("Falló con " + error);
}
hazAlgo(exitoCallback, falloCallback);
```

Las funciones modernas devuelven una promesa a la que puedes enganchar tus funciones de retorno.

```
let p = new Promise(function (resolve, reject) {
    if (/* condition */) {
        resolve(/* value */);
        // fulfilled successfully
    } else {
        reject(/* reason */);
        // error, rejected
    }
});
p.then((val) => console.log("fulfilled:", val)) //10
    .catch((err) => console.log("rejected:", err));
```

La promesa es un objeto en este caso lo asignamos a la variable “p”. Este objeto recibe como parámetro una función de callback la cual recibe el parámetro resolve y reject**.**

En caso de que se resuelva la promesa de forma correcta se retornara resolve(valor_retorno) esto indicará que la ejecución fue correcta y retorna el valor deseado. En caso de que ocurra un error o excepción se retornada reject(valor_error), en este caso se indica un error o excepción y la causa como parámetro.

Luego tratamos la promesa (como consumidores) aplicando el método “then” y “catch” Cuando se resuelva la misma de forma correcta (resolve) ejecuta el then y recibe el parámetro devuelto. En caso de

error o excepción (reject) ejecuta el catch.

Características:

- Las funciones callback nunca serán llamadas antes de la terminación de la ejecución actual del bucle de eventos de JavaScript.
- Las funciones callback añadidas con .then serán llamadas después del éxito o fracaso de la operación asíncrona, como arriba.
- Pueden ser añadidas múltiples funciones callback llamando a .then varias veces, para ser ejecutadas independientemente en el orden de inserción.
- Pero el beneficio más inmediato de las promesas es el encadenamiento.

Async / Await

Las incorporaciones más recientes al lenguaje JavaScript, son las funciones async y la palabra clave await, parte de la edición ECMAScript 2017. Estas características, básicamente, actúan como azúcar sintáctico, haciendo el código asíncrono fácil de escribir y leer más tarde. Hacen que el código asíncrono se parezca más al código síncrono de la vieja escuela, por lo que merece la pena aprenderlo.

Primero tenemos la palabra clave "async", que se coloca delante de la declaración de una función, para convertirla en función "async" (asíncrona). Una función "async", es una función que sabe cómo esperar la posibilidad de que la palabra clave "await" sea utilizada para invocar código asíncrono.

await solo trabaja dentro de las funciones async. Esta puede ser puesta frente a cualquier función async basada en una promesa para pausar tu código en esa línea hasta que se cumpla la promesa, entonces retorna el valor resultante. Mientras tanto, otro código que puede estar esperando una oportunidad para ejecutarse, puede hacerlo.

Ejemplo de promise con async / await:

```
//Promise
fetch('coffee.jpg')
    .then(response => {
        if (!response.ok) {
            throw new Error('HTTP error! status: ${response.status}');
        } else {
            return response.blob();
        }
    }).then(myBlob => {
        let objectURL = URL.createObjectURL(myBlob);
        let image = document.createElement('img');
        image.src = objectURL;
        document.body.appendChild(image);
    })
    .catch(e => {
        console.log('There has been a problem with your fetch operation: ' +
e.message);
    });
```

```
//ASync / await
async function myFetch() {
    try {
        let response = await fetch('coffee.jpg');
        if (!response.ok) {
            throw new Error('HTTP error! status: ${response.status}');
        } else {
            let myBlob = await response.blob();
            let objectURL = URL.createObjectURL(myBlob);
            let image = document.createElement('img');
            image.src = objectURL;
            document.body.appendChild(image);
        }
    } catch (e) {
        console.log('There has been a problem with your fetch operation: ' +
e.message);
    }
}
```

Javascript – Clases

Ahora JavaScript tendrá clases, muy parecidas las funciones constructoras de objetos que realizamos en el estándar anterior, pero ahora bajo el paradigma de clases, con todo lo que eso conlleva, como, por ejemplo, herencia.

```
class LibroTecnico extends Libro {
    constructor(tematica, paginas) {
        super(tematica, paginas);
        this.capitulos = [];
        this.precio = " ";
        // ..
    }
    metodo() {
        //..
    }
}
```

This

La variable this muchas veces se vuelve un dolor de cabeza. antiguamente teníamos que cachearlo en otra variable ya que solo hace referencia al contexto en el que nos encontremos. Por ejemplo, en el siguiente código si no hacemos var that = this dentro de la función document.addEventListener, this haría referencia a la función que pasamos por Callback y no podríamos llamar a foo()

```
//ES3
var obj = {
    foo: function () {...},
    bar: function () {
        var that = this;
        document.addEventListener("click", function (e) {
            that.foo();
        });
    }
}
```

Con ECMAScript5 la cosa cambió un poco, y gracias al método bind podíamos indicarle que this hace referencia a un contexto y no a otro.

```
//ES5
var obj = {
    foo: function () {...},
    bar: function () {
        document.addEventListener("click", function (e) {
            this.foo();
        }.bind(this));
    }
}
```

Ahora con ES6 y la función Arrow => la cosa es todavía más visual y sencilla.

```
//ES6
var obj = {
    foo: function () {...},
    bar: function () {
        document.addEventListener("click", (e) => this.foo());
    }
}
```

2.2. Instalar node js

Instalar node js

Node.js es un entorno de ejecución JavaScript multiplataforma de código abierto. Ejecuta código JavaScript fuera de un navegador.

Cada navegador tiene un motor JavaScript incorporado para procesar archivos JavaScript contenidos en sitios web. Google Chrome usa el motor V8, que está construido usando C + +. Node.js también utiliza este motor ultrarrápido para interpretar archivos JavaScript.

Node.js utiliza un modelo basado en eventos. Esto significa que Node.js espera a que se lleven a cabo ciertos eventos. Luego actúa sobre esos eventos. Los eventos pueden ser cualquier cosa, desde un clic hasta una solicitud HTTP. También podemos declarar nuestros propios eventos personalizados y hacer que Node.js escuche esos eventos.

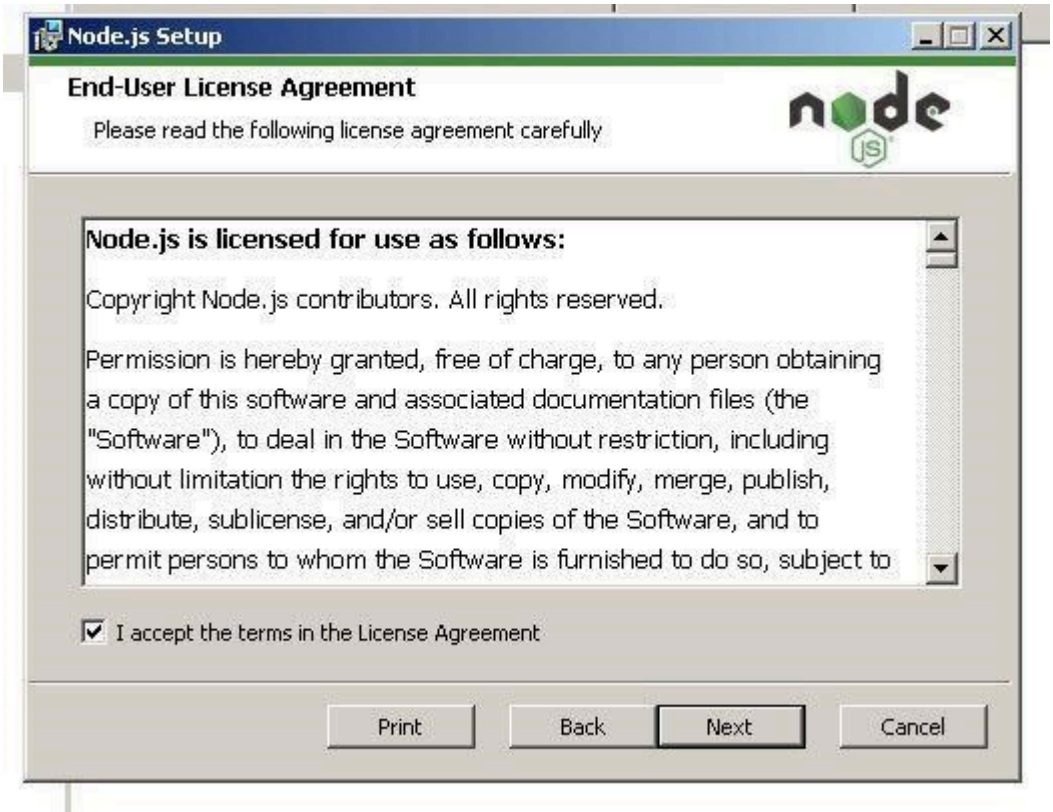
Node.js utiliza un modelo de E/S sin bloqueo. Sabemos que las tareas de E/S toman mucho más tiempo que las tareas de procesamiento. Node.js usa funciones de devolución de llamada para manejar tales solicitudes.

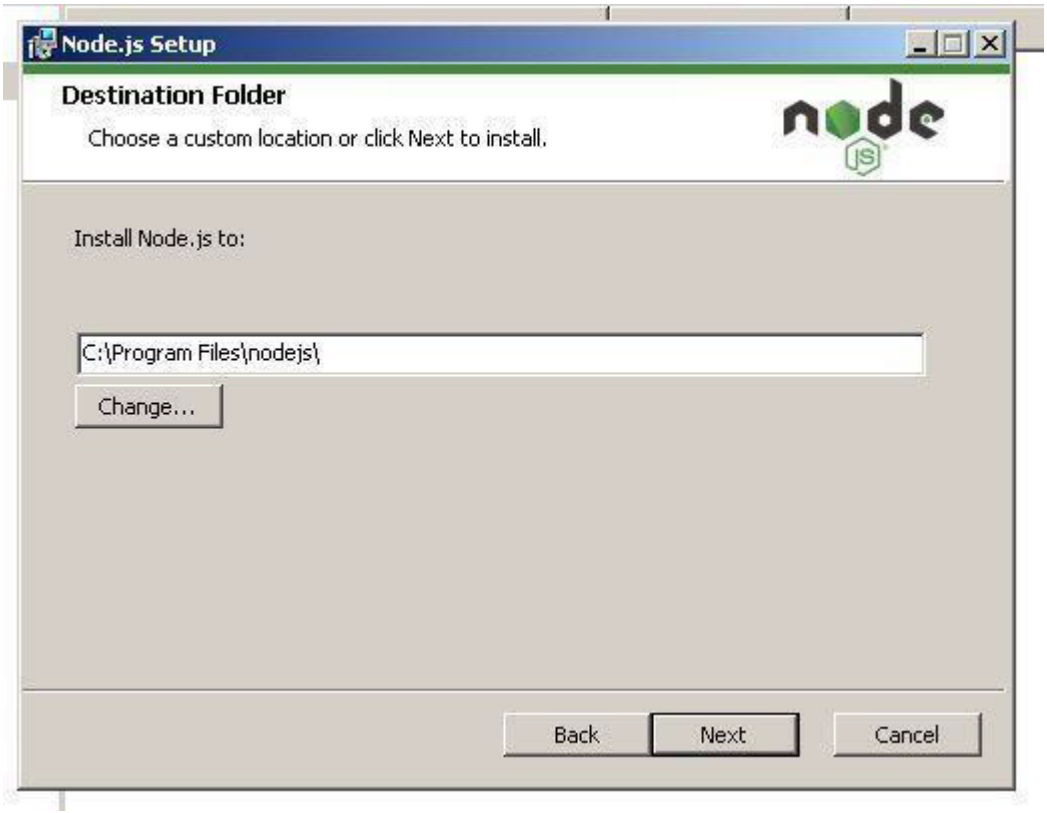
Instalación

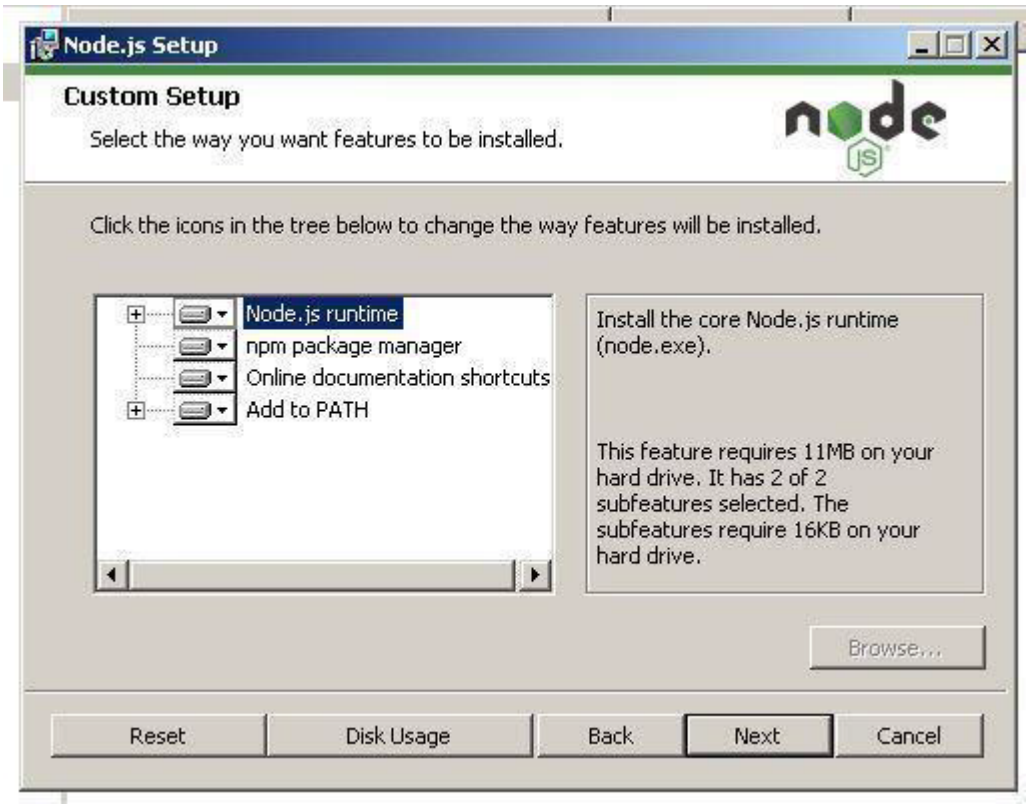
Para poder instalar una aplicación de react js desde el CLI, debemos previamente instalar node js.

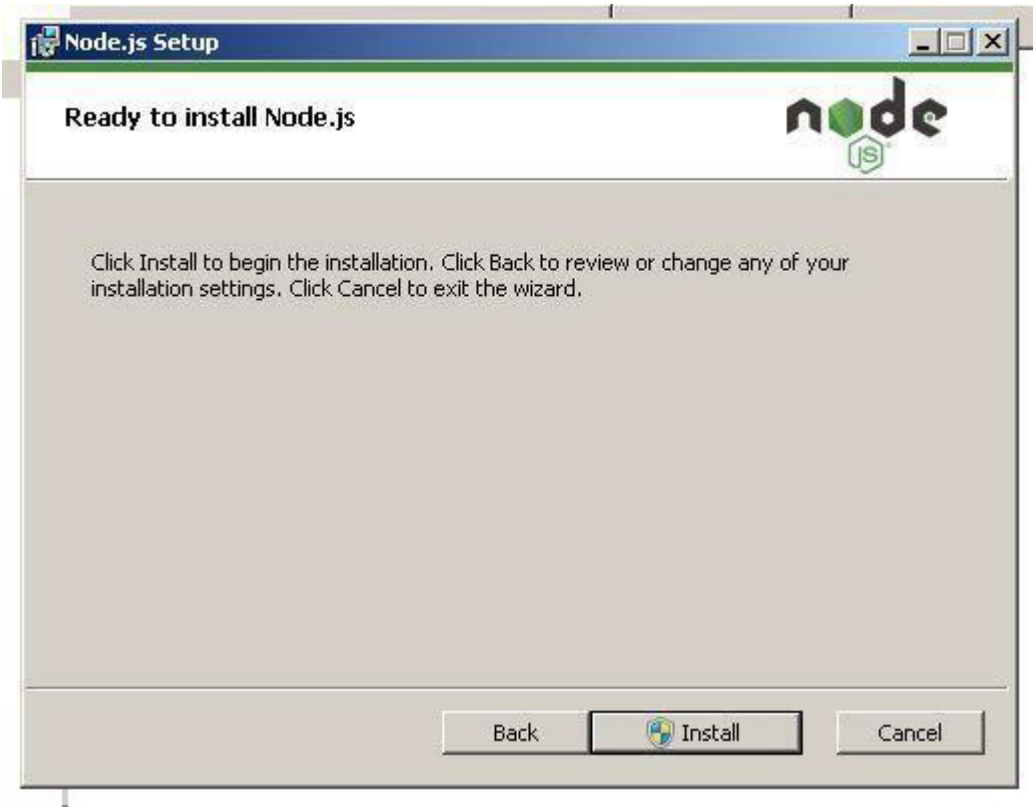
1. Ingresar a: ***https://nodejs.org/en/***
2. Descargar la última versión LTS de node Js
3. Ejecutar el archivo descargado y seguir los siguientes pasos:


Node.js Setup
Welcome to the Node.js Setup Wizard
The Setup Wizard will install Node.js on your computer.
Back
Next
Cancel


Node.js Setup
End-User License Agreement
Please read the following license agreement carefully
Node.js is licensed for use as follows:
Copyright Node.js contributors. All rights reserved.
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to
I accept the terms in the License Agreement
Print
Back
Next
Cancel

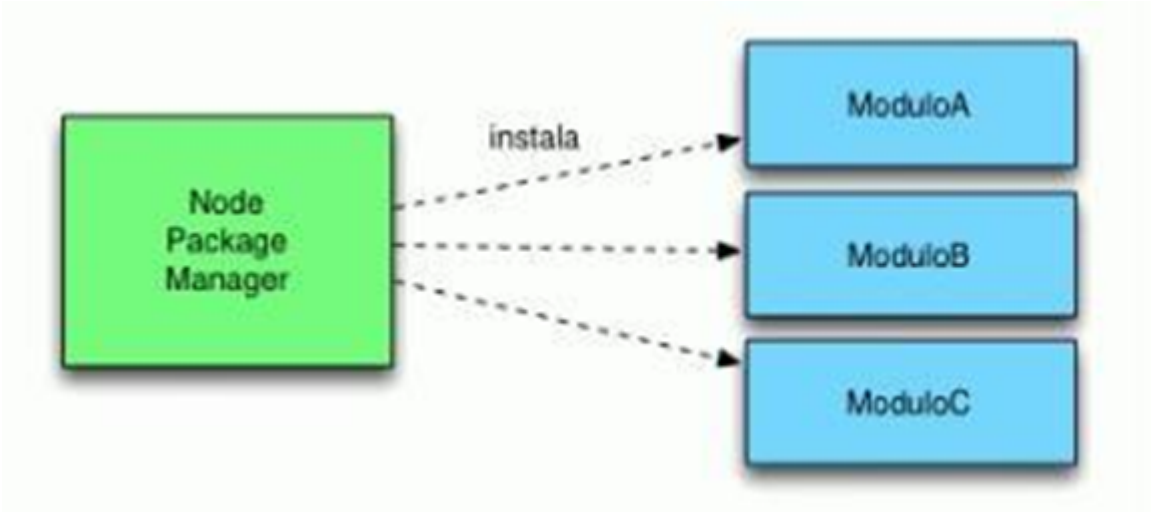

Node.js Setup
Destination Folder
Choose a custom location or click Next to install.
Install Node.js to:
C:\Program Files\nodejs\
Change...
Back
Next
Cancel


Node.js Setup
Custom Setup
Select the way you want features to be installed.
Click the icons in the tree below to change the way features will be installed.
Node.js runtime
npm package manager
Online documentation shortcuts
Add to PATH
Install the core Node.js runtime (node.exe).
This feature requires 11MB on your hard drive. It has 2 of 2 subfeatures selected. The subfeatures require 16KB on your hard drive.
Browse...
Reset
Disk Usage
Back
Next
Cancel



¿Qué es NPM?

Cuando usamos Node.js rápidamente tenemos que instalar módulos nuevos (librerías) ya que Node al ser un sistema fuertemente modular viene prácticamente vacío. Así que para la mayoría de las operaciones deberemos instalar módulos adicionales. Esta operación se realiza de forma muy sencilla con la herramienta npm (Node Package Manager).



¿Por qué necesita un Administrador de Paquetes?

Supongamos que no hubiera administradores de paquetes. En ese caso, tendría que hacer lo siguiente manualmente:

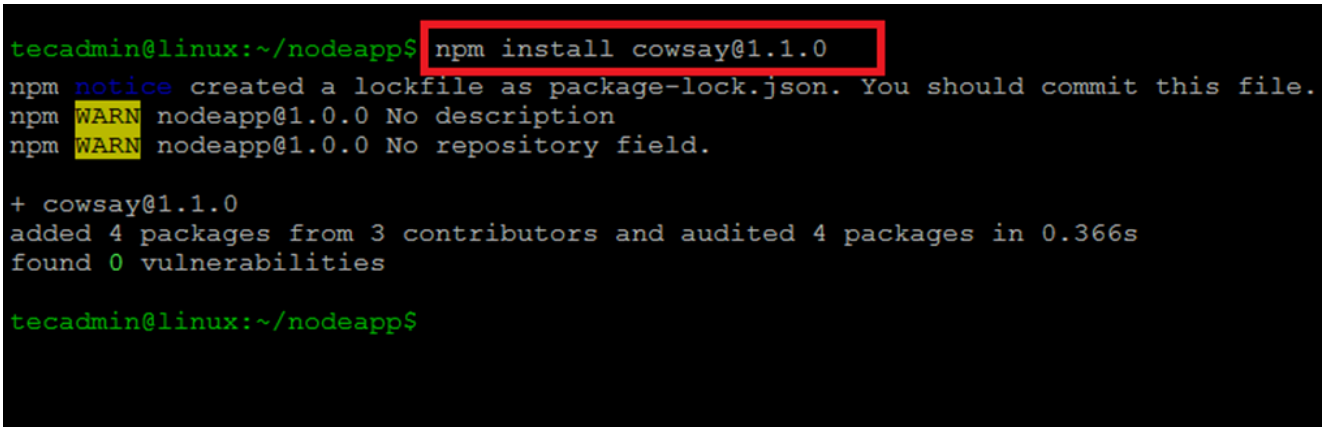
- Encuentre todos los paquetes correctos para su proyecto
- Verifique que los paquetes no tengan vulnerabilidades conocidas
- Descarga los paquetes
- Instálelos en el lugar apropiado
- Mantenga un registro de las nuevas actualizaciones para todos sus paquetes
- Actualice cada paquete cada vez que haya una nueva versión
- Elimina los paquetes que ya no necesites

Un administrador de paquetes es una herramienta que usan los desarrolladores para buscar, descargar, instalar, configurar, actualizar y desinstalar automáticamente los paquetes de una computadora.

NPM (Node Package Manager) e Yarn (Yet Another Resource Negotiator) son dos administradores de paquetes de uso popular.

Un registro de paquetes es una base de datos (almacenamiento) para miles de paquetes (bibliotecas, complementos, marcos o herramientas).

En otras palabras, un registro de paquetes es el lugar desde el que se publican y se instalan los paquetes.



¿Qué es un Módulo en JavaScript?

En términos simples, un módulo es una pieza de código JavaScript reutilizable. Podría ser un .js archivo o un directorio que contenga .js archivos. Puede exportar el contenido de estos archivos y usarlos en otros archivos.

Los módulos ayudan a los desarrolladores a adherirse al principio DRY (Don't Repeat Yourself) en la programación. También ayudan a dividir la lógica compleja en fragmentos pequeños, simples y manejables.

Tipos de módulos de Node

Hay tres tipos principales de módulos de Node con los que trabajará como desarrollador de Node.js. Incluyen lo siguiente.

- Módulos integrados
- Módulos locales
- Módulos de terceros

¿Por qué exportar módulos?

Querrá exportar módulos para poder usarlos en otras partes de su aplicación.

Los módulos pueden servir para diferentes propósitos. Pueden proporcionar utilidades simples para modificar cadenas. Pueden proporcionar métodos para realizar solicitudes de API. O incluso pueden proporcionar constantes y valores primitivos.

Cuando exporta un módulo, puede importarlo a otras partes de sus aplicaciones y consumirlo.

Tipos de exportaciones de archivos en JavaScript

Exportaciones predeterminadas

Aquí se explica cómo realizar una exportación predeterminada en un archivo JavaScript:

```
function getAllUser():
export default getAllUser
```

Tenga en cuenta que solo puede usar una exportación predeterminada en un archivo JavaScript. También puede exportar antes de la declaración, así:

```
export default function getAllUser():
```

Bibliografía y Webgrafía utilizada y sugerida

Fedosejev, A. (2015). React.js Essentials (1 ed.). EEUU, Packt.

Amler, . (2016). ReactJS by Example (1 ed.). EEUU, Packt.

Stein, J. (2016). ReactJS Cookbook (1 ed.). EEUU, Packt.
https://www.tutorialspoint.com/es6/es6_syntax.htm

http://www.desarrolloweb.com/manuales/20/ http://www.desarrolloweb.com/articulos/826.php
http://www.desarrolloweb.com/articulos/827.php http://www.desarrolloweb.com/articulos/846.php

http://www.desarrolloweb.com/articulos/861.php https://carlosazaustre.es/ecmascript-6-el-nuevo-estandar-de-javascript/

http://es6-features.org/

3. Javascript Repaso

-

3.1. JS - Variables

Almacenando la información que necesitas - Variables

¿Qué es una variable?

Una variable es un contenedor para un valor, como un número que podríamos usar en una suma, o una cadena que podríamos usar como parte de una oración. Pero una cosa especial acerca de las variables es que los valores que contienen pueden cambiar. Veamos un sencillo ejemplo:

```
<button>Presióname</button>
const button = document.querySelector('button');

button.onclick = function() {
  let name = prompt('¿Cuál es tu nombre?');
  alert('¡Hola ' + name + ', encantado de verte!');
}
```

En este ejemplo, al presionar el botón se ejecutan un par de líneas de código.

La primera línea muestra un cuadro en la pantalla que pide al lector que ingrese su nombre y luego almacena el valor en una variable.

La segunda línea muestra un mensaje de bienvenida que incluye su nombre, tomado del valor de la variable.

Para entender por qué esto es tan útil, pensemos en cómo escribiríamos este ejemplo sin usar una variable

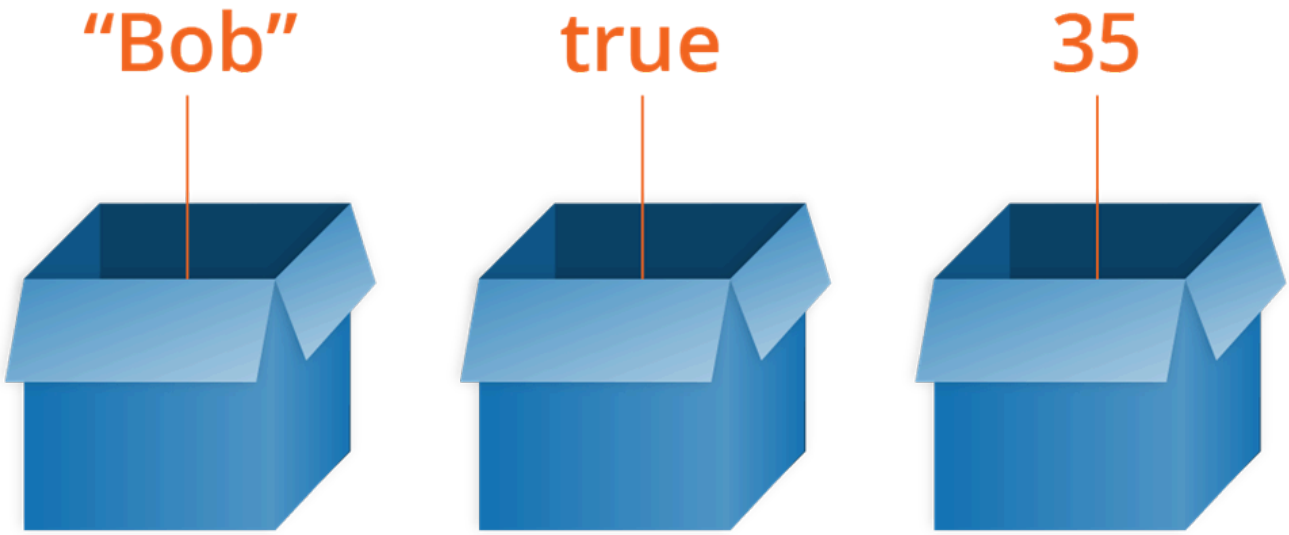
```
let name = prompt('¿Cuál es tu nombre?');

if (name === 'Adam') {
  alert('¡Hola Adam, encantado de verte!');
} else if (name === 'Alan') {
  alert('¡Hola Alan, encantado de verte!');
} else if (name === 'Bella') {
  alert('¡Hola Bella, encantado de verte!');
} else if (name === 'Bianca') {
  alert('¡Hola Bianca, encantado de verte!');
} else if (name === 'Chris') {
  alert('¡Hola Chris, encantado de verte!');
}

// ... y así sucesivamente ...
```

Si no tuviéramos variables disponibles, tendríamos que implementar un bloque de código gigante que verificará cuál era el nombre ingresado, y luego muestra el mensaje apropiado para cualquier nombre.

Las variables simplemente tienen sentido y, a medida que aprendas más sobre JavaScript, comenzarán a convertirse en una segunda naturaleza.

Otra cosa especial acerca de las variables es que pueden contener casi cualquier cosa, no sólo cadenas y números. **Las variables también pueden contener datos complejos e incluso funciones completas para hacer cosas asombrosas.**

Nota: Decimos que las variables contienen valores. Ésta es una importante distinción que debemos reconocer. Las variables no son los valores en sí mismos; son contenedores de valores. Puedes pensar en ellas como pequeñas cajas de cartón en las que puedes guardar cosas.



Declarar una variable

Para usar una variable, primero debes crearla, a esto lo llamamos declarar la variable. Para hacerlo, escribimos la palabra clave var o let seguida del nombre con el que deseas llamar a tu variable:

```
let myName;
let myAge;
```

Aquí estamos creando dos variables llamadas myName y myAge. Intenta escribir estas líneas en la consola de tu navegador web. Después de eso, intenta crear una variable (o dos) eligiendo su nombre.

Nota: En JavaScript, todas las instrucciones en el código deben terminar con un punto y coma (;) — tu código puede funcionar correctamente para líneas individuales, pero probablemente no lo hará cuando estés escribiendo varias líneas de código juntas. Trata de adquirir el hábito de incluirlo.

Puedes probar si estos valores existen ahora en el entorno de ejecución escribiendo solo el nombre de la variable, p. ej.

```
myName;
myAge;
```

Actualmente no tienen ningún valor; son contenedores vacíos. Cuando ingreses los nombres de las variables, deberías obtener devuelto un valor undefined. Si no existen, recibirás un mensaje de error; intenta escribir:

```
scoobyDoo;
```

Nota: No confundas una variable que existe pero no tiene un valor definido, con una variable que no existe en absoluto — son cosas muy diferentes. En la analogía de cajas que viste arriba, no existir significaría que no hay una caja (variable) para guardar un valor. Ningún valor definido significaría que HAY una caja, pero no tiene ningún valor dentro de ella.

Iniciar una variable

Una vez que hayas declarado una variable, la puedes iniciar con un valor. Para ello, escribe el nombre de la variable, seguido de un signo igual (**=**), seguido del valor que deseas darle. Por ejemplo:

```
myName = 'Chris';
myAge = 37;
```

Intenta volver a la consola ahora y escribe estas líneas. Deberías ver el valor que le has asignado a la variable devuelto en la consola para confirmarlo, en cada caso. Nuevamente, puedes devolver los valores de tus variables simplemente escribiendo su nombre en la consola; inténtalo nuevamente:

```
myName;
myAge;
```

Puedes declarar e iniciar una variable al mismo tiempo, así:

```
let myDog = 'Rover';
```

Esto probablemente es lo que harás la mayor parte del tiempo, ya que es más rápido que realizar las dos acciones en dos líneas separadas.

Diferencia entre var y let

Cuando se creó JavaScript por primera vez, solo existía var. Esto básicamente funciona bien en la mayoría de los casos, pero tiene algunos problemas en la forma en que trabaja — su diseño a veces puede ser confuso. Entonces, se creó let en versiones modernas de JavaScript, una nueva palabra clave para crear variables que funciona de manera algo diferente a var, solucionando sus problemas en el proceso.

Para empezar, si escribes un programa JavaScript de varias líneas que declara e inicia una variable, puedes declarar una variable con var después de iniciarla y seguirá funcionando. Por ejemplo:

```
myName = 'Chris';

function logName() {
  console.log(myName);
}

logName();

var myName;
```

Nota: Esto no funcionará al escribir líneas individuales en una consola de JavaScript, solo cuando se ejecutan varias líneas de JavaScript en un documento web.

Esto funciona debido a la **elevación.** La elevación (hoisting) ya no funciona con let. Si cambiamos var a let en el ejemplo anterior, fallaría con un error. Declarar una variable después de iniciarla resulta en un código confuso y más difícil de entender.

En segundo lugar, cuando usas var, puedes declarar la misma variable tantas veces como desees, pero con let no puedes. Lo siguiente funcionaría:

```
var myName = 'Chris';
var myName = 'Bob';
```

Pero lo siguiente arrojaría un error en la segunda línea:

```
let myName = 'Chris';
let myName = 'Bob';
```

Tendrías que hacer esto en su lugar:

```
let myName = 'Chris';
myName = 'Bob';
```

Nuevamente, esta es una sensata decisión del lenguaje. No hay razón para volver a declarar las variables — solo hace que las cosas sean más confusas.

Por estas y otras razones, se recomienda utilizar let tanto como sea posible en tu código, en lugar de var. No hay ninguna razón para usar var, a menos que necesites admitir versiones antiguas de Internet

Explorer con tu código (no es compatible con let hasta la versión 11; Edge el moderno navegador de Windows admite let perfectamente).

Actualizar una variable

Una vez que una variable se ha iniciado con un valor, puedes cambiar (o actualizar) ese valor simplemente dándole un valor diferente. Intenta ingresar las siguientes líneas en tu consola:

```
myName = 'Bob';
myAge = 40;
```

Un consejo sobre las reglas de nomenclatura de variables

Puedes llamar a una variable prácticamente como quieras, pero existen limitaciones. En general, debes limitarte a usar caracteres latinos (0-9, a-z, A-Z) y el caracter de subrayado.

- No debes usar otros caracteres porque pueden causar errores o ser difíciles de entender para una audiencia internacional.
- No use guiones bajos al comienzo de los nombres de las variables — esto se usa en ciertas construcciones de JavaScript para significar cosas específicas, por lo que puede resultar confuso.
- No uses números al comienzo de las variables. Esto no está permitido y provoca un error.
- Una convención segura a seguir es la llamada **"minúscula mayúsculas intercaladas"**, en la que se juntan varias palabras con minúsculas para la primera palabra completa y luego en mayúsculas las primeras letras de las siguientes palabras.
- Haz que los nombres de las variables sean intuitivos, para que describan los datos que contienen. No uses solo letras/números o frases grandes y largas.
- Las variables distinguen entre mayúsculas y minúsculas — por lo tanto myage es una variable diferente de myAge.
- Un último punto: también debes evitar el uso de palabras reservadas de JavaScript como nombres de variables — con esto, nos referimos a las palabras que componen la sintaxis real de JavaScript. Por lo tanto, no puedes usar palabras como var, function, let y for como nombres de variables. Los navegadores las reconocen como elementos de código diferentes, por lo que obtendrás errores.

Ejemplos de buenos nombres:

- age
- myAge
- init
- initialColor
- finalOutputValue

- audio1
- audio2

Ejemplos de nombres incorrectos:

- 1
- a
- _12
- myage
- MYAGE
- var
- Document
- skjfndskjfnbdskjfb
- thisisareallylongstupidvariablenameman

Ahora, intenta crear algunas variables más, con la guía anterior en mente.

Tipo de las variables

Hay algunos tipos de datos diferentes que podemos almacenar en variables. Hasta ahora hemos analizado los dos primeros, pero hay otros.

Números

Puedes almacenar números en variables, ya sea números enteros como 30 (también llamados enteros — "integer") o números decimales como 2.456 (también llamados números flotantes o de coma flotante — "number").

No es necesario declarar el tipo de las variables en JavaScript, a diferencia de otros lenguajes de programación. Cuando le das a una variable un valor numérico, no incluye comillas:

```
let myAge = 17;
```

Cadenas de caracteres (Strings)

Las strings (cadenas) son piezas de texto. Cuando le das a una variable un valor de cadena, debes encerrarlo entre comillas simples o dobles; de lo contrario, JavaScript intenta interpretarlo como otro nombre de variable.

```
let dolphinGoodbye = 'Hasta luego y gracias por todos los peces';
```

Booleanos

Los booleanos son valores verdadero/falso — pueden tener dos valores, true o false. Estos, generalmente se utilizan para probar una condición, después de lo cual se ejecuta el código según corresponda. Así, por ejemplo, un caso simple sería:

```
let iAmAlive = true;
```

Mientras que en realidad se usaría más así:

```
let test = 6 < 3;
```

Acá se está usando el operador "menor que" (**<**) para probar si 6 es menor que 3. Como era de esperar, devuelve false, ¡porque 6 no es menor que 3!

Arreglos

Un arreglo es un objeto único que contiene múltiples valores encerrados entre corchetes y separados por comas. Intenta ingresar las siguientes líneas en tu consola:

```
let myNameArray = ['Chris', 'Bob', 'Jim'];
let myNumberArray = [10, 15, 40];
```

Una vez que se definen estos arreglos, puedes acceder a cada valor por su ubicación dentro del arreglo. Prueba estas líneas:

```
myNameArray[0]; // debería devolver 'Chris'
myNumberArray[2]; // debe devolver 40
```

Los corchetes especifican un valor de índice correspondiente a la posición del valor que deseas devolver.

Posiblemente hayas notado que los arreglos en JavaScript tienen índice cero: **el primer elemento está en el índice 0**.

Objetos

En programación, un objeto es una estructura de código que modela un objeto de la vida real.

Puedes tener un objeto simple que represente una caja y contenga información sobre su ancho, largo y alto, o podrías tener un objeto que represente a una persona y contenga datos sobre su nombre, estatura, peso, qué idioma habla, cómo saludarlo, y más

Intenta ingresar la siguiente línea en tu consola:

```
let dog = { name : 'Spot', breed : 'Dalmatian' };
```

Para recuperar la información almacenada en el objeto, puedes utilizar la siguiente sintaxis:

```
dog.name
```

Tipado dinámico

JavaScript es un "lenguaje tipado dinámicamente", lo cual significa que, a diferencia de otros lenguajes, no es necesario especificar qué tipo de datos contendrá una variable (números, cadenas, arreglos, etc.).

Por ejemplo, si declaras una variable y le das un valor entre comillas, el navegador trata a la variable como una cadena (string):

```
let myString = 'Hello';
```

Incluso si el valor contiene números, sigue siendo una cadena, así que ten cuidado:

```
let myNumber = '500'; // Esto sigue siendo una cadena
typeof myNumber;
myNumber = 500; // mucho mejor -- ahora este es un número
typeof myNumber;
```

Intenta ingresar las cuatro líneas anteriores en tu consola una por una y ve cuáles son los resultados.

Notarás que estamos usando un operador especial llamado typeof — esto devuelve el tipo de datos de la variable que escribes después.

La primera vez que se llama, debe devolver string, ya que en ese punto la variable myNumber contiene una cadena, '500'.

Constantes en JavaScript

Muchos lenguajes de programación tienen el concepto de una *constante* — un valor que, una vez declarado, no se puede cambiar. Hay muchas razones por las que querrías hacer esto, desde la seguridad (si un script de un tercero cambia dichos valores, podría causar problemas) hasta la depuración y la comprensión del código (es más difícil cambiar accidentalmente valores que no se deben cambiar y estropear cosas claras).

En los primeros días de JavaScript, las constantes no existían. En JavaScript moderno, tenemos la palabra clave const, que nos permite almacenar valores que nunca se pueden cambiar:

```
const daysInWeek = 7;
const hoursInDay = 24;
```

const funciona exactamente de la misma manera que let, excepto que a const no le puedes dar un nuevo valor. En el siguiente ejemplo, la segunda línea arrojará un error:

```
const daysInWeek = 7;
daysInWeek = 8;
```

3.2. JS - Funciones

Construye tu propia función

Aprendizaje activo: construyamos una función.

La función personalizada que vamos a construir se llamará displayMessage(). Mostrará un cuadro de mensaje personalizado en una página web y actuará como un reemplazo personalizado para la función de alert() incorporada de un navegador.

Escriba lo siguiente en la consola de JavaScript de su navegador, en la página que desee:

```
alert('This is a message');
```

La función alert tiene un argumento — el string que se muestra en la alerta. Prueba a variar el string para cambiar el mensaje.

La función alert es limitada: puede cambiar el mensaje, pero no puede cambiar de manera sencilla nada más, como el color, icono o cualquier otra cosa.

La función básica

Para empezar, vamos a poner juntos una función básica.

Comience accediendo al archivo function-start.html y haciendo una copia local. Verás que el HTML es simple — el body únicamente tiene un botón.

También hemos proporcionado algunos estilos básicos de CSS para customizar el mensaje y un elemento <**script**> (en-US) vacío para poner nuestro JavaScript dentro.

Luego añade lo siguiente dentro del elemento <**script**>:

```
function displayMessage() {
}
```

Comenzamos con la palabra clave function, lo que significa que estamos definiendo una función. A esto le sigue el nombre que queremos darle a nuestra función, un conjunto de paréntesis y un conjunto de llaves.

Todos los parámetros que queremos darle a nuestra función van dentro de los paréntesis, y el código que se ejecuta cuando llamamos a la función va dentro de las llaves. Finalmente, agregue el siguiente código dentro de las llaves:

```
let html = document.querySelector('html');

let panel = document.createElement('div');
panel.setAttribute('class', 'msgBox');
html.appendChild(panel);

let msg = document.createElement('p');
msg.textContent = 'This is a message box';
panel.appendChild(msg);

let closeBtn = document.createElement('button');
closeBtn.textContent = 'x';
panel.appendChild(closeBtn);

closeBtn.onclick = function() {
  panel.parentNode.removeChild(panel);
}
```

La primera línea usa un función DOM API llamada document.querySelector() para seleccionar el elemento <html> y guardar una referencia a él en una variable llamada html, por lo que podemos hacer cosas para más adelante:

```
let html = document.querySelector('html');
```

La siguiente sección usa otra función del DOM API llamada Document.createElement() para crear un elemento y guardar una referencia a él en una variable llamada panel. Este elemento será el contenedor exterior de nuestro cuadro de mensaje.

Entonces usamos otra función del API DOM llamada Element.setAttribute() para configurar un atributo a class en nuestro panel con un valor de msgBox. Esto es para facilitar el estilo del elemento.

Si echas un vistazo al CSS en la página, verás que estamos utilizando un selector de clases.msgBox para dar estilo al contenedor del mensaje.

Finalmente, llamamos a una función del DOM llamada Node.appendChild() en la variable html que hemos guardado anteriormente, que anida un elemento dentro del otro como hijo de él.

Hemos especificado el panel como el hijo que queremos añadir dentro del elemento <html>. Debemos hacer esto ya que el elemento que creamos no aparecerá en la página por sí solo — tenemos que especificar donde ponerlo.

```
let panel = document.createElement('div');
panel.setAttribute('class', 'msgBox');
html.appendChild(panel);
```

Las siguientes dos secciones hacen uso de las mismas funciones createElement() y appendChild() que ya vimos para crear dos nuevos elementos — un <p> y un <button> — e insertarlo en la página como un hijo del panel . Usamos su propiedad Node.textContent — que representa el contenido de texto de un elemento: para insertar un mensaje dentro del párrafo y una 'x' dentro del botón. Este botón será lo que necesita hacer clic / activar cuando el usuario quiera cerrar el cuadro de mensaje.

```
let msg = document.createElement('p');
msg.textContent = 'This is a message box';
panel.appendChild(msg);

let closeBtn = document.createElement('button');
closeBtn.textContent = 'x';
panel.appendChild(closeBtn);
```

Finalmente, usamos el manejador de evento GlobalEventHandlers.onclick para hacerlo de modo que cuando se haga clic en el botón, se ejecute algún código para eliminar todo el panel de la página, para cerrar el cuadro de mensaje.

Brevemente, el handler onclick es una propiedad disponible en el botón (o, de hecho, en cualquier elemento de la página) que se puede configurar en una función para especificar qué código ejecutar cuando se hace clic en el botón.

Estamos haciendo el handler onclick igual que una función anónima, que contiene el código para ejecutar cuando se ha hecho click en el botón. La línea dentro de la función usa la función del DOM API Node.removeChild() para especificar que queremos eliminar un elemento secundario específico del elemento HTML— en este caso el panel .

```
closeBtn.onclick = function() {
  panel.parentNode.removeChild(panel);
}
```

Básicamente, todo este bloque de código está generando un bloque de HTML que se ve así, y lo está insertando en la página:

```
<div class="msgBox">
  <p>This is a message box</p>
  <button>x</button>
</div>
```

Fue un montón de código con el que trabajar: ¡no te preocupes demasiado si no recuerdas exactamente cómo funciona todo ahora! La parte principal en la que queremos centrarnos aquí es la estructura y el uso de la función, pero queríamos mostrar algo interesante para este ejemplo.

Llamando a la función

Ahora tienes la definición de tu función escrita en tu elemento <**script**> bien, pero no hará nada tal como está.

Intente incluir la siguiente línea debajo de su función para llamarla:

```
displayMessage();
```

Esta línea invoca la función, haciéndola correr inmediatamente. Cuando guarde el código y lo vuelva a cargar en el navegador, verá que el pequeño cuadro de mensaje aparece inmediatamente, solo una vez. Después de todo, solo lo llamamos una vez.

Ahora abra las herramientas de desarrollo de su navegador en la página de ejemplo, vaya a la consola de JavaScript y escriba la línea nuevamente allí, ¡verá que aparece nuevamente! Ahora tenemos una función reutilizable que podemos llamar en cualquier momento que queramos.

Pero probablemente queremos que aparezca en respuesta a las acciones del usuario y del sistema. En una aplicación real, tal cuadro de mensaje probablemente se llamará en respuesta a la disponibilidad de nuevos datos, a un error, al usuario que intenta eliminar su perfil ("¿está seguro de esto?"), o al usuario que agrega un nuevo contacto y la operación se está completando con éxito ... etc.

En esta demostración, obtendremos el cuadro de mensaje que aparecerá cuando el usuario haga clic en el botón. Elimina la línea anterior que agregaste.

A continuación, seleccionaremos el botón y guardaremos una referencia a él en una variable. Agregue la siguiente línea a su código, encima de la definición de la función:

```
let btn = document.querySelector('button');
```

Finalmente, agregue la siguiente línea debajo de la anterior:

```
btn.onclick = displayMessage;
```

De una forma similar que nuestra línea dentro de la función closeBtn.onclick..., aquí estamos llamando a algún código en respuesta a un botón al hacer clic. Pero en este caso, en lugar de llamar a una función anónima que contiene algún código, estamos llamando directamente a nuestro nombre de función.

Intente guardar y actualizar la página: ahora debería ver aparecer el cuadro de mensaje cuando haga clic en el botón.

Quizás te estés preguntando por qué no hemos incluido los paréntesis después del nombre de la función. Esto se debe a que no queremos llamar a la función inmediatamente, solo después de hacer clic en el botón.

Si intentas cambiar la línea a:

```
btn.onclick = displayMessage();
```

Al guardar y volver a cargar, verás que aparece el cuadro de mensaje sin hacer clic en el botón. Los paréntesis en este contexto a veces se denominan "operador de invocación de función". Solo los utiliza cuando desea ejecutar la función inmediatamente en el ámbito actual. Del mismo modo, el código dentro de la función anónima no se ejecuta inmediatamente, ya que está dentro del alcance de la función.

Si has intentado el último experimento, asegúrate de deshacer el último cambio antes de continuar.

Mejora de la función con parámetros.

Tal como está, la función aún no es muy útil, no queremos mostrar el mismo mensaje predeterminado cada vez. Mejoremos nuestra función agregando algunos parámetros, permitiéndonos llamarla con algunas opciones diferentes.

En primer lugar, actualice la primera línea de la función:

```
function displayMessage() {
```

colocando esta:

```
function displayMessage(msgText, msgType) {
```

Ahora, cuando llamamos a la función, podemos proporcionar dos valores variables dentro de los paréntesis para especificar el mensaje que se mostrará en el cuadro de mensaje y el tipo de mensaje que es.

Para utilizar el primer parámetro, actualiza la siguiente línea dentro de su función:

```
msg.textContent = 'This is a message box';
```

a

```
msg.textContent = msgText;
```

Por último, pero no menos importante, ahora necesita actualizar su llamada de función para incluir un texto de mensaje actualizado. Cambia la siguiente línea:

```
btn.onclick = displayMessage;
```

por este código:

```
btn.onclick = function() {
  displayMessage('Woo, this is a different message!');
};
```

Si queremos especificar parámetros dentro de paréntesis para la función a la que estamos llamando, no podemos llamarla directamente, necesitamos colocarla dentro de una función anónima para que no esté en el ámbito inmediato y, por lo tanto, no se llame de inmediato. Ahora no se llamará hasta que se haga clic en el botón.

Vuelva a cargar e intenta el código nuevamente y verás que aún funciona bien, ¡excepto que ahora también puede variar el mensaje dentro del parámetro para obtener diferentes mensajes mostrados en el cuadro!

Un parámetro más complejo.

El siguiente parámetro va a implicar un poco más de trabajo: lo configuraremos de modo que, dependiendo de la configuración del parámetro msgType, la función mostrará un icono diferente y un color de fondo diferente.

En primer lugar, descarga los iconos necesarios para este ejercicio (warning y chat) de GitHub. Guárdalos en una nueva carpeta llamada icons en la misma localización que tu HTML.

A continuación, encuentra el CSS dentro de tu archivo HTML. Haremos algunos cambios para dar paso a los iconos. Primero, actualiza el ancho de .msgBox desde:

```
width: 200px;
```

por lo siguiente:

```
width: 242px;
```

Luego, añade las siguientes líneas dentro de la regla.msgBox p { ... }:

```
padding-left: 82px;
background-position: 25px center;
background-repeat: no-repeat;
```

Ahora necesitamos añadir código a la función displayMessage() para manejar la visualización de los iconos. Agrega el siguiente bloque justo encima de la llave de cierre (}) de tu función:

```
if (msgType === 'warning') {
  msg.style.backgroundImage = 'url(icons/warning.png)';
  panel.style.backgroundColor = 'red';
} else if (msgType === 'chat') {
  msg.style.backgroundImage = 'url(icons/chat.png)';
  panel.style.backgroundColor = 'aqua';
} else {
  msg.style.paddingLeft = '20px';
}
```

Aquí, si el parámetro msgType se establece como 'warning', se muestra el icono de advertencia y el color de fondo del panel se establece en rojo. Si se establece en 'chat', se muestra el icono de chat y el color de fondo del panel se establece en azul aguamarina.

Si el parámetro msgType no está configurado en absoluto (o en algo diferente), entonces la parte else { ... } del código entra en juego, y al párrafo simplemente se le da un relleno predeterminado y ningún icono, sin el conjunto de colores del panel de fondo ya sea. Esto proporciona un estado predeterminado si no se proporciona ningún parámetro msgType , lo que significa que es un parámetro opcional.

Vamos a probar nuestra función actualizada, prueba a actualizar la llamada a displayMessage() de esto:

```
displayMessage('Woo, this is a different message!');
```

por uno de estos:

```
displayMessage('Your inbox is almost full -- delete some mails', 'warning');
displayMessage('Brian: Hi there, how are you today?','chat');
```

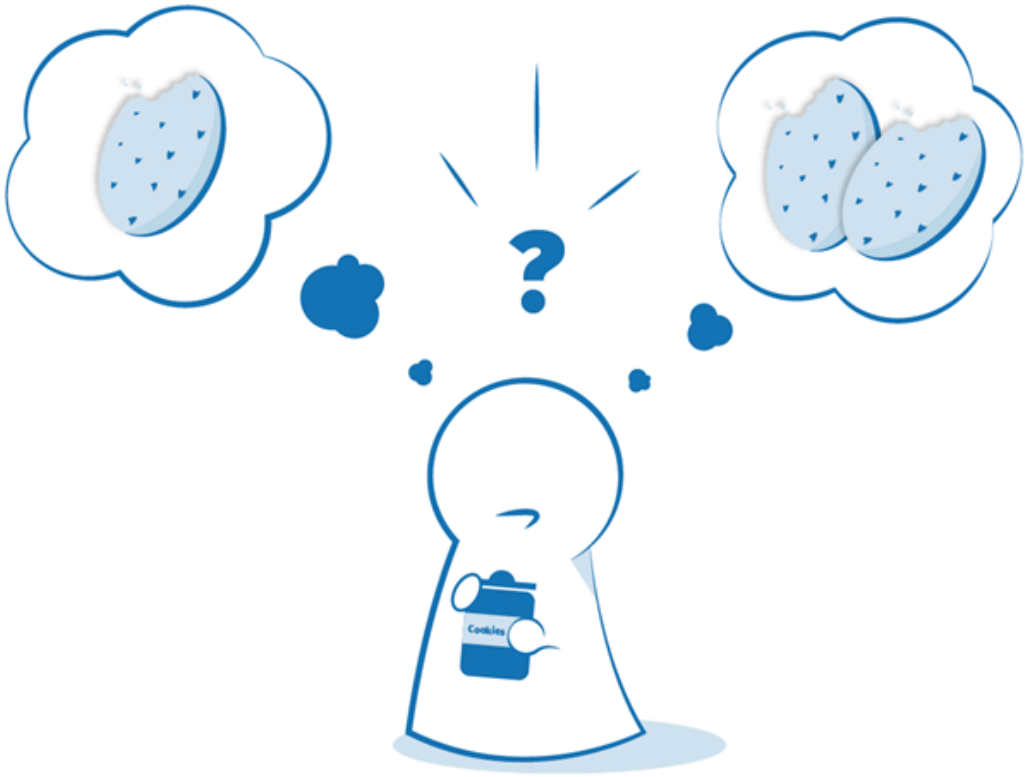
Puedes ver cuán útil se está volviendo nuestra (ahora no tan) poca función.

3.3. JS - condicionales

Tomando decisiones en tu código — condicionales

Puedes hacerlo en una condición..!

Los seres humanos (y otros animales) toman decisiones todo el tiempo que afectan sus vidas, desde la más insignificante ("¿Debería comer una galleta o dos?") hasta la más grande (¿Debería quedarme en mi país y trabajar en la granja de mi padre, o debería mudarme a Estados Unidos y estudiar astrofísica?).



Declaraciones if ... else

Echemos un vistazo a la declaración condicional más común que usarás en JavaScript.

Sintaxis if ... else básica.

Una sintaxis básica if...else luce así.

```
if (condición) {
  código a ejecutar si la condición es verdadera
} else {
  ejecuta este otro código si la condición es falsa
}
```

Aquí tenemos:

1. La palabra clave if seguida de unos paréntesis.
2. Una condición a probar, puesta dentro de los paréntesis (típicamente "¿es este valor mayor que este otro valor?", o "¿existe este valor?"). Esta condición usará los **operadores de comparación** y retorna un valor true o false (verdadero o falso).
3. Un conjunto de llaves, en las cuales tenemos algún código — puede ser cualquier código que deseemos, código que se ejecutará sólamente si la condición retorna true.
4. La palabra clave else.
5. Otro conjunto de llaves, dentro de las cuales tendremos otro código — puede ser cualquier código que deseemos, y sólo se ejecutará si la condición no es true.

Este código es fácil de leer — está diciendo "**si (if)** la **condición** retorna verdadero (true), entonces ejecute el código A, **sino (else)** ejecute el código B"

Habrás notado que no tienes que incluir else y el segundo bloque de llaves — La siguiente declaración también es perfectamente válida.

```
if (condición) {
  ejecuta el código de al ser verdadera la condición
}

ejecuta otro código
```

Sin embargo, hay que ser cuidadosos — en este caso, el segundo bloque no es controlado por una declaración condicional, así que **siempre** se ejecutará, sin importar si la condicional devuelve true o false. Esto no es necesariamente algo malo, pero puede ser algo que no quieras — a menudo desearás ejecutar un bloque de código u otro, no ambos.

Como punto final, habrá ocasiones donde veas declaraciones if...else escritas sin un conjunto de llaves, de esta manera:

```
if (condición) ejecuta código de ser verdadero (true)
else ejecuta este otro código
```

Este código es perfectamente válido, pero no es recomendado usarlo — es mucho más fácil leer el código y determinar qué sucede haciendo uso de las llaves para delimitar los bloques de código y usar varias líneas y sangrías.

Un ejemplo real

Para comprender mejor la sintaxis, realicemos un ejemplo real. Imagínese a un niño a quien su madre o padre le pide ayuda con una tarea. El padre podría decir: "¡Hola, cariño! Si me ayudas yendo y haciendo las compras, te daré un subsidio adicional para que puedas pagar ese juguete que querías". En JavaScript, podríamos representar esto así:

```
let compraRealizada = false;

if (compraRealizada === true) {
  let subsidioAdicional = 10;
} else {
  let subsidioAdicional = 5;
}
```

La variable compraRealizada escrita en este código dará siempre como resultado un retorno de valor false, lo cuál significa una desilusión para nuestro pobre hijo. Depende de nosotros proporcionar un mecanismo para que el padre cambie el valor de la variable compraRealizada a true si el niño realizó la compra.

Nota: Podrás ver una versión más completa de este ejemplo en GitHub (también podrás verlo corriendo en vivo.)

else if

El último ejemplo nos proporcionó dos opciones o resultados, pero ¿qué ocurre si queremos más de dos?

Hay una forma de encadenar opciones / resultados adicionales extras a if...else — usando else if. Cada opción extra requiere un bloque adicional para poner en medio de bloque if() { ... } y else { ... } — Vea el siguiente ejemplo un poco más complicado, que podría ser parte de una aplicación para un simple pronóstico del tiempo:

```
<label for="clima">Seleccione el tipo de clima de hoy: </label>
<select id="clima">
  <option value="">--Haga una elección--</option>
  <option value="soleado">Soleado</option>
  <option value="lluvioso">Lluvioso</option>
  <option value="nevando">Nevando</option>
  <option value="nublado">Nublado</option>
</select>

<p></p>
let seleccionar = document.querySelector('select');
let parrafo = document.querySelector('p');

seleccionar.addEventListener('change', establecerClima);
```

```
function establecerClima() {
  let eleccion = seleccionar.value;

  if (eleccion === 'soleado') {
    parrafo.textContent = 'El día esta agradable y soleado hoy. ¡Use pantalones
cortos! Ve a la playa o al parque y come un helado.';
  } else if (eleccion === 'lluvioso') {
    parrafo.textContent = 'Está lloviendo, tome un abrigo para lluvia y un
paraguas, y no se quede por fuera mucho tiempo.';
  } else if (eleccion === 'nevando') {
    parrafo.textContent = 'Está nevando — ¡está congelando! Lo mejor es quedarse en
casa con una taza caliente de chocolate, o hacer un muñeco de nieve.';
  } else if (eleccion === 'nublado') {
    parrafo.textContent = 'No está lloviendo, pero el cielo está gris y nublado;
podría llover en cualquier momento, así que lleve un saco solo por si acaso.';
  } else {
    parrafo.textContent = '';
  }
}
```

1. Aquí tenemos un elemento HTML <select> que nos permite realizar varias elecciones sobre el clima, y un párrafo simple.
2. En el JavaScript, estamos almacenando una referencia para ambos elementos <select> y <p> , y añadiendo un Event Listener o en español un Detector de Eventos al elemento <select> así cuando su valor cambie se ejecuta la función establecerClima().
3. Cuando la función es ejecutada, primero establecemos la variable eleccion con el valor obtenido del elemento <select>. Luego usamos una declaración condicional para mostrar distintos textos dentro del párrafo <p> dependiendo del valor de la variable eleccion. Note como todas las condicionales son probadas en los bloques else if() {...} , a excepción del primero, el cual es probado en el primer bloque if() {...}.
4. La última elección, dentro del bloque else {...}, es básicamente el "último recurso" como opción— El código dentro de este bloque se ejecutará si ninguna de las condiciones es true. En este caso, sirve para vaciar el contenido del párrafo si nada ha sido seleccionado, por ejemplo, si el usuario decide elegir de nuevo "--Haga una elección--" mostrado al inicio.

Nota: Puedes encontrar este ejemplo en GitHub (También podrás verlo correr en vivo.)

Una nota en los operadores de comparación

Los operadores de comparación son usados para probar las condiciones dentro de nuestra declaración condicional. Nuestras opciones son:

- === y !== — prueba si un valor es exactamente igual a otro, o sino es idéntico a otro valor.
- < y > — prueba si un valor es menor o mayor que otro.
- <= y >= — prueba si un valor es menor e igual o mayor e igual que otro.

Queremos hacer una mención especial al probar los valores (true/false) , y un patrón común que te encontrarás una y otra vez. Cualquier valor que no sea false, undefined, null, 0, NaN, o una cadena vacía string (") realmente retorna el valor true cuando es probada como una declaración condicional, por lo

tanto puedes simplemente usar el nombre de una variable para probar si es true, o si al menos existe (i.e. no está definido.) Por ejemplo:

```
let queso = 'Cheddar';

if (queso) {
  console.log('¡Siii! Hay queso para hacer tostadas con queso.');
} else {
  console.log('No hay tostadas con queso para ti hoy :(.');
}
```

En el ejemplo anterior la variable queso contiene el valor 'Cheddar', y como su valor está definido o no es false, undefined, null, 0, NaN y (' ') es considerado como true lo cual hará mostrar el mensaje dentro del primer bloque de llaves.

Y, devolviéndonos al ejemplo previo del niño haciendo las compras para su padre, lo podrías haber escrito así:

```
let compraRealizada = false;

if (compraRealizada) { //no necesitas especificar explícitamente '=== true'
   let subsidioAdicional = 10;
} else {
   let subsidioAdicional = 5;
}
```

Anidando if ... else

Está perfectamente permitido poner una declaración if...else dentro de otra declaración if...else — para anidarlas. Por ejemplo, podemos actualizar nuestra aplicación del clima para mostrar una serie de opciones dependiendo de cual sea la temperatura:

```
if (elección === 'soleado') {
  if (temperatura < 86) {
    parrafo.textContent = 'Está a ' + temperatura + ' grados afuera -- agradable y soleado. Vamos a la playa, o al parque, y comer helado.';
  } else if (temperatura >= 86) {
    parrafo.textContent = 'Está a ' + temperatura + ' grados afuera -- ¡QUÉ CALOR! Si deseas salir, asegúrate de aplicarte bloqueador solar.';
  }
}
```

Aunque el código funciona en conjunto, cada declaración if...else funciona completamente independiente del otro.

Operadores lógicos: AND, OR y NOT

Si quieres probar múltiples condiciones sin escribir declaraciones if...else anidados, los operadores lógicos pueden ayudarte. Cuando se usa en condiciones, los primeros dos hacen lo siguiente:

- && — **AND**; le permite encadenar dos o más expresiones para que todas ellas se tengan que evaluar individualmente true para que la expresión entera retorne true.
- || — **OR**; le permite encadenar dos o más expresiones para que una o más de ellas se tengan que evaluar individualmente true para que la expresión entera retorne true.

Para poner un ejemplo de **AND**, el anterior código puede ser reescrito de esta manera:

```
if (eleccion === 'soleado' && temperatura < 86) {
  parrafo.textContent = 'Está a ' + temperatura + ' grados afuera -- agradable y soleado. Vamos a la playa, o al parque, y comer helado.';
} else if (eleccion === 'soleado' && temperatura >= 86) {
  parrafo.textContent = 'Está a ' + temperatura + ' grados afuera -- ¡QUÉ CALOR! Si deseas salir, asegúrate de aplicarte bloqueador solar.';
}
```

Así que por ejemplo, el primer bloque sólo se ejecutará si la variable eleccion === 'soleado' *y* temperatura < 86 devuelven un valor verdadero o true.

Observemos un ejemplo rápido del operador **OR**:

```
if (carritoDeHelados || estadoDeLaCasa === 'en llamas') {
  console.log('Debes salir de la casa rápidamente.');
} else {
  console.log('Es mejor que te quedes dentro de casa');
}
```

El último tipo de operador lógico, **NOT**, es expresado con el operador **!**, puede ser usado para negar una expresión. Vamos a combinarlo con el operador **OR** en el siguiente ejemplo:

```
if (!(carritoDeHelados || estadoDeLaCasa === 'en llamas')) {
  console.log('Es mejor que te quedes dentro de casa');
} else {
  console.log('Debes salir de la casa rápidamente.');
}
```

En el anterior ejemplo, si las declaraciones del operador OR retornan un valor true, el operador NOT negará toda la expresión dentro de los paréntesis, por lo tanto retornará un valor false.

Puedes combinar los operadores que quieras dentro de las sentencias, en cualquier estructura. El siguiente ejemplo ejecuta el código dentro del condicional sólo si ambas sentencias OR devuelven verdadero, lo que significa que la instrucción general AND devolverá verdadero:

```
if ((x === 5 || y > 3 || z <= 10) && (logueado || nombreUsuario === 'Steve')) {
  // ejecuta el código
}
```

Un error común cuando se utiliza el operador OR en las declaraciones condicionales es intentar verificar el valor de la variable una sola vez, y luego darle una lista de valores que podrían retornar verdadero separados por operadores ||. Por ejemplo:

```
if (x === 5 || 7 || 10 || 20) {
  // ejecuta mi código
}
```

En este caso la condición if(...) siempre evaluará a verdadero siendo que 7 (u otro valor que no sea 0) siempre será verdadero. Esta condición lo que realmente está diciendo es que "if x es igual a 5, o 7 es verdadero— lo cual siempre es". ¡Esto no es lógicamente lo que queremos! Para hacer que esto funcione, tenemos que especificar una prueba completa para cada lado del operador OR:

```
if (x === 5 || x === 7 || x === 10 ||x === 20) {
  // ejecuta mi código
}
```

Declaraciones con switch

El condicional if...else hace un buen trabajo permitiéndonos realizar un buen código, pero esto viene con sus desventajas. Hay variedad de casos donde necesitarás realizar varias elecciones, y cada una requiere una cantidad razonable de código para ser ejecutado y/o sus condicionales son complejas (operadores lógicos múltiples).

Para los casos en los que solo se desea establecer una variable para una determinada opción de valores o imprimir una declaración particular dependiendo de una condición, la sintaxis puede ser un poco engorrosa, especialmente si se tiene una gran cantidad de opciones.

Para estos casos los switch son de gran ayuda — toman una sola expresión / valor como una entrada, y luego pasan a través de una serie de opciones hasta que encuentran una que coincida con ese valor, ejecutando el código correspondiente que va junto con ella. Aquí hay un pseudocódigo más para hacerte una idea:

```
switch (expresion) {
  case choice1:
    ejecuta este código
    break;

  case choice2:
    ejecuta este código
    break;

  // Se pueden incluir todos los casos que quieras

  default:
    por si acaso, corre este código
}
```

Aquí tenemos:

1. La palabra clave switch, seguida por un conjunto de paréntesis.
2. Una expresión o valor dentro de los paréntesis.
3. La palabra clave case, seguida de una elección con la expresión / valor que podría ser, seguido de dos puntos.
4. Algún código a correr si la elección coincide con la expresión.
5. Una declaración llamada break, seguida de un punto y coma.
   Si la elección previa coincide con la expresión / valor, el explorador dejará de ejecutar el bloque de código aquí, y continuará a la siguiente línea de código.
   Si la opción anterior coincide con la expresión / valor, aquí el navegador deja de ejecutar el bloque de código y pasa a cualquier código que aparece debajo de la declaración de switch.
6. Como muchos otros casos, los que quieras.
7. La palabra clave default, seguido exactamente del mismo patrón de código que en los casos anteriores , excepto que el valor predeterminado no tiene opciones después de él, y no es necesario que se use break porque no hay nada que ejecutar después de este bloque de todas formas. Esta es la opción predeterminada o por defecto que se ejecuta si ninguna de las opciones coinciden.

Nota: No tiene que incluir la sección default; se puede omitir con seguridad si no hay posibilidades de que la expresión termine igualando un valor desconocido. Sin embargo, si existe la posibilidad de que esto ocurra, debe incluirlo para evitar casos desconocidos o comportamientos extraños en tu código.

Un ejemplo con switch

Echemos un vistazo a un ejemplo real: reescribiremos nuestra aplicación de pronóstico del tiempo para usar una declaración de cambio en su lugar:

```
<label for="weather">Select the weather type today: </label>
<select id="weather">
  <option value="">--Make a choice--</option>
  <option value="sunny">Sunny</option>
  <option value="rainy">Rainy</option>
  <option value="snowing">Snowing</option>
  <option value="overcast">Overcast</option>
</select>

<p></p>
let select = document.querySelector('select');
let para = document.querySelector('p');

select.addEventListener('change', setWeather);


function setWeather() {
  let choice = select.value;

  switch (choice) {
    case 'sunny':
      para.textContent = 'It is nice and sunny outside today. Wear shorts! Go to
the beach, or the park, and get an ice cream.';
      break;
    case 'rainy':
      para.textContent = 'Rain is falling outside; take a rain coat and a brolly,
and don\'t stay out for too long.';
      break;
    case 'snowing':
      para.textContent = 'The snow is coming down - it is freezing! Best to stay in
with a cup of hot chocolate, or go build a snowman.';
      break;
    case 'overcast':
      para.textContent = 'It isn\'t raining, but the sky is grey and gloomy; it
could turn any minute, so take a rain coat just in case.';
      break;
    default:
      para.textContent = '';
  }
}
```

Nota: También puedes encontrar este ejemplo en GitHub (también puedes verlo en ejecución aquí.)

Operador Ternario

Hay una última sintaxis que queremos presentarte antes de que juegues con algunos ejemplos. El **operador ternario o condicional** es una pequeña sintaxis que prueba una condición y devuelve un

valor/expresión, si es true, y otro si es false — Esto puede ser útil en algunas situaciones, y puede ocupar mucho menos código que un bloque if...else si simplemente tienes dos opciones que se eligen a través de una condición true/false. El pseudocódigo se ve así:

```
( condición ) ? ejecuta este código : ejecuta este código en su lugar
```

Veamos un ejemplo simple:

```
let greeting = ( isBirthday ) ? 'Happy birthday Mrs. Smith -- we hope you have a
great day!' : 'Good morning Mrs. Smith.';
```

Aquí tenemos una variable llamada isBirthday — si esta es true, le damos a nuestro invitado un mensaje de feliz cumpleaños; si no, le damos el saludo diario estándar.

Ejemplo con operador ternario

No sólo puedes establecer valores variables con el operador ternario; También puedes ejecutar funciones o líneas de código — lo que quieras. El siguiente ejemplo muestra un selector de tema simple donde el estilo del sitio se aplica utilizando un operador ternario.

```
<label for="theme">Select theme: </label>
<select id="theme">
  <option value="white">White</option>
  <option value="black">Black</option>
</select>

<h1>This is my website</h1>
let select = document.querySelector('select');
let html = document.querySelector('html');
document.body.style.padding = '10px';

function update(bgColor, textColor) {
  html.style.backgroundColor = bgColor;
  html.style.color = textColor;
}

select.onchange = function() {
  ( select.value === 'black' ) ? update('black','white') : update('white','black');
}
```

Aquí tenemos un elemento <select> para elegir un tema (blanco o negro), más un simple <h1> para mostrar el título de un sitio web. También tenemos una función llamada update(), que toma dos colores como parámetros (entradas). El color de fondo del sitio web se establece en el primer color proporcionado y el color del texto se establece en el segundo color proporcionado.

Finalmente, también tenemos un detector de eventos onchange que sirve para ejecutar una función que contiene un operador ternario.

Comienza con una condición de prueba — select.value === 'black'. Si esto devuelve true, ejecutamos la función update() con parámetros de blanco y negro, lo que significa que terminamos con un color de

fondo negro y un color de texto blanco. Si devuelve false, ejecutamos las función update() con parámetros de blanco y negro, lo que significa que el color del sitio está invertido.

Nota: También puedes encontrar este ejemplo en GitHub (y verlo en ejecución aquí.)

Aprendizaje activo: Un calendario simple

En este ejemplo, nos ayudará a terminar una aplicación de calendario simple. En el código tienes:

- Un elemento <select> para permitir al usuario elegir entre diferentes meses.
- Un controlador de eventos onchange para detectar cuándo se cambia el valor seleccionado en el menú de <select>.
- Una función llamada createCalendar() que dibuja el calendario y muestra el mes correcto en el elemento <h1>.

Necesitamos que escriba una declaración condicional dentro de la función del controlador onchange justo debajo del comentario // AGREGAR DECLARACIÓN DE CAMBIO:. Debería:

1. Mire el mes seleccionado (almacenado en la variable choice. Este será el valor del elemento <select> después de que cambie el valor, por ejemplo "January")
2. Establezca una variable llamada days para que sea igual al número de días del mes seleccionado. Para hacer esto, tendrá que buscar el número de días en cada mes del año. Puede ignorar los años bisiestos a los efectos de este ejemplo.

Sugerencias:

- Se le aconseja que utilice el operador lógico **OR** para agrupar varios meses en una sola condición; Muchos de ellos comparten el mismo número de días.
- Piense qué número de días es el más común y utilícelo como valor predeterminado.

Si comete un error, siempre puede restablecer el ejemplo con el botón "Reset". Si se queda realmente atascado, presione "Show solution" para ver una solución.

Aprendizaje activo: ¡Más opciones de colores!

En este ejemplo, usaremos el ejemplo del operador ternario que vimos anteriormente y convertiremos el operador ternario en una declaración que nos permitirá aplicar más opciones.

Como podemos ver, el <select> no tiene dos opciones, sino cinco. Deberás agregar un switch justo debajo del comentario // AGREGAR DECLARACIÓN DE CAMBIO:

- Debe aceptar la variable choice como su expresión de entrada.
- Para cada caso, choice debe ser igual a cada una de las posibles opciones, (white, black, purple, yellow, or psychedelic). Debe tener en cuenta que los valores diferencian las minúsculas y mayúsculas. y deben ser iguales a los valores del elemento <option>.

- Para cada caso, la funcion update() debe ser ejecutada, y recibir dos valores de color, el primero es el color del fondo y el segundo el color del texto. Debes recordar que los colores son strings y deben estar entre comillas.

3.4. JS - Arrays

Arrays - Arreglos o Matrices

¿Qué es una matriz?

Las matrices se describen como "objetos tipo lista"; básicamente son objetos individuales que contienen múltiples valores almacenados en una lista. Los objetos de matriz pueden almacenarse en variables y tratarse de la misma manera que cualquier otro tipo de valor, la diferencia es que podemos acceder individualmente a cada valor dentro de la lista y hacer cosas útiles y eficientes con la lista, como recorrerlo con un bucle y hacer una misma cosa a cada valor.

Tal vez tenemos una serie de productos y sus precios almacenados en una matriz, y queremos recorrerlos e imprimirlos en una factura, sumando todos los precios e imprimiendo el total en la parte inferior.

Si no tuviéramos matrices, tendríamos que almacenar cada elemento en una variable separada, luego llamar al código que hace la impresión y agregarlo por separado para cada artículo. Esto sería mucho más largo de escribir, menos eficiente y más propenso a errores. Si tuviéramos 10 elementos para agregar a la factura, ya sería suficientemente malo, pero ¿ qué pasa con 100 o 1000 artículos? Volveremos a este ejemplo más adelante.

Creando una matriz

Las matrices se construyen con corchetes, que contiene una lista de elementos separados por comas.

1. Digamos que queríamos almacenar una lista de compras en una matriz — haríamos algo como lo siguiente. Ingresa las siguientes líneas en la consola:

```
let shopping = ['bread', 'milk', 'cheese', 'hummus', 'noodles'];
shopping;
```

En este caso, cada elemento de la matriz es una cadena, pero ten en cuenta que puedes almacenar cualquier elemento en una matriz — cadena, número, objeto, otra variable, incluso otra matriz. También puedes mezclar y combinar tipos de elementos — no todos tienen que ser números, cadenas, etc.

Prueba estos:

```
let sequence = [1, 1, 2, 3, 5, 8, 13];
let random = ['tree', 795, [0, 1, 2]];
```

2. Intenta crear un par de matrices por tu cuenta, antes de continuar.

Accediendo y modificando elementos de la matriz

Puedes entonces acceder a elementos individuales en la matriz mediante la notación de corchetes, del mismo modo que accediste a las letras de una cadena.

1. Ingresa lo siguiente en tu consola:

```
shopping[0];
// returns "bread"
```

También puedes modificar un elemento en una matriz simplemente dando a un ítem de la matriz un nuevo valor. Prueba esto:

```
shopping[0] = 'tahini';
shopping;
// shopping will now return [ "tahini", "milk", "cheese", "hummus",
"noodles" ]
```

Ten en cuenta que una matriz dentro de otra matriz se llama matriz multidimensional. Puedes acceder a los elementos de una matriz que estén dentro de otra, encadenando dos pares de corchetes.

Por ejemplo, para acceder a uno de los elementos dentro de la matriz, que a su vez, es el tercer elemento dentro de la matriz random, podríamos hacer algo como esto:

```
random[2][2];
```

3. Intenta seguir jugando y haciendo algunas modificaciones más a tus ejemplos de matriz antes de continuar.

Encontrar la longitud de una matriz

Puedes averiguar la longitud de una matriz (cuántos elementos contiene) exactamente de la misma manera que determinar la longitud (en caracteres) de una cadena— utilizando la propiedad length. Prueba lo siguiente:

```
sequence.length;
// should return 7
```

Esto tiene otros usos, pero se usa más comúnmente para indicarle a un ciclo que continúe hasta que haya recorrido todos los elementos de la matriz. Así por ejemplo:

```
let sequence = [1, 1, 2, 3, 5, 8, 13];
for (var i = 0; i < sequence.length; i++) {
  console.log(sequence[i]);
}
```

Aprenderás acerca de bucles correctamente en un artículo futuro, pero brevemente, éste código dice:

1. Comienza el bucle en el elemento de la posición 0 en la matriz.
2. Detén el bucle en el número de ítem igual a la longitud de la matriz. Esto funcionará para una matriz de cualquier longitud, pero en este caso el ciclo se detendrá en el elemento número 7 (esto es bueno, ya que el último elemento — que queremos que recorra el bucle — es 6).
3. Para cada elemento, imprime en la consola del navegador con console.log().

Alguno métodos de matriz útiles

En esta sección veremos algunos métodos bastante útiles relacionados con matrices que nos permiten dividir cadenas en elementos de matriz y viceversa, y agregar nuevos elementos en matrices.

Conversión entre matrices y cadenas

A menudo se te presentarán algunos datos brutos contenidos en una cadena larga y grande, y es posible que desees separar los elementos útiles de una forma más conveniente y luego hacerle cosas, como mostrarlos en una tabla de datos.

Para hacer esto, podemos usar el método split(). En su forma más simple, esto toma un único parámetro, el caracter que quieres separar de la cadena, y devuelve las subcadenas entre el separador como elementos en una matriz.

1. Vamos a jugar con esto, para ver cómo funciona. Primero, crea una cadena en tu consola:

```
let myData = 'Manchester,London,Liverpool,Birmingham,Leeds,Carlisle';
```

Ahora dividámoslo en cada coma:

```
let myArray = myData.split(',');
myArray;
```

Finalmente, intenta encontrar la longitud de tu nueva matriz y recuperar algunos elementos de ella:

```
myArray.length;
myArray[0]; // the first item in the array
myArray[1]; // the second item in the array
myArray[myArray.length-1]; // the last item in the array
```

También puedes ir en la dirección opuesta usando el método join(). Prueba lo siguiente:

```
let myNewString = myArray.join(',');
myNewString;
```

Otra forma de convertir una matriz en cadena es usar el método toString(). Es posiblemente más simple que join() ya que no toma un parámetro, pero es más limitado. Con join() puedes especificar diferentes separadores (intenta ejecutar con un caracter diferente a la coma).

```
let dogNames = ['Rocket','Flash','Bella','Slugger'];
dogNames.toString(); //Rocket,Flash,Bella,Slugger
```

Agregar y eliminar elementos de la matriz

Todavía no hemos cubierto la posibilidad de agregar y eliminar elementos de la matriz — echemos un vistazo a esto ahora. Usaremos la matriz myArray:

```
let myArray = ['Manchester', 'London', 'Liverpool', 'Birmingham', 'Leeds',
'Carlisle'];
```

Antes que nada, para añadir o eliminar un elemento al final de una matriz podemos usar push() y pop() respectivamente.

1. primero usemos push() — nota que necesitas incluir uno o más elementos que desees agregar al final de tu matriz. Prueba esto:

```
myArray.push('Cardiff');
myArray;
myArray.push('Bradford', 'Brighton');
myArray;
```

La nueva longitud de la matriz se devuelve cuando finaliza la llamada al método. Si quisieras almacenar la nueva longitud de matriz en una variable, podrías hacer algo como esto:

```
let newLength = myArray.push('Bristol');
myArray;
newLength;
```

Eliminar el último elemento de una matriz es tan simple como ejecutar pop() en ella. Prueba esto:

```
myArray.pop();
```

El elemento que sé eliminó se devuelve cuando se completa la llamada al método. Para guardar este elemento en una variable, puedes hacer lo siguiente:

```
let removedItem = myArray.pop();
myArray;
removedItem;
```

unshift() y shift() funcionan exactamente igual de push() y pop(), respectivamente, excepto que funcionan al principio de la matriz, no al final.

Primero unshift() — prueba el siguiente comando:

```
myArray.unshift('Edinburgh');
myArray;
```

Ahora shift(); ¡prueba estos!

```
let removedItem = myArray.shift();
myArray;
removedItem;
```

Aprendizaje activo: ¡Imprimiendo esos productos!

Volvamos al ejemplo que describimos anteriormente — imprima los nombres de los productos y los precios en una factura, luego, sume los precios e imprímelos en la parte inferior. En el ejemplo editable a continuación, hay comentarios que contienen números — cada uno de estos marca un lugar donde debe agregar algo al código. Ellos son los siguientes:

1. Debajo de // number 1 hay un número de cadena, cada una de las cuales contiene un nombre de producto y un precio separados por dos puntos. Nos gustaría que convirtieras esto en una matriz y lo almacenamos en una matriz llamada products.
2. En la misma línea que el comentario // number 2 es el comienzo de un ciclo for. En esta línea, actualmente tenemos i <= 0, que es una prueba condicional que hace que el bucle for se detenga inmediatamente, porque dice "detener cuando i es menor o igual 0", y i comienza en 0. Nos gustaría que reemplazaras esto con una prueba condicional que detenga el ciclo cuando i no sea inferior a la longitud la matriz products .
3. justo debajo del comentario // number 3 queremos que escriba una línea de código que divide el elemento actual de la matriz (**nombre**:precio) en dos elementos separados, uno que contiene solo el nombre y otros que contienen solo el precio.
4. Como parte de la línea de código anterior, también querrás convertir el precio de una cadena a un número.

5. Hay una variable llamada total que se crea y se le da un valor de 0 en la parte superior del código. Dentro del ciclo (debajo de // number 4) queremos que agregues una línea que añade el precio actual del artículo a ese total en cada iteración del ciclo, de modo que al final del código el total correcto se imprima en la factura. Es posible que necesites un **operador de asignación** para hacer esto.
6. Queremos que cambies la línea justo debajo // number 5 para que la variable itemText se iguale a "nombre de elemento actual — $precio de elemento actual", por ejemplo "Zapatos — $23.99" en cada caso, por lo que la información correcta del artículo está impreso en la factura. Esto es simplemente una concatenación de cadenas, lo que debería ser familiar para ti.

Aprendizaje Activo: Top 5 búsquedas

Un buen uso para los métodos de matriz como push() y pop() es cuando estás manteniendo un registro de elementos actualmente activos en una aplicación web.

En una escena animada por ejemplo, es posible que tengas una matriz de objetos que representan los gráficos de fondo que se muestran actualmente, y es posible que sólo desees que se muestren 50 a la vez, por razones de rendimiento o desorden. A medida que se crean y agregan nuevos objetos a la matriz, se puede eliminar los más antiguos de la matriz para mantener el número deseado.

En este ejemplo vamos a mostrar un uso mucho más simple — aquí te daremos un sitio de búsqueda falso, con un cuadro de búsqueda. La idea es que cuando los términos se ingresan en un cuadro de búsqueda, se muestran el top 5 de términos de búsqueda previos en la lista. Cuando el número de términos supera el 5, el último término comienza a borrarse cada vez que agregas un nuevo término a la parte superior, por lo que siempre se muestran los 5 términos anteriores.

Nota: En una aplicación de búsqueda real, probablemente puedas hacer clic en los términos de búsqueda anteriores para volver a los términos de búsqueda anteriores y ¡se mostrarán los resultados de búsqueda reales! Solamente lo mantendremos simple por ahora.

Para completar la aplicación necesitamos:

1. Agregar una línea debajo del comentario // number 1 que agrega el valor actual ingresado en la entrada de la búsqueda al inicio de la matriz. Esto se puede recuperar usando searchInput.value.
2. Agrega una línea debajo del comentario // number 2 que elimina el valor actualmente al final de la matriz.

3.5. JS – Función Retorno de Variables

Una función retorna valores

Algunas funciones no devuelven un valor significativo después de su finalización, pero otras sí, y es importante comprender cuáles son sus valores, cómo utilizarlos en su código y cómo hacer que sus propias funciones personalizadas devuelvan valores útiles.

¿Qué son los valores de retorno?

Los valores de retorno son exactamente como suenan: los valores devueltos por la función cuando se completa. Ya has alcanzado los valores de retorno varias veces, aunque es posible que no hayas pensado en ellos explícitamente. Volvamos a un código familiar:

```
var myText = 'I am a string';
var newString = myText.replace('string', 'sausage');
console.log(newString);
// la función de cadena replace () toma una cadena,
// sustituyendo una subcadena con otra y devolviendo
// una cadena nueva con la sustitución realizada
```

Vimos exactamente este bloque de código en nuestro primer artículo de función. Estamos invocando la función replace () en la cadena myText, y le pasamos dos parámetros: la subcadena a encontrar y la subcadena con la que reemplazarla.

Cuando esta función se completa (termina de ejecutarse), devuelve un valor, que es una nueva cadena con el reemplazo realizado. En el código anterior, estamos guardando este valor de retorno como el valor de la variable newString.

Algunas funciones no devuelven un valor de retorno como tal. Por ejemplo, en la función displayMessage () que creamos anteriormente, no se devuelve ningún valor específico como resultado de la función que se invoca. Simplemente hace que aparezca un cuadro en algún lugar de la pantalla, ¡eso es todo!

Generalmente, se usa un valor de retorno donde la función es un paso intermedio en un cálculo de algún tipo. Quieres llegar a un resultado final, que involucra algunos valores. Esos valores deben ser calculados por una función, que luego devuelve los resultados para que puedan usarse en la siguiente etapa del cálculo.

Uso de valores de retorno en sus propias funciones

Para devolver un valor de una función personalizada, debe usar la palabra clave return. La funcion draw() dibuja 100 círculos aleatorios en algún lugar del <canvas>

```
function draw() {
  ctx.clearRect(0,0,WIDTH,HEIGHT);
  for (var i = 0; i < 100; i++) {
    ctx.beginPath();
    ctx.fillStyle = 'rgba(255,0,0,0.5)';
    ctx.arc(random(WIDTH), random(HEIGHT), random(50), 0, 2 * Math.PI);
    ctx.fill();
  }
}
```

Dentro de cada iteración del bucle, se realizan tres llamadas a la función random() para generar un valor aleatorio para la **coordenada x**, **la coordenada y** y **el radio del círculo actual**, respectivamente.

La función random() toma un parámetro, un número entero, y devuelve un número aleatorio entero entre 0 y ese número. Se parece a esto:

```
function randomNumber(number) {
  return Math.floor(Math.random()*number);
}
```

Esto podría escribirse de la siguiente manera:

```
function randomNumber(number) {
  var result = Math.floor(Math.random()*number);
  return result;
}
```

Pero la primera versión es más rápida de escribir y más compacta.

Devolvemos el resultado del cálculo Math.floor(Math.random()*number) cada vez que se llama a la función. Este valor de retorno aparece en el punto en que se llamó a la función y el código continúa. Entonces, por ejemplo, si ejecutamos la siguiente línea:

```
ctx.arc(random(WIDTH), random(HEIGHT), random(50), 0, 2 * Math.PI);
```

y las tres llamadas random() devolvieron los valores 500, 200 y 35, respectivamente, la línea en realidad se ejecutaría como si fuera esto:

```
ctx.arc(500, 200, 35, 0, 2 * Math.PI);
```

Las llamadas de la función se ejecutan primero y sus valores de retorno se sustituyen por la función, antes de que se ejecute la línea en sí.

Aprendizaje activo: nuestra propia función de valor de retorno

Probemos escribir nuestras propias funciones con valores de retorno.

1. En primer lugar, haga una copia local del archivo function-library.html de GitHub. Esta es una página HTML simple que contiene un campo de texto <input> y un párrafo. También hay un elemento <**script**> en el que hemos almacenado una referencia a ambos elementos HTML en dos variables. Esta pequeña página le permitirá ingresar un número en el cuadro de texto y mostrar diferentes números relacionados con él en el párrafo a continuación.
2. Agreguemos algunas funciones útiles a este elemento <script>. Debajo de las dos líneas existentes de JavaScript, agregue las siguientes definiciones de funciones:

```
function squared(num) {
  return num * num;
}

function cubed(num) {
  return num * num * num;
}

function factorial(num) {
  var x = num;
  while (x > 1) {
    num *= x-1;
    x--;
  }
  return num;
}
```

Las funciones squared() y cubed(): devuelven el cuadrado o el cubo del número dado como parámetro. La función factorial() devuelve el factorial del número dado.

A continuación, vamos a incluir una forma de imprimir información sobre el número ingresado en la entrada de texto. Introduzca el siguiente controlador de eventos debajo de las funciones existentes:

```
input.onchange = function() {
  var num = input.value;
  if (isNaN(num)) {
    para.textContent = 'You need to enter a number!';
  } else {
    para.textContent = num + ' squared is ' + squared(num) + '. ' +
                       num + ' cubed is ' + cubed(num) + '. ' +
                       num + ' factorial is ' + factorial(num) + '.';
  }
}
```

3. Aquí estamos creando un controlador de eventos onchange que se ejecuta cada vez que el evento de cambio se activa en la entrada de texto, es decir, cuando se ingresa un nuevo valor en la entrada de texto y se envía (ingrese un valor y luego presione la pestaña, por ejemplo). Cuando se ejecuta esta función anónima, el valor existente ingresado en la entrada se almacena en la variable num

   A continuación, hacemos una prueba condicional: si el valor ingresado no es un número, imprimimos un mensaje de error en el párrafo. La prueba analiza si la expresión isNaN(num) devuelve verdadero. Usamos la función isNaN() para probar si el valor num no es un número; si es así, devuelve verdadero, y si no, falso.

   Si la prueba devuelve falso, el valor numérico es un número, por lo que imprimimos una oración dentro del elemento de párrafo que indica cuál es el cuadrado, el cubo y el factorial del número. La oración llama a las funciones squared(), cubed() y factorial() para obtener los valores requeridos.

4. Guarde su código, cárgalo en un navegador y pruébelo.

4. JavaScript Avanzado

Temario:

- Funciones.
- DOM.
- Eventos.

4.1. JavaScript Avanzado

Funciones

Cuando se desarrolla una aplicación compleja, es muy habitual utilizar una y otra vez las mismas instrucciones. Un script para una tienda de comercio electrónico, por ejemplo, tiene que calcular el precio total de los productos varias veces, para añadir los impuestos y los gastos de envío. Cuando una serie de instrucciones se repiten una y otra vez, se complica demasiado el código fuente de la aplicación, ya que:

● El código de la aplicación es mucho más largo porque muchas instrucciones están repetidas.

● Si se quiere modificar alguna de las instrucciones repetidas, se deben hacer tantas modificaciones como veces se haya escrito esa instrucción, lo que se convierte en un trabajo muy pesado y muy propenso a cometer errores.

Las funciones son la solución a todos estos problemas, tanto en JavaScript como en el resto de lenguajes de programación. Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente. En el siguiente ejemplo, las instrucciones que suman los dos números y muestran un mensaje con el resultado se repiten una y otra vez:

```
var resultado;

var numero1 = 3;
var numero2 = 5;

// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
alert("El resultado es " + resultado);

numero1 = 10;
numero2 = 7;
// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
alert("El resultado es " + resultado);

numero1 = 5;
numero2 = 8;

// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
 alert("El resultado es " + resultado);
```

Aunque es un ejemplo muy sencillo, parece evidente que repetir las mismas instrucciones a lo largo de todo el código no es algo recomendable. La solución que proponen las funciones consiste en extraer las instrucciones que se repiten y sustituirlas por una instrucción del tipo "en este punto, se ejecutan las instrucciones que se han extraído".

Por lo tanto, en primer lugar se debe crear la función básica con las instrucciones comunes. Las funciones en JavaScript se definen mediante la palabra reservada function, seguida del nombre de la función. Su definición formal es la siguiente:

function nombre_funcion() {

……..

}

El nombre de la función se utiliza para llamar a esa función cuando sea necesario. El concepto es el mismo que con las variables, a las que se les asigna un nombre único para poder utilizarlas dentro del código.

```
function suma_y_muestra() {
    resultado = numero1 + numero2;
    alert("El resultado es " + resultado);
 }
```

De esta forma, el ejemplo anterior queria asi:

```
function suma_y_muestra() {
  resultado = numero1 + numero2;
  alert("El resultado es " + resultado);
}

var resultado;
var numero1 = 3;
var numero2 = 5;

suma_y_muestra()

numero1 = 10;
numero2 = 7;

suma_y_muestra();

numero1 = 5;
numero2 = 8;

suma_y_muestra();
```

Otra posibilidad de las funciones es la de recibir parámetros. Estos son valores que pueden ser usados dentro del código que la función ejecuta. De esta manera las funciones se vuelven sumamente

reutilizables. Cuando definimos la función, damos nombre a los parámetros que queremos recibir y estos estarán disponibles como variables dentro de la función.

Cuando llamemos a la función solo debemos pasar la variable o valor que deseemos usar como parámetro de la función en la posición correcta.

```
function saludar(nombre) {
  alert("Hola " + nombre);
}

saludar("Gabriela")
saludar("Antonio")
```

Por último, las funciones también pueden devolver un valor o resultado mediante el uso de la palabra clave return. Dicho valor puede ser almacenado en una variable o utilizado en alguna de las estructuras de control o bucles que vimos previamente.

```
function numero_al_cubo(numero) {
  return numero * numero * numero;
}

var tres_al_cubo =  numero_al_cubo(3);
alert(tres_al_cubo);
```

DOM

El DOM (Document Object Model en español Modelo de Objetos del Documento) es una API definida para representar e interactuar con cualquier documento HTML o XML. El DOM es un modelo de documento que se carga en el navegador web y que representa el documento como un árbol de nodos, en donde cada nodo representa una parte del documento (puede tratarse de un elemento, una cadena de texto o un comentario).

El DOM es una de las APIs más usadas en la Web, pues permite ejecutar código en el navegador para acceder e interactuar con cualquier nodo del documento. Estos nodos pueden crearse, moverse o modificarse. Pueden añadirse a estos nodos manejadores de eventos (event listeners en inglés) que se ejecutarán/activarán cuando ocurra el evento indicado en este manejador.

El DOM surgió a partir de la implementación de JavaScript en los navegadores. A esta primera versión también se la conoce como DOM 0 o "Legacy DOM". Hoy en día el grupo WHATWG es el encargado de actualizar el estándar de DOM.

Selectores de elementos

Los selectores api proveen métodos que hacen más fácil y rápido devolver elementos del nodo Element del DOM mediante emparejamiento de un conjunto de selectores. Esto es mucho más rápido que las técnicas anteriores, donde fuera necesario, por ejemplo usar un loop en un código JavaScript para localizar el ítem específico que quisieras encontrar.

document.querySelector()

Devuelve la primera coincidencia del (elemento) Element nodo dentro de las subramas del nodo. Si no se encuentra un nodo coincidente, se devuelve null .

document. querySelectorAll()

Devuelve un listado de nodos NodeList conteniendo todos los elementos del nodo coincidentes(Element) dentro de las subramas del nodo, o Devuelve un Listado de Nodos vacío NodeList si no se encuentran coincidencias.

Estos métodos aceptan uno o más selectores separados por comas entre cada selector para determinar qué elemento o elementos deben ser devueltos. por ejemplo para seleccionar todos los elementos (p) del párrafo en un documento donde la clase CSS sea tanto warning or note, podes hacer lo siguiente:

```
var special = document.querySelectorAll( "p.warning, p.note" );
```

También por usar query para etiquetas id. Por ejemplo:

```
var destacados = document.querySelector( "#principal,
#bienvenidos, #notas" );
```

Luego de ejecutar el código de arriba, la variable contiene el primer elemento del documento, su ID puede ser uno de los siguientes: principal, bienvenidos, o notas. Podes usar cualquier selector CSS con los métodos querySelector() y querySelectorAll().

document.getElementById()

Este método permite seleccionar una elemento del DOM usando el atributo id del mismo.

```
var mensaje = document.getElementById('mensaje');
```

document.getElementsByClassName()

Este método devuelve un array o lista de elementos que contengan la clase que se especifique como parámetro.

```
var items = document.getElementsByClassName('item');
```

document.getElementsByTagName()

Este método devuelve un array o lista de elementos cuya etiqueta html sea la que se especifique como parámetro.

```
var parrafos = document.getElementsByTagName('p');
```

Crear elementos y agregarlos al DOM

Podemos crear cualquier tipo de elemento usando el método document.createElement(). Este método recibe como parámetro el nombre de la etiqueta del elemento que deseamos crear, por ejemplo li.

Una vez creado el elemento, este no será visible al usuario hasta que lo agreguemos explícitamente al DOM, para ello, los elementos o nodos del DOM cuentan con un método llamado appendChild() que recibe como parámetro el elemento que se desee agregar.

```
<body>
    <div id="div1">El texto siguiente es dinámico</div>

    <script>
        function agregarElemento() {
            var newDiv = document.createElement('div');
            newDiv.innerText = 'Hola!';

            document.body.appendChild(newDiv);
        }
        agregarElemento();

    </script>
</body>
```

Eliminar elementos del DOM

Para eliminar elementos del DOM solo debemos obtener una referencia al elemento usando cualquiera de los métodos de selección que mencionamos previamente, y llamar a al método remove().

```
<div id="div1">Este es el div1</div>
<div id="div2">Este es el div2</div>
<div id="div3">Este es el div3</div>

<script>
     var elemento = document.getElementById('div2');
     elemento.remove();
</script>
```

innerHTML e innerText

Las propiedades de los elementos del DOM innerHTML e innerText hacen referencia al contenido de los nodos. Mientras que innerText permite insertar y manipular contenido sencillo dentro de los nodos, innerHTML nos permite incluir código HTML más complejo sin la necesidad de crear los elementos a mano.

```
<body>
  <p id="demo">Este elemnto tiene espacio extra y contiene
  <span>una etiqueta span</span></p>
  <script>

      function getHTML() {
          alert(document.getElementById('demo').innerHTML);
      }

      function getInnerText() {
          alert(document.getElementById('demo').innerText);
      }
  </script>
</body>
```

Atributos de los elementos

Podemos acceder a los atributos de cualquier elemento que hayamos seleccionado simplemente usando el nombre del atributo como propiedad del nodo, Por ejemplo

```
var miLink = document.getElementsByTagName('a')[0];
alert(miLink.href);
```

De esta forma veremos el mensaje de alerta con el contenido del atributo href de nuestro link. Del mismo modo podemos escribir la propiedad y cambiar su valor según necesitemos.

Estilos de los elementos

Los estilos CSS se pueden manipular mediante javascript accediendo a la propiedad style que tienen todos los elementos o nodos del DOM. Hay que tener en cuenta que las propiedades cuyo nombre lleva un guión en su nombre (estilo llamado kebab-case) cómo font-size se convierten al estilo camelCase, resultando en fontSize.

```
var intro = document.getElementById('intro');
intro.style.backgroundColor = '#ff00ff';
```

Eventos

Javascript permite responder a eventos en el DOM mediante el uso de funciones.

Existen 3 formas distintas de registrar gestores de eventos para un elemento del DOM:

EventTarget.addEventListener

```
myButton.addEventListener('click', function(){
  alert('Hello world');
 }, false);
```

Esta es la forma recomendada y la más moderna de las 3. Permite registrar más de un listener por evento, lo que brinda una mayor flexibilidad.

Atributo HTML

```
<button onclick="alert('Hola Mundo')">Saludar</button>
```

Esta forma es la menos recomendada porque incrementa el tamaño del HTML y dificulta su lectura, además de que resulta muy complejo incluir múltiples instrucciones.

Propiedad del elemento DOM

```
myButton.onclick = function(event){
  alert('Hello world');
};
```

Este método solo permite registrar un listener por evento.

Bibliografía:

- Eguíluz Pérez, Javier. Introducción a JavaScript. España: www.librosweb.es 2008.