

JavaScript / TypeScript	2
3.1.- Introducción a JavaScript	2
3.2.- Elementos Básicos del Lenguaje	3
Tipos de datos	3
Strings	3
Propiedades y métodos de Strings	4
Numbers	5
Booleans	6
Operadores	6
Funciones	7
Condicionales	8
Arrays	8
3.3.- Objetos JavaScript.	11
3.4.- Proyecto aplicación JavaScript.	12
3.5.- Introducción a TypeScript.	12
3.2.- Elementos básicos de TypeScript.	12
3.6.- Conceptos de ES6.	13
Operadores aritméticos	13
Operadores relacionales	13
Operadores lógicos	14
sentencia if	15
switch	17
bucles	18
Funciones	21
funciones parametrizadas	22
parámetros de la función por defecto	22
Parámetros de descanso	23
Función Anónimo	24
El Constructor de funciones	24
Recursividad y JavaScript Funciones	25
Funciones Lambda	26
Función Lambda - anatomía	26
La expresión lambda	26
Cuadros de diálogo	26
El cuadro de diálogo de confirmación	27
Diálogo de consulta	28

Impresión de la página	29
Objetos	30
El Object() Constructor	31
El Método Object.create	32
El Object. assign() Función	32
Objeto desestructuración	33
Boolean	33
El objeto Boolean representa dos valores, ya sea "true" o "false" . Si se omite el parámetro de valor o es 0, -0, null, falso, NaN, indefinido, o la cadena vacía ("") , el objeto tiene un valor inicial de falsa.	33
Utilice la siguiente sintaxis para crear un boolean object .	33
var val = new Boolean(value) ;	33
toSource ()	33
toString ()	34
Propiedad booleana Prototipo	34

JavaScript / TypeScript

3.1.- Introducción a JavaScript

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.

JavaScript se utiliza principalmente del lado del cliente (es decir, se ejecuta en nuestro ordenador, no en el servidor) permitiendo crear efectos atractivos y dinámicos en las páginas web. Los navegadores modernos interpretan el código **JavaScript** integrado en las páginas web.

3.2.- Elementos Básicos del Lenguaje

Variables

En JavaScript, una variable contiene un valor. Cuando se utiliza la variable, se hace referencia a los datos que representa.

Las variables se usan para almacenar, recuperar y manipular los valores que aparecen en el código. Intenta asignar a las variables nombres descriptivos. Así, a los demás les resultará más fácil entender qué hace el código.

La primera vez que aparece una variable en un script, es su declaración. La primera mención de la variable la configura en la memoria, para que puedas hacer referencia a ella más adelante en el script. Hay que declarar las variables antes de utilizarlas. Para ello, se usa la palabra clave **var**.

JavaScript

```
// Simple Declaración.  
var count;  
// Múltiples declaraciones de variables.  
var count, amount, level;  
// Declaración de variables con inicialización.  
var count = 0, amount = 100;
```

Tipos de datos

Strings

Las cadenas son valores compuestos de texto y pueden contener letras, números, símbolos, etc

Las cadenas están contenidas dentro de un par de comillas simples (') o comillas dobles (").

```
var text = 'Hola esta es una forma de declarar una cadena';
```

```
var text2 = "Hola esta es otra forma de declarar una cadena";
```

En las cadenas también podemos incluir comillas de la siguiente forma:

```
var text = 'Hola esta es una forma de "declarar una cadena";
```

```
var text2 = "Hola esta es \'otra forma de declarar una cadena\'";
```

Propiedades y métodos de Strings

Las cadenas tienen sus propias variables y funciones incorporadas, también conocidas como propiedades y métodos. Éstos son algunos de los más comunes.

Length

La propiedad de **length** de una cadena controla cuántos caracteres tiene.

```
var text = 'Curso JavaScript'.length;
```

toLowerCase

El método **toLowerCase** de una cadena devuelve una copia de la cadena con sus letras convertidas en minúsculas. Los números, símbolos y otros caracteres no se ven afectados.

```
var text = 'Curso JavaScript se imparte hoy'.toLowerCase();
```

toUpperCase

El método **toUpperCase** de una cadena devuelve una copia de la cadena con sus letras convertidas en mayúsculas. Los números, símbolos y otros caracteres no se ven afectados.

```
var text = 'Curso JavaScript se imparte hoy'.toUpperCase();
```

trim

El método **trim** de una cadena devuelve una copia de la cadena con los caracteres de inicio y fin de los espacios en blanco eliminados.

```
var text = 'Curso JavaScript    se imparte hoy    '.trim();
```

Numbers

Los números son valores que se pueden utilizar en operaciones matemáticas. No necesita ninguna sintaxis especial para los números, simplemente se escriben directamente en JavaScript.

```
var monto = 1234;
```

Decimales y Fracciones

JavaScript no distingue entre números enteros y decimales, por lo que puede utilizarlos juntos sin tener que convertir de uno a otro.

```
var monto = 11;  
var fraccion = 12.23;
```

Booleans

Los booleanos son valores que pueden ser sólo uno de dos cosas: verdadero o falso (true / false)

```
var sabado = true;  
var viernes = false;
```

Operadores

Los operadores son los símbolos entre valores que permiten diferentes operaciones como suma, resta, multiplicación y más.

JavaScript tiene decenas de operadores, así que vamos a centrarnos en los que se usan mas a menudo.

```
var suma = 11 + 12;  
var resta = 50 + 15;  
var division = 12 / 4;  
var producto = 3 * 12;
```

Funciones

Las funciones son bloques de código que se pueden nombrar y reutilizar

```
function suma(a , b){  
    return a + b;  
}
```

```
function producto(a , b){  
    return a * b;  
}
```

```
function resta(a , b){  
    return a - b;  
}
```

```
function division(a , b){  
    return a / b;  
}
```

```
function saludo(){  
    return 'Hola a todos....';  
}
```

Condicionales

Los condicionales controlan el comportamiento en JavaScript y determinan si se pueden ejecutar o no fragmentos de código.

```
if ('Sabado'.trim().length = 6) {  
    var istrue = true;  
}else{  
    var istrue = false;  
}
```

Arrays

Las matrices son valores tipo contenedor que pueden contener otros valores. Los valores dentro de una matriz se llaman elementos.

```
var desayuno = ['cafe', 'pan'];
```

Acceso a los elementos

Para acceder a uno de los elementos dentro de una matriz, necesitará utilizar los corchetes y un número como este: myArray [3]. Los arrays JavaScript comienzan en 0, por lo que el primer elemento siempre estará dentro de [0].

```
var familia = ["Mamá", "Papá", "Hermano", "Hermana"];  
familia[2];
```


length

Para obtener el último elemento, puede utilizar paréntesis y '1' menos que la propiedad **length** de la matriz.

```
var comida = ["Pozole", "Picadillo", "Sopa", "Frijoles"];  
comida[comida.length - 1];
```

```
var actores = ["Felicia", "Nathan", "Neil"];  
actores[actores.length - 1];
```

Esto también funciona para establecer el valor de un elemento.

```
var colores = ["rojo", "verde", "azul"];  
colors[1] = "amarillo";  
|colors;
```

Propiedades y métodos

Las matrices tienen sus propias variables y funciones incorporadas, también conocidas como propiedades y métodos. Éstos son algunos de los más comunes.

Length

La propiedad **length** de una matriz almacena el número de elementos dentro de la matriz.

```
var colores = ["rojo", "verde", "azul"];  
colores.length;
```

Concat

El método concat de una matriz devuelve una nueva matriz que combina los valores de dos matrices.

```
var comida = ["tortilla chips"].concat(["salsa", "queso", "guacamole"]);
```

pop

El método pop de una matriz elimina el último elemento de la matriz y devuelve el valor de ese elemento

```
var planetas = ["Jupiter", "Saturno", "Urano", "Neptuno"].pop();
```

push

El método push de una matriz añade un elemento a la matriz y devuelve la longitud de la matriz.

```
var planetas = ["Jupiter", "Saturno", "Urano", "Neptuno"].push("Marte");
```

3.3.- Objetos JavaScript.

Los objetos son valores que pueden contener otros valores. Utilizan las llaves para nombrar los valores, que son mucho como variables.

```
var curso = {  
  name : "JavaScript",  
  inicio : "8",  
  fin : "12"  
}
```

Obteniendo Claves

Para obtener la clave de un objeto, tiene dos opciones:

Puede utilizar la notación de punto con el nombre de la clave después de un punto (.).

```
var curso = {  
  name : "JavaScript",  
  inicio : "8",  
  fin : "12"  
}  
  
curso.name;
```

O puede utilizar la notación de paréntesis con el nombre de la clave dentro de una cadena dentro de corchetes ["]

```
var curso = {  
  name : "JavaScript",  
  inicio : "8",  
  fin : "12"  
}  
  
curso["name"];
```

3.4.- Proyecto aplicación JavaScript.

3.5.- Introducción a TypeScript.

TypeScript es un lenguaje de programación de código abierto desarrollado y presentado por Microsoft hace unos tres años. Es un superconjunto de JavaScript que esencialmente añade capacidades de POO como es el tipado estático y objetos basados en clases.

TypeScript es un lenguaje de programación de código abierto con herramientas de programación orientada a objetos, muy favorable si se tienen proyectos grandes.

TypeScript es un lenguaje de programación de código abierto desarrollado por Microsoft, el cual cuenta con herramientas de programación orientada a objetos, muy favorable si se tienen proyectos grandes. Anders Hejlsberg, arquitecto principal del desarrollo del lenguaje de programación C#, es el principal participante en el desarrollo de este lenguaje.

TypeScript convierte su código en Javascript común. Es llamado también Superset de Javascript, lo que significa que si el navegador está basado en Javascript, este nunca llegará a saber que el código original fue realizado con TypeScript y ejecutará el Javascript como lenguaje original.

¿Qué es un superset?

Se trata de un lenguaje escrito sobre otro lenguaje. En este caso Typescript es eso, un lenguaje basado en el original, ofreciéndonos grandes beneficios como el descrito anteriormente, aunque existen otros beneficios. Por ejemplo, mientras otros superset de JavaScript nos alejan del código original, Typescript, por el contrario, es muy similar a Javascript y a C# gracias a que su creador posee conocimientos de ambos lenguajes.

Actualmente Angular 2, uno de los frameworks más famosos de JavaScript, está siendo desarrollado en TypeScript, para lo cual conocer este lenguaje será fundamental para entender y darle un mejor uso a la nueva versión de Angular.

3.2.- Elementos básicos de TypeScript.

Para crear un proyecto con TypeScript hay que crear el archivo **tsconfig.json**, (más adelante se explicara todo sobre ese archivo)

tsc -init

En TypeScript existe lo que se conoce como modo observador, quiere decir que estará

al pendiente de cualquier cambio que suceda con los archivos con extensiones (".ts") y automáticamente lo va a compilar a su versión de Java Script, para ello existe un comando que es el siguiente:

```
tsc archivo.ts --watch
tsc archivo.ts -w
tsc *.ts -w
```

Esto va a iniciar un servicio que va a estar escuchando cualquier cambio

3.6.- Conceptos de ES6.

Operadores aritméticos

Suponga que los valores en las variables de a y b son 10 y 5, respectivamente

```
1  var num1 = 10;
2  var num2 = 2;
3  var res = 0;
4
5  res = num1+num2;
6  console.log("Suma: " + res);
7
8  res = num1-num2;
9  console.log("Resta: " + res);
10
11 res = num1*num2;
12 console.log("Multiplicacion: " + res);
13
14 res = num1/num2;
15 console.log("Division: " + res);
16
17 num1++;
18 console.log("Valor de num1 despues del incremento: " + num1);
19
20 num2--;
21 console.log("Valor de num2 despues del decremento: " + num2);
```

Operadores relacionales

Los operadores relacionales de prueba o definir el tipo de relación entre dos entidades. Los operadores relacionales devuelven un valor booleano, es decir, verdadero / falso.

Supongamos que el valor de A es 10 y B es 20.

```
1  var num1 = 5;
2  var num2 = 9;
3
4  console.log("valor de num1: " + num1) ;
5  console.log("valor de num2: " + num2) ;
6
7  var res = num1 > num2;
8  console.log("num1 mayor que num2: " + res) ;
9
10 res = num1 < num2;
11 console.log("num1 menor que num2: " + res) ;
12
13 res = num1 >= num2;
14 console.log("num1 mayor o igual que num2: " + res) ;
15
16 res = num1 <= num2;
17 console.log("num1 menor o igual que num2: " + res) ;
18
19 res = num1 == num2;
20 console.log("num1 es igual a num2: " + res) ;
21
22 res = num1 != num2;
23 console.log("num1 es igual a num2: " + res) ;|
```

Operadores lógicos

Los operadores lógicos se utilizan para combinar dos o más condiciones. Los operadores lógicos, también, devuelven un valor booleano. Supongamos que el valor de la variable A es 10 y B es 20.

```

1  var promedio = 20;
2  var porcentaje = 90;
3
4  console.log("Valor de promedio: " + avg + " ,valor de porcentaje " + percentage) ;
5
6  var res = ((avg > 50) && (percentage > 80) );
7  console.log("(avg>50) &&(percentage>80): ", res);
8
9  var res = ((avg > 50) || (percentage > 80) );
10 console.log("(avg>50) ||(percentage>80): ", res);
11
12 var res = !((avg > 50) && (percentage > 80) );
13 console.log("!((avg>50) &&(percentage>80)): ", res);

```

sentencia if

El 'if ... else' constructo evalúa una condición antes de ejecutar un bloque de código. Ésta es la sintaxis.

```

if(boolean_expression) {
    // statement(s) will execute if the Boolean expression is true
}

```

Si la expresión booleana es verdadera, entonces el bloque de código dentro de la instrucción if será ejecutado. Si la expresión booleana es falsa, entonces el primer conjunto de código después del final de la instrucción if (after the closing curly brace) será ejecutado.

```

1  var num = 5
2  if (num>0) {
3      console.log("El numero es positivo")
4  }

```

Un caso puede ser seguido por un bloque else opcional. El bloque else se ejecutará si la expresión booleana probado por si se evalúa como falsa.

Ésta es la sintaxis.

```

if(boolean_expression) {
    // statement(s) will execute if the Boolean expression is true
}

```

```
} else {  
  // statement(s) will execute if the Boolean expression is false  
}
```

```
1  var num = 12;  
2  
3  if (num > 10) {  
4    console.log("Es mayor a 10") ;  
5  } else {  
6    console.log("Es menor a 10") ;  
7  }
```

La otra cosa ... si la escalera es útil para probar múltiples condiciones. Ésta es la sintaxis de la misma.

```
if (boolean_expression1) {  
  //statements if the expression1 evaluates to true  
} else if (boolean_expression2) {  
  //statements if the expression2 evaluates to true  
} else {  
  //statements if both expression1 and expression2 result to false  
}
```

Al utilizar si ... else, hay algunos puntos a tener en cuenta.

- Un caso puede tener cero o de nadie más y que debe venir después de cualquier otra persona si de.
- Un caso puede tener de cero a muchos de else if y deben venir antes de la cosa.
- Una vez que una persona si tiene éxito, ninguno de los restantes más si es o de lo contrario se pondrá a prueba de.

```
1  var num = 2  
2  if(num > 0) {  
3    console.log(num + "Es positivo")  
4  } else if(num < 0) {  
5    console.log(num + "Es negativo")  
6  } else {  
7    console.log(num + " No es positivo ni negativo")  
8  }
```


switch

La sentencia switch evalúa una expresión, coincide con el valor de la expresión a una cláusula de caso y ejecuta las instrucciones asociadas con ese caso.

Ésta es la sintaxis.

```
switch(variable_expression) {  
    case constant_expr1: {  
        //statements;  
        break;  
    }  
    case constant_expr2: {  
        //statements;  
        break;  
    }  
    default: {  
        //statements;  
        break;  
    }  
}
```

El valor de la variable_expression se prueba contra todos los casos en el interruptor. Si la variable coincide con uno de los casos, se ejecuta el bloque de código correspondiente. Si ninguna expresión case coincide con el valor de la variable_expression, el código dentro del bloque por defecto está asociado.

Las siguientes reglas se aplican a una sentencia switch -

- Puede haber cualquier número de declaraciones de casos dentro de un interruptor.
- Las declaraciones de casos sólo pueden incluir constantes. No puede ser una variable o una expresión.
- El tipo de datos de la variable_expression y la expresión de la constante deben coincidir.
- A menos que poner una pausa después de cada bloque de código, la ejecución desemboca en el siguiente bloque.
- La expresión case debe ser único.
- El bloque por defecto es opcional.

```

1  var grado = "A";
2  switch(grado) {
3      case "A" : {
4          console.log("Excelente") ;
5          break;
6      }
7      case "B" : {
8          console.log("Bueno") ;
9          break;
10     }
11     case "C" : {
12         console.log("Regular") ;
13         break;
14     }
15     case "D" : {
16         console.log("Reprobado") ;
17         break;
18     }
19     default: {
20         console.log("Opcion invalida") ;
21         break;
22     }
23 }

```

bucles

A veces, ciertas instrucciones requieren la ejecución repetida. Los bucles son una forma ideal de hacer lo mismo. Un bucle representa un conjunto de instrucciones que debe repetirse. En el contexto de un bucle, una repetición que se denomina como una iteration .

La siguiente figura muestra la clasificación de los bucles.

BUCLE DEFINIDA

Un bucle cuyo número de iteraciones son definido / fijo que se denomina como un definite loop . El 'bucle' es una implementación de un definite loop .

```

for (initial_count_value; termination-condition; step) {
    //statements
}

```

El bucle tiene tres partes: el inicializador (i = num) , la condición (i >= 1) y la expresión final (i--) .

El programa calcula el factorial del número 5 y muestra el mismo. El bucle genera la secuencia de números de 5 a 1, calculando el producto de los números en cada iteración.

Múltiples misiones y expresiones finales se pueden combinar en un bucle, mediante el operador de coma (,) . Por ejemplo, el siguiente bucle for imprime los primeros ocho números de Fibonacci -

```
1 for(let temp, i = 0, j = 1; j<30; temp = i, i = j, j = i + temp)
2 console.log(j) ;
```

EL BUCLE FOR ... IN

El bucle for ... in se utiliza para recorrer las propiedades del objeto.

A continuación se muestra la sintaxis de 'for ... in' bucle.

```
for (variablename in object) {
    statement or block to execute
}
```

En cada iteración, una propiedad del objeto se asigna al nombre de la variable y este bucle continúa hasta que todas las propiedades del objeto se agotan.

```
1 var obj = {a:1, b:2, c:3};
2 for (var prop in obj) {
3     console.log(obj[prop]) ;
4 }
```

BUCLE INDEFINIDO

Un bucle indefinido se utiliza cuando el número de iteraciones en un bucle es indeterminado o desconocido.

bucles indefinidos pueden ser implementados utilizando

El bucle while ejecuta las instrucciones cada vez que la condición especificada se evalúa como verdadera.

A continuación se presenta la sintaxis para el bucle while.

```
while (expression) {  
    Statement(s) to be executed if expression is true  
}
```

```
1  var num = 5;  
2  var factorial = 1;  
3  
4  while(num >= 1) {  
5      factorial = factorial * num;  
6      num--;  
7  }  
8  console.log("El factorial es: "+factorial) ;
```

EL BUCLE DO ... WHILE

El do...while bucle es similar al bucle while excepto que el do...while bucle no evalúa la condición por primera vez se ejecuta el bucle. Sin embargo, la condición se evalúa para las iteraciones posteriores. En otras palabras, el bloque de código se ejecutará al menos una vez en un do...while loop.

A continuación se presenta la sintaxis para el bucle while.

```
do {  
    Statement(s) to be executed;  
}  
while (expression) ;
```

```
1  var n = 10;  
2  do {  
3      console.log(n) ;  
4      n--;  
5  }  
6  while(n>=0) ; |
```

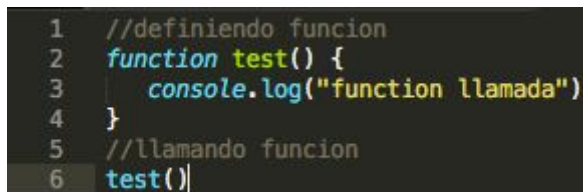
Funciones

Son los bloques de construcción de código legible, fácil de mantener, y reutilizable. Las funciones se definen utilizando la palabra clave función. Ésta es la sintaxis para la definición de una función estándar.

```
function function_name() {  
    // function body  
}
```

Para forzar la ejecución de la función, que debe ser llamado. Esto se conoce como invocación de la función. Ésta es la sintaxis para invocar una función.

```
function_name()
```



```
1 //definiendo funcion  
2 function test() {  
3     console.log("function llamada")  
4 }  
5 //llamando funcion  
6 test()
```

Las funciones también pueden devolver el valor junto con el control, de vuelta a la persona que llama. Tales funciones se denominan como devolver funciones.

Ésta es la sintaxis para la función que devuelve.

```
function function_name() {  
    //statements  
    return value;  
}
```

- Una función que devuelve debe finalizar con una instrucción de retorno.
- Una función puede devolver un valor en la mayoría. En otras palabras, no puede haber sólo una instrucción de retorno por función.
- La instrucción de retorno debe ser la última sentencia de la función.

El siguiente fragmento de código es un ejemplo de una función que devuelve

```

1  function retStr() {
2      return "Hola Mundo!!!"
3  }
4  var val = retStr()
5  console.log(val)

```

funciones parametrizadas

Los parámetros son un mecanismo para pasar valores a las funciones. Parámetros forman parte de la firma de la función. Los valores de los parámetros se pasan a la función durante su invocación. A menos que se especifique de forma explícita, el número de valores pasados a una función debe coincidir con el número de parámetros definidos.

A continuación se presenta la sintaxis de definición de una función parametrizada.

```

function func_name( param1,param2 ,.....paramN) {
    .....
    .....
}

```

Example – Parameterized Function

El ejemplo define un complemento función que acepta dos parámetros n1 y n2 e imprime su suma. Los valores de los parámetros se pasan a la función cuando se invoca.

```

1  function add( n1,n2) {
2      var sum = n1 + n2
3      console.log("La suma de los valores introducidos "+sum)
4  }
5  add(12,13) |

```

parámetros de la función por defecto

En ES6, una función permite que los parámetros pueden inicializar con valores por defecto, si no hay valores se transmiten a él o ella no está definido. La misma se ilustra en el siguiente código.

```

1  function add(a, b = 1) {
2    return a+b;
3  }
4  console.log(add(4) )

```

La función anterior, establece el valor de b a 1 por defecto. La función siempre tendrá en cuenta el parámetro b para soportar el valor de 1 a menos que un valor ha sido aprobado de forma explícita. El resultado siguiente se muestra en la ejecución exitosa del código de seguridad

El valor predeterminado del parámetro se sobrescribirá si la función pasa a un valor de forma explícita

```

1  function add(a, b = 1) {
2    return a + b;
3  }
4  console.log(add(4,2) )

```

Parámetros de descanso

parámetros de descanso son similares a los argumentos variables en Java. parámetros de descanso no restringe el número de valores que puede pasar a una función. Sin embargo, los valores pasados deben ser todos del mismo tipo. En otras palabras, los parámetros de descanso actúan como marcadores de posición para múltiples argumentos del mismo tipo.

Para declarar un parámetro de reposo, el nombre del parámetro se prefija con tres periodos, conocido como el operador de difusión. El siguiente ejemplo ilustra la misma.

```

1  function fun1(...params) {
2    console.log(params.length) ;
3  }
4  fun1() ;
5  fun1(5) ;
6  fun1(5, 6, 7) ; |

```

Función Anónimo

Las funciones que no están unidas a un identificador (function name) se denominan como funciones anónimas. Estas funciones se declaran de forma dinámica en tiempo de ejecución. Las funciones anónimas pueden aceptar entradas y salidas de volver, al igual que lo hacen las funciones estándar. Una función anónima por lo general no es accesible después de su creación inicial.

Las variables se pueden asignar una función anónima. Tal expresión se llama una function expression .

Ésta es la sintaxis para la función anónima.

```
var res = function( [arguments] ) { ... }
```

```
1  var f = function() { return "hello"}
2  console.log(f() ) |
```

```
1  var func = function(x,y) { return x*y };
2  function product() {
3      var result;
4      result = func(10,20) ;
5      console.log("The product : "+result)
6  }
7  product()
```

El Constructor de funciones

La declaración aunque no es la única manera de definir una nueva función; puede definir su función de forma dinámica utilizando Function() constructor junto con el nuevo operador.

Ésta es la sintaxis para crear una función utilizando Function() constructor junto con el nuevo operador.


```
var variablename = new Function(Arg1, Arg2..., "Function Body") ;
```

La Function() constructor espera que cualquier número de argumentos de cadena. El último argumento es el cuerpo de la función - que puede contener declaraciones arbitrarias JavaScript, separadas entre sí por punto y coma.

La Function() constructor no se pasa ningún argumento que especifica un nombre para la función que crea.

```
1 var func = new Function("x", "y" , "return x*y;") ;
2 function product() {
3     var result;
4     result = func(10,20) ;
5     console.log("The product : "+result)
6 }
7 product()
```

En el ejemplo anterior, la Function() constructor se utiliza para definir una función anónima. La función acepta dos parámetros y devuelve su producto.

Recursividad y JavaScript Funciones

La recursividad es una técnica para la iteración en una operación por tener una función llama a sí mismo varias veces hasta que se llega a un resultado. La recursividad se aplica mejor cuando se necesita llamar la misma función varias veces con diferentes parámetros dentro de un bucle.

```
1 ▼ function factorial(num) {
2     if(num<=0) {
3         return 1;
4     } else {
5         return (num * factorial(num-1) )
6     }
7 }
8 console.log(factorial(6) )
```

Funciones Lambda

Lambda se refiere a funciones anónimas en la programación. Las funciones lambda son un mecanismo concisa de representar funciones anónimas. Estas funciones también se llaman como Arrow functions .

Función Lambda - anatomía

Hay 3 partes a una función lambda -

- Parameters - Una función puede tener opcionalmente parámetros.
- La fat arrow notation/lambda notation (\Rightarrow) : Se denomina también como el va a operador.
- Statements - Representa el conjunto de instrucciones de la función.

La expresión lambda

Es una expresión función anónima que apunta a una sola línea de código. Ésta es la sintaxis para el mismo.

```
[[param1, parma2,...param n] ] =>statement;
```

```
1  var test = (x) =>10+x
2
3  console.log(test(10) )
```

Cuadros de diálogo

JavaScript soporta tres tipos importantes de los cuadros de diálogo. Estos cuadros de diálogo se pueden utilizar para subir y alerta, o para obtener la confirmación de cualquier entrada o tener un tipo de entrada de los usuarios. Aquí vamos a discutir cada cuadro de diálogo uno a uno.

El cuadro de diálogo de alerta

Un cuadro de diálogo de alerta se utiliza sobre todo para enviar un mensaje de advertencia a los usuarios. Por ejemplo, si un campo de entrada requiere que se introduzca un texto, pero el usuario no proporciona ninguna entrada, a continuación, como parte de la validación, se puede utilizar un cuadro de alerta para enviar un mensaje de advertencia.

Sin embargo, un cuadro de alerta todavía se puede utilizar para los mensajes más amigables. Cuadro de alerta proporciona sólo un botón "OK" para seleccionar y proceder.

```
1 <html>
2 <head>
3   <script type = "text/javascript">
4     function Warn() {
5       alert ("Mensaje de advertencia!") ;
6       document.write ("Mensaje de advertencia!") ;
7     }
8   </script>
9 </head>
10
11 <body>
12   <p>Click: </p>
13   <form>
14     <input type = "button" value = "Click Me" onclick = "Warn() ;" />
15   </form>
16 </body>
17 </html>
```

El cuadro de diálogo de confirmación

Un cuadro de diálogo de confirmación se utiliza principalmente para tomar el consentimiento del usuario en cualquier opción. Se muestra un cuadro de diálogo con dos botones: Aceptar y Cancelar.

Si el usuario hace clic en el botón Aceptar, el método de la ventana `confirm()` devolverá `true`. Si el usuario hace clic en el botón Cancelar, y `confirm()` devuelve `false`. Puede utilizar un cuadro de diálogo de confirmación de la siguiente manera.

```

1  <html>
2  <head>
3      <script type = "text/javascript">
4          function getConfirmation() {
5              var retVal = confirm("Desea continuar ?") ;
6
7              if( retVal == true ) {
8                  document.write ("El usuario quiere continuar!") ;
9                  return true;
10             } else {
11                 Document.write ("El usuario no quiere continuar!") ;
12                 return false;
13             }
14         }
15     </script>
16 </head>
17
18 <body>
19     <p>Click: </p>
20     <form>
21         <input type = "button" value = "Click Me" onclick = "getConfirmation()" ;" />
22     </form>
23 </body>
24 </html>

```

Diálogo de consulta

El cuadro de diálogo del sistema es muy útil cuando se desea para que aparezca un cuadro de texto para obtener una entrada del usuario. Por lo tanto, le permite interactuar con el usuario. El usuario tiene que llenar el campo y haga clic en OK.

Este cuadro de diálogo se muestra utilizando un método llamado `prompt()` que toma dos parámetros: (i) una etiqueta que desea mostrar en el cuadro de texto y (ii) una secuencia predeterminada para mostrar en el cuadro de texto.

Este cuadro de diálogo tiene dos botones: Aceptar y Cancelar. Si el usuario hace clic en el botón Aceptar, el método de la ventana `prompt()` devolverá el valor introducido en el cuadro de texto. Si el usuario hace clic en el botón Cancelar, el método de la ventana `prompt()` devuelve un valor nulo.

```

1  <html>
2    <head>
3      <script type = "text/javascript">
4        function getValue() {
5          var retVal = prompt("Digite su nombre: ", "tu nombre aqui") ;
6          document.write("Has ingresado: " + retVal) ;
7        }
8      </script>
9    </head>
10
11   <body>
12     <p>Click: </p>
13     <form>
14       <input type = "button" value = "Click Me" onclick = "getValue() ;" />
15     </form>
16   </body>
17 </html>

```

Impresión de la página

Muchas veces se desea colocar un botón en su página web para imprimir el contenido de esa página web a través de una impresora real. JavaScript ayuda a implementar esta funcionalidad utilizando la función de impresión del objeto de la ventana.

La función de impresión JavaScript `window.print()` imprime la página web actual cuando se ejecuta. Puede llamar directamente a esta función mediante el evento `onclick` como se muestra en el siguiente ejemplo.

```

1  <html>
2    <body>
3      <form>
4        <input type = "button" value = "Imprimir" onclick = "window.print()" />
5      </form>
6    </body>
7  </html>

```

Objetos

JavaScript apoya la extensión tipos de datos. objetos JavaScript son una gran manera de definir los tipos de datos personalizados.

Un object es una instancia que contiene un conjunto de pares de valores clave. A diferencia de los tipos de datos primitivos, los objetos pueden representar múltiples o complejas valores y pueden cambiar a lo largo de su tiempo de vida. Los valores pueden ser valores o funciones escalares o incluso variedad de otros objetos.

Las variaciones sintácticas para la definición de un objeto se discute más.

inicializadores de objeto

Al igual que los tipos primitivos, los objetos tienen una sintaxis literal: curly bracesv ({and}) . Siguiendo es la sintaxis para la definición de un objeto.

```
var identifier = {  
  Key1:value, Key2: function () {  
    //functions  
  },  
  Key3: ["content1", "content2"]  
}
```

El contenido de un objeto se denominan properties (or members) , y las propiedades consisten en un name (or key) y value . Los nombres de propiedad deben ser cadenas o símbolos, y los valores pueden ser de cualquier tipo (including other objects) .

Al igual que todas las variables JavaScript, tanto el nombre del objeto (which could be a normal variable) y el nombre de la propiedad entre mayúsculas y minúsculas. Accede a las propiedades de un objeto con un simple punto-notación.

Lo que sigue es la sintaxis para acceder a Propiedades del objeto.

```

1  var person = {
2      firstname:"Tom",
3      lastname:"Hanks",
4      func:function() {return "Hello!!"},
5  };
6  //access the object values
7  console.log(person.firstname)
8  console.log(person.lastname)
9  console.log(person.func() )

```

El Object() Constructor

JavaScript proporciona una función especial llamada constructor Object() para construir el objeto. El nuevo operador se utiliza para crear una instancia de un objeto. Para crear un objeto, el nuevo operador es seguido por el método de constructor.

Siguiente es la sintaxis para la definición de un objeto.

```

var obj_name = new Object() ;
obj_name.property = value;
OR
obj_name["key"] = value

```

Lo que sigue es la sintaxis para acceder a una propiedad.

```

Object_name.property_key
OR
Object_name["property_key"]

```

```

1  var myCar = new Object() ;
2  myCar.make = "Ford"; |
3  myCar.model = "Mustang";
4  myCar.year = 1987;
5
6  console.log(myCar["make"])
7  console.log(myCar["model"])
8  console.log(myCar["year"])

```

propiedades no asignados de un objeto no están definidos.

```

1  var myCar = new Object() ;
2  myCar.make = "Ford";
3  console.log(myCar["model"])

```

El Método Object.create

Los objetos también pueden ser creados usando el `Object.create()` método. Se le permite crear el prototipo para el objeto que desee, sin tener que definir una función constructora.

```
1  var roles = {  
2    type: "Admin" ,  
3    displayType : function() {  
4      console.log(this.type) ;  
5    }  
6  }  
7  var super_role = Object.create(roles) ;  
8  super_role.displayType() ;  
9  
10 var guest_role = Object.create(roles) ;  
11 guest_role.type = "Guest";  
12 guest_role.displayType() ;
```

El Object. assign() Función

El `Object.assign()` método se utiliza para copiar los valores de todas las propiedades propias enumerables de uno o más objetos de origen a un objeto de destino. Se devolverá el objeto de destino.

Ésta es la sintaxis para el mismo.

```
1  var det = { name:"Tom", ID:"E1001" };  
2  var copy = Object.assign({}, det) ;  
3  console.log(copy) ;  
4  for (let val in copy) {  
5    console.log(copy[val])  
6  }
```



```

1  var o1 = { a: 10 };
2  var o2 = { b: 20 };
3  var o3 = { c: 30 };
4  var obj = Object.assign(o1, o2, o3) ;
5  console.log(obj) ; |

```

Objeto desestructuración

El término destructuring se refiere a romper la estructura de una entidad. La sintaxis asignación desestructuración en JavaScript hace que sea posible extraer datos de matrices u objetos en variables distintas. La misma se ilustra en el siguiente ejemplo.

```

1  var emp = { name: 'John', Id: 3 }
2  var {name, Id} = emp
3  console.log(name)
4  console.log(Id)

```

Boolean

El objeto Boolean representa dos valores, ya sea "true" o "false" . Si se omite el parámetro de valor o es 0, -0, null, falso, NaN, indefinido, o la cadena vacía ("") , el objeto tiene un valor inicial de falsa.

Utilice la siguiente sintaxis para crear un boolean object .

```
var val = new Boolean(value) ;
```

toSource ()

Javascript boolean toSource() método devuelve una cadena que representa el código fuente del objeto.

Note - Este método no es compatible con todos los navegadores.

Ésta es la sintaxis para el mismo.

`boolean.toSource()`

```
1 <html>
2   <head>
3     <title>JavaScript toSource() Method</title>
4   </head>
5
6   <body>
7     <script type = "text/javascript">
8       function book(title, publisher, price) {
9         this.title = title;
10        this.publisher = publisher;
11        this.price = price;
12      }
13      var newBook = new book("Adulterio","Paulo Cohelo",250) ;
14      document.write("el nuevo libro es : "+ newBook.toSource() );
15    </script>
16  </body>
17
18 </html>
```

`toString()`

Este método devuelve una cadena de cualquiera de "true" o "false" dependiendo del valor del objeto.

Sintaxis

Ésta es la sintaxis para el mismo.

`boolean.toString()`

```
1 <html>
2   <head>
3     <title>JavaScript toString() Method</title>
4   </head>
5
6   <body>
7     <script type = "text/javascript">
8       var flag = new Boolean(false) ;
9       document.write( "flag es: " + flag.toString() );
10    </script>
11  </body>
12
13 </html>
```

Propiedad booleana Prototipo

El prototype la propiedad le permite añadir propiedades y métodos de cualquier objeto (Number, Boolean, String and Date, etc.) .

Note - El prototipo es una propiedad global que está disponible con casi todos los objetos.

Utilice la siguiente sintaxis para crear un prototipo de Boole.

```
object.prototype.name = value
```

```
1 <html>
2
3 <head>
4   <title>User-defined objects</title>
5   <script type = "text/javascript">
6     function book(title, author) {
7       this.title = title;
8       this.author = author;
9     }
10  </script>
11 </head>
12
13 <body>
14   <script type = "text/javascript">
15     var myBook = new book("Adulterio", "Paulo Coelho") ;
16     book.prototype.price = null;
17     myBook.price = 100;
18     document.write("Titulo : " + myBook.title + "<br>") ;
19     document.write("Autor: " + myBook.author + "<br>") ;
20     document.write("Precio: " + myBook.price + "<br>") ;
21   </script>
22 </body>
23
24 </html>
```