JavaScript

Guía Completa de JavaScript para el Examen

Esta guía ha sido diseñada para ofrecerte una referencia completa y detallada de los conceptos esenciales de JavaScript que necesitas dominar para tu examen. Al no contar con acceso a internet durante el examen, esta documentación te servirá como tu principal recurso de consulta. ¡Asegúrate de familiarizarte con ella y entender cada sección!

Tabla de Contenidos

1. Programación Básica
    - Variables
    - Tipos de Datos
    - Operadores
    - Estructuras de Control de Flujo
    - Funciones y Propiedades Básicas
2. Programación Avanzada
    - Funciones
    - Ámbito de Variables (Scope)
    - Sentencias `break` y `continue`
    - Estructuras de Control Adicionales
    - Temporizadores
3. Document Object Model (DOM)
    - Introducción al DOM
    - Selección de Elementos

- Manipulación del DOM
- Atributos y Propiedades
- Objetos del Navegador

4. Eventos
   - Conceptos Básicos de Eventos
   - Tipos de Eventos
   - Manejadores de Eventos
   - Obteniendo Información del Evento
   - Objeto `event`
   - Propagación y Delegación de Eventos
5. Notas Adicionales y Consejos
   - Carga del Documento
   - Consejos para el Examen

Capítulo 1: Programación Básica

Variables

Declaración de Variables

- `var`: Evitar su uso; tiene alcance global o de función.
- `let`: Para variables que pueden cambiar; alcance de bloque.
- `const`: Para valores constantes; no pueden reasignarse; alcance de bloque.

```
let edad = 25;
const PI = 3.1416;
```

Reglas de Nomenclatura

- Deben comenzar con una letra, `$` o `_`.
- No pueden ser palabras reservadas.
- Sensibles a mayúsculas y minúsculas (`nombre` ≠ `Nombre`).

Tipos de Datos

Primitivos

- **Número**: Enteros y decimales.

```
let entero = 10;
let decimal = 3.14;
```

- **Cadena de Texto (`string`)**: Texto entre comillas simples o dobles.

```
let saludo = "Hola Mundo";
```

  - **Caracteres Especiales**:
    - `\n`: Nueva línea
    - `\t`: Tabulador
    - `\'`: Comilla simple
    - `\"`: Comilla doble
    - `\\`: Barra invertida

- **Booleano**: `true` o `false`.

```
let esVisible = true;
```

- **Undefined**: Variable declarada pero sin asignar.
- **Null**: Intencionalmente sin valor.

Complejos

- **Array**: Lista ordenada de valores.

```
let frutas = ["Manzana", "Banana", "Cereza"];
```

- **Objeto**: Colección de pares clave-valor.

```
let persona = {
  nombre: "Juan",
```

```
  edad: 30,
};
```

Operadores

Aritméticos

- **Suma (`+`)**
- **Resta (`-`)**
- **Multiplicación (`*`)**
- **División (`/`)**
- **Módulo (`%`)**: Resto de la división.

```
let resto = 17 % 5; // resto = 2
```

Asignación

- `=`: Asignación simple.
- `+=`, `-=`, `*=`, `/=`, `%=`: Operadores de asignación compuesta.

```
let x = 10;
x += 5; // x = x + 5
```

Comparación

- **Igualdad Débil (`==`)**: Compara valor, realiza coerción de tipo.
- **Igualdad Estricta (`===`)**: Compara valor y tipo.
- **Desigualdad (`!=`, `!==`)**
- **Mayor, Menor, Mayor o Igual, Menor o Igual (`>`, `<`, `>=`, `<=`)**

Lógicos

- **AND (`&&`)**: Verdadero si ambas expresiones son verdaderas.
- **OR (`||`)**: Verdadero si al menos una expresión es verdadera.
- **NOT (`!`)**: Invierte el valor lógico.

```
let esAdulto = true;
if (!esAdulto) {
  // Código si no es adulto
}
```

Incremento y Decremento

- **Incremento (`++`)**
- **Decremento (`-`)**

```
let contador = 0;
contador++; // contador = 1
```

Estructuras de Control de Flujo

Condicionales

- `if`

```
if (condicion) {
  // Código si la condición es verdadera
}
```

- `if...else`

```
if (condicion) {
  // Código si verdadera
} else {
  // Código si falsa
}
```

- `if...else if...else`

```
if (condicion1) {
  // Código
} else if (condicion2) {
  // Código
} else {
```

```
  // Código
}
```

Bucles

- `for`

```
for (let i = 0; i < 10; i++) {
  // Código que se repite 10 veces
}
```

- `while`

```
while (condicion) {
  // Código mientras la condición sea verdadera
}
```

- `do...while`

```
do {
  // Código que se ejecuta al menos una vez
} while (condicion);
```

- `for...of` (para arrays)

```
for (let fruta of frutas) {
  console.log(fruta);
}
```

- `for...in` (para objetos)

```
for (let clave in persona) {
  console.log(clave, persona[clave]);
}
```

Funciones y Propiedades Básicas

Conversión de Tipos

- **Convertir String a Número**

```
let cadena = "42";
let numero = Number(cadena); // numero = 42
```

- **Convertir Número a String**

```
let num = 123;
let texto = String(num); // texto = "123"
```

- **Funciones Alternativas**
  - `parseInt(cadena)`: Convierte cadena a número entero.

```
let entero = parseInt("123"); // entero = 123
```

  - `parseFloat(cadena)`: Convierte cadena a número decimal.

```
let decimal = parseFloat("3.14"); // decimal = 3.14
```

Funciones para Cadenas de Texto

Los métodos para manipular cadenas de texto nos permiten realizar operaciones comunes y útiles en programación.

- **Sintaxis del Objeto String**

  Aunque en JavaScript las cadenas se pueden manipular directamente, también es posible crear un objeto String:

```
let texto = new String("Hola Mundo");
```

- `length`: Devuelve la longitud de la cadena.

```
let saludo = "Hola Mundo";
let longitud = saludo.length; // 10
```

- `charAt(indice)`: Devuelve el carácter en la posición indicada.

```
let letra = saludo.charAt(0); // "H"
```

- `substring(ind1, ind2)`: Devuelve el texto comprendido entre los índices.

```
let subcadena = saludo.substring(0, 4); // "Hola"
```

- `indexOf(cadena)`: Devuelve el índice de la primera aparición de la subcadena.

```
let indice = saludo.indexOf("Mundo"); // 5
```

- `replace(cadenaVieja, cadenaNueva)`: Reemplaza una subcadena por otra.

```
let nuevoSaludo = saludo.replace("Mundo", "JavaScript");
// "Hola JavaScript"
```

- `toLowerCase()`: Transforma la cadena a minúsculas.

```
let minusculas = saludo.toLowerCase(); // "hola mundo"
```

- `toUpperCase()`: Transforma la cadena a mayúsculas.

```
let mayusculas = saludo.toUpperCase(); // "HOLA MUNDO"
```

- `split(separador)`: Convierte la cadena en un array, separando por el delimitador indicado.

```
let palabras = saludo.split(" "); // ["Hola", "Mundo"]
```

Funciones Matemáticas

JavaScript proporciona una serie de métodos matemáticos a través del objeto `Math`, que permiten realizar operaciones complejas.

- **Métodos Matemáticos Comunes:**

Método	Descripción	Ejemplo
`Math.sin(x)`	Devuelve el seno de un ángulo (en radianes).	`Math.sin(0)` // 0

`Math.cos(x)`	Devuelve el coseno de un ángulo (en radianes).	`Math.cos(0)` // 1
`Math.tan(x)`	Devuelve la tangente de un ángulo (en radianes).	`Math.tan(0)` // 0
`Math.abs(x)`	Devuelve el valor absoluto de un número.	`Math.abs(-5)` // 5
`Math.log(x)`	Devuelve el logaritmo natural de un número.	`Math.log(1)` // 0
`Math.max(x, y, ...)`	Devuelve el número mayor entre los proporcionados.	`Math.max(5, 10)` // 10
`Math.min(x, y, ...)`	Devuelve el número menor entre los proporcionados.	`Math.min(5, 10)` // 5
`Math.pow(base, exponente)`	Calcula la potencia de un número.	`Math.pow(2, 3)` // 8
`Math.sqrt(x)`	Calcula la raíz cuadrada de un número.	`Math.sqrt(16)` // 4
`Math.round(x)`	Redondea un número al entero más cercano.	`Math.round(3.6)` // 4
`Math.random()`	Genera un número aleatorio entre 0 y 1.	`Math.random()` // Ejemplo: 0.823432
`Math.floor(x)`	Redondea hacia abajo al entero más cercano.	`Math.floor(3.9)` // 3
`Math.ceil(x)`	Redondea hacia arriba al entero más cercano.	`Math.ceil(3.1)` // 4

- **Generar Número Aleatorio en un Rango:**

```
// Número aleatorio entre 0 y 10
let num = Math.random() * 10;

// Número aleatorio entero entre 1 y 100
let enteroAleatorio = Math.floor(Math.random() * 100) +
1;
```

- **Funciones Trigonométricas Inversas:**

```
Math.asin(x); // Arco seno de x
Math.acos(x); // Arco coseno de x
```

```
Math.atan(x); // Arco tangente de x
```

Función `eval()`

- `eval(cadena)`: Evalúa una cadena de texto como código JavaScript.

```
let expresion = "2 + 2";
let resultado = eval(expresion); // resultado = 4
```

> Nota: El uso de eval() es generalmente desaconsejado por razones de seguridad y rendimiento. Úsalo con precaución.

Capítulo 2: Programación Avanzada

Funciones

Declaración de Funciones

- **Función Declarada**

```
function nombreFuncion(parametros) {
  // Código
  return resultado;
}
```

- **Función Expresada (Anónima)**

```
let sumar = function (a, b) {
  return a + b;
};
```

- **Funciones Flecha (ES6+)**

```
let restar = (a, b) => a - b;
```

Parámetros y Argumentos

- **Parámetros**: Variables en la definición.
- **Argumentos**: Valores al invocar la función.

```
function saludar(nombre) {
  return "Hola, " + nombre;
}
saludar("Ana"); // "Hola, Ana"
```

Valores de Retorno

- `return`: Finaliza la función y devuelve un valor.

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

Ámbito de Variables (Scope)

- **Local**: Dentro de una función o bloque.
- **Global**: Fuera de funciones y bloques.

```
let global = "Variable global";

function ejemplo() {
  let local = "Variable local";
  console.log(global); // Accede a variable global
}

console.log(local); // Error: local no está definida
```

Sentencias `break` y `continue`

- `break`: Sale del bucle actual.

```
for (let i = 0; i < 10; i++) {
  if (i === 5) break;
```

```
  console.log(i); // 0 a 4
}
```

- `continue`: Salta a la siguiente iteración.

```
for (let i = 0; i < 10; i++) {
  if (i % 2 === 0) continue;
  console.log(i); // Números impares
}
```

Estructuras de Control Adicionales

`switch`

- Estructura condicional múltiple.

```
switch (expresion) {
  case valor1:
    // Código
    break;
  case valor2:
    // Código
    break;
  default:
    // Código por defecto
}
```

Operador Ternario

- Sintaxis concisa para condiciones simples.

```
let acceso = edad >= 18 ? "Permitido" : "Denegado";
```

Temporizadores

`setTimeout()`

- Ejecuta una función después de un retraso específico (en milisegundos).

```
setTimeout(functionLlamada, tiempo);
```

- **Ejemplo**:

```
function mostrarMensaje() {
  alert("¡Hola después de 2 segundos!");
}

setTimeout("mostrarMensaje()", 2000);
```

> Nota: Es preferible pasar la función directamente sin comillas:

```
setTimeout(mostrarMensaje, 2000);
```

Capítulo 3: Document Object Model (DOM)

> ¡Este capítulo es el más importante! Asegúrate de comprender cada concepto.

Introducción al DOM

- **Definición**: Representación en forma de árbol de todos los elementos de una página web.
- **Nodos**:
  - **Document**: Nodo raíz que representa todo el documento.
  - **Element**: Cada etiqueta HTML.
  - **Text**: El contenido de texto dentro de un elemento.
  - **Attribute**: Atributos de los elementos.
  - **Comment**: Comentarios en el código HTML.

Selección de Elementos

Métodos Clásicos

- `document.getElementById(id)`

```
let titulo = document.getElementById("tituloPrincipal");
```

- `document.getElementsByTagName(etiqueta)`

```
let parrafos = document.getElementsByTagName("p");
```

- `document.getElementsByClassName(clase)`

```
let elementosDestacados = document.getElementsByClassNam
e("destacado");
```

- `document.getElementsByName(name)`

```
let camposFormulario = document.getElementsByName("usuar
io");
```

Métodos Modernos

- `document.querySelector(selector)`
  - Selecciona el primer elemento que coincide con el selector CSS.

```
let primerElemento = document.querySelector(".clase .
subclase");
```

- `document.querySelectorAll(selector)`
  - Selecciona todos los elementos que coinciden.

```
let todosLosEnlaces = document.querySelectorAll("a");
```

Manipulación del DOM

Creación de Elementos

- **Crear un nuevo elemento**

```
let nuevoDiv = document.createElement("div");
```

- **Crear un nodo de texto**

```
let texto = document.createTextNode("Hola Mundo");
```

- **Añadir contenido al elemento**

```
nuevoDiv.appendChild(texto);
```

Inserción en el DOM

- **Añadir al final del cuerpo**

```
document.body.appendChild(nuevoDiv);
```

- **Insertar antes de un elemento existente usando** `insertBefore()`

```
let referencia = document.getElementById("referencia");
let padre = referencia.parentNode;
padre.insertBefore(nuevoDiv, referencia);
```

  - **Explicación:** `insertBefore()` inserta un nodo antes de otro nodo hijo específico. Se aplica al nodo padre.

    ```
    padre.insertBefore(nuevoNodo, nodoReferencia);
    ```

- **Insertar en una posición específica**

```
let lista = document.getElementById("miLista");
let nuevoElemento = document.createElement("li");
nuevoElemento.textContent = "Nuevo ítem";

let primerItem = lista.firstChild;
lista.insertBefore(nuevoElemento, primerItem); // Insert
a al inicio de la lista
```

Clonación de Elementos usando `cloneNode()`

- **Clonar un nodo existente**

```
let elementoOriginal = document.getElementById("elemento
Original");
let copiaElemento = elementoOriginal.cloneNode(true);
```

  - **Parámetro** `true`: Clona el nodo y todos sus descendientes (clonación profunda).
  - **Parámetro** `false`: Clona solo el nodo sin sus hijos (clonación superficial).

- **Insertar el nodo clonado en el DOM**

```
document.body.appendChild(copiaElemento);
```

- **Ejemplo Práctico**

```
// Clonar un elemento de lista y añadirlo al final
let itemOriginal = document.querySelector(".lista .ite
m");
let copiaItem = itemOriginal.cloneNode(true);

let lista = document.querySelector(".lista");
lista.appendChild(copiaItem);
```

Reemplazo de Elementos usando `replaceChild()`

- **Reemplazar un elemento por otro**

```
let nuevoElemento = document.createElement("section");
nuevoElemento.textContent = "Contenido nuevo";

let elementoViejo = document.getElementById("viejo");
let padre = elementoViejo.parentNode;
padre.replaceChild(nuevoElemento, elementoViejo);
```

- **Explicación**: `replaceChild()` reemplaza un nodo hijo por otro. Se aplica al nodo padre.

```
padre.replaceChild(nuevoNodo, nodoExistente);
```

Eliminación de Elementos

- **Eliminar un elemento existente**

```
let elementoAEliminar = document.getElementById("elemento");
elementoAEliminar.parentNode.removeChild(elementoAEliminar);
```

Atributos y Propiedades

Manipulación de Atributos

- **Obtener un atributo**

```
let enlace = document.getElementById("enlace");
let url = enlace.getAttribute("href");
```

- **Establecer un atributo**

```
enlace.setAttribute("href", "https://nueva-url.com");
```

- **Eliminar un atributo**

```
enlace.removeAttribute("target");
```

Manipulación de Clases

- **Añadir una clase**

```
elemento.classList.add("nueva-clase");
```

- **Eliminar una clase**

```
elemento.classList.remove("clase-antigua");
```

- **Comprobar si tiene una clase**

```
elemento.classList.contains("activo");
```

- **Alternar una clase**

```
elemento.classList.toggle("visible");
```

Estilos en Línea

- **Modificar estilos directamente**

```
elemento.style.color = "red";
elemento.style.backgroundColor = "#f0f0f0";
```

> Nota: Las propiedades CSS con guiones se convierten a camelCase en JavaScript.

Trabajando con Contenido de Texto

- **Modificar el contenido de texto de un elemento**

```
elemento.textContent = "Nuevo contenido de texto";
```

- **Obtener el contenido de texto**

```
let texto = elemento.textContent;
```

Trabajando con Formularios

- **Obtener el valor de un input**

```
let input = document.getElementById("miInput");
let valor = input.value;
```

- **Establecer el valor de un input**

```
input.value = "Nuevo valor";
```

Objetos del Navegador

Además del DOM, JavaScript proporciona objetos que permiten interactuar con diferentes aspectos del navegador y la pantalla del usuario. Los más comunes son `window`, `document`, `navigator` y `screen`.

Objeto `window`

El objeto `window` representa la ventana del navegador y es el objeto global en el contexto del navegador. Permite controlar aspectos de la ventana y abrir nuevas ventanas.

Métodos y Propiedades del Objeto `window`:

Método/Propiedad	Descripción	Sintaxis
`open()`	Abre una nueva ventana.	`var nuevaVentana = window.open(url, nombre, atributos);`
`close()`	Cierra la ventana actual o la especificada.	`nuevaVentana.close();`
`opener`	Referencia a la ventana que abrió la ventana actual.	`var openerVentana = nuevaVentana.opener;`
`closed`	Indica si la ventana ha sido cerrada.	`var estaCerrada = nuevaVentana.closed;`
`location`	Obtiene o establece la URL de la ventana.	`window.location = "https://www.ejemplo.com";`
`print()`	Abre el diálogo de impresión para la ventana actual.	`window.print();`
`alert()`	Muestra un cuadro de diálogo de alerta.	`window.alert("Mensaje");`
`confirm()`	Muestra un cuadro de diálogo de confirmación.	`var respuesta = window.confirm("¿Estás seguro?");`
`prompt()`	Muestra un cuadro de diálogo para ingresar texto.	`var entrada = window.prompt("Ingrese su nombre:", "Valor por defecto");`

`status`	Establece el texto en la barra de estado del navegador (no soportado en todos los navegadores).	`window.status = "Cargando...";`
`setTimeout()`	Ejecuta una función después de un retraso especificado.	`setTimeout(funcion, tiempo);`
`showModalDialog()`	Crea una ventana modal (obsoleto en navegadores modernos).	`var retorno = window.showModalDialog(url, argumentos, atributos);`

Objeto `document`

El objeto `document` representa el documento HTML cargado en la ventana. Proporciona métodos y propiedades para interactuar con el contenido de la página.

Métodos y Propiedades del Objeto `document`:

Método/Propiedad	Descripción	Sintaxis
`write()`	Escribe contenido directamente en el documento.	`document.write("Texto");`
`writeln()`	Escribe contenido y añade un salto de línea.	`document.writeln("Texto");`
`bgColor`	Establece el color de fondo de la página.	`document.bgColor = "#FFFFFF";`
`fgColor`	Establece el color del texto de la página.	`document.fgColor = "#000000";`
`linkColor`	Establece el color de los enlaces no visitados.	`document.linkColor = "#0000FF";`
`vlinkColor`	Establece el color de los enlaces visitados.	`document.vlinkColor = "#800080";`
`location`	Obtiene la URL actual del documento.	`var urlActual = document.location;`
`lastModified`	Obtiene la fecha de última modificación del documento.	`var fechaModificacion = document.lastModified;`
`referrer`	Obtiene la URL de la página que enlazó al documento actual.	`var urlReferrer = document.referrer;`

Objeto `navigator`

El objeto `navigator` contiene información sobre el navegador del usuario.

Propiedades del Objeto `navigator`:

Propiedad	Descripción	Sintaxis
`appName`	Nombre del navegador.	`var nombreNavegador = navigator.appName;`
`appVersion`	Versión del navegador.	`var versionNavegador = navigator.appVersion;`
`userAgent`	Información del agente de usuario.	`var agenteUsuario = navigator.userAgent;`
`platform`	Plataforma del sistema operativo.	`var plataforma = navigator.platform;`
`language`	Idioma del navegador.	`var idioma = navigator.language;`

Objeto `screen`

El objeto `screen` proporciona información sobre la configuración de la pantalla del usuario.

Propiedades del Objeto `screen`:

Propiedad	Descripción	Sintaxis
`width`	Ancho total de la pantalla en píxeles.	`var anchoPantalla = screen.width;`
`height`	Alto total de la pantalla en píxeles.	`var altoPantalla = screen.height;`
`colorDepth`	Profundidad de color de la pantalla (bits por píxel).	`var profundidadColor = screen.colorDepth;`

Capítulo 4: Eventos

Conceptos Básicos de Eventos

- **Eventos**: Acciones que ocurren en la página (clics, teclas presionadas, carga de la página, etc.).
- **Programación Basada en Eventos**: El código se ejecuta en respuesta a acciones del usuario o del navegador.

- **Nomenclatura de Eventos**: Los nombres de los eventos comienzan con el prefijo `on` seguido del nombre en inglés de la acción asociada. Por ejemplo, `onclick`, `onmouseover`.

Tipos de Eventos

En JavaScript, cada elemento HTML puede tener asociados diferentes tipos de eventos. Un mismo evento puede aplicarse a múltiples elementos, y un elemento puede tener múltiples eventos.

Eventos Comunes

Evento	Descripción	Elementos Aplicables	Ejemplo
`onblur`	Cuando un elemento pierde el foco	`<input>`, `<select>`, `<textarea>`, `<button>`	`elemento.onblur = funcion;`
`onchange`	Cuando el valor de un elemento cambia	`<input>`, `<select>`, `<textarea>`	`elemento.onchange = funcion;`
`onclick`	Cuando se hace clic en un elemento	Todos los elementos	`elemento.onclick = funcion;`
`ondblclick`	Doble clic sobre un elemento	Todos los elementos	`elemento.ondblclick = funcion;`
`onfocus`	Cuando un elemento recibe el foco	`<input>`, `<select>`, `<textarea>`, `<button>`	`elemento.onfocus = funcion;`
`onkeydown`	Al presionar una tecla (sin soltar)	Elementos de formulario y `<body>`	`elemento.onkeydown = funcion;`
`onkeypress`	Al presionar una tecla	Elementos de formulario y `<body>`	`elemento.onkeypress = funcion;`
`onkeyup`	Al soltar una tecla	Elementos de formulario y `<body>`	`elemento.onkeyup = funcion;`
`onload`	Cuando la página ha cargado	`<body>`	`window.onload = funcion;`

	completamente		
`onmousedown`	Al presionar un botón del ratón	Todos los elementos	`elemento.onmousedown = funcion;`
`onmousemove`	Al mover el ratón sobre un elemento	Todos los elementos	`elemento.onmousemove = funcion;`
`onmouseout`	Cuando el ratón sale de un elemento	Todos los elementos	`elemento.onmouseout = funcion;`
`onmouseover`	Cuando el ratón entra en un elemento	Todos los elementos	`elemento.onmouseover = funcion;`
`onmouseup`	Al soltar un botón del ratón	Todos los elementos	`elemento.onmouseup = funcion;`
`onreset`	Al reiniciar un formulario	`<form>`	`formulario.onreset = funcion;`
`onresize`	Al cambiar el tamaño de la ventana	`<body>`	`window.onresize = funcion;`
`onselect`	Cuando se selecciona un texto	`<input>`, `<textarea>`	`elemento.onselect = funcion;`
`onsubmit`	Al enviar un formulario	`<form>`	`<form onsubmit="return funcion();">`
`onunload`	Al abandonar la página	`<body>`	`window.onunload = funcion;`

Manejadores de Eventos

Asignación Mediante HTML (No Recomendado)

```
<button onclick="alert('Hola')">Haz clic</button>
```

Asignación Mediante Propiedades DOM

```
elemento.onclick = funcion;
```

- **Ejemplo**:

```
let boton = document.getElementById("miBoton");
boton.onclick = function () {
  alert("¡Botón presionado!");
};
```

Uso de `addEventListener` (Recomendado)

- **Sintaxis**

```
elemento.addEventListener("evento", funcion, usoCaptur
a);
```

  - `evento`: Tipo de evento (e.g., `"click"`, `"mouseover"`).
  - `funcion`: Función que maneja el evento.
  - `usoCaptura`: Booleano que indica la fase de propagación (por defecto, `false` para burbuja).

- **Ejemplo**

```
boton.addEventListener("click", funcion);
```

Obteniendo Información del Evento

Para acceder al objeto `event` y obtener información sobre el evento ocurrido, se puede definir una función manejadora que reciba el evento como parámetro.

```
function manejadorEventos(elEvento) {
  var evento = elEvento || window.event;
  // Código que utiliza el objeto evento
}
```

Ejemplo Práctico

```
function resalta(elEvento) {
  var evento = elEvento || window.event;
  switch (evento.type) {
    case "mouseover":
```

```
      this.style.borderColor = "black";
      break;
    case "mouseout":
      this.style.borderColor = "silver";
      break;
  }
}

window.onload = function () {
  var seccion = document.getElementById("seccion");
  seccion.onmouseover = resalta;
  seccion.onmouseout = resalta;
};
```

HTML Asociado:

```
<div
  id="seccion"
  style="width:150px; height:60px; border:thin solid silve
r"
>
  Sección de contenidos...
</div>
```

- **Explicación**: La función `resalta` cambia el color del borde del elemento según el tipo de evento (`mouseover` o `mouseout`). Utiliza `this` para referirse al elemento que generó el evento.

Objeto `event`

- **Acceso al Objeto `event`**

  ```
  elemento.onclick = function (event) {
    // Código que utiliza el objeto event
  };
  ```

Propiedades Comunes

- `event.type`: Tipo de evento.

- `event.target`: Elemento que originó el evento.
- `event.currentTarget`: Elemento al que se le asignó el manejador.
- `event.preventDefault()`: Evita la acción por defecto del evento.
- `event.stopPropagation()`: Detiene la propagación del evento.

Propagación y Delegación de Eventos

Fases de Propagación

1. **Captura**: El evento se propaga desde el elemento más externo hasta el objetivo.
2. **Objetivo**: El evento alcanza el elemento objetivo.
3. **Burbuja**: El evento se propaga desde el objetivo hacia los elementos padres.

Detener la Propagación

- `event.stopPropagation()`

```
elemento.addEventListener("click", function (event) {
  event.stopPropagation();
  // Código específico
});
```

Delegación de Eventos

- Asignar un evento a un elemento padre para manejar eventos en elementos hijos.

```
contenedor.addEventListener("click", function (event) {
  if (event.target.matches(".boton")) {
    // Acción para elementos con clase 'boton'
  }
});
```

Prevención de Comportamientos por Defecto

- `event.preventDefault()`

- Evita acciones como seguir un enlace o enviar un formulario.

```
enlace.addEventListener("click", function (event) {
  event.preventDefault();
  // Código alternativo
});
```

Notas Adicionales y Consejos

Carga del Documento

`window.onload`

- Se ejecuta cuando todos los recursos (imágenes, scripts, etc.) han cargado.

```
window.onload = function () {
  // Código a ejecutar
};
```

`DOMContentLoaded`

- Se ejecuta cuando el DOM ha sido completamente cargado, sin esperar a recursos externos.

```
document.addEventListener("DOMContentLoaded", function
() {
  // Código a ejecutar
});
```

Consejos para el Examen

- **Prioriza DOM y Eventos**: Estos temas son fundamentales y probablemente serán el foco principal del examen.
- **Practica Manipulación del DOM**: Crea scripts que añadan, modifiquen, clonen y reemplacen elementos en una página usando `insertBefore()`, `replaceChild()` y `cloneNode()`.

- **Entiende la Propagación de Eventos**: Familiarízate con `event.stopPropagation()` y `event.preventDefault()`.
- **Repasa el Scope de Variables**: Asegúrate de diferenciar entre variables locales y globales.
- **Utiliza Buenas Prácticas**: Emplea `let` y `const` en lugar de `var`, y prefiere `addEventListener` sobre manejadores en línea.
- **Escribe Código Limpio y Comentado**: Facilitará tu comprensión durante el examen.

¡Mucho éxito en tu examen! Estudia estos conceptos y practica con ejemplos para fortalecer tu comprensión. Recuerda que la clave está en entender cómo y por qué funcionan las cosas en JavaScript. 🚀