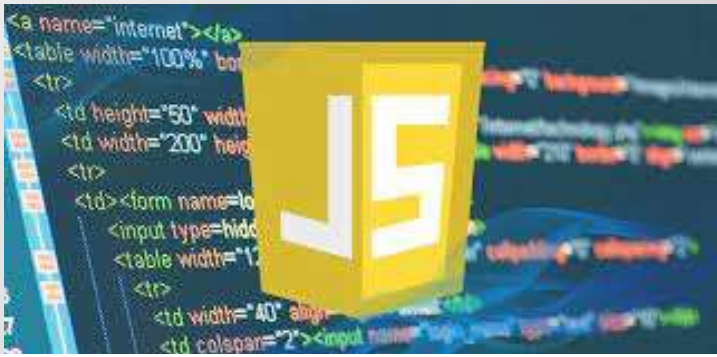
JavaScript

Parte II

Diseño de Sitios Web



Funciones anónimas

- Otra manera de declarar una función es usando funciones anónimas. Las funciones anónimas son funciones sin un nombre o identificador. Debido a esto, se pueden pasar a otras funciones o asignar a variables. Cuando una función anónima se asigna a una variable, el nombre de la variable es el que usamos para llamar a la función, tal como hacemos en el siguiente ejemplo.

```
var mifuncion = function(valor) {
valor = valor * 2;
return valor;
};
var total = 2;
for (var f = 0; f < 10; f++) {
total = mifuncion(total); }
alert("El total es " + total); // "El total es 2048"
```

Funciones Anónimas

- Las funciones anónimas se pueden ejecutar al instante agregando paréntesis al final de su declaración. Esto es útil cuando queremos asignar el resultado de una operación compleja a una variable. La función procesa la operación y devuelve el resultado. En este caso, no es la función lo que se asigna a la variable, sino el valor que devuelve la misma.

```
var mivalor = function(valor) {
valor = valor * 2;
return valor;
}(35);
alert("El valor es " + mivalor); // "El valor es 70"
```

Funciones Estándar

- Son predefinidas por el lenguaje
- Las funciones estándar son funciones globales que podemos llamar desde cualquier parte del código; solo tenemos que llamarlas como lo hacemos con cualquier otra función con los valores que queremos procesar entre paréntesis.

Funciones Estándar

- **isNaN(**valor**)**—Esta función devuelve true (verdadero) si el valor entre paréntesis no esun número.
- **parseInt(**valor**)**—Esta función convierte una cadena de caracteres con un número en un número entero que podemos procesar en operaciones aritméticas.
- **parseFloat(**valor**)**—Esta función convierte una cadena de caracteres con un número en un número decimal que podemos procesar en operaciones aritméticas.
- **encodeURIComponent(**valor**)**—Esta función codifica una cadena de caracteres. Se utiliza para codificar los caracteres de un texto que puede crear problemas cuando se inserta en una URL.
- **decodeURIComponent(**valor**)**—Esta función decodifica una cadena de caracteres.

Ejemplo funciones estándar

```
var mivalor = "Hola";
if (isNaN(mivalor)) {
alert(mivalor + " no es un número");
} else {
alert(mivalor + " es un número");
}
var nombre = "Juan Perez";
var codificado = encodeURIComponent(nombre);
var miURL = "http://www.ejemplo.com/contacto.html?nombre=" +
codificado;
alert(miURL);
```

Clases

- Javascript no es un lenguaje de programación orientado a objetos propiamente dicho. Ya queno cumple con alguna de las restricciones de la OO.
- Se pueden crear nuevos objetos y utilizar objetos que ya creados.
- Recordamos que un objeto se crea a partir de una clase
- La clase es la definición de las características y funcionalidades de un objeto.
- Con las clases no se trabaja

Objetos en Javascript

- Los objetos son estructuras de información capaces de contener variables (llamadas *propiedades*), así como funciones (llamadas *métodos*). Debido a que los objetos almacenan valores junto con funciones, son como programas independientes que se comunican entre sí para realizar tareas comunes.
- Permiten organizar nuestro código de esta manera, podemos crear unidades de procesamiento independientes capaces de realizar tareas y que cuentan con toda la información que necesitan para hacerlo.

Declaración de Objetos

- Existen diferentes maneras de declarar objetos en JavaScript:
  - Notación literal. El objeto se declara como cualquier otra variable usando la palabra clave var, y las propiedades y métodos que definen el objeto se declaran entre llaves usando dos puntos después del nombre y una coma para separar cada declaración.
  - Definición de objetos a partir de constructores
  - Definición de objetos usando el operador new
- Es posible asi mismo definir en un objeto otro objeto, acceder a sus propiedades, métodos.
- Es posible seguir las reglas de la OO pero sólo a través de las practicas del programador

Notación literal

- Es una de las mas usadas, es mas pensado como una variable del tipo registro, se puede acceder a las propiedad y métodos el objeto
- No es posible crear dos instancias iguales de un objeto, ya que no construye el esqueleto general de la clase
- Sintaxis general

```
var nombre_objeto = { // misma idea de un registro o variable
propiedad1: valor,
Propiedad2: valor};
nombre_objeto.propiedad1= valor1;
```

Ejemplo Objetos: Notación literal

```
var miobjeto = { // misma idea de un registro o variable
nombre: "Juan",
edad: 30   };
var mensaje = "Mi nombre es " + miobjeto.nombre + "\r\n";
mensaje += "Tengo " + miobjeto["edad"] + " años";
alert(mensaje);
//accediendo con variables a la propiedad
var nombrePropiedad = "nombre";
alert(miobjeto[nombrePropiedad]);
//asignando valores a las propiedades
miobjeto.nombre = "Martín";
miobjeto.trabajo = "Programador";
```

Ejemplo Objetos: Notación literal

```
var miobjeto = {
nombre: "Juan",
edad: 30,
motocicleta: { // objetos dentro de objetos
modelo: "Susuki",
fecha: 1981}
};
alert(miobjeto.nombre + " tiene una " +
miobjeto.motocicleta.modelo);
```

Métodos: notación literal

- Los métodos tienen la misma sintaxis que las propiedades: requieren dos puntos después del nombre y una coma para separar cada declaración, pero en lugar de valores, debemos asignarles funciones anónimas.
- Sintaxis

```
var nombre_objeto={
propiedad1=valor,
nombre_metodo: function() {
// código de la función
},
};
```

Ejemplo Métodos: Notación literal

```
var miobjeto = {
nombre: "Juan",
edad: 30,
mostrardatos: function() {
var mensaje = "Nombre: " + miobjeto.nombre + "\r\n";
mensaje += "Edad: " + miobjeto.edad;
alert(mensaje);
},
cambiarnombre: function(nombrenuevo) {
miobjeto.nombre = nombrenuevo;
}
};
```

Ejemplo métodos: invocación

```
miobjeto.mostrardatos(); // "Nombre: Juan Edad: 30"
miobjeto.cambiarnombre("José");
miobjeto.mostrardatos(); // "Nombre: José Edad: 30«
```

En la OO se utiliza la palabra clave this cuando se hace referencia a una propiedad o método de la misma clase, reemplazando el nombre del objeto si se encuentra dentro de la misma. Es posible en javascript utilizar el this como «comodin» que representa el objeto mismo.

Ejemplo: Uso de «This»

```
var miobjeto = {
nombre: “Juan”,
edad: 30,
cambiarnombre: function(nombrenuevo) {
var nombreviejo = this.nombre;
this.nombre = nombrenuevo;
return nombreviejo;
}
};
var anterior = miobjeto.cambiarnombre(“José”);
alert(“El nombre anterior era: “ + anterior); // “Juan”
```

Definición de objetos usando Constructores

- Usando notación literal podemos crear objetos individuales, pero si queremos crear copias de estos objetos con las mismas propiedades y métodos, tenemos que usar constructores.
- Un constructor es una función anónima que define un nuevo objeto y lo devuelve, creando copias del objeto (también llamadas *instancias*), cada una con sus propias propiedades, métodos y valores.

Ejemplo Objetos: usando constructores

```
var constructor = function() {
var obj = {
nombre: “Juan”,
edad: 30,
mostrarnombre: function() {
alert(this.nombre);
}    // demás métodos};
return obj;
};
var empleado = constructor();
empleado.mostrarnombre(); // “Juan”
```

Ejemplo Objetos: usando Constructores

```
var constructor = function(nombreinicial) {
var obj = {
nombre: nombreinicial,
edad: 30,
mostrarnombre: function() {
alert(this.nombre);
}      };
return obj;
};
var empleado = constructor(“Juan”);
empleado.mostrarnombre(); // “Juan”
var empleado2 = constructor(“Roberto”);
```

Definición de objetos con el operador new

- Con la notación literal y los constructores tenemos todo lo que necesitamos para crear objetos individuales o múltiples objetos basados en una misma definición, pero para ser coherente con otros lenguajes de programación orientada a objetos, JavaScript ofrece una tercera alternativa.
- Es una clase especial de constructor que trabaja con un operador llamado new (nuevo). El objeto se define mediante una función y luego se llama con el operador new para crear un objeto a partir de esa definición.

Operador new

- **Instanciación de objetos**: Instanciar un objeto es la acción de crear un “ejemplar” de una clase. Para acceder a sus propiedades debo utilizar el this, asi mismo para los métodos.
- Para crear un objeto a partir de una clase se utiliza la instrucción new:
  - var miObjeto = new miClase()
- Propiedades y métodos de los objetos
- **miObjeto.miPropiedad**
  - **miObjeto.miMetodo(param1, param2, ...)**
  - **miObjeto.miMetodo()**

Ejemplo Objetos: operador new

```
function MiObjeto() {
this.nombre = “Juan”;
this.edad = 30;
this.mostrarnombre = function(){
alert(this.nombre);
};
this.cambiarnombre = function(nombrenuevo){
this.nombre = nombrenuevo;
};
}
var empleado = new MiObjeto();
empleado.mostrarnombre(); // “Juan”
```

Ejemplo uso de new

```
function MiObjeto(nombreinicial, edadinicial){
this.nombre = nombreinicial; // si uso new debo usar this
this.edad = edadinicial;
this.mostrarnombre = function(){
alert(this.nombre);
};
this.cambiarnombre = function(nombrenuevo){
this.nombre = nombrenuevo;
};
}
var empleado = new MiObjeto("Roberto", 55);
empleado.mostrarnombre(); // "Roberto"
```

Herencia

- Una característica importante de los objetos es que podemos crearlos desde otros objetos.
- Cuando los objetos se crean a partir de otros objetos, pueden heredar sus propiedades y métodos, y también agregar los suyos propios. La herencia en JavaScript (cómo los objetos obtienen las mismas propiedades y métodos de otros objetos) se logra a través de prototipos.
- Un objeto no hereda las propiedades y los métodos de otro objeto directamente; lo hace desde el prototipo del objeto. Este prototipo queda encadenado a todos los objetos definidos a partir de él.
- Para esto hacemos uso del objeto de javascript Object

Ejemplo: Herencia

```
var miobjeto = {
nombre: “Juan”,
edad: 30,
mostrarnombre: function(){
alert(this.nombre);
},
cambiarnombre: function(nombrenuevo){
this.nombre = nombrenuevo;
}
};
```

Ejemplo Herencia

var empleado = Object.create(miobjeto);

empleado.cambiarnombre('Roberto');

empleado.mostrarnombre(); // "Roberto"

miobjeto.mostrarnombre(); // "Juan"

miobjeto.mostraredad = function(){

alert(this.edad);

};

empleado.mostraredad(); // 24

empleado.saludo=function(){ alert ("Hola " + this.nombre);

miobjeto.saludo???

Objetos Estándar

- Son lo objetos predefinidos por el lenguaje. Se definen mediante el operador **new**
- Estos objetos poseen propiedades y métodos que pueden ser utilizados por el programador. Es una herramienta muy potente para el procesamiento de ciertos «tipos de datos»
  - **String**, para el trabajo con cadenas de caracteres.
  - **Date**, para el trabajo con fechas.
  - **Math**, para realizar funciones matemáticas.
  - **Number**, para realizar algunas cosas con números
  - **Boolean**, trabajo con boleanos.
  - **Array** arreglos

Declaración de objetos estándar

- Estos constructores trabajan con el operador new para crear nuevos objetos. El siguiente ejemplo almacena un número.

```
var valor = new Number(5);
alert(valor); // 5
var valor1 = new Number("5");
alert(valor1); // 5
var lista = new Array(12, 35, 57, 8);
alert(lista); // 12,35,57,8
```

Objeto String

- Propiedad Length: Guarda el número de caracteres del String.
- Métodos:
- **toLowerCase()**—Este método convierte los caracteres a letras minúsculas.
- **toUpperCase()**—Este método convierte los caracteres a letras mayúsculas.
- **trim()**—Este método elimina espacios en blanco a ambos lados de la cadena de caracteres. JavaScript también incluye los métodos trimLeft() y trimRight() para limpiar la cadena de caracteres en un lado específico (izquierda o derecha).
- **substr(**comienzo, extensión**)**—Este método devuelve una nueva cadena de caracteres con caracteres extraídos de la cadena original. El atributo **comienzo** indica la posición del primer carácter a leer, y el atributo **extensión** determina cuántos caracteres queremos incluir. Si no se especifica la extensión, el método devuelve todos los caracteres hasta el final de la cadena.

Objeto String

- **Substring(**comienzo, final**)**—Este método devuelve una nueva cadena de caracteres con caracteres extraídos de la cadena original. Los atributos **comienzo** y **final** son números enteros que determinan las posiciones del primer y último carácter a incluir.
- **split(**separador, limite**)**—Este método divide la cadena de caracteres y devuelve un array con todas las partes. El atributo **separador** indica el carácter en el que se va a cortar la cadena y el atributo **limite** es un número entero que determina el número máximo de partes. Si no especificamos un límite, la cadena se cortará cada vez que se encuentre el separador.
- **startsWith(**valor**)**—Este método devuelve true si el texto especificado por el atributo
- **valor** se encuentra al comienzo de la cadena de caracteres.
- **endsWith(**valor**)**—Este método devuelve true si el texto especificado por el atributo
- **valor** se encuentra al final de la cadena de caracteres

Objeto String

- **Includes(**buscar, posición**)**—Este método busca el valor del atributo **buscar** dentro de la cadena y devuelve true o false de acuerdo con el resultado. El atributo **buscar** es el texto que queremos buscar, y el atributo **posición** determina el índice en el que queremos comenzar la búsqueda. Si el atributo **posición** no se especifica, la búsqueda comienza desde el inicio de la cadena.
- **Replace(**expresión, reemplazo**)**—Este método reemplaza la parte de la cadena de caracteres que coincide con el valor del atributo **expresión** por el texto especificado por el atributo **reemplazo**. El atributo **expresión** se puede especificar como una cadena de caracteres o como una expresión regular para buscar un texto con un formato particular

Objeto String

- **indexOf(**valor, posición**)**—Este método devuelve el índice en el que el texto especificado por el atributo **valor** se encuentra por primera vez. El atributo **posición** determina el índice en el que queremos comenzar la búsqueda. Si el atributo **posición** no se especifica, la búsqueda comienza desde el inicio de la cadena. El método devuelve el valor -1 si ninguna coincidencia es encontrada.
- **lastIndexOf(**valor, posición**)**—Este método devuelve el índice en el que se encuentra por primera vez el texto especificado por el atributo **valor**. A diferencia de indexOf(), este método realiza una búsqueda hacia atrás, desde el final de la cadena. El atributo **posición** determina el índice en el que queremos comenzar la búsqueda. Si no se especifica el atributo **posición**, la búsqueda comienza desde el final de la cadena. El método devuelve el valor -1 si no se encuentra ninguna coincidencia.

Ejemplo Objeto String

```
var texto = “Hola Mundo”;
var mensaje = “El texto tiene “ + texto.length + “ caracteres”;
alert(mensaje); // “El texto tiene 10 caracteres
var palabra = texto.substring(5, 7);
alert(palabra); // “Mu”
mensaje=mensaje.toUpperCase();
```

Objeto Array

- **Length**—Esta propiedad devuelve un número entero que representa la cantidad de valores en el array.
- **Push(**valores**)**—Este método agrega uno o más valores al final del array y devuelve la nueva extensión del array. También contamos con un método similar llamado unshift(), que agrega los valores al comienzo del array.
- **Pop()**—Este método elimina el último valor del array y lo devuelve. También contamos con un método similar llamado shift(), que elimina el primer valor del array.
- **Concat(**array**)**—Este método concatena el array con el array especificado por el atributo y devuelve un nuevo array con el resultado. Los arrays originales no se modifican.
- **Splice(**índice, remover, valores**)**—Este método agrega o elimina valores de un array y devuelve un nuevo array con los elementos eliminados. El atributo **índice** indica el índice en el que vamos a introducir las modificaciones, el atributo **remover** es el número de valores que queremos eliminar desde el índice, y el atributo **valores** es la lista de valores separados por coma que queremos agregar al array desde el índice. Si solo queremosagregar valores, el atributo **remover** se puede declarar con el valor 0, y si solo queremos eliminar valores, podemos ignorar el último atributo

Objeto Array

- **Slice(**comienzo, final**)**—Este método copia los valores de los atributos dentro de un nuevo array. Los atributos **comienzo** y **final** indican los índices del primer y último valor a copiar. El último valor no se incluye en el nuevo array.
- **indexOf(**valor, posición**)**—Este método devuelve el índice en el que se encuentra por primera vez el valor especificado por el atributo **valor**. El atributo **posición** determina el índice en el que queremos comenzar la búsqueda. Si el atributo **posición** no se especifica, la búsqueda comienza desde el inicio del array. El método devuelve el valor -1 si no se encuentra ninguna coincidencia.
- **lastIndexOf(**valor, posición**)**—Este método devuelve el índice en el que el valor especificado por el atributo **valor** se encuentra por primera vez. A diferencia de indexOf(), este método realiza una búsqueda de atrás hacia adelante. El atributo
- **posición** determina el índice en el que queremos comenzar la búsqueda. Si no se especifica el atributo **posición**, la búsqueda comienza desde el final del array. El método devuelve el valor -1 si no se encuentra ninguna coincidencia.

Objeto Array

- **Filter(**función**)**—Este método envía los valores del array a una función uno por uno y devuelve un nuevo array con todos los valores que aprueba la función. El atributo **función** es una referencia a una función o una función anónima a cargo de validar los valores. La función recibe tres valores: el valor a evaluar, su índice y una referencia al array. Después de procesar el valor, la función debe devolver un valor booleano para indicar si es válido o no.
- **every(**función**)**—Este método envía los valores del array a una función uno por uno y devuelve true cuando la función aprueba todos los valores. El atributo **función** es una referencia a una función o una función anónima a cargo de evaluar los valores. La función recibe tres valores: el valor a evaluar, su índice, y una referencia al array. Después de procesar el valor, la función debe devolver un valor booleano indicando si es válido o no. También contamos con un método similar llamado some() que devuelve true si la función aprueba al menos uno de los valores.
- **Join(**separador**)**—Este método crea una cadena de caracteres con todos los valores en el array. El atributo **separador** especifica el carácter o la cadena de caracteres que queremos incluir entre los valores.

Objeto Array

- **Reverse()**—Este método invierte el orden de los valores en el array.
- **Sort(**función**)**—Este método ordena los valores en el array. El atributo **función** es una referencia a una función o una función anónima a cargo de comparar los valores. La función recibe dos valores desde el array y debe devolver un valor booleano indicando su orden. Si no se especifica el atributo, el método ordena los elementos alfabéticamente y en orden ascendente.
- **Map(**función**)**—Este método envía los valores del array a una función uno por uno y crea un nuevo array con los valores que devuelve la función. El atributo función es una referencia a una función o una función anónima a cargo de procesar los valores. La función recibe tres valores: el valor a procesar, su índice y una referencia al array.

Ejemplo Objeto Array

```
var lista = [12, 5, 80, 34];
var total = 0;
for (var f = 0; f < lista.length; f++) {
total += lista[f];
}
alert(“El total es: “ + total); // “El total es: 131”
var mensaje = lista.join(“-“);
alert(mensaje); // “12-5-80-34”
```

Objeto Date

- Manipular fechas es una tarea complicada, no solo porque las fechas están compuestas de varios valores que representan diferentes componentes, sino porque estos componentes están relacionados. Si un valor sobrepasa su limite, afecta al resto de los valores de la fecha. Y los límites son distintos por cada componente
- **Date(**valor**)**—Este constructor crea un valor en milisegundos para representar una fecha basada en los valores provistos por el atributo. El atributo valor se puede declarar como una cadena de caracteres o como los componentes de una fecha separados por comas, en este orden: año, mes, día, horas, minutos, segundos y milisegundos.

Objeto Date

- **getFullYear()**—Este método devuelve un número entero que representa el año (un valor de 4 dígitos).
- **getMonth()**—Este método devuelve un número entero que representa el mes (un valor de 0 a 11).
- **getDate()**—Este método devuelve un número entero que representa el día del mes (un valor de 1 a 31).
- **getDay()**—Este método devuelve un número entero que representa el día de la semana (un valor de 0 a 6).
- **getHours()**—Este método devuelve un número que representa las horas (un valor de 0 a 23).
- **getMinutes()**—Este método devuelve un número entero que representa los minutos (un valor de 0 a 59).

Objeto Date

- **getSeconds()**—Este método devuelve un número entero que representa los segundos (un valor de 0 a 59).
- **getMilliseconds()**—Este método devuelve un número entero que representa los milisegundos (un valor de 0 a 999).
- **getTime()**—Este método devuelve un número entero en milisegundos que representa el intervalo desde el 1 de enero de 1970 hasta la fecha.
- **setFullYear(**año**)**—Este método especifica el año (un valor de 4 dígitos). También puede recibir valores para especificar el mes y el día.
- **setMonth(**mes**)**—Este método especifica el mes (un valor de 0 a 11). También puede recibir valores para especificar el día.
- **setDate(**día**)**—Este método especifica el día (un valor de 1 a 31).
- **setHours(**horas**)**—Este método especifica la hora (un valor de 0 a 23). También puede recibir valores para especificar los minutos y segundos.
- **setMinutes(**minutos**)**—Este método especifica los minutos (un valor de 0 a 59).
- **setSeconds(**segundos**)**—Este método especifica los segundos (un valor de 0 a 59).
- **setMilliseconds(**milisegundos**)**—Este método especifica los milisegundos (un valor de

Objeto Date

- Los objetos Date también ofrecen métodos para convertir una fecha en una cadena de caracteres.
- **toString()**—Este método convierte una fecha en una cadena de caracteres. El valor que devuelve se expresa en inglés americano y con el formato “Wed Jan 04 2017 22:32:48 GMT-0500 (EST)”.
- **toDateString()**—Este método convierte una fecha en una cadena de caracteres, pero solo devuelve la parte de la fecha, no la hora. El valor se expresa en inglés americano y con el formato “Wed Jan 04 2017”.
- **toTimeString()**—Este método convierte una fecha en una cadena de caracteres, pero solo devuelve la hora. El valor se expresa en inglés americano y con el formato “23:21:55 GMT-0500 (EST)”.

Ejemplo Date

```
var fecha = new Date();
alert(fecha);
var fecha1 = new Date(“January 20 2017”);
alert(fecha1);

var hoy = new Date();
var año = hoy.getFullYear();
alert(“El año es “ + ano);
```

Objeto Math

- **PI**—Esta propiedad devuelve el valor de PI.
- **E**—Esta propiedad devuelve la constante Euler.
- **LN10**—Esta propiedad devuelve el logaritmo natural de 10. El objeto también incluye las propiedades LN2 (algoritmo natural de 2), LOG2E (algoritmo base 2 de E) y LOG10E (algoritmo base 10 de E).
- **SQRT2**—Esta propiedad devuelve la raíz cuadrada de 2. El objeto también incluye la propiedad SQRT1_2 (la raíz cuadrada de 1/2).
- **Ceil(**valor**)**—Este método redondea un valor hacia arriba al siguiente entero y devuelve el resultado.
- **Floor(**valor**)**—Este método redondea un valor hacia abajo al siguiente entero y devuelve el resultado.
- **Round(**valor**)**—Este método redondea un valor al entero más cercano y devuelve el resultado.

Objeto Math

- **Trunc(**valor**)**—Este método elimina los dígitos de la fracción y devuelve un entero.
- **abs(**valor**)**—Este método devuelve el valor absoluto de un número (invierte valores negativos para obtener un número positivo).
- **Min(**valor**)**—Este método devuelve el número más pequeño de una lista de valores separados por comas.
- **Max(**valores**)**—Este método devuelve el número más grande de una lista de valores separados por comas.
- **Random()**—Este método devuelve un número al azar en un rango entre 0 y 1.

Math

- **Pow(**base, exponente**)**—Este método devuelve el resultado de elevar la base a la potencia del exponente.
- **Exp(**exponente**)**—Este método devuelve el resultado de elevar E a la potencia del exponente. El objeto también incluye el método expm1(), que devuelve el mismo resultado menos 1.
- **Sqrt(**valor**)**—Este método devuelve la raíz cuadrada de un valor.
- **Log10(**valor**), Sin(**valor**), Cos(**valor**), Tan(**valor**)**

Ejemplo Math

```
var cuadrado = Math.sqrt(4); // 2
var elevado = Math.pow(2, 2); // 4
var maximo = Math.max(cuadrado, elevado);
alert("El valor más grande es " +maximo); // "El valor más grande es 4"
```

Objeto Window

- Cada vez que abrimos el navegador o iniciamos una nueva pestaña, se crea un objeto global llamado Window para referenciar la ventana del navegador y proveer algunas propiedades y métodos esenciales. El objeto se almacena en una propiedad del objeto global de JavaScript llamada window. A través de esta propiedad podemos conectarnos con el navegador y el documento desde nuestro código.
- El objeto Window a su vez incluye otros objetos con los que provee información adicional relacionada con la ventana y el documento. Las siguientes son algunas de las propiedades disponibles para acceder a estos objetos.

Objeto window

- **Location**—Esta propiedad contiene un objeto Location con información acerca del origen del documento. También se puede usar como una propiedad para declarar o devolver la URL del documento (por ejemplo, window.location = “http://www.formasterminds.com”).
- **history**—Esta propiedad contiene un objeto History con propiedades y métodos para manipular el historial de navegación.
- **Navigator**—Esta propiedad contiene un objeto Navigator con información acerca de la aplicación y el dispositivo.
- **Document**—Esta propiedad contiene un objeto Document, que provee acceso a los objetos que representan los elementos HTML en el documento.

Objeto Window

- **innerWidth**—Esta propiedad devuelve el ancho de la ventana en píxeles.
- **innerHeight**—Esta propiedad devuelve la altura de la ventana en píxeles.
- **scrollX**—Esta propiedad devuelve el número de píxeles en los que el documento se ha desplazado horizontalmente.
- **scrollY**—Esta propiedad devuelve el número de píxeles en los que el documento se ha
- desplazado verticalmente.
- **Alert(**valor**)**—Este método muestra una ventana emergente en la pantalla que muestra el valor entre paréntesis.
- **Confirm(**mensaje**)**—Este método es similar a alert(), pero ofrece dos botones, Aceptar y Cancelar, para que el usuario elija qué hacer. El método devuelve true o false, según a la respuesta del usuario.
- **Prompt(**mensaje**)**—Este método muestra una ventana emergente con un campo de entrada para permitir al usuario introducir un valor. El método devuelve el valor que inserta el usuario.

Objeto Window

- **Open(**URL, ventana, parámetros**)**—Este método abre un documento en una nueva ventana. El atributo **URL** es la URL del documento que queremos abrir, el atributo **ventana** es el nombre de la ventana donde queremos mostrar el documento (si el nombre no se especifica o la ventana no existe, el documento se abre en una nueva ventana), y el atributo **parámetros** es una lista de parámetros de configuración separados por comas que configuran las características de la ventana (por ejemplo, resizable=no,scrollbars=no").
- **Close()** para cerrar una ventana abierta con este método.

Ejemplo Window

```
<head><script>
function realizar() {
location.reload();
} </script>
</head>
<body>
<section>
<h1>Sitio Web</h1>
<button type=”button” onclick=”realizar()”>Presione Aquí</button>
</section>
```

Ejemplo Window

```
<head>
<script>
function realizar() {
open(“http://www.formasterminds.com”);
}
</script> </head>
<body>
<section>
<h1>Sitio Web</h1>
<button type=”button” onclick=”realizar()”>Presione Aquí</button>
</section>
</body>
</html>
```

Objeto Document

- Casi todo en JavaScript se define como un objeto, y esto incluye los elementos en el documento. Cuando se carga un documento HTML, el navegador crea una estructura interna para procesarlo. La estructura se llama *DOM* (Document Object Model) y está compuesta por múltiples objetos de tipo Element (u otros tipos más específicos que heredan de Element), que representan cada elemento en el documento.
- Los objetos Element mantienen una conexión permanente con los elementos que representan. Cuando se modifica un objeto, su elemento también se modifica y el resultado se muestra en pantalla. Para ofrecer acceso a estos objetos y permitirnos alterar sus propiedades desde nuestro código JavaScript, los objetos se almacenan en un objeto llamado Document que se asigna a la propiedad document del objeto Window.

Objeto Document

- **forms**—Esta propiedad devuelve un array con referencias a todos los objetos Element que representan los elementos <form> en el documento.
- **images**—Esta propiedad devuelve un array con referencias a todos los objetos Element que representan los elementos <img> en el documento.
- **links**—Esta propiedad devuelve un array con referencias a todos los objetos Element que representan los elementos <a> en el documento.

Objeto Document

- El objeto Document también incluye los siguientes métodos para acceder a objetos individuales u obtener listas de objetos a partir de otros parámetros.
- **getElementById(**id**)**—Este método devuelve una referencia al objeto Element que representa el elemento identificado con el valor especificado por el atributo (el valor asignado al atributo id).
- **getElementsByClassName(**clase**)**—Este método devuelve un array con referencias a los objetos Element que representan los elementos identificados con la clase especificada por el atributo (el valor asignado al atributo clase).
- **getElementsByName(**nombre**)**—Este método devuelve un array con referencias a los objetos Element que representan los elementos identificados con el nombre especificado por el atributo (el valor asignado al atributo name).

Objeto Document

- **getElementsByTagName(**tipo**)**—Este método devuelve un array con referencias a los objetos Element que representan el tipo de elementos especificados por el atributo. El atributo es el nombre que identifica a cada tipo de elemento, como h1, p, img, div, etc.
- **querySelector(**selector**)**—Este método devuelve una referencia al objeto Element que representa el elemento que coincide con el selector especificado por el atributo. El método devuelve el primer elemento en el documento que coincide con el selector CSS
- **querySelectorAll(**selectores**)**—Este método devuelve un array con referencias a los objetos Element que representan los elementos que coinciden con los selectores especificados por el atributo. Se pueden declarar uno o más selectores separados por comas

Ejemplo Document getElementby

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8">
<title>JavaScript</title>
<script>
function iniciar() {
var elemento = document.getElementById("subtitulo");
}
</script>
</head>
```

Ejemplo Document getelementby

```
<body onload="iniciar()">
<section>
<h1>Website</h1>
<p id="subtitulo">El mejor sitio web!</p>
</section>
</body>
</html>
```

Ejemplo Document queryselector

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8">
<title>JavaScript</title>
<script>
function iniciar() {
var elemento = document.querySelector("section > p");
alert("El id es: " + elemento.id);
}
</script>
</head>
```

Ejemplo Document queryselector

```
<body onload="iniciar()">
<section>
<h1>Sitio Web</h1>
<p id="subtitulo">El mejor sitio web!</p>
</section>
</body>
</html>
```

Objeto Element

- Obtener una referencia para acceder a un elemento y leer sus atributos puede ser útil en algunas circunstancias, pero lo que convierte a JavaScript en un lenguaje dinámico es la posibilidad de modificar esos elementos y el documento. Con este propósito, los objetos
- Element contienen propiedades para manipular y definir los estilos de los elementos y sus contenidos. Una de estas propiedades es style, el cual contiene un objeto llamado Styles que a su vez incluye propiedades para modificar los estilos de los elementos.
- Los nombres de los estilos en JavaScript no son los mismos que en CSS.

Objeto Element

- **color**—Esta propiedad declara el color del contenido del elemento.
- **background**—Esta propiedad declara los estilos del fondo del elemento. También podemos trabajar con cada estilo de forma independiente usando las propiedades asociadas backgroundColor, backgroundImage, backgroundRepeat, backgroundPosition y backgroundAttachment.
- **border**—Esta propiedad declara los estilos del borde del elemento. Podemos modificar cada estilo de forma independiente con las propiedades asociadas borderColor, borderStyle, y borderWidth, o modificar cada borde individualmente usando las propiedades asociadas borderTop (borderTopColor, borderTopStyle, y borderTopWidth), borderBottom (borderBottomColor, borderBottomStyle, y borderBottomWidth), borderLeft (borderLeftColor, borderLeftStyle, y borderLeftWidth), y borderRight (borderRightColor, borderRightStyle, y borderRightWidth).
- **margin**—Esta propiedad declara el margen del elemento. También podemos usar las propiedades asociadas marginBottom, marginLeft, marginRight, y marginTop.

Objeto Element

- **padding**—Esta propiedad declara el relleno del elemento. También podemos usar las propiedades asociadas paddingBottom, paddingLeft, paddingRight, y paddingTop.
- **width**—Esta propiedad declara el ancho del elemento. Existen dos propiedades asociadas para declarar el ancho máximo y mínimo de un elemento: maxWidth y minWidth.
- **height**—Esta propiedad declara la altura del elemento. Existen dos propiedades asociadas para declarar la altura máxima y mínima de un elemento: maxHeight y minHeight.
- **visibility**—Esta propiedad determina si el elemento es visible o no.
- **display**—Esta propiedad define el tipo de caja usado para presentar el elemento.
- **position**—Esta propiedad define el tipo de posicionamiento usado para posicionar el elemento.
- **top**—Esta propiedad especifica la distancia entre el margen superior del elemento y el margen superior de su contenedor.
- **bottom**—Esta propiedad especifica la distancia entre el margen inferior del elemento y el margen inferior de su contenedor.

Objeto Element

- **Left/right**—Esta propiedad especifica la distancia entre el margen izquierdo/derecho del elemento y el margen izquierdo/derecho de su contenedor.
- **cssFloat**—Esta propiedad permite al elemento flotar hacia un lado o el otro.
- **overflow**—Esta propiedad especifica cómo se va a mostrar el contenido que excede los límites de la caja de su contenedor.
- **font**—Esta propiedad declara los estilos de la fuente. También podemos declarar los estilos individuales usando las propiedades asociadas fontFamily, fontSize, fontStyle, fontVariant y fontWeight.
- **textAlign**—Esta propiedad alinea el texto dentro del elemento.
- **verticalAlign**—Esta propiedad alinea elementos Inline verticalmente.
- **textDecoration**—Esta propiedad resalta el texto con una línea. También podemos declarar los estilos de forma individual asignando los valores true o false a las propiedades textDecorationBlink, textDecorationLineThrough, textDecorationNone, textDecorationOverline, y textDecorationUnderline.

Ejemplo Element

```
<script>
function iniciar() {
var elemento = document.getElementById("subtitulo");
elemento.style.width = "300px";
elemento.style.border = "1px solid #FF0000";
elemento.style.padding = "20px";
}
</script>
</head>
<body onload="iniciar()">
<p id="subtitulo">El mejor sitio web!</p>
```

Crear Objetos Element

- Existen un gran numero de propiedades y metodos de element, los que se mostraron aquí son algunos de los mas utilizados
- Cuando se agrega código HTML al documento a través de propiedades y métodos como innerHTML o insertAdjacentHTML(), el navegador analiza el documento y genera los objetos Element necesarios para representar los nuevos elementos. Aunque es normal utilizar este procedimiento para modificar la estructura de un documento, el objeto Document incluye métodos para trabajar directamente con los objetos Element.

Crear Objetos Element

- **createElement(**nombre**)**—Este método crea un nuevo objeto Element del tipo especificado por el atributo **nombre**.
- **appendChild(**elemento**)**—Este método inserta el elemento representado por un objeto Element como hijo de un elemento existente en el documento.
- **removeChild(**elemento**)**—Este método elimina un elemento hijo de otro elemento. El atributo debe ser una referencia del hijo a eliminarse.

Ejemplo Modificar HTML

```
<script>
function agregarelemento() {
var elemento = document.querySelector("section");
var elementonuevo = document.createElement("p");
elemento.appendChild(elementonuevo);
}
</script>
<body>
<section>
<h1>Sitio Web</h1>
<button type="button" onclick="agregarelemento()">Agregar Elemento</button>
</section>
</body>
```

Agregar elemento a HTML

- Si no queremos reemplazar todo el contenido de un elemento, sino agregar más contenido, podemos usar el método insertAdjacentHTML(). Este método puede agregar contenido antes o después del contenido actual y también fuera del elemento, dependiendo del valor asignado al primer atributo.

Ejemplo InsertAdjacentHTML

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>JavaScript</title>
<script>
function agregarelemento() {
var elemento = document.querySelector("section");
elemento.insertAdjacentHTML("beforeend", "<p>Este es un texto nuevo</p>");
}
</script>
</head>
<body>
<section>
<h1>Sitio Web</h1>
<button type="button" onclick="agregarelemento()">Agregar Contenido</button>
</section>
</body>
```

Ejemplo innerHTML

- La manera más sencilla de reemplazar el contenido de un elemento es con la propiedad innerHTML. Asignando un nuevo valor a esta propiedad, el contenido actual se reemplaza con el nuevo. El siguiente ejemplo reemplaza el contenido del elemento <p> con el texto "Este es mi sitio web" cuando hacemos clic en él.

Ejemplo InnerHTML

```
<!DOCTYPE html>
<head>
<script>
function cambiarcontenido() {
var elemento = document.getElementById("subtitulo");
elemento.innerHTML = "Este es mi sitio web";
}
</script> </head>
<body>
<section>
<h1>Sitio Web</h1>
<p id="subtitulo" onclick="cambiarcontenido()">El mejor sitio web!</p>
</section>
</body> </html>
```

Ejemplo InnerHTML procesa y modifica

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>JavaScript</title>
<script>
function cambiarcontenido() {
var elemento = document.getElementById("subtitulo");
var texto = elemento.innerHTML + " Somos los mejores!";
elemento.innerHTML = texto;
}
</script>
</head>
```

Ejemplo InnerHTML - cont

```
<body>
<section>
<h1>Sitio Web</h1>
<p id="subtitulo" onclick="cambiarcontenido()">El mejor sitio
web!</p>
</section>
</body>
</html>
```