Guia de JavaScript

Jorge Victorino, Miguel Barrero
Departamento de Ingeniería de Sistemas, C^{2}UC.

2020

1. Introducción.

JavaScript es un lenguaje dinámico orientado a objetos. Tiene tipos y operadores, objetos básicos y métodos. Su sintaxis viene de los lenguajes Java y C, por lo que muchas de las estructuras de esos lenguajes se aplican también a JavaScript (Mozilla, 2019).

2. Archivos JavaScript.

En el cualquier parte del cuerpo de un archivo HTML se pueden cargar archivos propios o librerías JavaScript con diferentes propósitos. Los archivos pueden ser llamados de forma local o remota.

```
1    <script src="https://threejs.org/build/three.js"/>
```

También se puede escribir JavaScript directamente sobre el HTML de la siguiente forma.

```
1    <script type="text/javascript">
2        const x = 'JavaScript!';
3    </script>
```

3. Comentarios línea y multilínea.

Los comentarios de una sola línea en JavaScript se realizan con una barra diagonal doble. Si se requiere multilínea se agrega una barra y un asterisco al inicio y al final de los comentarios.

```
1    <script type="text/javascript">
2        // Comentarios en una linea.
3        /* Comentarios para bloques de texto extensos
```

```
4               o especificaciones que lo ameriten.
5           */
6       </script>
```

4. Tipos de datos primitivos.

Javascript cuenta con los siguientes tipos de datos primitivos: números, cadenas, booleanos null y undefined.

4.1. Número

A diferencia de otros lenguajes JavaScript no diferencia entre tipos de datos enteros o reales.

```
1       <script type="text/javascript">
2           const a = 1;
3           const b = 3.4323;
4       </script>
```

4.2. Boleano

Representa valores de lógica binaria (1 ó 0). Pueden ser usados en variables tipo bandera (true o false).

```
1       <script type="text/javascript">
2           const luzEncendida = true;
3           const ejecutandoAnimacion = false;
4       </script>
```

4.3. Cadena de caracteres.

Es una secuencia ordenada de caracteres donde la longitud es arbitraria y finita. Comprende caracteres alfabéticos y numéricos. En JavaScript las cadenas de caracteres deben incluirse entre comilla doble ó comilla sencilla.

```
1       <script type="text/javascript">
2           const nombre = 'Mi nombre es miguel';
3           const personaje = 'I\'m a bumblebee';
4           const resultado = 'La suma de ' + a + ' + ' + b + ' es:
        ' + (a + b) + '.'
5           const resultado = `La suma de ${a} + ${b} es ${(a + b)
        }.`
6
7           // devuelve el tamanio de la cadena.
8           resultado.length;
```

```
9           // extrae la primera letra de la cadena.
10          resultado.charAt(0);
11          //remplaza un valor por otro dentro de la cadena.
12          resultado.replace("es", "fue");
13          // coloca la cadena en mayÃºscula sostenida.
14          resultado.toUpperCase();
15      </script>
```

4.4. Nulo e indefinido.

Undefined corresponde a una asingación automática cuando la variable no posee ningun valor y null cuando el programador indica que la variable se encuentra vacía.

```
1       <script type="text/javascript">
2           var valor;
3           console.log(valor);
4           const precio = null;
5       </script>
```

4.5. Tipado dinámico.

JavaScript es un lenguaje de tipeado dinámico. Esto significa que no hay declaración de tipos de datos para las variables.

```
1       <script type="text/javascript">
2
3           let x = 5;
4           x = 'cambio de tipo';
5       </script>
```

5. Objetos.

Los objetos en JavaScript se definen usando llaves {} que contienen colecciones de datos $[llave, valor]$ en pares.

```
1       <script type="text/javascript">
2           var o = new Object();
3           var d = {};
4           const obj = {
5               x: 5,
6               y: 'hola',
7           };
8       </script>
```

Para acceder a los datos de un objeto se debe declarar el nombre del objeto y el atributo que se requiere.

```
<script type="text/javascript">
    const n = obj.x;
    const m = obj['y'];
</script>
```

Se pueden crear objetos tipo prototipo a partir de la clausula *function*:

```
<script type="text/javascript">
    function Vehiculo(tipo, colorHSl, marca) {
        this.tipo = tipo;
        this.colorHSl = colorHSl;
        this.marca = marca;
    }
    // creando un objeto.
    var obj_tipo = new Vehiculo("Carro", 0.13, "Chevrolet")
;
    //acceder al contenido el objeto.
    var colorCarro = obj_tipo.color;
    // modificando elemento del objeto.
    obj_tipo.marca = "BMW";
```

6. Listas y matrices.

Son estructuras de datos simples que permiten almacenar conjuntos de elementos entre datos primitivos y objetos. Generalmente los elementos en una lista o matriz son homogeneos, es decir, del mismo tipo.

```
<script type="text/javascript">
    const lista = ['hola', 'estas', 45, 23];
    const numeros = [1, 32, 45, 69];
</script>
```

Se puede acceder a los elementos de la lista indicando el índice correspondiente. El primer elemento de la lista es el el 0 y el último es el tamaño de la lista - 1.

```
<script type="text/javascript">
    const z = lista[0];
    console.log(z);
</script>
```

Las matrices son un listas de listas. Ejemplo:

```
<script type="text/javascript">
    const matriz = [ [1, 2], [3, 4] ];
</script>
```

Estas estructuras de datos en JavaScript cuentan con algunos métodos que son útiles al momento de programar.

```
<script type="text/javascript">
    // creando una lista sin elementos
    var listaNumeros = [];

    // agregando elementos a la lista
    listaNumeros.push(2);
    listaNumeros.push(5,6,7);
    console.log(listaNumeros);

    // identificando la longitud de la lista
    var tmLista = listaNumeros.length;
    console.log(tmLista);

    // devuelve una cadena para cada elemento.
    listaNumeros.toString();

    // quita el ultimo item de la lista.
    listaNumeros.pop();


    // devuelve una sub-matriz.
    listaNumeros.slice(inicio, fin);
</script>
```

Para ampliar información sobre métodos de listas y matrices ingrese **aquí**.

7. Variables.

Al momento de programar en JavaScript, dependiendo de la necesidad, se pueden declarar las variables de diferente forma:

- *var*: las declaraciones de este tipo tienen un alcance global o función (alcance local). Este tipo de variables pueden ser redeclaradas y actualizadas dentro de código del programa.
- *let*: se utiliza para definir y proteger variables dentro de un contexto local ó global. Este tipo no permite que la variable sea declarada con el mismo nombre dentro del mismo contexto; pero si permite que sea actualizada o que se defina en diferentes contextos.
- *const*: se usa para asignar un valor a una variable que no puede cambiar durante la ejecución del programa. Al igual que las *let* cuentan con alcance de bloque. Este tipo de variable no se puede volver a declarar o actualizar.

```
<script type="text/javascript">
    // acceso local y global.
    var test = "Hola";
    function prueba() {
        var hola = "saludo";
        console.log(test);
    }
    console.log(hola); // error: hola no esta definida

    let libr = "three.js";
    let intensidadLuz = 4;

    if (intensidadLuz > 3) {
        let mensaje = "intensidad mayor a 3";
        console.log(mensaje); // intensidad mayor a 3
    }
    console.log(mensaje) // mensaje no esta definida.

    // error: identificador libr ya ha sido declarado.
    let libr = "OpenGL";

    libr = "OpenGL"; // actualizando la variable
    let encendida = "luz encendida";

    if (true) {
        let encendida = "luz apagada";
        console.log(encendida); // luz apagada.
    }
    console.log(encendida); // luz encendida.

</script>
```

Nota: JavaScript discrimina entre mayúsculas y minúsculas en la declaración de variables.

```
<script type="text/javascript">
    const manzanas = 4;
    const Manzanas = 5;
</script>
```

Nota: la codificación en JavaScript se sujeta al estándar de convenciones. Revise el código de convenciones **aquí**.

8. Operadores.

Los operadores numéricos de JavaScript son $+$, $-$, $*$, y $\%$ (esta última es la operación módulo o resto de la división).

```
<script type="text/javascript">

    x += 5;
    x = x + 5;
```

```
5           x = 5;
6           x -= 1;
7
8           const sum = 1 + 1; // sumar.
9           const sub = 5 - 3; // restar.
10          const div = 10 / 2; // divir.
11          const mult = 5 * 20; // multiplicar.
12
13          // concatena una cadena con otra.
14          "hola" + "mundo";
15
16          101 % 10; // 1
17          -101 % 10; // -1
18
19          const a = 2;
20          const b = a ** 2; // b = 2 * 2;
21      </script>
```

Con los operadores aritméticos pueden darse asignaciones de la siguiente forma:

```
1       <script type="text/javascript">
2           a += 5; // a = a + 5
3           a -= 5; // a = a - 5
4           a /= 5; // a = a / 5
5           a *= 5; // a = a * 5
6           a **= 5; // a = a ** 5
7           a %= 5; // a = a % 5
8       </script>
```

En JavaScript se pueden hacer comparaciones con $<$, $>$, $<=$ y $>=$ la cuales devuelven un valor lógico que podría controlar un bloque de instrucciones en instrucciones selectivas ó cíclicas.

```
1       <script type="text/javascript">
2           // devuelve verdadero
3           "gato" == "gato";
4           1 == true;
5
6           /* devuelve falso,
7           la igualdad es estricta por tipo de dato.*/
8           1 === true;
9           1 === "1";
10          //devuelve verdadero, puesto que son de diferente tipo.
11          1 !== "1";
12      </script>
```

JavaScript también cuenta con operadores lógicos para conjunción, disyunción o negación según sea el caso.

```
1       <script type="text/javascript">
2           const x = true;
```

```
        const y = true;
        const z = false;

        x && y; // true and true -> true
        x && z; // true and false -> false

        x || y; // true or true -> true
        x || z; // true or false -> true

        !x; // not true -> false
        !z; // not false -> true
    </script>
```

9. Estructuras selectivas y cíclicas.

Las estructuras de selectivas de JavaScript son similares a otros lenguajes como C y Java. Las estructura selectivas *if* y *else* pueden ser usadas de la siguiente forma:

```
    <script type="text/javascript">
        var nombre = 'Miguel';

        if (nombre == 'Juan') {
            nombre += 'Pardo';
        } else if (nombre == 'Antonio') {
            nombre += 'Guerra';
        } else {
            nombre += 'Barrero';
        }
    </script>
```

Para el caso del switch se puede usar para múltiples opciones. El valor de la opción puede ser un número o cadena. Es necesario terminar la invocación del método para cada caso con la instrucción break, de lo contrario la ejecución será para el próximo caso en el cual, esta instrucción, si se encuentre definida.

```
    <script type="text/javascript">
        switch(accion) {
            case 'cargar':
                cargarImagen();
                break;
            case 'reiniciar':
                reiniciarEscena();
                break;
            default:
                iniciar();
    </script>
```

Los ciclos estándar en JavaScript pueden construirse con *while* y *for*.

```
<script type="text/javascript">
    var c = 5;
    while (c < 10) {
        c++;
    }

    var entrada = 1;
    do {
        entrada++;
    } while (entrada < 11);

    var contador = 0;
    for (var i = 0; contador < 5; i++) {
        contador++;
        console.log(contador);
    }
</script>
```

En JavaScript se pueden hacer ciclos de diferentes formas a la convencional. Para esto es necesario que el elemento que se quiera recorrer sea iterable.

```
<script type="text/javascript">
    const numeros = [1, 2, 3, 4, 5, 6];
    let sumaLista = 0;

    for (const valor of numeros) {
      sumaLista += valor;
    }

    sumaLista = 0;
    for (var i in numeros) {
       sumaLista += i;
    }

    sumalista = 0;
    numeros.forEach((elemento) =>{
        sumaLista += elemento;
    });
</script>
```

Es posible que en alguna parte de su código usted necesite hacer esto mismo pero con objetos. Se sabe que los objetos no son iterables, sin embargo a continuación se muestra un ejemplo básico de como se podría realizar.

```
<script type="text/javascript">
    const notasEstudiantesThreeJS = {
      jorgeVictorino: 5,
      darwinMartinez: 5,
      miguelBarrero: 4
    }
    const notas = Object.values(notasEstudiantesThreeJS);

    let totalNotas = 0;
    notas.forEach((nota) => {
```

```
11              totalNotas += nota;
12          } );
13      </script>
```

Otra forma de recorrer listas incorporando sus elementos a una función. El objetivo es aplicar esa función a cada elemento de la lista. A continuación se muestra un ejemplo.

```
1       <script type="text/javascript">
2           const lista = [1, 2, 3, 4];
3
4           let mult = 0;
5           lista.forEach(function (elemento) {
6             mult *= elemento;
7           });
8       </script>
```

10. Funciones.

Las funciones en JavaScript son muy recurrentes a la hora de definir y dividir funcionalidades puntuales de un programa más general. La declaración de una función en JavaScript se da de la siguiente forma:

```
1       <script type="text/javascript">
2           function suma(x, y) {
3               var total = x + y;
4           return total;
5       }
6       </script>
```

Las funciones en JavaScript pueden tomar argumentos y retornar valores de diferente tipo. Sin embargo usted puede programar su función sin ningún valor de retorno.

El cuerpo de la función puede tener tanta cantidad de instrucciones como esta lo necesite. Dentro del conjunto de instrucciones que son declaradas declaradas, existe la posibilidad que usted necesite de algunas variables. Cada variable declarada dentro de la función estará definida dentro de un contexto local y no global.

Si usted definió una función sin argumentos, usted podría hacer el llamado de esa función añadiéndole los argumentos que necesite. Si usted quiere tomar los argumentos que pasó, deberá añadir la clausula *arguments* dentro la función para así tomar todos los valores que se encuentren en la función como una lista de valores.

```
<script type="text/javascript">
    function promedio() {
        var suma = 0;
        for (var i = 0, j = arguments.length; i < j; i++) {
            suma += arguments[i];
        }
        return suma / arguments.length;
    }
    promedio(1, 2, 3, 4, 5);
</script>
```

En JavaScript existen otros tipos de funciones que son los *callback*. Esto quiere decir que usted podrá pasar a una función otra función como argumento. Esto se da cuando se requiere encadenar una serie de funciones las cuales deben ejecutarse en orden en específico.

En JavaScript, las funciones son objetos. Debido a esto, las funciones pueden tomar funciones como argumentos, y pueden ser devueltas por otras funciones. Las funciones que hacen esto se denominan funciones de orden superior. Cualquier función que se pasa como argumento se denomina función de devolución de llamada o **callback function**.

```
<script type="text/javascript">
    function hacerTarea(asignatura, callback) {
      alert('Iniciando mi tarea de' + asignatura);
      callback();
    }
    hacerTarea('ciencias', function() {
      alert('Lista la tarea');
    });
    // una forma mas elegante.

    function hacerTaller(materia, callback) {
      alert('Iniciando mi taller de' + materia);
      callback();
    }
    function termineTaller(){
      alert('Termine mi taller');
    }
    hacerTaller('ciencias', termineTaller);
</script>
```

Puede ampliar sus conocimientos alrededor de funciones, ingrese **aquí**.

Referencias

Mozilla, M. w. d. (2019). Una re-introducción a javascript.