



PROGRAMACIÓN CON LENGUAJES DE GUION EN PÁGINAS WEB (UF1305)

Familia profesional: Informática y Comunicaciones

[Manual de contenidos](#)

Reservados todos los derechos. El contenido de esta obra está protegido por la ley, que establece penas de prisión o multas, además de las correspondientes indemnizaciones por daños y perjuicios, para quienes reprodujeren, plagiaren, distribuyeren o comunicaren públicamente en todo o en parte, una obra literaria, artística o científica, o su transformación, interpretación o ejecución artística fijada en cualquier tipo de soporte o comunicada a través de cualquier medio, sin la preceptiva autorización.

Todos los nombres propios de programas, sistemas operativos, equipos, hardware, programas de afiliación, páginas web, etc. que aparecen en esta publicación son marcas registradas de sus respectivas compañías, organizaciones y propietarios y tan solo se muestran a modo informativo.

Conzepto Comunicación Creativa

ÍNDICE

1. METODOLOGÍA DE LA PROGRAMACIÓN

1.1. Lógica de programación

- 1.1.1. Descripción y utilización de operaciones lógicas
- 1.1.2. Secuencias y partes de un programa

1.2. Ordinogramas

- 1.2.1. Descripción de un ordinograma
- 1.2.2. Elementos de un ordinograma
- 1.2.3. Operaciones en un programa
- 1.2.4. Implementación de elementos y operaciones en un ordinograma.

1.3. Pseudocódigos

- 1.3.1. Descripción de pseudocódigo
- 1.3.2. Creación del pseudocódigo.

1.4. Objetos

- 1.4.1. Descripción de objetos
- 1.4.2. Funciones de los objetos
- 1.4.3. Atributos de los objetos
- 1.4.4. Creación de objetos.

1.5. Ejemplos de códigos en diferentes lenguajes

- 1.5.1. Códigos en lenguajes estructurales
- 1.5.2. Códigos en lenguajes scripts
- 1.5.3. Códigos en lenguajes orientados a objetos

2. LENGUAJE DE GUIÓN

2.1. Características del lenguaje

- 2.1.1. Descripción del lenguaje orientado a eventos
- 2.1.2. Descripción del lenguaje interpretado
- 2.1.3. La interactividad del lenguaje de guión

2.2. Relación del lenguaje de guión y el lenguaje de marcas

- 2.2.1. Extensión de las capacidades del lenguaje de marcas
- 2.2.2. Adicción de propiedades interactivas

2.3. Sintaxis del lenguaje de guión

- 2.3.1. Etiquetas identificativas dentro del lenguaje de marcas

2.3.2. Especificaciones y características de las instrucciones

2.3.3. Elementos del lenguaje de guión

- a) Variables
- b) Operaciones
- c) Comparaciones
- d) Asignaciones

2.3.4. Objetos del lenguaje de guión

- a) Métodos
- b) Eventos
- c) Atributos
- d) Funciones

2.4. Tipos de scripts: inmediatos, diferidos e híbridos

2.4.1. Script dentro del cuerpo del lenguaje de marcas

- a) Ejecutables al abrir la página
- b) Ejecutables por un evento

2.4.2. Script dentro del encabezado del lenguaje de marcas

2.4.3. Script dentro del cuerpo del lenguaje de marcas

2.4.4. Ejecución de un script

- a) Ejecución al cargar la página
- b) Ejecución después de producirse un evento
- c) Ejecución del procedimiento dentro de la página
- d) Tiempos de ejecución
- e) Errores de ejecución

3. ELEMENTOS BÁSICOS DEL LENGUAJE DE GUIÓN

3.1. Variables e identificadores

3.1.1. Declaración de variables

3.1.2. Operaciones con variables

3.2. Tipos de datos

3.2.1. Datos booleanos

3.2.2. Datos numéricos

3.2.3. Datos de texto

3.2.4. Valores nulos

3.3. Operadores y expresiones

- 3.3.1. Operadores de asignación
- 3.3.2. Operadores de comparación
- 3.3.3. Operadores aritméticos
- 3.3.4. Operadores sobre bits
- 3.3.5. Operadores lógicos
- 3.3.6. Operadores de cadenas de caracteres
- 3.3.7. Operadores especiales
- 3.3.8. Expresiones de cadena
- 3.3.9. Expresiones aritméticas
- 3.3.10. Expresiones lógicas
- 3.3.11. Expresiones de objeto

3.4. Estructuras de control

- 3.4.1. Sentencia if
- 3.4.2. Sentencia while
- 3.4.3. Sentencia for
- 3.4.4. Sentencia break
- 3.4.5. Sentencia continue
- 3.4.6. Sentencia switch

3.5. Funciones

- 3.5.1. Definición de funciones
- 3.5.2. Sentencia return
- 3.5.3. Propiedades de las funciones
- 3.5.4. Funciones predefinidas del lenguaje de guión
- 3.5.5. Creación de funciones
- 3.5.6. Particularidades de las funciones en el lenguaje de guión

3.6. Instrucciones de entrada / salida

- 3.6.1. Descripción y funcionamiento de las instrucciones de entrada y salida
 - a) Lectura de teclado de datos
 - b) Almacenamiento en variables
 - c) Impresión en pantalla del resultado
- 3.6.2. Sentencia prompt
- 3.6.3. Sentencia document.write

4. DESARROLLO DE SCRIPTS

4.1. Herramientas de desarrollo, utilización

- 4.1.1. Crear scripts con herramientas de texto
- 4.1.2. Crear scripts con aplicaciones web
- 4.1.3. Recursos en web para la creación de scripts

4.2. Depuración de errores: errores de sintaxis y de ejecución

- 4.2.1. Definición de los tipos de errores
- 4.2.2. Escritura del programa fuente
- 4.2.3. Compilación del programa fuente
- 4.2.4. Corrección de errores de sintaxis
- 4.2.5. Corrección de errores de ejecución

4.3. Mensajes de error

- 4.3.1. Funciones para controlar los errores

5. GESTIÓN DE OBJETOS DEL LENGUAJE DE GUIÓN

5.1. Jerarquía de objetos

- 5.1.1. Descripción de objetos de la jerarquía
- 5.1.2. Propiedades compartidas de los objetos
- 5.1.3. Navegar por la jerarquía de los objetos

5.2. Propiedades y métodos de los objetos del navegador

- 5.2.1. El objeto superior Windows#
- 5.2.2. El objeto navigator
- 5.2.3. URL actual (location)
- 5.2.4. URL visitada por el usuario
- 5.2.5. Contenido del documento actual (document)
 - a) Título, color de fondo y formularios

5.3. Propiedades y métodos de los objetos del documento

- 5.3.1. Propiedades del objeto document
- 5.3.2. Ejemplos de propiedades de document
- 5.3.3. Métodos de document
- 5.3.4. Flujo de escritura del documento
- 5.3.5. Métodos open () y close () de document

5.4. Propiedades y métodos de los objetos del formulario

- 5.4.1. Propiedades principales del objeto form (name, action, method, target)

5.4.2. Métodos del objeto form (submit, reset, get, post)

5.5. Propiedades y métodos de los objetos del lenguaje

5.5.1. Document (escribir texto, color fuente, color fondo, obtener elementos del documento actual HTML, título de página)

5.5.2. Windows (open)

5.5.3. History (go)

5.5.4. Location (servidor)

5.5.5. Navigator (nombre, versión y detalles del navegador)

6. LOS EVENTOS DEL LENGUAJE DE GUIÓN

6.1. Utilización de eventos

6.1.1. Definición de eventos

6.1.2. Acciones asociadas a los eventos

6.1.3. Jerarquía de los eventos desde el objeto Windows

6.2. Eventos en elementos de formulario

6.2.1. Onselect (al seleccionar un elemento de un formulario)

6.2.2. Onchange (al cambiar el estado de un elemento de un formulario)

6.3. Eventos de ratón. Eventos de teclado

6.3.1. Eventos de ratón

a) Onmousedown (al pulsar sobre un elemento de la página)

b) Onmousemove (al mover el ratón por la página)

c) Onmouseout (al salir del área ocupada por un elemento de la página)

d) Onmouseover (al entrar el puntero del ratón en el área ocupada por un elemento de la página)

e) Onmouseup (al soltar el usuario el botón del ratón que anteriormente había pulsado)

6.3.2. Eventos de teclado

a) Onkeydown (al pulsar una tecla el usuario)

b) Onkeypress (al dejar pulsada una tecla un tiempo determinado)

c) Onkeyup (al liberar la tecla apretada)

6.4. Eventos de enfoque

6.4.1. Onblur (cuando un elemento pierde el foco de la aplicación)

6.4.2. Onfocus (cuando un elemento de la página o la ventana gana el foco de la aplicación)

6.5. Eventos de formulario

6.5.1. Onreset (al hacer clic en el botón de reset de un formulario)

6.5.2. Onsubmit (al pulsar el botón de enviar el formulario)

6.6. Eventos de ventana

6.6.1. Onmove (al mover la ventana del navegador)

6.6.2. Onresize (al redimensionar la ventana del navegador)

6.7. Otros eventos

6.7.1. Onunload (al abandonar una página)

6.7.2. Onload (al terminar de cargarse la página o imágenes)

6.7.3. Onclick (al hacer clic en el botón del ratón sobre un elemento de la página)

6.7.4. Ondragdrop (al soltar algo que se ha arrastrado sobre la página)

6.7.5. Onerror (al no poderse cargar un documento o una imagen)

6.7.6. Onabort (al detenerse la carga de una imagen de la página o irse de la página)

7. BÚSQUEDA Y ANÁLISIS DE SCRIPTS

7.1. Búsqueda en sitios especializados

7.1.1. Páginas oficiales

7.1.2. Tutoriales

7.1.3. Foros

7.1.4. Bibliotecas

7.2. Operadores booleanos

7.2.1. Funcionamiento de los operadores booleanos

7.2.2. Utilización en distintos buscadores

7.3. Técnicas de búsqueda

7.3.1. Expresiones

7.3.2. Definiciones de búsquedas

7.3.3. Especificaciones

7.4. Técnicas de refinamiento de búsquedas

7.4.1. Utilización de separadores

7.4.2. Utilización de elementos de unión

7.5. Reutilización de scripts

7.5.1. Scripts gratuitos

7.5.2. Generalización de códigos

OBJETIVOS

OBJETIVO GENERAL DEL MÓDULO

Crear y publicar páginas web que integren textos, imágenes y otros elementos, utilizando lenguajes de marcas y editores apropiados, según especificaciones y condiciones de "usabilidad" dadas y realizar los procedimientos de instalación y verificación de las mismas en el servidor correspondiente.

OBJETIVOS ESPECÍFICOS

C1: Identificar las estructuras de programación y los tipos de datos que se utilizan en la elaboración de *scripts*, de acuerdo a unas especificaciones recibidas.

CE1.1 Describir las estructuras secuencial, condicional y de iteración que se utilizan para agrupar y organizar las acciones de un programa.

CE1.2 Reconocer la sintaxis del lenguaje de guion que describen las estructuras de programación en la elaboración de *scripts*, de acuerdo a las especificaciones técnicas del lenguaje.

CE1.3 Explicar los tipos de datos que se utilizan para representar y almacenar los valores de las variables en la elaboración de *scripts*, de acuerdo a las especificaciones técnicas del lenguaje.

CE1.4 Identificar los operadores que se utilizan para hacer los cálculos y operaciones dentro de un *script*.

CE1.5 Citar las instrucciones proporcionadas por el lenguaje de guion para realizar operaciones de entrada y salida de datos, de acuerdo a las especificaciones técnicas del lenguaje.

CE1.6 Distinguir los métodos para ejecutar un *script* utilizando varios navegadores web.

CE1.7 Interpretar *scripts* que resuelvan un problema previamente especificado:— Identificar el tipo y el uso de los datos declarados dentro del *script*.

- Describir las estructuras de programación utilizadas para organizar las acciones del programa.
- Reconocer las instrucciones proporcionadas por el lenguaje de *script* utilizadas en las operaciones de manipulación, entrada y salida de datos.
- Insertar el *script* dentro de la página web utilizando las etiquetas apropiadas.
- Probar la funcionalidad del *script* utilizando un navegador.
- Detectar y corregir los errores de sintaxis y de ejecución.
- Documentar los cambios realizados en el *script*.

C2: Distinguir las propiedades y métodos de los objetos proporcionados por el lenguaje de guion, en función de las especificaciones técnicas del lenguaje.

CE2.1 Explicar los objetos del navegador, así como sus propiedades y métodos, que se utilizan para añadir funcionalidad a las páginas web teniendo en cuenta las especificaciones técnicas del lenguaje.

CE2.2 Identificar los objetos predefinidos por el lenguaje de guion para manejar nuevas estructuras y utilidades que añadirán nuevas funcionalidades a las páginas, de acuerdo a las especificaciones técnicas del lenguaje.

CE2.3 Describir e identificar los objetos del documento que permiten añadir interactividad entre el usuario y el *script*, así como sus propiedades y métodos.

CE2.4 Describir los eventos que proporciona el lenguaje de guion: de ratón, de teclado, de enfoque, de formulario y de carga, entre otros, para interactuar con el usuario y relacionarlos con los objetos del lenguaje.

CE2.5 Interpretar *scripts* que añaden efectos estéticos a la presentación de las páginas:

- Identificar los objetos sobre los que se aplican los efectos estéticos.
- Identificar las propiedades y métodos utilizados para añadir los efectos.
- Reconocer los eventos utilizados para la realización de las acciones.
- Describir la función o funciones de efectos identificando los parámetros de la misma.
- Realizar cambios en el *script* siguiendo unas especificaciones recibidas.
- Detectar y corregir los errores de sintaxis y de ejecución.
- Documentar los cambios realizados.

CE2.6 Interpretar *scripts* en los que se validan las entradas de datos de los campos de un formulario:

- Identificar los objetos del formulario que son validados dentro del *script*.
- Identificar las propiedades y métodos utilizados para validar cada entrada.
- Reconocer las funciones proporcionadas por el lenguaje de guión utilizadas para la validación de datos.
- Describir los eventos que se utilizan en la realización de las acciones.
- Describir la función o funciones de validación identificando los parámetros de la misma.
- Realizar cambios en el *script* siguiendo unas especificaciones recibidas.
- Detectar y corregir los errores de sintaxis y de ejecución.
- Documentar el *script* realizado.

CE2.7 Interpretar la documentación técnica asociada al lenguaje de guion, incluso si está editada en la lengua extranjera de uso más frecuente en el sector, utilizándola de ayuda en la integración de *scripts*.

C3: Identificar *scripts* ya desarrollados que se adapten a las funcionalidades especificadas e integrarlos en las páginas web de acuerdo a unas especificaciones recibidas.

CE3.1 Localizar y descargar el componente ya desarrollado ya sea desde Internet o desde las colecciones indicadas siguiendo las especificaciones recibidas.

CE3.2 Identificar los objetos, sus propiedades y sus métodos y su funcionalidad dentro del *script* ya desarrollado con el fin de ajustarlos a la página donde se va a integrar.

CE3.3 Identificar los eventos incluidos en el *script* para distinguir las interacciones con el usuario.
CE3.4 Describir la lógica de funcionamiento del *script* identificando las estructuras de programación y los datos con los que opera.

CE3.5 Integrar *scripts* ya desarrollados en una página web, para añadir funcionalidades específicas de acuerdo a las especificaciones recibidas:

- Descargar el componente ya desarrollado.
- Utilizar una herramienta de edición de script.
- Modificar las propiedades y los atributos de los objetos que componen el script para ajustarlo a las especificaciones recibidas.
- Comprobar la disponibilidad de utilización del script teniendo en cuenta los derechos de autor y la legislación vigente.
- Integrar el script a la página web previamente indicada.
- Probar la funcionalidad de la página resultante utilizando un navegador.
- Corregir los errores detectados.
- Documentar los procesos realizados.

CONTENIDOS

1. METODOLOGÍA DE LA PROGRAMACIÓN

La metodología de la programación consiste en un conjunto de métodos, principios y reglas capaces de desarrollar un programa que sea capaz de resolver un problema algorítmico.

Esta metodología se suele estructurar paso por paso de manera secuencial. Comienza con la definición del problema y termina con la realización de un programa que resuelve el problema.

La metodología de la programación consta de una serie de pasos. Estos pasos ayudarán a tener una visión mucho más clara del problema del que se parte y cómo va a resolverlo. Los pasos son:

- **Definición del problema:** Tiene que saber exactamente cuál es el problema que quiere resolver y entenderlo claramente para su posterior análisis.
- **Análisis del problema:** Cuando el problema esté bien definido, debe analizarlo para, así, poder llegar a una solución satisfactoria. Para poder definir el problema con precisión, debe especificar cómo va a llevar a cabo la entrada, proceso y salida de datos.
- **Diseño de la solución:** Para que un computador sea capaz de resolver un problema debe proporcionarle los pasos a seguir, ya que, las máquinas no son capaces por sí solas de hacer esta tarea. A los pasos secuenciales que describen las soluciones al problema que la máquina va a ejecutar se llama algoritmos. La información que le proporciona al algoritmo constituye la entrada de datos, por el contrario, la información que produce el algoritmo constituye la salida de datos.

Para diseñar la solución al problema planteado debe romper el problema en pequeños problemas para que así sea más fácil de solucionar. A la descomposición del problema original en problemas más simples se le denomina diseño descendente. Para representar algoritmos en su diseño se utilizarán pseudocódigos o diagramas de flujo. **Codificación:** Se denomina codificación al pasar un algoritmo a la computadora escrito en un lenguaje de programación. Al pasar un algoritmo a un lenguaje de programación, debe seguir las reglas gramaticales y la sintaxis de dicho lenguaje.

- **Compilación y ejecución:** Se habla de compilación cuando el lenguaje de programación revisa que no haya errores en el código. Una vez se haya revisado se pasará a la ejecución.
- **Verificación y pruebas:** En esta parte se corrigen que revisar si el programa aporta los resultados que quiere. Debe revisar si hay un error de tipo lógico de semántica o de ejecución.
- **Depuración:** La depuración es la parte donde se corrigen los errores que se hayan generado. Para ello, debe ir a la parte correspondiente del código y volver a escribirlo.



Más Info

RECURSO MULTIMEDIA



1.1. Lógica de programación

La lógica de la programación es una técnica utilizada para desarrollar instrucciones de manera secuencial y, así, lograr un determinado objetivo. La lógica es la base de todo el conocimiento en programación, ya que enseña a cómo escribir un código determinado para aprender a comunicarse con una computadora y que esta sea capaz de interpretarlo de manera correcta. Gracias a la lógica de la programación puede organizar y planificar las instrucciones que quiere dar a la computadora en un algoritmo. Lo más importante para un programador es montar la estructura del programa de manera que este pueda ser ejecutado por una computadora. Para ello, debe ser consciente de que la computadora no piensa de la misma forma que un ser humano, que no es inteligente y, por lo tanto, no sabe qué es lo que tiene que hacer, y que no es capaz de comprender mensajes subjetivos. Es por ello, que organizar la información de forma clara y en el orden correcto es primordial para que las instrucciones de un programa sean ejecutadas de manera adecuada.

Como ha dicho anteriormente las instrucciones que da a las computadoras deben estar organizadas y planificadas en algoritmos. Un algoritmo es una secuencia de instrucciones para la ejecución de una tarea. Es decir, puede decir que una receta de cocina es un algoritmo, ya que, indica todos los procedimientos a realizar para llevar a cabo una tarea. La información en un algoritmo no puede ser redundante ni subjetiva. Tiene que ser clara y concisa para que la computadora pueda interpretarlo correctamente. Vea un ejemplo:

Si a un humano le explican cómo hacer una tortilla francesa, simplemente, le puede dar las siguientes instrucciones.

- Coja dos huevos.
- Bata los dos huevos.
- Échelos a la sartén cuando esté caliente.

Este algoritmo sería incomprendible para una máquina porque dentro de esas instrucciones, aunque para usted sean claras, son ambiguas y subjetivas para una máquina. Por lo tanto, una máquina entendería el siguiente algoritmo:

- Coja dos huevos.
- Coja un cuenco.
- Rompa los dos huevos.
- Échelos al cuenco.
- Coja un tenedor.
- Bata los huevos.
- Coja una sartén.
- Eche aceite en la sartén.
- Ponga la sartén a 90°C.
- Vierta el contenido del cuenco en la sartén.
- Espere 10 minutos.
- Dele la vuelta al contenido de la sartén.
- Espere 10 minutos.
- Retire el contenido de la sartén y sírvulos.
- Fin

Como puede ver, esta lista de instrucciones es mucho más detallada que la anterior y no deja ninguna instrucción a libre interpretación. Además, las acciones están descritas en una secuencia correcta y bien definida, porque sería imposible batir los huevos antes de romperlos y echarlos en un cuenco. Pues esta sería la interpretación de un algoritmo para que una máquina pudiera comprenderlo y ejecutar su tarea de forma correcta.

Hay diversas formas de representar un algoritmo. Para ello, más adelante verá lo que son los ordinogramas o diagramas de flujo y el pseudocódigo. Lo que diferencia al algoritmo que representa de manera gráfica y el algoritmo que la máquina entiende, es que uno está representado con lenguaje humano y otro, está representado con lenguaje de máquina o lo que es lo mismo, con código.



TOME NOTA

El flujo de un programa puede seguir diferentes caminos y convertirse en un algoritmo de gran complejidad. Esto dependerá del fin con el que el programa haya sido diseñado. Si un programa está diseñado para dar solo dos soluciones el algoritmo será muy sencillo, pero si, por el contrario, el programa está diseñado para dar 100 soluciones posibles el algoritmo tenderá a ser más complejo, ya que debe programar las 100 soluciones posibles.



Más Info

RECURSO MULTIMEDIA



1.1.1. Descripción y utilización de operaciones lógicas

Las operaciones lógicas son operaciones matemáticas de condición que proporcionan un resultado booleano, es decir, obtendrá un resultado de 0 o 1, o lo que es lo mismo verdadero o falso. Sus operandos, también, son valores lógicos o asimilables. Los operadores lógicos se pueden combinar entre ellos, por lo que dos o más operadores lógicos conformarían lo que denominaría una función lógica.

Estos operadores lógicos se pueden usar para combinar condiciones. Dependiendo del tipo de operador lógico tendrá operadores lógicos de búsqueda, de proximidad, etc. Los operadores lógicos de búsqueda agrupan diferentes términos con el fin de refinar una búsqueda. Por ejemplo, si tiene una base de datos y quiere buscar a Juan Gómez, no solo basta con buscar el término Juan, ya que podría obtener miles de resultados. Entonces, debe buscar ambos términos Juan y Gómez. En este curso verá los operadores lógicos de Javascript, ya que, todo lo que va a estudiar está relacionado con el desarrollo web. Aunque, la mayoría de los lenguajes de programación usan operadores lógicos similares.

Existen tres tipos de operadores lógicos AND (&&), OR (||) y NOT (!). A continuación, va a ver cómo funcionan los operadores lógicos, ya que, más adelante verá su funcionalidad en profundidad.

El operador lógico AND, o Y lógico, se escribe con la doble repetición del símbolo ampersand (&&), este símbolo se encuentra pulsando la tecla SHIFT + 6. La función de este tipo de operador es unir dos o más condiciones. Todas las condiciones que una este operador deberán ser verdaderas para que se cumpla la operación. Vea un ejemplo: Imagine que quiere comprar unos pantalones que sean vaqueros, largos y azules. Pues tendría tres condiciones que representaría como: vaqueros && largos && azules. Si alguna de estas condiciones no se cumple, entonces no compraría el pantalón. Pues es así como funciona en programación el Y lógico.

El operador lógico OR, u O lógico, se escribe con la doble repetición del símbolo al que se denomina barra vertical o tubería (||), este símbolo se encuentra pulsando en el teclado la tecla ALT GR + 1. La función de este tipo de operador lógico es conectar dos o más condiciones, pero esta vez no tienen

por qué cumplirse todas las condiciones, ya que, este operador es disyuntivo. Volviendo al ejemplo anterior de los pantalones, pero en este caso va a centrarse solo en que sean largos o cortos. Pues Imagine que está buscando unos pantalones cortos o largos y no importa cuál encuentre, solo importa que sean pantalones. Lo representaría de la siguiente manera: vaqueros && cortos || largos && azules. Esta operación buscaría unos vaqueros azules, con la diferencia de que pueden ser largos o cortos. Es decir, esta operación se cumplirá cuando uno de ambos resultados se cumpla.

El operador lógico NOT, o NO lógico, se escribe con el símbolo de admiración de cierre (!), este símbolo se encuentra pulsando la tecla SHIFT + 1. Este operador es un poco más complejo que los dos anteriores, ya que, como verá más adelante se puede combinar con diversos operadores de asignación e incluso con funciones. Este operador niega el resultado de una expresión, es decir, si una expresión tiene, de forma natural, como resultado verdadero al añadir el operador NOT obtendrá el resultado contrario. Vuelva a retomar el ejemplo anterior. Quiere unos pantalones, pero no quiere que sean vaqueros, por lo tanto si escribe pantalones = vaqueros, obtendría unos pantalones vaqueros, pero si altera la expresión y escribe pantalones != vaqueros, estaría diciendo que no quiere unos pantalones vaqueros sino de otro tipo. A continuación, vea otro ejemplo de forma numérica. Por ejemplo, si tiene la expresión 1 = 8, obtendría una solución falsa, ya que, 1 no es igual a 8. Pero, si por el contrario le añade a esa expresión una negación lógica, 1 != 8, obtendría una expresión verdadera. Como puede ver, la negación lógica niega el resultado natural de una expresión.



Los operadores lógicos se pueden combinar entre ellos y entre otros tipos de operadores, como los de comparación, asignación, etc. Estas operaciones, también, pueden combinar dos o más condiciones u operandos.

TOME NOTA

1.1.2. Secuencias y partes de un programa

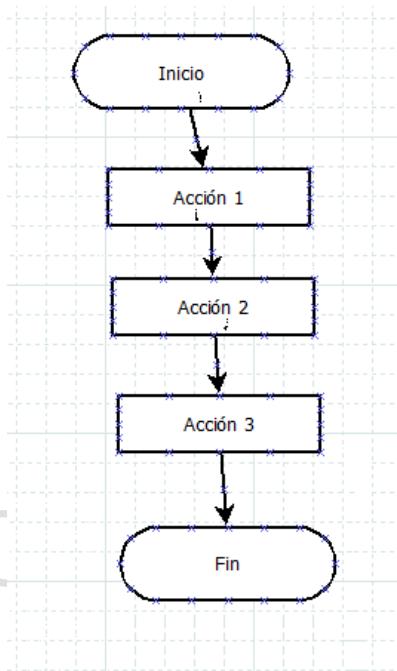
Como ha visto anteriormente la programación debe constar de una estructura ordenada y concisa. En la programación estructurada hay un inicio y un fin de las acciones perfectamente definido

Una parte muy importante de un programa, aunque pocas veces se le da el valor que debería tener, son los comentarios. Los comentarios son líneas de texto que el programador agrega a su código y que sirven para aclarar partes del código para que en un futuro sea más fácil su lectura. Estas líneas no son consideradas como parte del código, por lo tanto, el compilador no las interpretará como tal.

La programación estructurada parte de tres reglas: la secuencia, la decisión o condición y la iteración o ciclo.

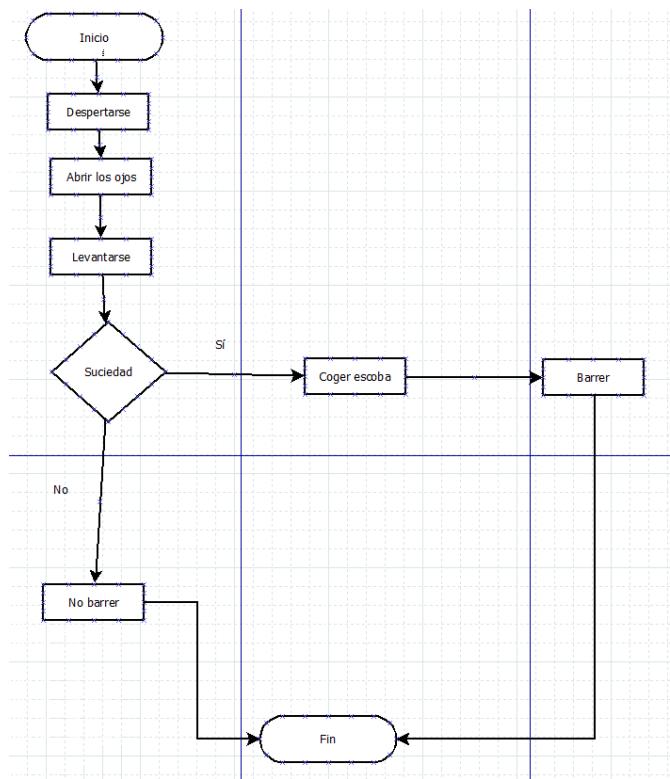
Secuencia: Cuando se habla de secuencia se está refiriendo al hecho de que las instrucciones del programa van a leerse de principio a fin, desde la primera línea de código hasta la última. A esto, también, le puede llamar flujo secuencial. El flujo secuencial de un programa se ejecuta de arriba hacia abajo, pero normalmente este flujo es casi inexistente, ya que, la mayoría de programas que conoce hoy día hacen llamadas a funciones y, por lo tanto, se crean saltos en la secuencia del programa. Resumidamente, las funciones son trozos de código que contienen instrucciones que se ejecutan solo cuando una acción llama a esta función. Esto facilita la creación de código, ya que, hace ahorrar líneas de código al no tener que repetir esa estructura una y otra vez dentro de su código.

Más adelante verá en profundidad qué son las funciones y puede entender mucho mejor cómo y por qué se realizan este tipo de saltos en la secuencia de un programa. En la siguiente imagen puede ver cómo se ejecutaría secuencialmente un programa:



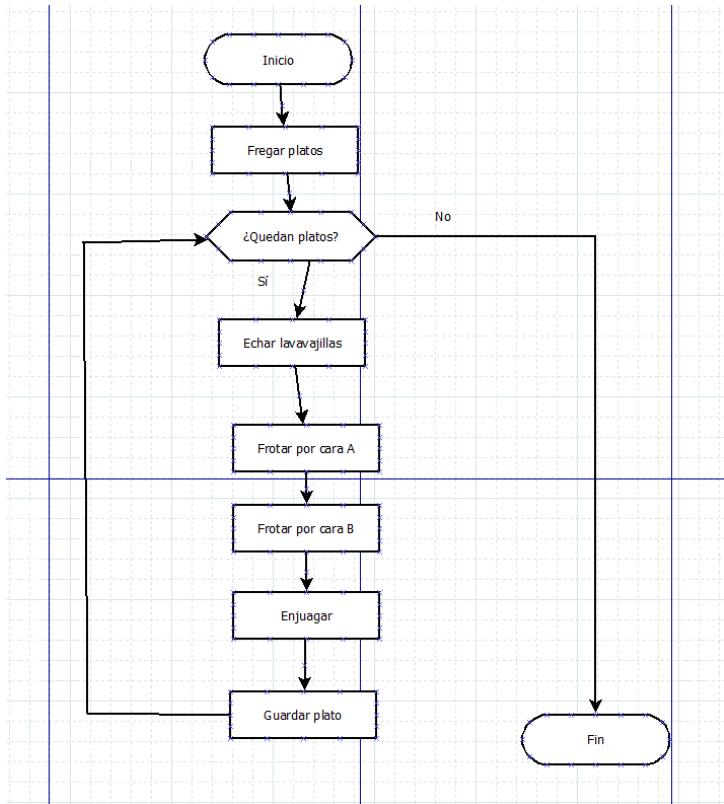
En esta imagen puede ver gráficamente cómo el flujo del programa va ejecutando secuencia por secuencia hasta llegar al final. Como ha visto anteriormente, el flujo del programa siempre va a ir de arriba hacia abajo, a no ser que alguna estructura del programa cambie su flujo.

Condición: Cuando se habla de decisión o condición se hace referencia a la estructura de la programación, está indicando que según ciertas condiciones dadas se ejecutará o no un conjunto de instrucciones. Estas condiciones permiten dividir su código en ramificaciones, las cuales cambiarán el flujo de ejecución dependiendo de si las instrucciones dadas son ciertas o no. En el siguiente ejemplo puede ver gráficamente cómo funciona la condición dentro de un código estructurado:



Como puede ver, la acción que representa este diagrama de flujo es un conjunto de acciones que llegan a una condición. Una vez llega a esa condición, el flujo del programa se bifurcará dependiendo de la condición, si hay suciedad se cogerá la escoba y se barrera y si no, se terminará el programa.

Iteración: La iteración o ciclo son estructuras cíclicas que permiten repetir un conjunto de instrucciones una cantidad de veces determinada o indeterminada. Esto dependerá de la condición que le otorgue. Esta estructura, también llamada loop en inglés, es una estructura de control dentro de un algoritmo que ordena que ciertas instrucciones se repitan una y otra vez hasta que se cumpla una condición. Vea un ejemplo gráfico para entenderlo mejor:



Como puede observar en la imagen, hay un conjunto de acciones hasta llegar a la iteración. Una vez llega a la iteración, lo que hará el programa es repetir todas las acciones que están dentro de la iteración una y otra vez hasta que la respuesta a la pregunta *¿Quedan platos?* Sea negativa. Los ciclos o iteraciones te ayudan a programar tareas repetitivas sin necesidad de escribir para la misma acción una y otra vez el código.



TOME NOTA

Las iteraciones son muy parecidas a las condiciones. Sin embargo, se diferencian en que las condicionales son instrucciones de control donde dependiendo si la condición se cumple o no, toma un camino u otro, por otra parte, las iteraciones se usan para repetir una acción o un bloque de acciones un número indeterminado de veces hasta que la condición se cumpla o no se cumpla (dependiendo de cómo la haya programado). Para ello, en cada ciclo se validará la condición.

1.2. Ordinogramas

Los ordinogramas o diagramas de flujo nacen de la necesidad de plasmar de manera gráfica el diseño de un programa, el flujo de datos manipulados por el mismo y la secuencia lógica de las operaciones que constituyen el algoritmo. Para ello, se utilizan herramientas especializadas en este tipo de tarea como, por ejemplo, el programa Dia, que es el que utilice para crear los diagramas de flujo.

Antes de adentrarse en cómo realizar un ordinograma, verá las cualidades de un algoritmo, ya que, antes de poder diseñar gráficamente cómo funciona un algoritmo, tiene que saber cómo encontrar la

solución más óptima y que cumpla más fielmente las cualidades deseables de un algoritmo bien diseñado. Un algoritmo debe ser:

- **Finito.** Un algoritmo siempre tiene que finalizar tras un número finito de acciones. Cuando el algoritmo *Dividir platos* sea un programa, siempre se ejecutará de la misma manera, ya que, siempre se seguirán las mismas acciones descritas en el cuerpo del algoritmo, desde la primera hasta la última, una por una y en el orden establecido.
- **Preciso.** Todas las acciones descritas en un algoritmo deberán estar bien establecidas y definidas, es decir, ninguna de estas acciones puede ser ambigua, por lo que solo se podrán interpretar de una única manera. Es por ello que, si ejecuta el programa varias veces con los mismos datos de entrada, debe obtener los mismos datos de salida.
- **Claro.** Normalmente en programación un problema se puede resolver de distintas formas. Por lo que, una de las tareas más importante para un diseñador de algoritmos es encontrar la manera más legible para el ser humano de encontrar la solución al problema.
- **General.** Un algoritmo debe servir para resolver problemas generales. Por ejemplo, el programa *Dividir* no solo deberá servir para dividir dos números determinados, sino que deberá servir para dividir cualquier número.
- **Eficiente.** El programa resultante de codificar un algoritmo deberá consumir los mínimos recursos posibles de la computadora. A esto también se le llama optimización.
- **Sencillo.** Siempre hay que intentar encontrar un equilibrio entre claridad y eficiencia, ya que, a veces encontrar la solución algorítmica más eficiente puede resultar en un algoritmo muy complejo y esto afecta a la claridad del mismo. Es por ello, que hay que intentar que la solución de este algoritmo sea sencilla, aunque pueda perder un poco de eficiencia del mismo. Escribir algoritmos eficientes, claros y sencillos se consigue a base de práctica.
- **Modular.** No hay que olvidarse de que un algoritmo, normalmente, va a formar parte de la solución a un problema mayor y que, a su vez, este algoritmo puede descomponerse en otros, siempre y cuando esto favorezca la claridad del mismo.



Más Info

0

RECURSO MULTIMEDIA



1.2.1. Descripción de un ordinograma

Los ordinogramas o diagramas de flujo engloban tanto la representación gráfica de la circulación de los datos, como la representación gráfica de la secuencia de operaciones que se deben de realizar dentro del mismo. En la fase de programación es el programador el que crea un ordinograma para cada programa y, a partir de él, se realizará la codificación en el lenguaje de programación correspondiente.

Un ordinograma que representa un programa, debe representar claramente algunos de los elementos esenciales del mismo:

- Comienzo del programa.
- Operaciones.
- Secuencias en las que se realizan las acciones.
- Final del programa.

Antes de construir un ordinograma se deberá tener en cuenta los siguientes puntos:

- Hay que identificar las ideas principales e incluirlas en el diagrama de flujo.
- Hay que definir qué es lo que se espera obtener del diagrama de flujo.
- Identificar quién lo usará y cómo.
- Establecer el nivel de detalle requerido.
- Y, por último, determinar los límites del proceso a describir.

Para construir un ordinograma correctamente debe seguir las siguientes reglas:

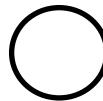
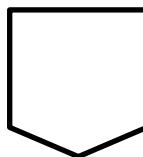
- Debe existir un inicio y un fin.
- El orden del flujo de datos será de arriba hacia abajo y de izquierda a derecha.
- Se aconseja utilizar un símbolo por acción.
- Dentro de los símbolos no se escribirán instrucciones propias de un lenguaje de programación. Así conseguirá que la representación gráfica del programa sea independiente del lenguaje de programación y, por lo tanto, pueda codificarlo en cualquier lenguaje.
- La secuencia se indica con flechas.
- A todos los símbolos, excepto al de inicio llega una sola línea.
- De todos los símbolos, excepto los de fin y decisión, sale una sola línea.
- Se evitará que los conectores tengan líneas demasiado largas. Puede haber varios conectores de salida con la misma etiqueta, pero solo uno de entrada con la misma etiqueta.
- Se usarán símbolos de conexión para facilitar la visualización del diagrama.
- Las conexiones de entrada tendrán un símbolo de conexión con una flecha que va hasta el diagrama de flujo.
- En las conexiones de salida saldrán una flecha del diagrama de flujo que llegará hasta el símbolo de conexión de salida.
- Por cada conexión de entrada podrá haber una o más de salida.
- Las conexiones de entrada y salida estarán etiquetadas para aclarar el enlace.

- El conector circular también se usa para unir todas las vías que genera todas las condiciones en un único hilo de control.

1.2.2. Elementos de un ordinograma

Los elementos de un ordinograma siguen un estándar oficial que regula el significado de los símbolos utilizados para representar gráficamente los algoritmos. Este estándar lo estableció el Instituto Nacional Estadounidense de Estándares (ANSI) en 1960 y, posteriormente, la Organización Internacional de Normalización (ISO) adoptó estos símbolos en 1970. En este apartado va a ver los principales elementos de un ordinograma:

Forma ANSI/ISO	Nombre	Disposición
	Terminal	Este símbolo indica el Inicio o el fin de un programa. Tiene forma de rectángulo redondeado. En su interior lleva escrita la palabra Inicio o Fin dependiendo de para qué se esté usando.
	Línea de flujo o flecha	Este símbolo muestra el orden de operación de los procesos. Las líneas de flujo se representarán saliendo de un símbolo y apuntando a otro.
	Tratamiento o Proceso	Esta figura, representada como un rectángulo, simboliza todas las variedades de procesos, es decir, procesos que modifiquen el valor original. Por ejemplo, la suma de dos variables.
	Entrada / Salida	Esta figura, representada por un paralelogramo, simboliza la entrada y salida de datos, es decir, ingresar datos o mostrar resultados.
	Decisión	Esta figura, representada por un rombo, muestra una operación condicional que determinará cuál de los dos caminos seguirá el programa. Este tipo de operación suele ser lógica, es decir, que los resultados pueden ser verdadero o falso.

	Preparación	Esta figura indica que hay que comprobar algo en el proceso antes de proceder. Normalmente, esta figura se utiliza cuando quiere representar un bucle FOR.
	Subrutina o proceso predefinido	Este símbolo se utiliza cuando quiere llamar a un proceso que ya ha sido definido previamente. Por ejemplo, el proceso Área del cuadrado, multiplicaría la base por la altura. Pues si quiere más adelante llamar al proceso Área del cuadrado sin tener que volver a repetir el proceso, utilizaría este símbolo en su lugar.
	Conejero o conector interno	Este símbolo indica que habrá un reagrupamiento de una instrucción de control.
	Conejero externo	Este símbolo indica un enlace entre dos partes de un diagrama en dos páginas diferentes.
	Entrada por teclado	Este símbolo indica que el programa debe esperar a que el usuario introduzca un dato que se guarda en una variable.
	Impresora	Este símbolo indica la salida de datos que serán leídos por el usuario. Por ejemplo, se usaría para pedir usuario y contraseña en un login.



TOME NOTA

Estos son los símbolos básicos para empezar a componer un diagrama de flujo. Hay muchos más símbolos que se pueden implementar para alcanzar mayor número de detalle en la creación de un diagrama de flujo.

1.2.3. Operaciones en un programa

En este apartado va a ver cómo plasmar en un diagrama de flujo las instrucciones, operaciones y estructuras más utilizadas en la creación de un programa.

Procesos. Los procesos en programación son instrucciones que indican al sistema la operación u operaciones que tiene que realizar tomando unos datos determinados. Existen una gran variedad de operaciones, que verá a continuación. Entre ellas están las operaciones aritméticas, operaciones de asignación, operaciones de comparación, lógicas, etc. Los procesos en un diagrama de flujo, están representados por un rectángulo, el cual lleva en su interior la instrucción a seguir.

Instrucción

Instrucción de asignación. Las instrucciones de asignación consisten en asociar a una variable el resultado de la operación de una expresión. Cuando quiere representar este tipo de instrucción en un diagrama de flujo, crea un rectángulo y, dentro de él, escriba la expresión y la variable donde se guardará el resultado.

En una asignación, la variable debe ser del mismo tipo que la expresión asignada.

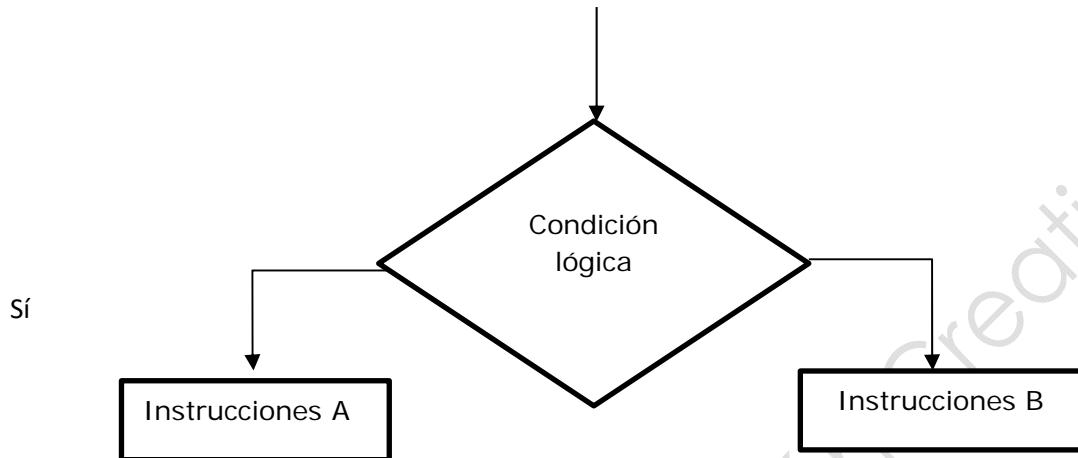
Variable ← Expresión

Instrucciones de entrada / salida. En un ordinograma, las instrucciones de entrada y de salida se representan escritas en el interior de un paralelogramo. Las instrucciones de entrada son instrucciones que asignan a una o más variables valores recogidos del exterior, ya sea a través del teclado, ratón, etc. Sin embargo, los valores de salida consisten en llevar hacia el exterior los datos obtenidos de la evaluación de expresiones. Estos datos se envían a través de la pantalla normalmente.

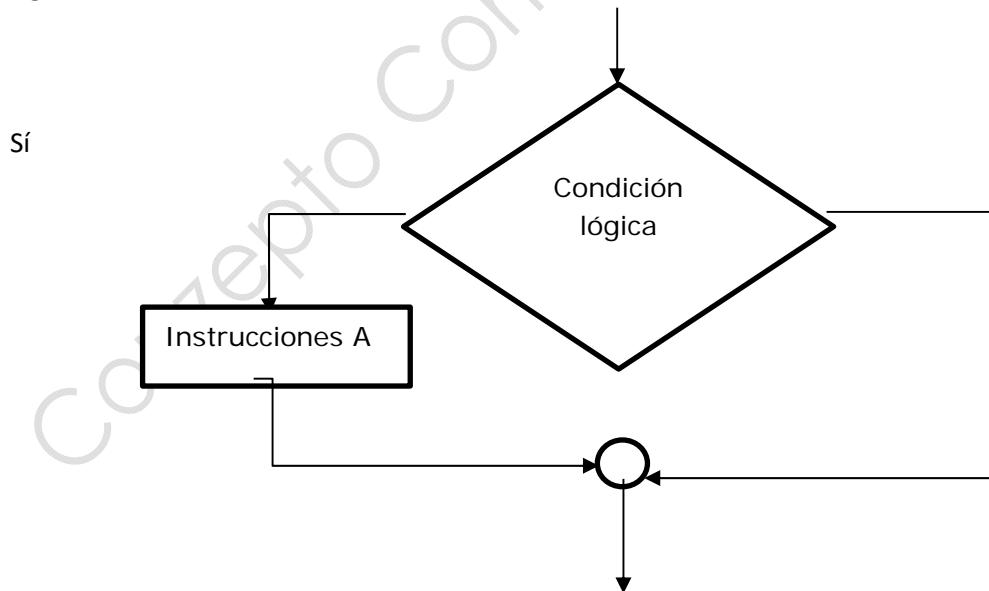
Ler
(variables)

Escribir
(expresión)

Instrucción de decisión o condición (IF / ELSE). Las condiciones o decisiones son expresiones que devuelven un valor booleano, es decir, verdadero o falso. Dependiendo de este valor, se ejecutará un bloque de instrucciones u otro. En un ordinograma se representa esta instrucción de la siguiente manera.

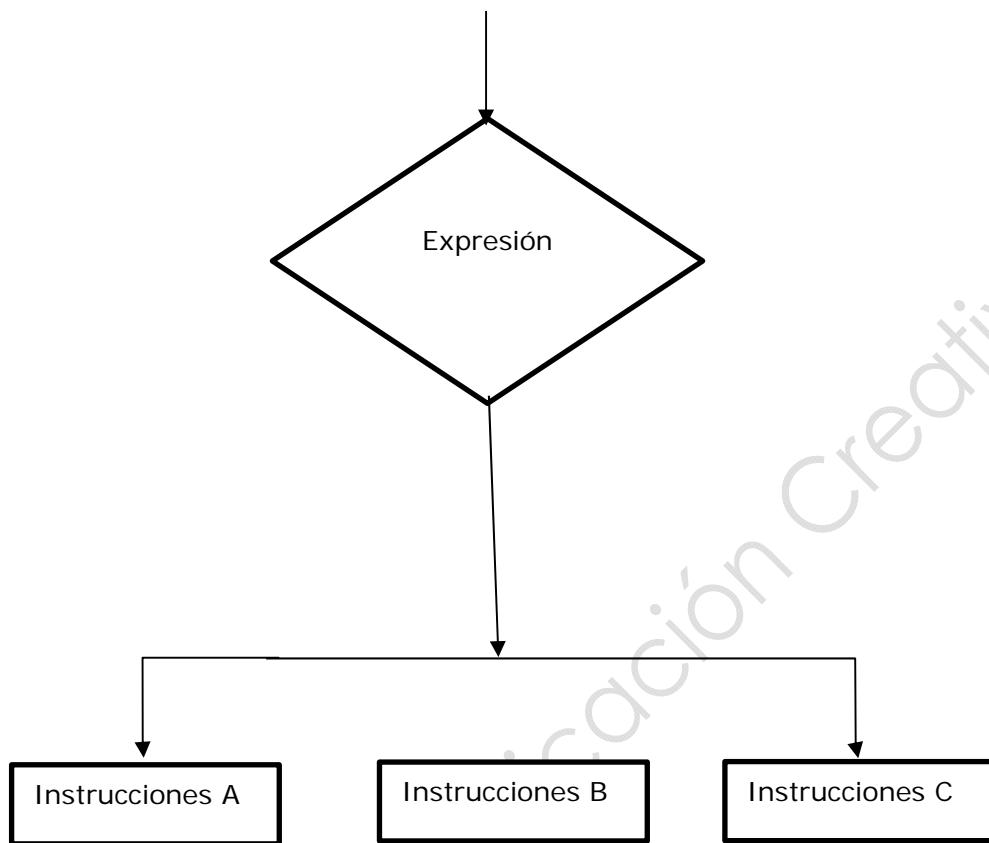


Instrucción condicional simple (IF). Este tipo de instrucción es una variante de la instrucción anterior. En este tipo de instrucción si en la evaluación de la expresión lógica devuelve un resultado verdadero, se comenzará a ejecutar el bloque de instrucción, en cambio, si devuelve un resultado falso, el programa seguirá ejecutándose con naturalidad. Esta instrucción se representa de la siguiente manera.

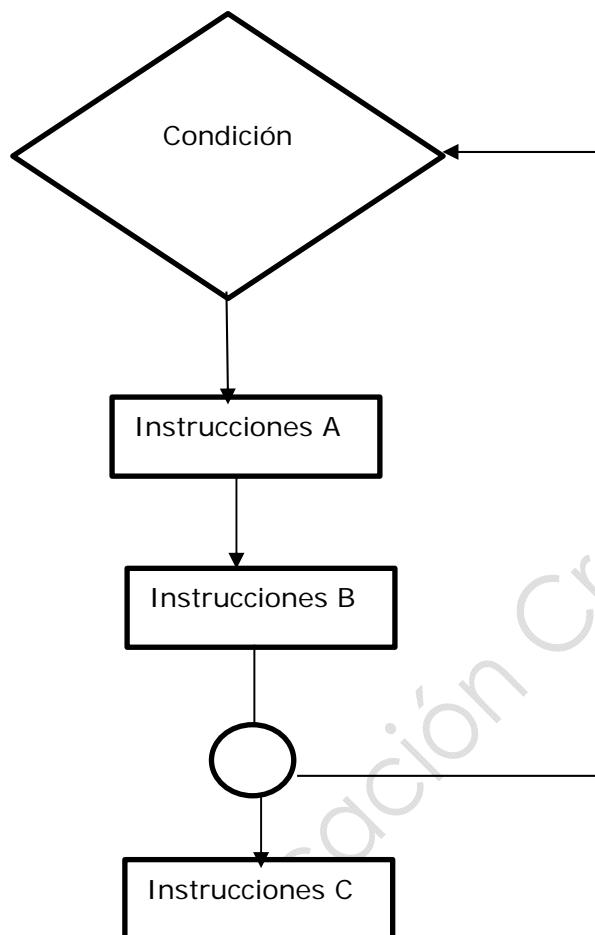


Instrucción de selección múltiple (SWITCH). La instrucción de selección múltiple puede albergar múltiples opciones, como su nombre indica, partiendo de la evaluación de una misma expresión. Esto

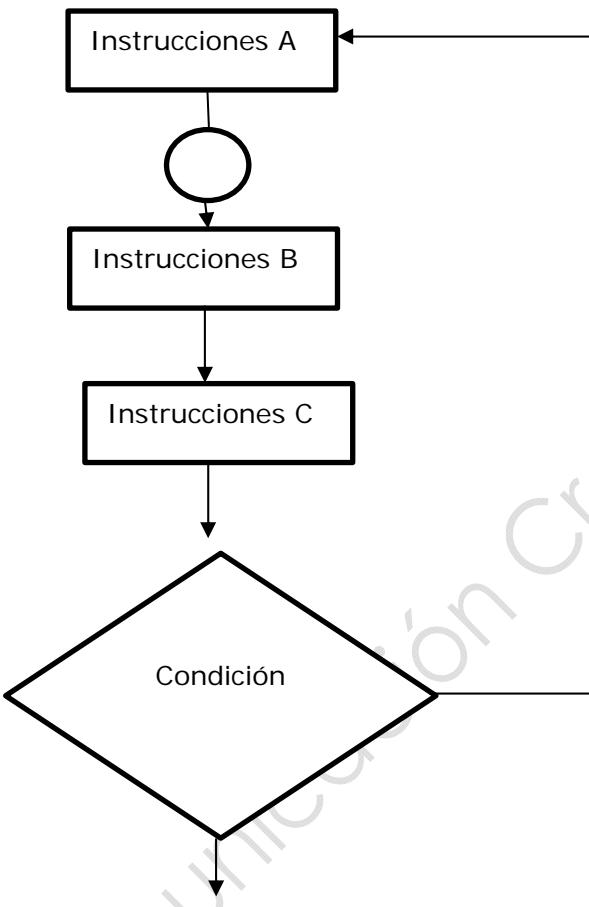
en programación se escribiría como Switch, mientras que las opciones serían case 1, case 2, case 3, etc. En un ordinograma se representa de la siguiente manera.



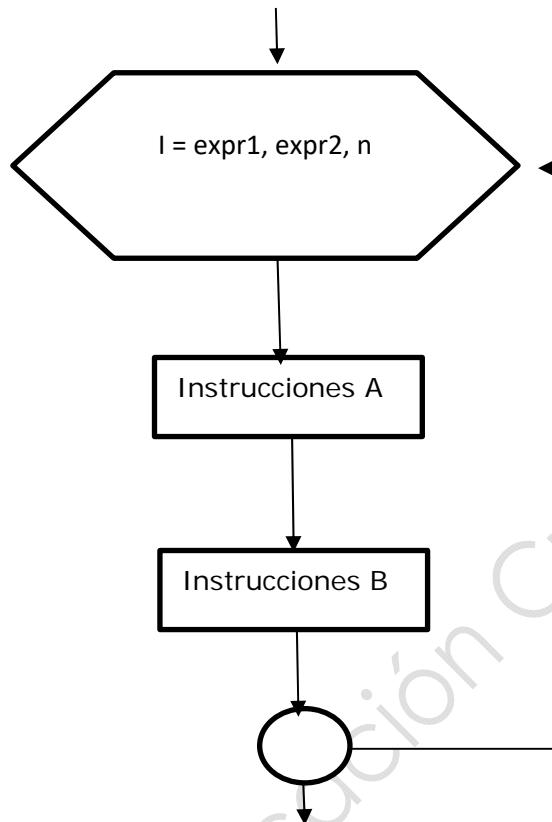
Estructura iterativa o bucle WHILE. Este tipo de bucle se utiliza cuando realmente no sabe el número de iteraciones que se van a realizar, sin embargo, sí sabe qué condición debe cumplirse, por lo que, si la condición es verdadera, se ejecuta el bloque de sentencias, en cambio, si la condición es falsa el programa no llega a entrar en el bucle y continuaría ejecutando las Instrucciones C. En este tipo de bucle no existe ninguna variable que muestre el número de vueltas que ha realizado o falta por realizar. El único control que tendrá es la condición que está al comienzo del bucle. Así pues, lo primero que se realiza es la comprobación de la condición al principio y cada vez que realice una iteración. Este tipo de bucles es uno de los más utilizados en programación. Se representa de la siguiente manera.



Estructura iterativa o bucle DO WHILE. Este bucle se diferencia de los bucles WHILE y FOR porque en estos bucles la condición se evalúa al principio del bucle, por lo que, si la condición es falsa, el bucle no se realiza. Sin embargo, en el bucle DO WHILE, la condición se evalúa al final del bucle, esto permite que el bucle realice al menos una iteración. Como pasa con los bucles WHILE, en este bucle tampoco se sabe qué número de iteraciones se va a realizar, ya que, depende de la condición que se encuentra al final de bucle. Si la condición es verdadera, se vuelve a realizar otra iteración, en cambio, si es falsa el programa sale del bucle. Este tipo de bucles se utilizan, por ejemplo, cuando una entrada de información deba cumplir una serie de condiciones y no quiera que el programa continúe hasta que se cumplan. La representación algorítmica del bucle DO WHILE es la siguiente.



Estructura iterativa o bucle FOR. Los bucles FOR se utilizan cuando sabe qué número de iteraciones debe realizar el bucle. Para ello, se utiliza una variable de tipo entero como contador. Cuando se declara un bucle FOR debe especificar el valor inicial de dicha variable, el número de vueltas que va a dar y el incremento o decremento que se deberá añadir en cada vuelta. A la variable contador del bucle FOR, normalmente, se le denomina i. Por ejemplo: i = 0,5,1. El primero de los parámetros, que es 0, indica que la variable comenzará a contar desde 0. El segundo parámetro, indica el número de vueltas que dará el bucle, en este caso son 5. Y, el tercer parámetro indica que por cada vuelta se incrementará en 1 el valor inicial. Así en la primera vuelta la variable i valdrá 1, en la segunda 2, y así sucesivamente. A continuación, verá cómo se representa de manera algorítmica el bucle FOR.



Es muy importante tener en cuenta que la finalizar en bucle FOR, el valor de i será $i = i + \text{incremento} / \text{decremento}$. Esto quiere decir que, si ha dado 5 vueltas y finaliza el contador, la variable i valdrá 6. Además, tiene que tener en cuenta que esta variable alterará su valor por cada iteración y que el valor anterior desaparecerá. Esto no ocurre con los demás bucles.

TOME NOTA

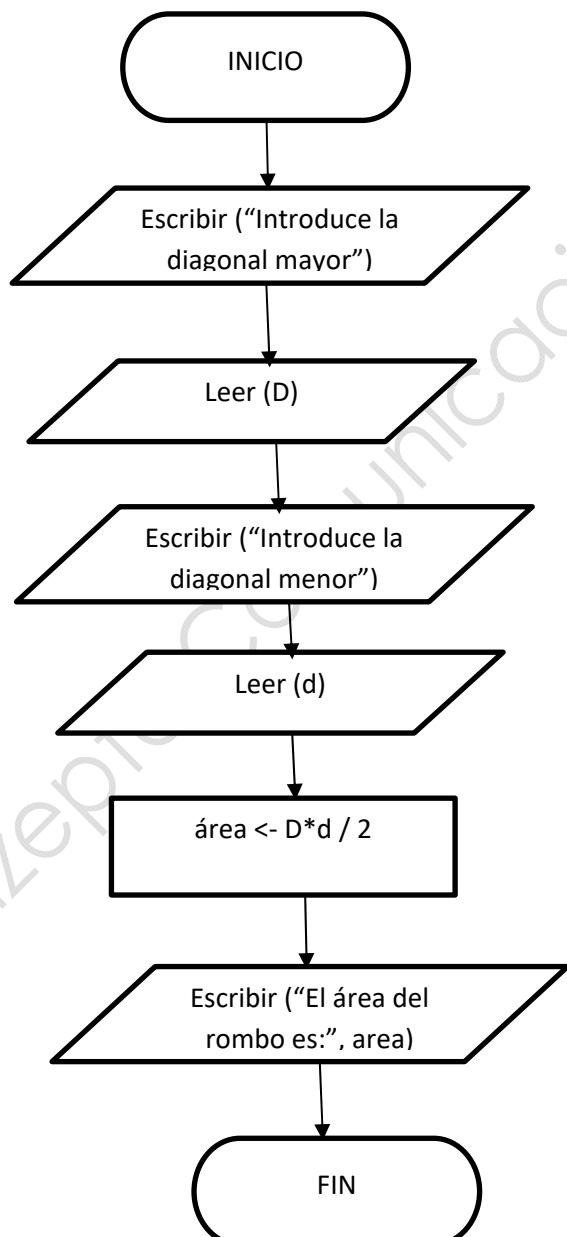
En una asignación, la variable debe ser del mismo tipo que la expresión asignada.

1.2.4. Implementación de elementos y operaciones en un ordinograma

En este apartado va a combinar todo lo aprendido sobre los ordinogramas. Y va a ver algunos ejemplos algorítmicos de programas reales.

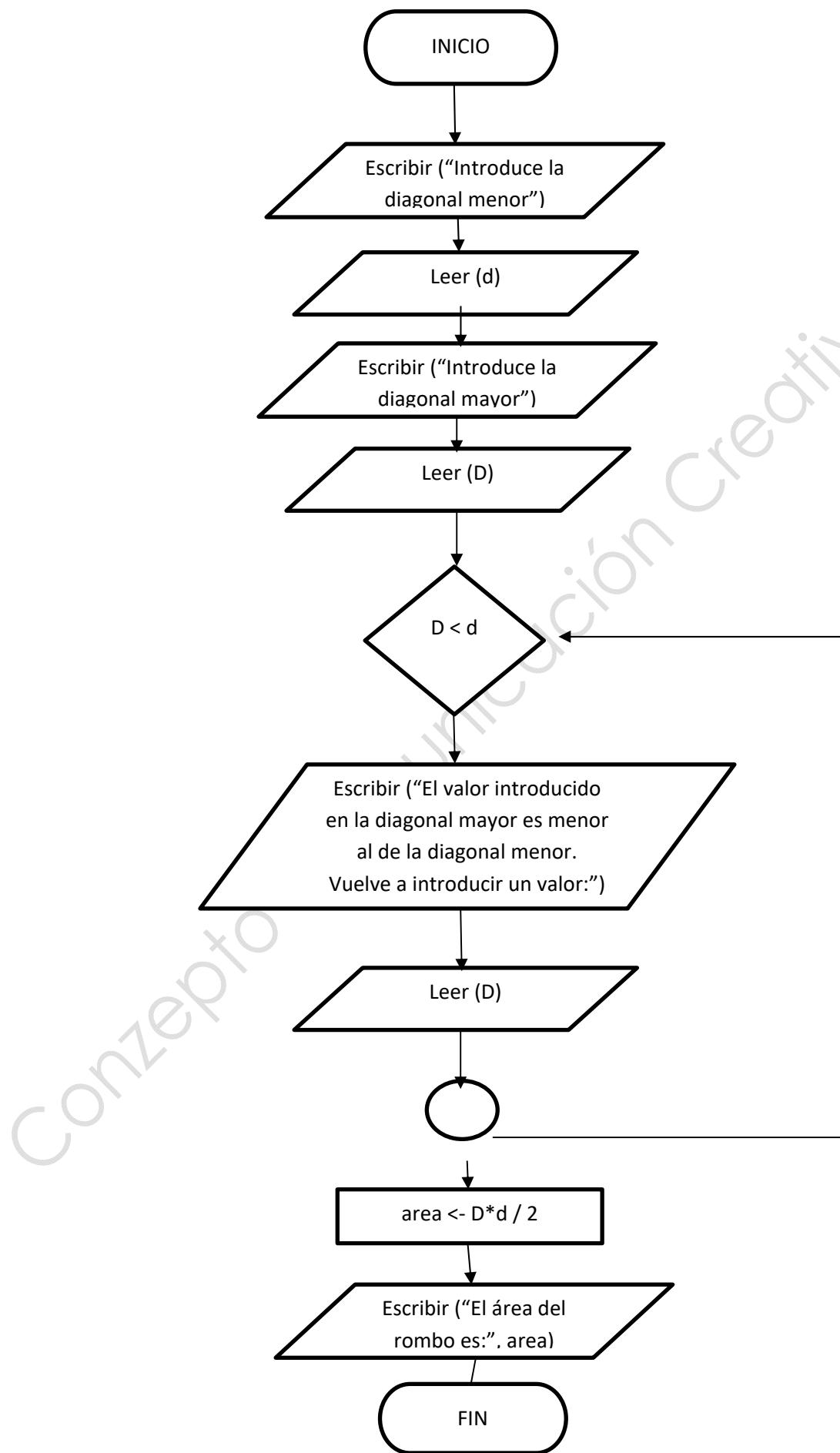
Va a crear un ordinograma que calcule el área de un rombo. Para ello, sigue los siguientes pasos:

- Pedir por teclado la diagonal mayor (D).
- Pedir por teclado la diagonal menor (d).
- Calcular el área del rombo.
- Mostrar por pantalla el resultado.



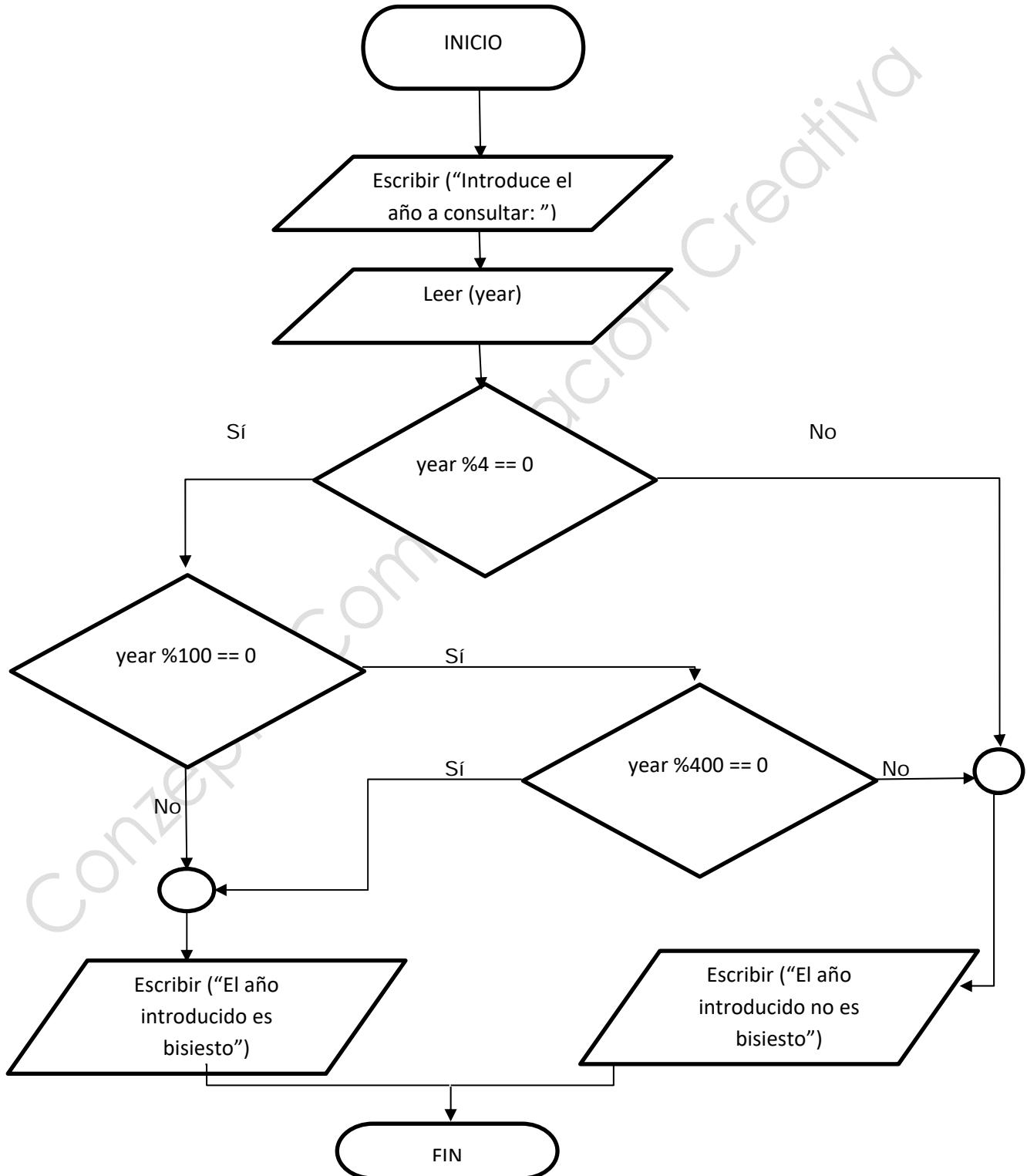
A continuación, va a crear el mismo programa, pero añadiéndole mayor complejidad. En el siguiente programa, va a hacer que la diagonal mayor deba ser un número mayor que el que haya escrito para la diagonal menor. Para ello, seguirá los siguientes pasos:

- Pedir por teclado la diagonal menor.
- Pedir por teclado la diagonal mayor.
- Si el resultado es mayor que la diagonal menor, continuará el programa.
- Si el resultado es menor que la diagonal menor, saldrá en pantalla un aviso.
- Calcular el área del rombo.



Por último, va a crear un programa que sea capaz de decir si el año que introduce es bisiesto o no. Los pasos a seguir son:

- Pedir por teclado el año.
- Leer el año.
- Si el año no es bisiesto, escribir no es bisiesto.
- Si el año es bisiesto, escribir es bisiesto.



1.3. Pseudocódigos

El pseudocódigo, como su nombre indica, no es un código de programación verdadero, sino una forma de expresar los distintos pasos que va realizar un programa de la manera más parecida al lenguaje humano. Con el pseudocódigo puede escribir de forma más clara y genérica las instrucciones que debe ejecutar un programa. Los ordinogramas junto con el pseudocódigo componen una herramienta muy útil que da la posibilidad de comprender de forma gráfica los conceptos que necesita entender para crear un algoritmo que resuelva un problema determinado.

Cuando está comenzando a aprender a programar es sumamente importante entender bien el concepto de pseudocódigo, ya que, antes de aprender algún lenguaje de programación, debe aprender la lógica de cómo se comunica una máquina y es eso, precisamente, lo que va a aportar el pseudocódigo, un método para visualizar la solución de un algoritmo de manera detallada.

El pseudocódigo es una manera clara y sencilla de expresar las distintas instrucciones que debe realizar un programa hasta alcanzar su objetivo, sin tener que utilizar un lenguaje de programación específico. Esto es muy importante, ya que, gracias al pseudocódigo no estaría limitando el diseño del algoritmo a las características propias de un lenguaje de programación determinado.



TOME NOTA

Una definición más acertada y exacta del pseudocódigo es que se trata de la definición de un algoritmo informático de programación de alto nivel compacto e informal que utiliza convenciones estructurales de un lenguaje de programación verdadero pero que a su vez es totalmente genérico.

Lo extraño del concepto de pseudocódigo es que depende de las convenciones de desarrollo de un lenguaje de programación, pero a la vez es independiente de él, ya que, elimina las limitaciones estructurales al no tener que pensar en cómo implementar una misma idea en diferentes lenguajes de programación. Podría comparar el pseudocódigo con un lenguaje universal que todo el mundo pudiera entender, pero que, a la vez, fuese independiente de los lenguajes nacionales.

El pseudocódigo, como tal, no puede ejecutarse en una computadora, ya que no está estructurado para que pueda ser comprensible para un ordenador pues, como se ha mencionado anteriormente, es un código falso. La principal utilidad de este pseudocódigo es que sea un código que pueda ser interpretado a simple vista por los humanos. Es por ello, que también se llama Lenguaje de descripción algorítmico.

Para que el pseudocódigo pueda ser comprendido por usuarios sin conocimiento alguno en programación, el pseudocódigo utiliza las convenciones estructurales de un lenguaje de programación general. Aunque, se suele omitir partes de la estructura como todo lo relativo a un lenguaje de programación específico, por ejemplo, las subrutinas, las variables, etc.

Por último, cabe destacar que el pseudocódigo es un lenguaje intermedio entre el lenguaje humano y el lenguaje de máquina.

RECURSO MULTIMEDIA



1.3.1. Descripción de pseudocódigo

En este apartado va a aprender a representar con pseudocódigo las instrucciones de un algoritmo. Para ello, parte de la base que ha aprendido sobre cómo representar los ordinogramas.

Instrucciones. A continuación, va a ver ejemplos de cómo se representan las instrucciones en pseudocódigo. Para ello, va a usar una sintaxis parecida al lenguaje Javascript, ya que, es el lenguaje de programación que usa generalmente para el desarrollo web. Una estructura secuencial sencilla se describiría en pseudocódigo de la siguiente manera:

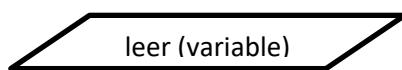
```
instrucción_1;  
instrucción_2;  
instrucción_3;  
instrucción_n;
```

Instrucción

Instrucciones de entrada / salida. Para representar la entrada y salida de datos, los datos se escriben de manera textual de la misma forma que hacía en los ordinogramas. Las instrucciones de entrada y salida de datos se representan de la siguiente manera:

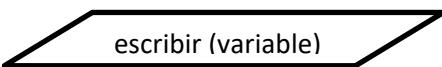
Instrucción de entrada:

```
leer (variable);
```



Instrucción de salida:

```
escribir (variable);
```



Como podrá haber comprobado, todas las sentencias terminan en punto y coma. Esto sucede porque está utilizando una sintaxis similar a la que utilizaría si programa con Javascript.

Operaciones de asignación. Para representar una operación de asignación puede hacerlo de dos maneras:

Variable=valor;

Variable <- expresión

O, también puede usar la siguiente expresión:

Variable <- valor;

Operaciones. Todas las operaciones que realiza en un programa deben estar almacenadas en variables, por lo que, tanto como en pseudocódigo como en un lenguaje de programación real, debe representar todos los tipos de operaciones con sus respectivas variables. A estas variables, le puede asignar el nombre que quiera. Las operaciones en pseudocódigo se representarán de la siguiente manera:

Multiplicación <- a*b;

División <- a/b;

Que también puede representar como:

Multiplicación = a*b;

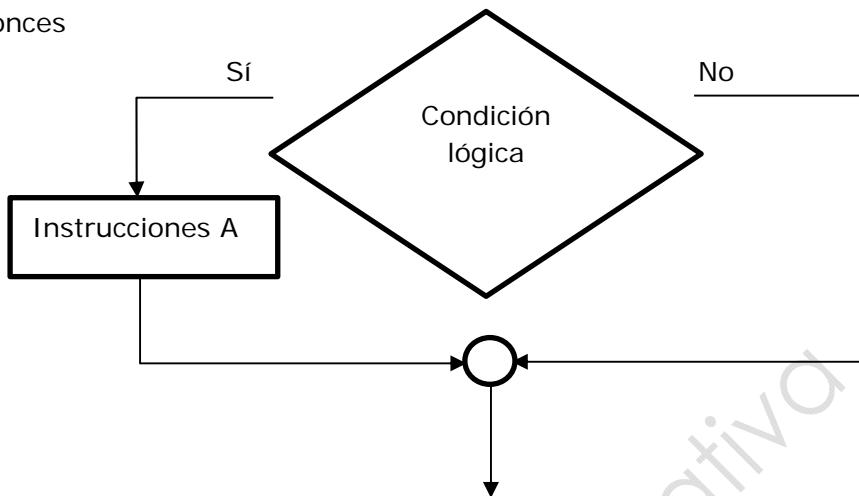
División = a/b;

Estructura IF. En las estructuras de condición y bucle, verá que la primera línea no termina en punto y coma como hasta ahora había hecho. Esto se debe a que las estructuras de decisión siempre van acompañadas de una condición y, a continuación, un bloque de instrucciones. Al final de ese bloque de instrucciones, debe indicar el final de la estructura. Cabe destacar que varias estructuras de decisión pueden anidarse entre sí. La estructura IF se representa en pseudocódigo de la siguiente manera:

Si (expresión lógica) Entonces

Instrucciones;

Fin Si



Estructura IF/ELSE. La estructura IF/ELSE deriva de la estructura IF. La diferencia que tiene esta estructura con la estructura IF, es que en esta estructura va a haber dos bloques de código donde uno se ejecutará y el otro no dependiendo de si la expresión lógica se cumple o no. Esta estructura se representa de la siguiente manera:

Si (expresión lógica) Entonces

Instrucciones;

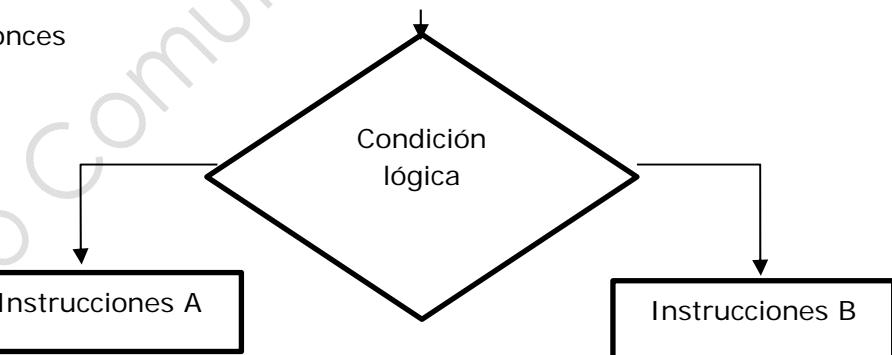
SiNo

Instrucciones;

FinSi

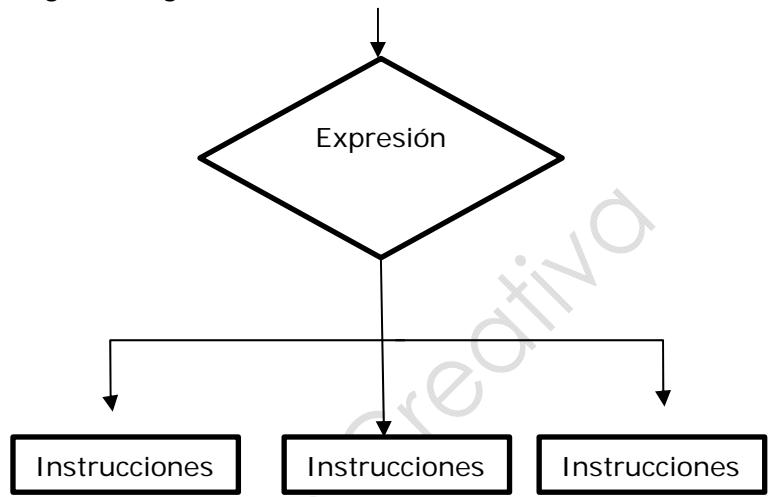
Instrucciones A

Instrucciones B



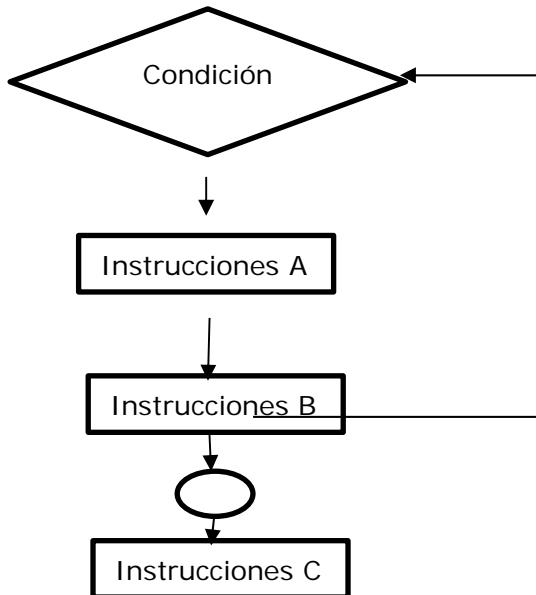
Estructura SWITCH. La estructura SWITCH es una estructura de selección múltiple en la que, tomando como base una variable, se elegirá una ruta específica. Si ninguna de las rutas satisface la correspondencia entre el valor de la variable y el selector, se ejecutará la acción por defecto. La estructura SWITCH se representa en pseudocódigo de la siguiente manera:

```
Según (variable) Hacer
    Caso valor_1
        Instrucciones_1;
    Caso valor_2
        Instrucciones_2;
    Caso valor_n
        Instrucciones_n;
    Por defecto
        Instrucciones_defecto;
Fin Según
```



Estructura WHILE. Recorde que la estructura WHILE se utiliza cuando no sabe cuántas vueltas debe hacer un bucle y, por lo tanto, es una estructura muy utilizada en programación. Representa la estructura WHILE en pseudocódigo de la siguiente manera:

```
Mientras (expresión lógica) Hacer
    Instrucciones;
Fin Mientras
```



Estructura DO WHILE. Esta estructura deriva de la estructura WHILE, la diferencia es que en este tipo de bucle se ejecuta el bucle al menos una vez. Este tipo de estructura se representa en pseudocódigo de la siguiente manera:

Hacer

 Instrucción 1;

 Instrucción 2;

 Instrucción n;

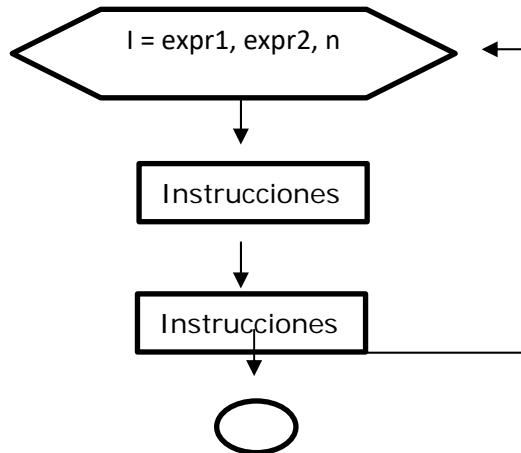
Mientras (expresión lógica);

Estructura FOR. El bucle FOR es una estructura de control muy común que se usa cuando se desea repetir un número repetido de veces un bloque de instrucciones. Para ello, se usa una variable que incrementa o decrementa en cada vuelta del bucle. La forma de representarlo en pseudocódigo es la siguiente:

Para i=x Hasta i=y Con i=i++/i—Hacer

 Instrucciones;

Fin para



Estructuras anidadas. Por último, va a ver cómo anidar alguna de las estructuras que ha aprendido. Las estructuras anidadas se utilizan cuando necesita que una estructura de control tome decisiones tomando como base más de una expresión lógica. Puede anidar todas las estructuras de control que necesite para que su programa sea funcional. Vea unos ejemplos a continuación:

Si (expresión lógica) Entonces

Instrucciones;

SiNo

Si (expresión lógica) Entonces

Instrucciones;

SiNo

Instrucciones;

FinSi

FinSi

Esto mismo, puede hacerlo con las demás estructuras de control. Incluso, puede usar varias estructuras diferentes anidadas entre sí.

1.3.2. Creación del pseudocódigo

Como ha visto anteriormente, la ventaja que tiene el pseudocódigo sobre los diagramas de flujo es que desarrollar un algoritmo en pseudocódigo ocupa menos espacio que en un diagrama de flujo. Otra de las ventajas que tiene es que permite representar de manera más fácil operaciones repetitivas y complejas. Además, pasar un algoritmo de pseudocódigo a un lenguaje de programación real es más sencillo, ya que, el pseudocódigo es más parecido al código real que un diagrama de flujo.

Otra parte importante del pseudocódigo es que permite representar de manera sencilla una función. Una función es un bloque de código identificado por medio de un nombre. Ese bloque de código queda “escondido” hasta que, posteriormente, llama a la función en la parte del programa que la necesite para que el programa la ejecute. Gracias a esto, no tiene que reescribir el código una y otra vez.

Va a ver un ejemplo de una función escrita en pseudocódigo:

Función factorial (número)

Si número === 0 Entonces

Devolver 1;

SiNo

Devolver número * factorial (--número)

Fin Si

Esta función calcula el factorial de un número. Si quisiera usarla en su programa, tan solo debe llamarla en la parte de código que desee. Vea un ejemplo:

Algoritmo_Cálculo

Var n;

Escribir "Introduce un número para calcular su factorial",n;

Leer n;

Escribir "El resultado es: n", factorial(n);

FinAlgoritmo

Por último, va a pasar el programa, que hizo anteriormente en diagrama de flujo, que calcula la diagonal de un rombo a pseudocódigo.

Algoritmo_Rombo

Var d,D;

Escribir "Introduce la daigonal menor",d;

Leer d;

Escribir "Introduce la diagonal mayor", D;

Leer D;

Mientras D<d Hacer

Escribir "El valor introducido en la diagonal mayor es menor al de la diagonal menor. Vuelve a introducir un valor", D;

Leer D;

FinMientras

Área = D*d/2;

Escribir "El área del rombo es: ", área;

FinAlgoritmo

1.4. Objetos

Para comenzar a hablar de objetos, antes debe conocer lo que es la POO, o lo que es lo mismo, la programación orientada a objetos. La programación orientada a objetos es una forma diferente de organización de un programa. Es decir, con el mismo lenguaje de programación puede programar como tradicionalmente se ha hecho, o puede decidir programar con una estructura de programación orientada a objetos. La POO nace por la necesidad que había a la hora de crear programas complejos, ya que, en la programación tradicional había que repetir una y otra vez los trozos de código que se necesitaba ejecutar en un mismo programa. Por ello, la estructura del programa se hacía difícil de controlar cuanto más grande era un programa.

La diferencia entre los métodos de programación tradicionales y los métodos de programación modernos es que antes se le daba más importancia a la lógica de la programación que a los datos, sin embargo, hoy día se les da más importancia a los datos que a lógica de programación.

Actualmente, el método más utilizado para crear programas es la POO, con ella los programas se modelan en torno a objetos que aglutan toda la funcionalidad relacionada con ellos. De esta manera, en vez de crear funciones inconexas unas con otras, crearía lo que se denomina clases. Este concepto de clases lo verá más adelante en profundidad.

Con más detalle se describe en el siguiente vídeo, de más de tres minutos de duración, qué es y en qué consisten los objetos:



1.4.1. Descripción de objetos

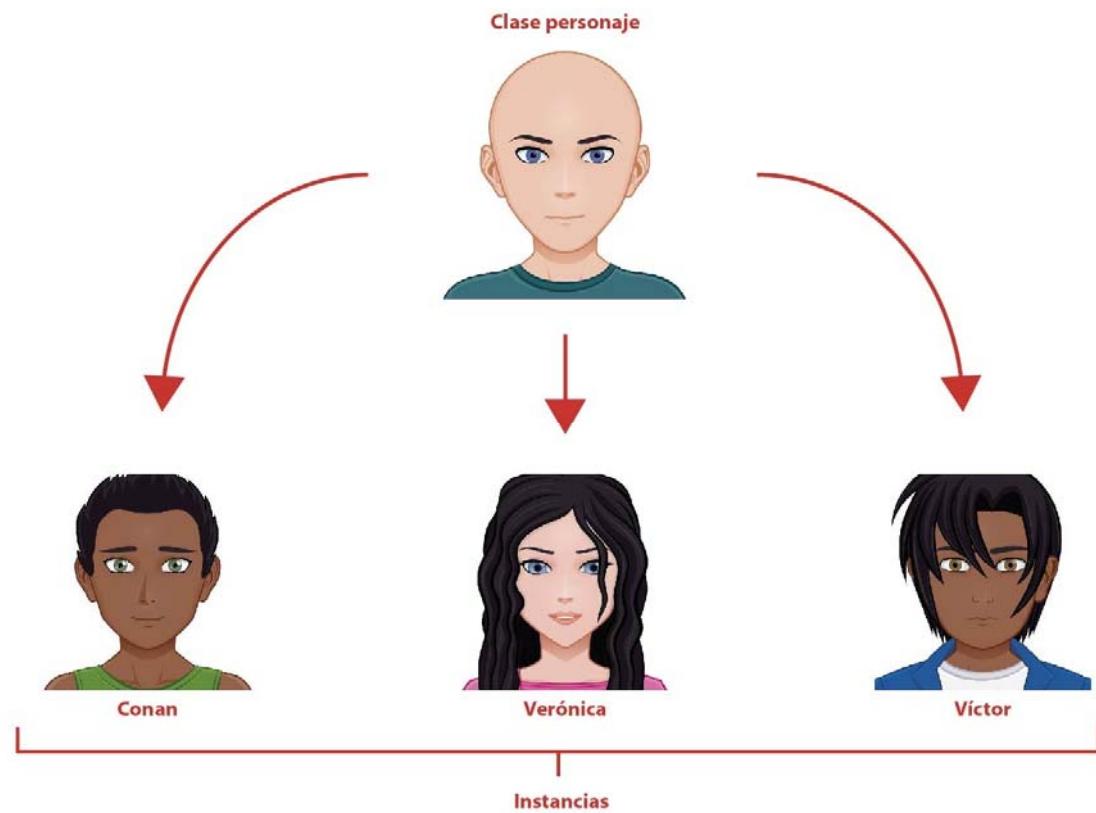
La POO resulta bastante confusa al principio de empezar a utilizarla, es por ello, que debe tener muy claro los conceptos de cómo estructurarla.

El primer concepto que debe tener claro es la diferencia entre clase y objeto.

Clase. Una clase es como una plantilla que define de manera genérica cómo van a ser todos los objetos de determinado tipo que se contienen en ella. Imagine que está programando un juego y que tiene que crear una clase que represente a los personajes. Pues, crearía la clase Personaje. Estos personajes tendrían una serie de atributos, o propiedades, como Nombre, Edad, Altura, Complexión,

y una serie de comportamientos que pueden tener como Caminar(), Hablar(), Saltar(). Estos comportamientos, se implementan como métodos de clase o funciones.

Objetos. Una clase por sí sola es, tan solo, un concepto sin entidad real. Para poder utilizar las clases en un programa hay que crear instancias. Instanciar una clase consiste en crear un nuevo objeto de la misma. Es decir, un objeto, por sí solo, es ya una entidad concreta que se crea a partir de la plantilla que es la clase. Por lo tanto, este nuevo objeto ya tiene existencia real, pues ocupa memoria y puede ser utilizado en el programa. En este caso un objeto sería un personaje que se llama Conan de 25 años que mide 1,90 metros y tiene complexión atlética. Este personaje podría hablar, saltar y caminar, ya que son los comportamientos que están definidos en esta clase. Vea un ejemplo gráfico:



Como puede ver, la clase define de forma genérica cómo son los personajes y, en cambio, los objetos son personajes concretos, con sus propios rasgos. Poder manejar los datos y comportamientos de cada objeto de manera independiente evita tener que coordinar datos de manera global. Es por ello que, la POO es muy potente porque permite modelar de manera sencilla datos y comportamientos del mundo real.

1.4.2. Funciones de los objetos

La función más importante de los objetos es la capacidad de poder representar cosas del mundo real. Cualquier cosa del mundo real es susceptible de ser modelada como un objeto. Desde una persona, un automóvil, o, incluso, cosas abstractas como un proceso.

Para poder manejar de manera eficiente las clases y los objetos, antes, tiene que aprender los cuatro pilares fundamentales de la POO.

Encapsulación. La encapsulación es la característica que permite que todo lo referente a un objeto quede aislado dentro de este. Es decir, que todos los datos referentes a un objeto queden encerrados dentro de este y solo se pueda acceder a ellos a través de los miembros que la clase proporcione, los cuales son las propiedades y los métodos.

Si coge el ejemplo anterior de los personajes, toda la información sobre estos y cualquier dato interno que se utilice y que no, necesariamente, se ve desde exterior del objeto, está circunscrito al ámbito de dicho personaje. Por lo tanto, si internamente tiene un dato, que es el nombre del personaje, accede a él a través de la propiedad pública Nombre que define la clase que representa a los personajes. Así es como da acceso solo a los datos que interese y de la manera que interese.

Abstracción. La abstracción está muy relacionada con el concepto anterior. El principio de abstracción implica que la clase debe representar las características de la entidad hacia el mundo exterior, pero ocultando su complejidad. Es decir, puede usar una serie de atributos y comportamientos que tiene el objeto, pero sin preocuparse de qué pasa dentro. Por lo tanto, en una clase se debe exponer para su uso, solo, lo que sea necesario. A un programa le es irrelevante lo que ocurra dentro de la clase al usar los objetos.

Ponga un ejemplo con los personajes de su juego. Imagine que tiene un dato interno que llame energía y que no es accesible desde fuera. Sin embargo, cada vez que su personaje anda, corre o salta gasta esa energía y el valor de ese dato disminuye. En este caso el programa solo tendría acceso a un objeto llamado Conan y llamaría a la función Saltar(), por lo que, lo que ocurra dentro de la función Saltar() no es relevante para el programa.

La abstracción está muy relacionada con la encapsulación, pero va un paso más allá, ya que, esta no solo controla el acceso a la información, sino que, también, oculta la complejidad de los procesos que este implementando.

Herencia. La herencia en programación funciona de la misma manera que la herencia biológica. En POO cuando una clase hereda de otra, obtiene todos los rasgos que tuviese la primera. Así, dado que una clase es un patrón que define cómo es y cómo se comporta una cierta entidad, una clase que hereda de otra obtiene todos los rasgos de la primera y añade otros nuevos. Además, la clase que hereda puede modificar algunos de los rasgos que ha heredado.

La clase de la que se heredan los rasgos se denomina clase base y la clase que hereda los rasgos se denomina clase derivada.

Tomando como ejemplo la clase Personajes que sería la clase base, puede crear clases más específicas que serían clases derivadas de Personajes. Por ejemplo, podría crear una clase que fuera Explorador y Guerrero. Todos los objetos de estas dos clases derivadas heredarían las propiedades y métodos de la clase Personajes, pudiendo especializar alguno de estos e, incluso, añadir propiedades y métodos propios. Por ejemplo, en la clase Explorador podría crear un método nuevo que fuese Explorar() y en la de Guerrero podría añadir una propiedad que fuera Apodo, por lo que, el guerrero que cree tendría su nombre que hereda de la clase Personaje y, además, tendría un apodo con el que se conoce al guerrero.

La herencia es una de las características más potentes de la POO, ya que, fomenta la reutilización del código permitiendo al mismo tiempo la especialización del mismo.

Polimorfismo. El polimorfismo es la cualidad de tener muchas formas. En POO este concepto se refiere al hecho de que varios objetos de diferentes clases, pero con una base común, se pueden usar de manera indistinta sin tener que saber de qué clase son exactamente.

Imagine que en el juego que quiere programar tiene una cantidad considerable de distintos personajes en un mismo escenario, y quiere que hablen todos. Como cada personaje habla de forma diferente, ya que, cada uno es de un tipo, ya sea guerrero o explorador, sería muy complejo tener que localizar primero a los personajes de un tipo, hacerlos hablar y después tener que hacer lo mismo con los de otro tipo. La idea es que se puedan tratar a todos como personajes, independientemente del tipo específico de personaje que sea y, simplemente, ordenarles que hablen.

Al derivar todos de la clase Personaje, todos pueden hablar y al llamar al método Hablar(), cada uno de ellos utilizará el proceso que se haya programado en su clase específica. Todo esto ocurre de manera transparente para el programador. Y, precisamente, esto es el polimorfismo.

Para resumir, el polimorfismo se permite utilizar objetos de manera genérica, aunque, internamente se comporten según su variedad específica.



TOME NOTA

Todos los lenguajes orientados a objetos cumplen estos cuatro principios, por lo que se facilita mucho la tarea de programar, se minimizan los errores, se escribe el código más rápido y se puede mantener de manera más sencilla cuando haya que implementar modificaciones en el futuro.

1.4.3. Atributos de los objetos

Como ha visto anteriormente las clases definen atributos y métodos. Los atributos son las características individuales que diferencian un objeto de otro. Los atributos se guardan en variables llamadas variables de instancia y cada objeto puede tener valores distintos para cada variable. Las variables de instancia son declaradas en la clase, pero sus valores son fijados y cambiados en el objeto.

Además de variables de instancia, también, hay variables de clase, las cuales se aplican a la clase y a todas sus instancias. Por ejemplo, la clase Coche, deberá tener un atributo que sea Ruedas y su valor será 4. Vea un ejemplo de cómo serían los atributos de una clase:

```
class vehículo

{

    Constructor (tipo, ruedas, motor){

        This.tipo = tipo

        This.ruedas = ruedas

        This.motor = motor

    }

}
```

Así, es como quedarían los atributos en una clase. Como puede observar, son parecidas a las variables de las que ha hablado anteriormente, la única diferencia es que, en este caso, las variables pertenecen a una clase. Este código pertenece al lenguaje Javascript que es con el que programará su web. Los demás lenguajes de POO son similares a este y, por lo tanto, si domina este tipo de programación en Javascript, será capaz de programar en otros lenguajes. En el siguiente apartado verá más detalladamente esta estructura, explicándola paso por paso.

1.4.4. Creación de objetos

Va a ver cómo se crea paso por paso un objeto en Javascript que es el lenguaje que utilice para el desarrollo web. Para crear un objeto de manera simple tan solo debe de escribir el nombre del objeto y sus propiedades dentro de él. Por ejemplo, va a crear el objeto armario:

```
Const armario={

    Material:'madera',

    Capacidad: '100L',

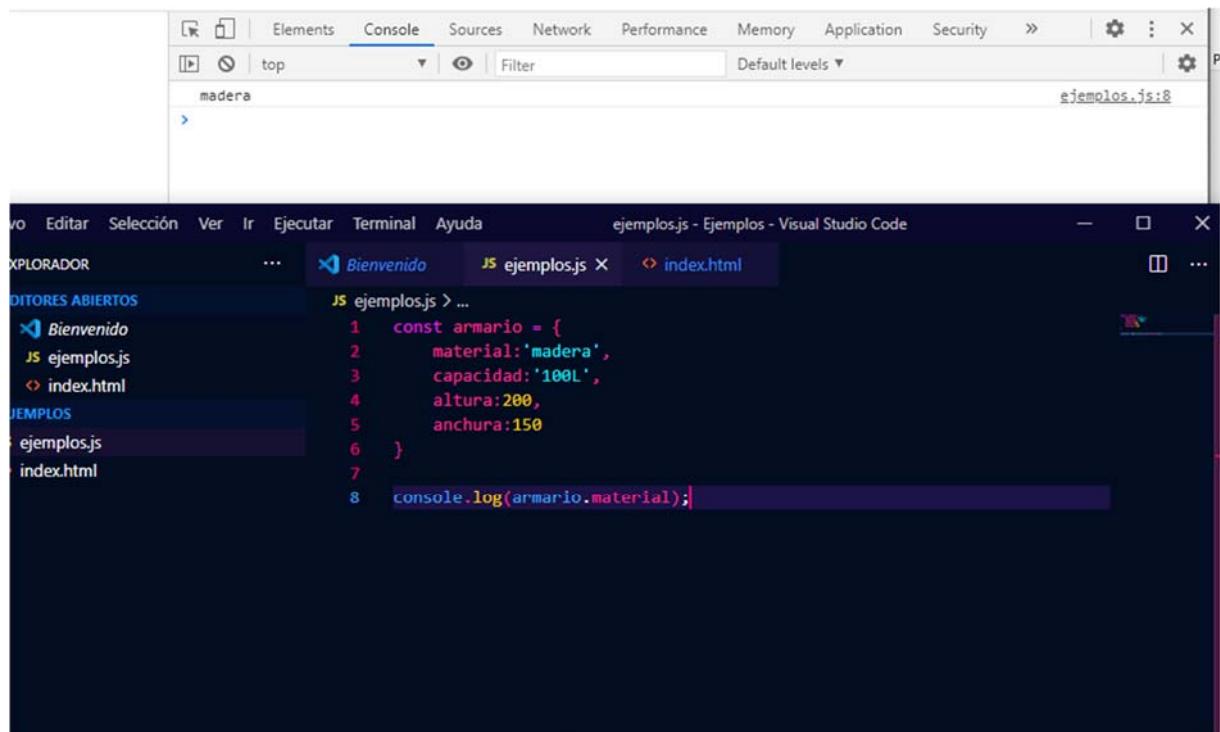
    Altura: 200,

    Anchura: 150

}
```

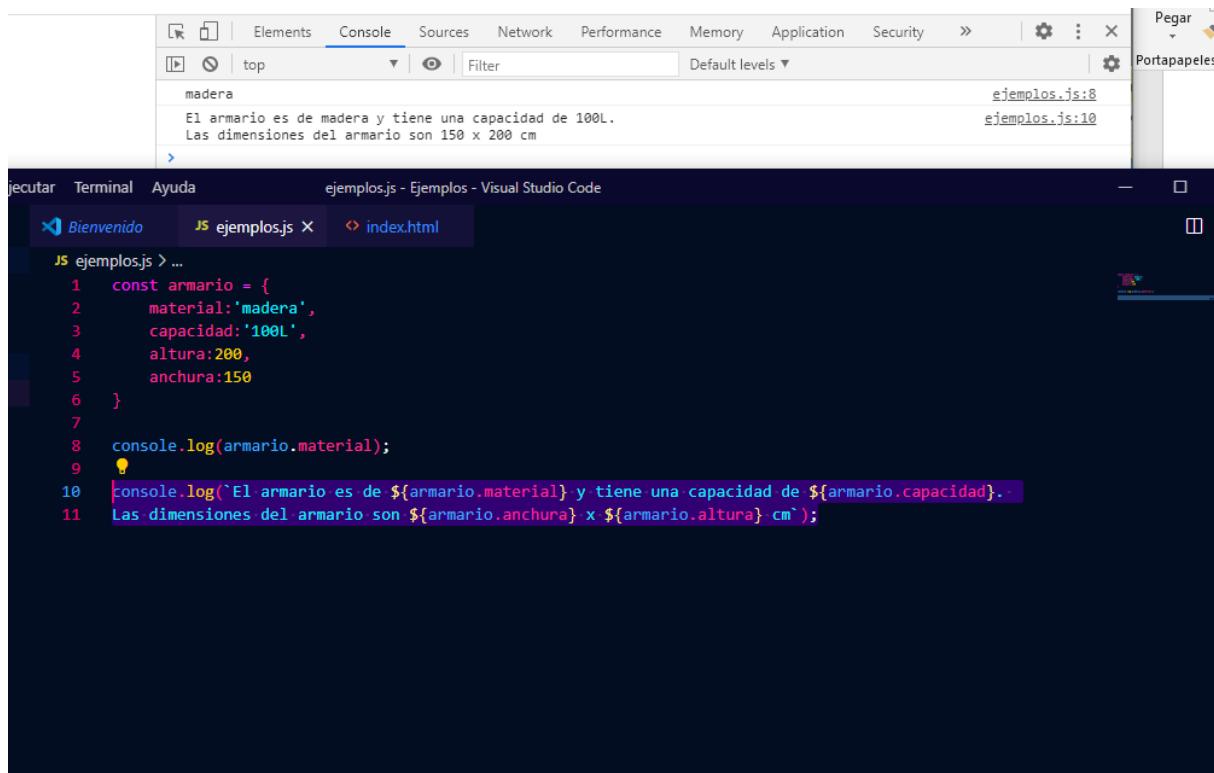
Con este código, habrá creado el objeto armario dándole todas sus propiedades. Para acceder a las propiedades y acciones del objeto va a utilizar la nomenclatura del punto. La nomenclatura del punto se utiliza poniendo el nombre del objeto, un punto y la propiedad a la que quiere acceder. Por ejemplo, si quiere acceder a la propiedad material del objeto armario, debería escribir:

```
Console.log(armario.material);
```



Si quiere imprimir los valores de un objeto en una frase, puede utilizar lo que se denomina template string o plantillas de cadena. Estas plantillas habilitan el uso de expresiones incrustadas. Con ellas, es posible utilizar cadenas de caracteres de más de una línea e incluso puede incrustar los valores de los objetos. Gracias a estas plantillas, si cambia los valores de los objetos no tendrá que reescribir la frase de nuevo. Por ejemplo, imagine que tiene un catálogo de armarios en su web y en la descripción quiere poner el tipo de material, las dimensiones y la capacidad. Tan solo con escribir la frase una vez, puede usarla para todos los armarios. Las plantillas de cadena se delimitan con tildes invertidas. Vea cómo hacer con código:

```
Console.log (`El armario es de ${armario.material} y tiene una capacidad de ${armario.capacidad}. Las dimensiones son ${armario.anchura} x ${armario.altura} cm.`);
```



The screenshot shows a Visual Studio Code interface. The top status bar indicates 'ejemplos.js - Ejemplos - Visual Studio Code'. Below it, the 'Console' tab is active, showing the output of the code execution. The code itself defines an object 'armario' with properties like material ('madera'), capacity ('100L'), height ('200'), and width ('150'). It then logs the material and capacity to the console. The bottom part of the screenshot shows the code editor with the same 'ejemplos.js' file open.

```
madera
El armario es de madera y tiene una capacidad de 100L.
Las dimensiones del armario son 150 x 200 cm
ejemplos.js:8
ejemplos.js:10

ejecutar Terminal Ayuda ejemplos.js - Ejemplos - Visual Studio Code
JS ejemplos.js index.html

1 const armario = {
2     material:'madera',
3     capacidad:'100L',
4     altura:200,
5     anchura:150
6 }
7
8 console.log(armario.material);
9
10 console.log(`El armario es de ${armario.material} y tiene una capacidad de ${armario.capacidad}.`);
11 Las dimensiones del armario son ${armario.anchura} x ${armario.altura} cm`);
```

Lo que ha visto hasta ahora es la creación de objetos de manera sencilla pero poco útil, ya que, si quisiera crear otro objeto armario habría que repetir el mismo código y nombrarlo de manera diferente, aunque el nuevo armario tuviera las mismas propiedades que el primero. Para poder crear objetos de manera eficiente, tiene que crear antes una clase que reúna las características principales que van a tener los objetos de esa clase. Como se vio anteriormente, una clase es como una plantilla, la cual usa para crear todos los objetos que quiera sin tener que reescribir el código una y otra vez.

Las clases necesitan una función constructora. A la función constructora se le llama cada vez que se crea un objeto, es por ello que esta función es siempre obligatoria para todas las clases que cree. A las funciones constructoras no se les invoca de la misma manera que a otras funciones, hay que invocarlas de una manera especial. Vea un ejemplo de clase:

```
1 class Muebles{
2     constructor(tipo,material,capacidad){
3         this.tipo = tipo
4         this.material = material
5         this.capacidad = capacidad
6     }
7 }
```

Como puede ver en la imagen al crear una clase, por convención, debe poner su nombre en mayúscula. En este caso, ha creado la clase `Muebles`. A continuación, ha creado la función

constructora la cual deberá llamarse siempre constructor. A la función constructora le ha dado parámetros que son: tipo, material y capacidad. Estos parámetros son los que va a tener el objeto de base, aunque después se pueden añadir propiedades al objeto y esas propiedades se pasarán como parámetros a la función constructor. Lo siguiente que debe hacer es asignar los valores al objeto, es por ello, que utiliza la palabra `this` la cual hace referencia al objeto. Con esta palabra, está asignando valores al objeto. Esto se puede traducir como:

- Tipo del objeto = tipo del parámetro
- Material del objeto = material del parámetro
- Capacidad del objeto = capacidad del parámetro

Simplemente está asociando las propiedades el objeto con las propiedades que está enviando por parámetro.

También, puede asignar propiedades que no haya en los parámetros, pero siempre utilizando `this` para referenciar el objeto. Por ejemplo: `this.datos = `${tipo} ${material} ${capacidad}``

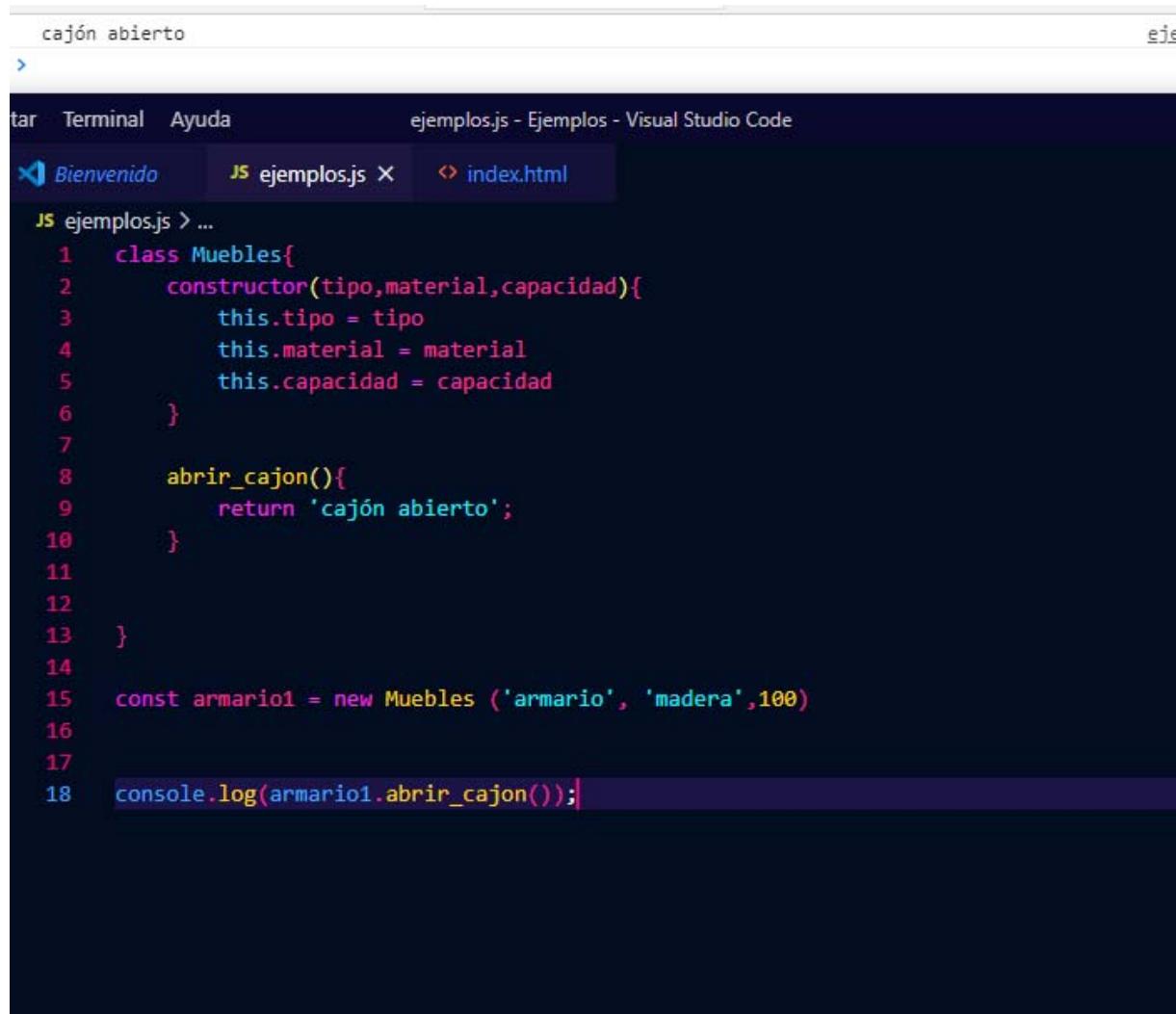
Cuando se habla de funciones dentro de un objeto, en vez de llamarlas funciones, se llamarán métodos. Los métodos de un objeto deben crearlos siempre dentro de la clase, pero fuera del constructor. Por ejemplo, a partir de la clase `Muebles`, puede crear un método que sea `abrir_cajón()` que quedaría de la siguiente manera:

```
Abrir_cajón(){  
    Return 'Cajón abierto';  
}
```

Para crear un objeto a partir de una clase antes debe instanciarlo. Para ello, si quiere crear el objeto `armario` debe hacerlo de la siguiente forma:

```
Const armario1 = new Muebles('armario', 'madera', 100)
```

Una vez instanciado el objeto, puede acceder a sus propiedades y métodos utilizando la nomenclatura del punto.



```
cajón abierto ejemplos.js - Ejemplos - Visual Studio Code

tar Terminal Ayuda
JS Bienvenido JS ejemplos.js X index.html

JS ejemplos.js > ...
1  class Muebles{
2      constructor(tipo,material,capacidad){
3          this.tipo = tipo
4          this.material = material
5          this.capacidad = capacidad
6      }
7
8      abrir_cajon(){
9          return 'cajón abierto';
10     }
11
12 }
13
14
15 const armario1 = new Muebles ('armario', 'madera',100)
16
17
18 console.log(armario1.abrir_cajon());
```

Al igual que ha creado el objeto armario1, puede crear todos los objetos del mismo tipo que quiera. Puede crear una mesa, otro armario, una silla, etc.

```
class Muebles{  
    constructor(tipo,material,capacidad){  
        this.tipo = tipo  
        this.material = material  
        this.capacidad = capacidad  
    }  
  
    abrir_cajon(){  
        return 'cajón abierto';  
    }  
  
}  
  
const armario1 = new Muebles ('armario', 'madera',100)  
  
const armario2 = new Muebles ('armario', 'metal', 90)  
  
const mesa1 = new Muebles ['mesa', 'plástico', 4]
```

En definitiva, puede crear cuantos métodos necesite. Por ejemplo, en la siguiente imagen verá dos métodos llamados descripción_masc() y descripción_fem() que podría utilizar para que apareciera en pantalla la descripción de un producto. Como vais a ver a continuación, gracias a la POO no tiene que reescribir el código para cada objeto, sino que puede usar los métodos que necesite dependiendo del objeto que haya creado. En este ejemplo, va a ver cómo utilizar la descripción masculina para los muebles con artículos masculinos y la descripción femenina para los que tienen artículos femeninos:

The screenshot shows the Visual Studio Code interface. The title bar says 'ejemplos.js - Ejemplos - Visual Studio Code'. The left sidebar shows a tree view with 'ejemplos.js > Muebles > abrir_cajon'. The main editor area contains the following JavaScript code:

```
1  class Muebles{
2      constructor(tipo,material,capacidad){
3          this.tipo = tipo
4          this.material = material
5          this.capacidad = capacidad
6      }
7
8      abrir_cajon(){
9          return 'cajón abierto';
10     }
11
12
13     descripcion_masc(){
14         return `El producto es un ${this.tipo} de ${this.material}`
15     }
16
17     descripcion_fem(){
18         return `El producto es una ${this.tipo} de ${this.material}`
19     }
20 }
21
22 const armario1 = new Muebles ('armario', 'madera',100)
23
24 const armario2 = new Muebles ('armario', 'metal', 90)
25
26 const mesa1 = new Muebles ('mesa', 'plástico', 4)
27
28 console.log (armario1.descripcion_masc());
29 console.log(armario2.descripcion_masc());
30 console.log(mesa1.descripcion_fem());
```

The 'Console' tab in the top bar is selected, showing the output of the code execution:

```
El producto es un armario de madera
El producto es un armario de metal
El producto es una mesa de plástico
```

File paths for each log entry are listed on the right: 'ejemplos.js:28', 'ejemplos.js:29', and 'ejemplos.js:30'.

En el siguiente vídeo se describen estos objetos en javascript:

RECURSO MULTIMEDIA



1.5. Ejemplos de códigos en diferentes lenguajes

Como ya sabe, existen múltiples lenguajes de programación y diferentes formas de organizarlos. En este apartado verá algunos de los tipos de lenguajes que existen y sus clasificaciones.

En diversas ocasiones, puede encontrar con que un mismo lenguaje puede pertenecer tanto a lenguaje de *script*, como a estructurado u orientado a objetos. Esto se debe a que hay lenguajes que

no son puramente de un tipo, ya que, tienen implementaciones que les permiten disfrutar de esa versatilidad.

1.5.1. Códigos en lenguajes estructurados

La programación estructurada es un paradigma de programación basado en utilizar subrutinas o funciones. Este tipo de programación, únicamente, utiliza tres estructuras de control:

- **Secuencia:** como vió anteriormente una secuencia es la ejecución de una sentencia tras otra.
- **Condicional:** es la ejecución de una sentencia o conjunto de sentencias según el valor resultante de una variable booleana.
- **Iteración:** es la ejecución de una sentencia o conjunto de sentencias mientras el resultado de una variable booleana sea verdadero. También se llama a esta estructura ciclo o bucle.
Este paradigma se fundamenta en el teorema que establece que toda función computable puede ser implementada en un lenguaje de programación que combine solo estas tres estructuras.

Las ventajas de la programación estructurada es que los programas son más fáciles de entender, ya que, pueden ser leídos de forma secuencial sin necesidad de tener que rastrear saltos de líneas dentro de los bloques de código para poder entender la lógica interna. Además, la estructura de los programas es clara, pues las sentencias están más relacionadas entre sí. También, los códigos son reutilizables para futuras aplicaciones y se pueden hacer modificaciones o correcciones de manera más sencilla. Todo ello conlleva la reducción del esfuerzo en las pruebas de depuración de errores y permite a los programadores escribir el código de manera más fácil y rápida.

Este tipo de programación también tiene inconvenientes como, por ejemplo, si el programa se hace demasiado grande sería difícil de manejar, ya que, todo el código se concentra en un único bloque. También, cabe destacar que, aunque este tipo de código permita la reutilización de código, no permite hacerlo de manera eficiente.

Algunos ejemplos de lenguaje estructurado son C, Pascal, Fortran. A continuación, va a ver un ejemplo de código de cómo se programaría una calculadora en C:

```
#include <conio.h>
#include <stdio.h>

int main()
{
    char opcion;
    int n1, n2;
```

```
do
{
    printf( "\n >>> MENU CALCULADORA <<<" );
    printf( "\n\n 1. Sumar dos n%cmeros.", 163 );
    printf( "\n 2. Restar dos n%cmeros.", 163 );
    printf( "\n 3. Multiplicar dos n%cmeros.", 163 );
    printf( "\n 4. Dividir dos n%cmeros.", 163 );
    printf( "\n 5. Salir.\n" );
    /* Filtra la opción elegida por el usuario */
    do
    {
        printf( "\n Introduzca opci%cn (1-5): ", 162 );
        fflush( stdin );
        scanf( "%c", &opcion );
    } while ( opcion < '1' || opcion > '5' );
    /* La opción sólo puede ser '1', '2', '3', '4' o '5' */
    switch ( opcion )
    {
        /* Opción 1: Sumar */
        case '1': printf( "\n Introduzca primer sumando: " );

```

```
scanf( "%d", &n1);

printf( "\n Introduzca segundo sumando: " );

scanf( "%d", &n2);

printf( "\n %d + %d = %d\n", n1, n2, n1 + n2 );

break;

/* Opción 2: Restar */

case '2': printf( "\n Introduzca minuendo: " );

scanf( "%d", &n1);

printf( "\n Introduzca sustraendo: " );

scanf( "%d", &n2);

printf( "\n %d - %d = %d\n", n1, n2, n1 - n2 );

break;

/* Opción 3: Multiplicar */

case '3': printf( "\n Introduzca primer operando: " );

scanf( "%d", &n1);

printf( "\n Introduzca segundo operando: " );

scanf( "%d", &n2);

printf( "\n %d * %d = %d\n", n1, n2, n1 * n2 );

break;

/* Opción 4: División entera */
```

```
case '4': printf( "\n  Introduzca dividendo: " );

scanf( "%d", &n1);

printf( "\n  Introduzca divisor: " );

scanf( "%d", &n2);

if ( n2 != 0 )

    printf( "\n  %d div %d = %d ( Resto = %d )\n", n1, n2, n1 / n2, n1 % n2 );

else

    printf( "\n  ERROR: No se puede dividir entre cero.\n" );

}

} while ( opcion != '5' );

return 0;

}
```

1.5.2. Códigos en lenguajes scripts

Los lenguajes de *script* son lenguajes de programación que permiten el control de otros programas o aplicaciones. Normalmente, las tareas complejas que realizan las aplicaciones involucran la ejecución de numerosos programas individuales que necesitan coordinarse entre sí. Los lenguajes *script* son los que presentan las características más apropiadas para hacer esta tarea, ya que, son lenguajes interpretados y esto hace que sean más rápidos en su ejecución.

Los lenguajes de *script* dan más importancia a la flexibilidad y a la facilidad en el desarrollo de programas. Un programa escrito en lenguaje de *script* suele recibir el nombre de *script* o guion. También se les denomina *script* a las páginas web y a los lenguajes de programación que pueden incorporarse en programas escritos en otros lenguajes para extender sus capacidades.

La mayoría de los lenguajes de *script* presentan características comunes como, por ejemplo, permitir su uso interactivo y como programa, sus expresiones suelen ser concisas, no tienen declaraciones o pueden prescindir de ellas utilizando reglas de alcance simples. Normalmente, los *scripts* tienen un tipado dinámico y flexible, es decir, los lenguajes de *script* no requieren que las variables se declaren

Por lo tanto, los valores que se asignan a las variables son los que determinan su tipo. Además, permiten un fácil acceso al sistema operativo subyacente.

Vea algunos ejemplos de lenguajes de script. Por ejemplo, **el lenguaje Bash** es un lenguaje capaz de interpretar comandos en un sistema operativo. Con él, se pueden escribir y almacenar scripts capaces de realizar tareas en el sistema operativo con tan solo ejecutar el archivo donde se almacena el script.

Por otro lado, también tiene los **lenguajes como PHP** que están destinado a desarrollar aplicaciones para web, ya que, favorecen la conexión entre los servidores y la interfaz de usuario. En el caso de Javascript, se utiliza principalmente del lado del cliente permitiendo crear efectos atractivos y dinámicos en su página web. Además, verá cómo necesita integrarlo con HTML para poder ejecutar el script. A continuación, verá un ejemplo:

```
<html>  
<head>  
<title>Ejemplo de Javascript</title>  
</head>  
<body>  
<script type="text/javascript">  
document.write('Hola Mundo');  
</script>  
</body>  
</html>
```

Normalmente, esta no es la forma con la que integra Javascript en un HTML, sino que llama al archivo .js de manera que el código quede más limpio. Vea un ejemplo:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Title</title>

</head>

<body>

<script src="ejemplos.js"></script>

</body>

</html>
```

1.5.3. Códigos en lenguajes orientados a objetos

Se le denomina lenguaje de programación orientado a objetos a cualquier lenguaje que implemente los conceptos definidos por la programación orientada a objetos como ha visto en este tema. Cabe destacar que no todos los lenguajes orientados a objetos son puramente orientados a objetos, sino que son híbridos que combinan la POO con otros paradigmas.

Como ejemplo, puede destacar PHP, que además de ser un lenguaje de *script* también se puede usar como lenguaje de programación orientado a objetos.

Por otra parte, también puede hablar de Javascript que hasta su versión EcmaScript 6 no implementó los conceptos de la POO. Otros lenguajes de programación orientada a objetos pueden ser C++, C#, Java, etc.

A continuación, va a ver un ejemplo de código de C# que pertenece a la clase Coche.

```
csharp_ejemplo.cs
1  namespace Ejemplo
2  {
3      class Coche
4      {
5          private string _marca;
6          private int _ruedas;
7          private int _plazas;
8          private string _color;
9
10         public Coche (string marca, int ruedas, int plazas, string color){
11             this._marca=marca;
12             this._ruedas=ruedas;
13             this._plazas=plazas;
14             this._color=color;
15         }
16
17         public string marca {get => _marca; set => _marca=value;}
18         public int ruedas {get => _ruedas; set => _ruedas;}
19         public int plazas {get => _plazas; set => _plazas;}
20         public string color {get => _color; set => _color;}
21     }
22 }
23 }
```

2. LENGUAJE DE GUION

En el mundo de la informática a los lenguajes de guion también se les denomina lenguajes de *script*. Los *scripts* son códigos de programación, generalmente, sencillos contenidos en archivos. Los *scripts* constituyen un conjunto de instrucciones que el ordenador ejecuta de manera secuencial. Normalmente, se utilizan para controlar el comportamiento de un programa o para interactuar con un sistema operativo.

Uno de los ejemplos más clásicos que tiene de lenguajes de guion en el mundo de la informática son los archivos batch de procesamiento por lotes, generalmente conocidos por su extensión .bat, que es la primera extensión utilizada por Microsoft para los archivos por lotes. Esta extensión puede ejecutarse en la mayoría de los sistemas operativos de Microsoft, incluyendo el famoso MS-DOS. La nueva extensión para los archivos .bat es la extensión .cmd. Por otra parte, tiene los archivos batch en UNIX/Linux lo identifica por las extensiones .bash o .sh.

Los archivos batch son aplicaciones formadas, simplemente, por pequeños archivos de texto que contienen algunas instrucciones escritas en lenguaje MSDOS, a estos códigos también se les conoce como comandos. Al ejecutar estos archivos, Windows automáticamente inicia la consola o interprete de comandos de MSDOS y ejecuta el lote de instrucciones.

Los archivos batch, en la actualidad, se suelen utilizar para crear aplicaciones propias que permitan ejecutar tareas tediosas y repetitivas y así, evitar la instalación de software de terceros. Los archivos batch son ideales para varias funciones, por ejemplo, hacer múltiples copias, extraer y listar información, eliminar virus, planificar tareas, ejecutar limpiezas en el sistema, hacer respaldos de datos e información, automatizar descargar de internet, etc.

	<p><i>Si ejecuta un archivo .cmd en un sistema operativo antiguo de Microsoft, este no lo podrá interpretar. Es por ello, que si maneja sistemas operativos antiguos es mejor usar archivos .bat.</i></p>
TOME NOTA	

2.1. Características del lenguaje

Se les denomina guion o *script* a los programas que están escritos bajo un determinado lenguaje de programación interpretado. A diferencia de otros tipos de lenguajes de programación, los lenguajes de *script* no utilizan un compilador, sino que estos utilizan un intérprete para traducir el *script* a lenguaje de máquina. Las ventajas de los lenguajes interpretados es que son más rápidos y sencillos de desarrollar. Además, son normalmente lenguajes multiplataforma, ya que, cada sistema operativo tiene su propio intérprete. Sin embargo, los lenguajes interpretados son más lentos en su ejecución, pues cada vez que ejecuta el lenguaje, tiene que pasar por su intérprete para traducirlo a lenguaje de máquina y que, así, pueda entenderlo el ordenador.



Un lenguaje de script puede ser considerado como multiplataforma si su intérprete está disponible en múltiples plataformas y la secuencia de comandos solo utiliza los servicios proporcionados por el lenguaje.

TOME NOTA

Como vio en el apartado anterior, los archivos batch son archivos de script y, por lo tanto, en Windows se utiliza un intérprete de comandos llamado cmd.exe, siendo su antecesor command.com. Este es el primer programa que se ejecuta después del inicio de Windows y, por lo tanto, posee el rol de la configuración del sistema ejecutando el archivo autoexec.bat.

Los principales lenguajes de *script* que se usan actualmente son:

- JavaScript/ECMAScript
- PHP
- Python
- Ruby
- VBA
- R

La característica principal de todos estos lenguajes de *script* es que pueden realizar diferentes acciones dentro de un entorno de tiempo de ejecución particular, como automatizar la ejecución de tareas, mejorar la funcionalidad del software principal, realizar configuraciones y extraer datos de conjuntos de datos.

Como vió anteriormente, los scripts son una serie de comandos que son interpretados uno por uno por una aplicación intérprete. Aunque el *script* guía a la plataforma sobre lo que debe hacer, la ejecución la realiza el entorno de ejecución y no el propio lenguaje de *scripting*. Esto es lo que diferencia a los lenguajes de *script* de los lenguajes de programación como, por ejemplo, Java. Otras de las características de los lenguajes de *script* son:

- Incorporan tipos de datos estructurados y de alto nivel.
- Permiten un fácil acceso al sistema operativo subyacente.
- Disponen de expresiones regulares y emparejamiento de patrones sofisticados.
- Tipado dinámico y flexible.
- Permiten su uso interactivo y como programas.
- Sus expresiones suelen ser concisas.
- No tienen declaraciones o pueden prescindir de ellas utilizando reglas de alcance simples.

La mayoría de los lenguajes de *script* disponen de reglas simples para determinar el alcance de los nombres, aunque difieren unos de otros.

Sobre el tipado dinámico y flexible, la mayoría de los lenguajes de script no requieren que las variables se declaren, es decir, los valores que se asignan a las variables son los que determinan su tipo. Por ejemplo, en JavaScript escribiría var num; sin tener que determinar el tipo de dato, cosa que en C sí debe hacer, por ejemplo, la misma variable sería int num; si quisiera que albergara un número entero.

Por último, cabe destacar que los lenguajes de guion se clasifican en dos tipos:

- *Scripts* del lado del cliente.
- *Scripts* del lado del servidor.

Los *scripts* del lado del cliente son los que se ejecutan en el ordenador del usuario que está accediendo a una página web, por ejemplo, JavaScript sería mayoritariamente un lenguaje script del lado del cliente. Por otro lado, los lenguajes de *script* del lado del servidor son los que se ejecutan en el servidor que aloja el contenido de la página web, en este caso. Como lenguaje de *script* del lado del servidor puede poner como ejemplo PHP, ya que, es uno de los lenguajes más utilizado en servidores.

RECURSO MULTIMEDIA



2.1.1. Descripción del lenguaje orientado a eventos

La programación orientada a eventos es un paradigma de programación en el que la estructura y la ejecución de los programas van determinados por los sucesos o acciones que ocurren en el sistema.

A diferencia de la programación estructurada donde es el programador el que define cuál va a ser el flujo del programa, en la programación orientada a eventos es el propio usuario, al provocar los eventos, el que dirige el flujo del programa.

La programación orientada a eventos es muy fácil de usar y es adecuada para principiantes en programación. Con los lenguajes de programación orientados a eventos se pueden realizar, en poco tiempo, aplicaciones sencillas y muy funcionales utilizando interfaces gráficas en las que se insertan componentes o controles a los que se les programa eventos como, por ejemplo, ocurre con el lenguaje de programación VBA que puede utilizar en aplicaciones de Microsoft como, por ejemplo,

Access donde encuentra una interfaz gráfica para programar, así, los eventos. Aunque, también puede programar VBA como cualquier otro lenguaje de programación sin interfaz gráfica.

En definitiva, los eventos permiten interactuar con los objetos, por ejemplo, si pasa el ratón por encima de una imagen y esta hace zoom. O, si hace clic en un botón y este cambia de color o de forma.

2.1.2. Descripción del lenguaje interpretado

Como ha visto anteriormente, un lenguaje interpretado es aquel que es ejecutado a través de un intérprete. A este tipo de lenguajes también se les denomina lenguaje de guion o *script*. Los lenguajes *script* reúnen algunas características como:

- Son independientes de la plataforma donde son ejecutados.
- Se genera el código sin recurrir a la compilación del mismo.
- El código es fácil de depurar.
- Los *scripts* suelen ser de tamaño pequeño y, por lo tanto, ocupan poco espacio.
- Son dinámicos.

Por otro lado, una de las desventajas de los lenguajes de script es que son menos eficientes frente a un programa compilado, ya que, cada instrucción es ejecutada en tiempo de ejecución. Los scripts para ser ejecutados, como ya sabe, necesitan un intérprete, por lo que, si en la plataforma que quiere ejecutar el programa de *script* no existe un intérprete para traducir el *script*, este no podrá ser ejecutado.

Cabe destacar que, los lenguajes de alto nivel se pueden clasificar en tres ramas:

- Lenguajes puramente compilados como, por ejemplo, C.
- Lenguajes parcialmente compilados como, por ejemplo, Java.
- Lenguajes interpretados como, por ejemplo, JavaScript.

Los lenguajes parcialmente compilados son lenguajes que se compilan con un lenguaje intermedio que, a su vez, ese lenguaje intermedio es interpretado.

2.1.3. La interactividad del lenguaje de guion

El lenguaje de marcas HTML, con el que se crean las páginas webs, es un lenguaje estático. Esto quiere decir que la página web es totalmente plana, el usuario no puede interactuar con ningún elemento de la página. Para ello, utiliza JavaScript.



TOME NOTA

Gracias a este lenguaje de script puede aportar dinamismo a su web implementando el uso de eventos.

Por ejemplo, si pasa el puntero del ratón por encima de un botón, este cambia de color o si clica al botón, este cambia de forma.

En JavaScript dispone de muchísimos eventos para poder llevar a cabo cualquier interacción. Así pues, si tiene un botón que se llama “Mostrar perfil”, deberá haber un evento *Onclick* (al hacer clic), que abra la página de su perfil. Además, puede aplicar a ese mismo botón un evento *Onmouseover* (cuando el ratón pase por encima del botón) y puede hacer que el botón cambie de forma y color. Más adelante verá, en profundidad, los eventos más importantes que puede utilizar con JavaScript.

2.2. Relación del lenguaje de guion y el lenguaje de marcas

Cuando Se habla de desarrollo web el lenguaje de guion por excelencia es JavaScript, mientras que el lenguaje de marcas al que se refiere es HTML.

HTML significa, por sus siglas en inglés *Hypertext Markup Language*, o lo que es lo mismo, Lenguaje de Marcas de Hipertexto. Este lenguaje es el componente más básico de una web, es por ello que, define el significado y la estructura del contenido web. HTML utiliza marcas, precedidas por los símbolos < y >, para etiquetar texto, imágenes y otros contenidos para mostrarlos en un navegador web. La estructura HTML más básica de una web es la siguiente:

```
<HTML>
  <HEAD>
    <TITLE></TITLE>
  </HEAD>
  <BODY>
    </BODY>
</HTML>
```

Vea a continuación para qué sirven las diferentes etiquetas que componen la estructura básica de cualquier web:

<HTML>. Esta etiqueta indica el comienzo y el final del documento HTML, es decir, dentro de estas estructuras será donde codifica lo que el navegador va a interpretar de su web.

<HEAD>. Esta etiqueta corresponde al encabezado y descripción del documento HTML. En ella, se encontrará información como los metadatos del documento, incluyendo su título, enlaces a *scripts* y hojas de estilos.

<TITLE>. Dentro de esta etiqueta puede escribir el título de su página web. Este título será el que se muestre en la barra del navegador.

<BODY>. Esta etiqueta representa el contenido de un documento HTML. Solo puede haber un elemento <BODY> en el documento. Dentro de esta etiqueta será donde codifique la estructura de su página, ya sea añadiéndole imágenes, vídeos, tablas, columnas, etc.

	<p><i>En HTML, por lo general, las etiquetas tienen una apertura y un cierre, pero hay algunas etiquetas que no cumplen este criterio como, por ejemplo, la etiqueta
 que se utiliza para dejar espacios en blanco. A este tipo de etiquetas, se les llama Self-Closing Tags.</i></p>
<p>TOME NOTA</p>	

Como se ha dicho anteriormente, el HTML es la estructura básica de una web, pero para que la web tenga un contenido más dinámico donde el usuario pueda interactuar con los elementos, debe insertar trozos de *script* que permitan darle un comportamiento dinámico a los elementos. Hay dos formas de insertar JavaScript en HTML y hacer que funcionen juntos.

- La primera forma de insertar JavaScript, es insertar directamente el código de *script* al documento HTML. Esto se puede hacer mediante las etiquetas <script> y </script>, las cuales deben envolver todo el código JavaScript que escriba. A su vez, estas dos etiquetas puede situarlas dentro del <HEAD> o dentro del <BODY>.

La carga del archivo HTML será diferente dependiendo de dónde haya agregado el código JavaScript en el archivo. Casi siempre, se recomienda que el código *script* vaya dentro del <HEAD> para que permanezca separado del contenido del archivo HTML. Esto ayudará a que el archivo sea más claro y legible. Sin embargo, poner el código *script* dentro del <BODY> puede ayudar a mejorar la velocidad de carga de la web, ya que el *script* se procesará después de haber cargado el sitio web.

Vea ambos ejemplos:

En esta imagen ve cómo el código *script* está insertado dentro del <HEAD>.

```
↳ index.html > html > body
1   <!DOCTYPE html>
2   <html lang="es">
3     <head>
4       <meta charset="UTF-8">
5       <title>Título de nuestra web</title>
6       <script>
7         var time = new Date();
8         console.log(time.getHours() + ":" + time.getMinutes() + ":" + time.getSeconds());
9       </script>
10    </head>
11    <body>
12    </body>
13  </html>
```

Por otro lado, así es cómo quedaría el *script* dentro del <BODY>.

```
↳ index.html > html > body > script
1   <!DOCTYPE html>
2   <html lang="es">
3     <head>
4       <meta charset="UTF-8">
5       <title>Título de nuestra web</title>
6     </head>
7     <body>
8       <script>
9         var time = new Date();
10        console.log(time.getHours() + ":" + time.getMinutes() + ":" + time.getSeconds());
11      </script>
12    </body>
13  </html>
```

- La segunda forma de agregar código de *script* en un documento HTML es poner el *script* en un archivo separado. A veces, insertar *script* directamente en un archivo HTML no es la mejor opción, ya que, algunos scripts pueden ocupar varias páginas, por lo tanto, lo mejor es mantener ese *script* separado del archivo HTML. De hecho, la forma más recomendable de relacionar un *script* con HTML es de esta manera. Para ello, dentro del archivo HTML referencia el archivo JavaScript que quiere insertar. Los beneficios de agregar código JavaScript desde un archivo externo son los siguientes:

Cuando los códigos HTML y JavaScript están separados, se cumple el principio de diseño de separación y hace que todo sea mucho más sostenible y reutilizable.

La legibilidad y el mantenimiento del código son mucho más sencillos.

Los archivos JavaScript en caché mejoran el rendimiento general del sitio web al disminuir el tiempo que tardan las páginas en cargarse.

Para referenciar un archivo JavaScript desde HTML lo hará de la siguiente manera:

```
④ index.html > ...
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <title>Título de nuestra web</title>
6  </head>
7  <body>
8
9  </body>
10 <script src="ejemplos.js"></script>
11 </html>
12
13
14
```



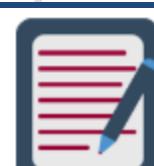
TOME NOTA

La manera más eficiente y profesional de insertar un script en HTML siempre será referenciándolo desde un archivo externo.

2.2.1. Extensión de las capacidades del lenguaje de marcas

HTML5 extiende más allá las capacidades del HTML original. En HTML5 tiene etiquetas semánticas para cada sección de la página. Por ejemplo, tiene la etiqueta `<footer>` que, como su nombre indica, sirve para añadir todo lo que componga el pie de página. También, tiene la etiqueta `<nav>` que sirve para colocar la botonera de navegación principal.

Como ya verá más adelante, el HTML5 ha significado un gran paso evolutivo dentro de los lenguajes de marcas, ya que, ha permitido crear webs mucho más eficientes frente al SEO y dinámicas.

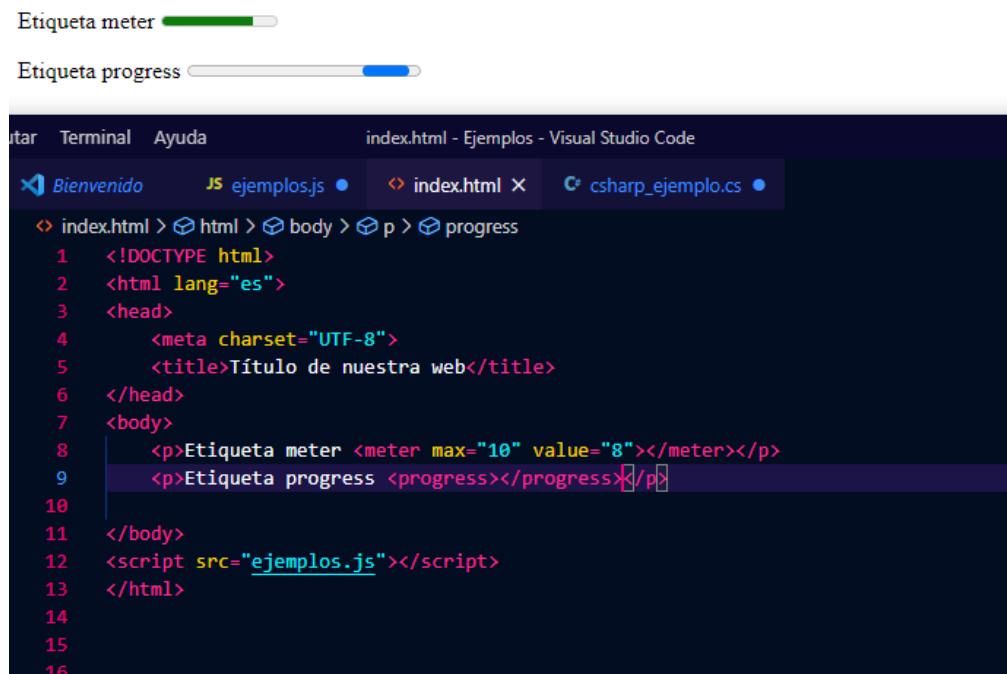


TOME NOTA

Para crear un documento con HTML5 solo se debe escribir `<!DOCTYPE html>` antes de la etiqueta `<html>`.

2.2.2. Adicción de propiedades interactivas

En HTML5 se han añadido una serie de propiedades interactivas que antes no existían y que tenía que añadir con software de terceros como, por ejemplo, Flash. Entre ellas están la etiqueta `<progress>` que informa al usuario que debe esperar y la etiqueta `<meter>` que representa valores sobre un total. Vea el siguiente ejemplo:



The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows files: Bienvenido (dark blue), ejemplos.js (light blue), index.html (purple), and csharp_ejemplo.cs (light blue).
- Code Editor:** Displays the following HTML code:

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <title>Título de nuestra web</title>
6 </head>
7 <body>
8   <p>Etiqueta meter <meter max="10" value="8"></meter></p>
9   <p>Etiqueta progress <progress></progress></p>
10 </body>
11 <script src="ejemplos.js"></script>
12 </html>
13
14
15
16
```
- Status Bar:** Shows "index.html - Ejemplos - Visual Studio Code".

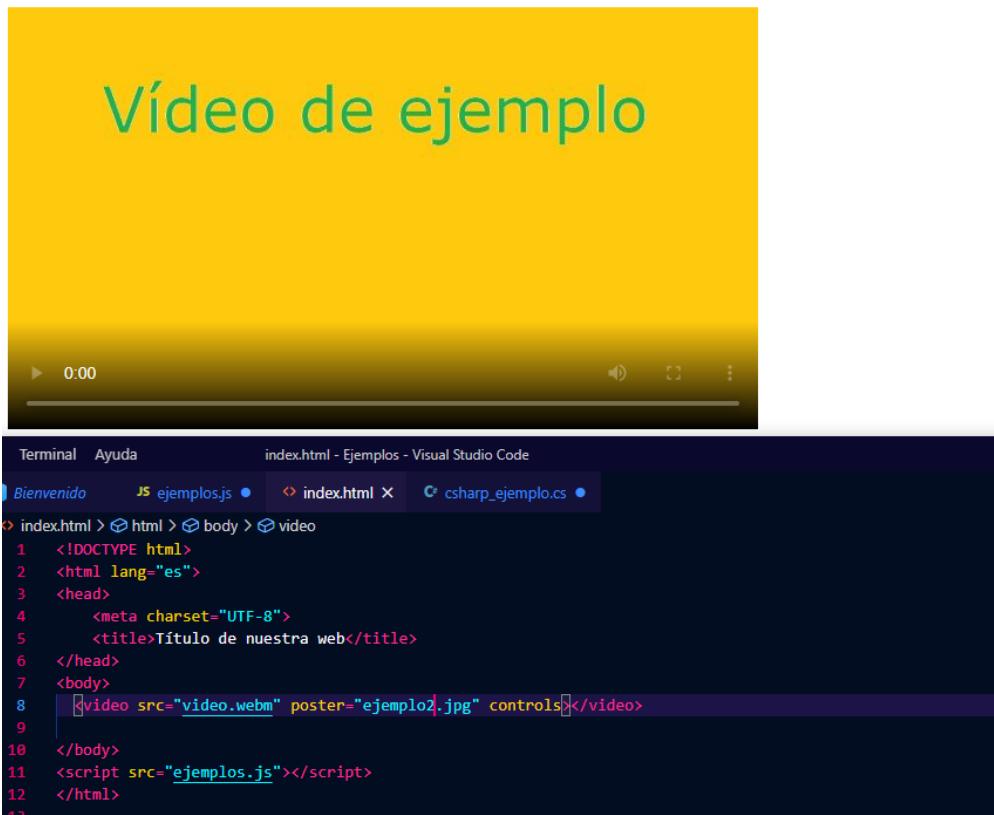
Además, en cuanto al contenido interactivo como vídeos o sonido, ya no será necesario un reproductor flash. Para ello, se han creado las etiquetas `<video>` y `<audio>` que incorporan los controles. A continuación, puede ver un ejemplo:

En esta imagen puede ver que la etiqueta `<video>` tiene como atributos poster y controls. El atributo `posters`, nos permite poner una imagen personalizada en vez de dejar el primer fotograma por defecto. Por otro lado, el atributo `controls` nos permite añadir los controles del vídeo tales como play, pausa, volumen, etc. Además de estos atributos, también se le pueden añadir otros como `autoplay`, si quiere que el vídeo comience automáticamente. `Muted`, para que el vídeo no tenga volumen. O, `loop` para que el vídeo se ejecute una y otra vez en forma de bucle.



Recuerde que este tipo de etiquetas pertenecen exclusivamente a la versión 5 de HTML, por lo que, si está programando en versiones anteriores, estas funciones no van estar permitidas. También, cabe destacar que los navegadores que no permitan la versión HTML5 no podrán ejecutar estas funcionalidades.

TOME NOTA



2.3. Sintaxis del lenguaje de guion

La sintaxis de un lenguaje de programación se define como el conjunto de reglas que deben seguirse al escribir el código fuente de los programas para considerarse como correctos para ese lenguaje.

La sintaxis de JavaScript es muy similar a la de otros lenguajes de programación como Java y C. Las normas básicas que definen la sintaxis de JavaScript son las siguientes:

- **No se tienen en cuenta los espacios en blanco y las nuevas líneas.** Como sucede con XHTML, el intérprete de JavaScript ignora cualquier espacio sobrante, por lo que el código se puede ordenar de forma adecuada para entenderlo mejor, ya sea tabulando las líneas, añadiendo espacios o creando nuevas líneas.
- **Se distinguen las mayúsculas y minúsculas.** Hay lenguajes, como HTML, en que el uso de mayúsculas o minúsculas no interfiere con el funcionamiento de la web. En cambio, si en JavaScript se intercambian mayúsculas y minúsculas el *script* no funciona. A esta propiedad también se le da el nombre de case sensitive.
- **No se define el tipo de las variables.** Al crear una variable no es necesario indicar el tipo de dato que se almacenará en ella. De esta forma, una misma variable puede almacenar diferentes tipos de datos durante la ejecución de *un script*.
- **No es necesario terminar cada sentencia con el carácter de punto y coma.** En la mayoría de lenguajes de programación, es obligatorio terminar cada sentencia con el carácter (;).

Aunque JavaScript no obliga a hacerlo, es conveniente seguir la tradición de terminar cada sentencia con punto y coma.

- **Se pueden incluir comentarios.** Los comentarios se utilizan para añadir información en el código fuente del programa. Aunque el contenido de los comentarios no se visualiza por pantalla, sí que se envía al navegador junto con el resto del *script*, por lo que es necesario extremar las precauciones sobre la información incluida en los comentarios. En JavaScript hay dos maneras de comentar. La primera es comentar una sola línea. Para ello, solo tiene que escribir doble barra // al principio de la línea. Vea un ejemplo:

```
// Soy una línea comentada en JavaScript
```

Para comentar varias líneas, utilice un signo de apertura /* y un signo de cierre */. Vea un ejemplo:

```
/* Soy un comentario de  
varias líneas en Javascript*/
```

RECURSO MULTIMEDIA

2.3.1. Etiquetas identificativas dentro del lenguaje de marcas

Tal y como ha visto anteriormente, HTML usa etiquetas para formar la estructura de una web y, casi siempre, tendrá una etiqueta de apertura y una de cierre. La sintaxis de HTML es como la siguiente: <etiqueta> información que quiere poner </etiqueta>.

Además de esto, hay etiquetas que tienen atributos que son palabras especiales que se escriben dentro de las etiquetas de apertura, y que sirven para controlar el comportamiento del elemento.

Antes que nada, debe recordar que las etiquetas esenciales para crear un archivo HTML son <html> y </html>. Además, si quiere usar HTML5 debe escribir al principio <!DOCTYPE html>.

A continuación, va a ver algunas de las etiquetas HTML más utilizadas.

Etiquetas de secciones. Las etiquetas de secciones van a utilizarse para delimitar las partes del contenido de la página. Por ejemplo, `<head>` o `<body>` son etiquetas de sección. A continuación, se muestran algunas de ellas:

Etiqueta	Descripción
<code><section></code>	Define una sección del documento
<code><nav></code>	Define la sección que contiene los enlaces de navegación
<code><article></code>	Define un contenido autónomo que podría existir independientemente del resto del contenido. Por ejemplo, los posts de un blog.
<code><h1><h2><h3></code>	Los elementos de cabecera implementan seis niveles de cabeceras de documentos. <code><h1></code> es la de mayor importancia y <code><h6></code> es la de menor. Un elemento de cabecera describe brevemente el tema de la sección que introduce.
<code><header></code>	Define la cabecera de una página. Usualmente contiene un logotipo, el título del sitio web y una tabla de navegación de contenidos.
<code><footer></code>	Define el pie de página. Usualmente contiene un mensaje de derechos de autor, algunos enlaces a información legal o direcciones para dar información de retroalimentación.
<code><address></code>	Define una sección que contiene información de contacto.
<code><main></code>	Define el contenido principal o importante en el documento. Solamente existe un elemento <code><main></code> en el documento
<code><body></code>	Representa el contenido principal de un documento HTML. Solo puede haber un elemento <code><body></code> en un documento.

Etiquetas de agrupación de contenido. Este tipo de etiquetas se utilizan para agrupar contenido en un documento. Entre ellas tiene las más básicas que son las de párrafo y las de listas:

Etiquetas	Descripción
<code><p></code>	Define una parte que debe mostrarse como párrafo.
<code></code>	Define una lista ordenada de artículos.
<code></code>	Define una lista de artículos sin ordenar.
<code></code>	Define un artículo de una lista enumerada.
<code><dl></code>	Define una lista de definiciones, es decir, una lista de términos y sus definiciones.
<code><dt></code>	Representa un término definido por la etiqueta <code><dd></code>
<code><dd></code>	Representa la definición de los términos listados como <code><dt></code>
<code><figure></code>	Representa una figura ilustrada como parte del documento.
<code><figcaption></code>	Representa la leyenda de una figura.
<code><div></code>	Representa un contenedor genérico sin ningún significado especial.

Etiquetas semánticas a nivel de texto. Este tipo de etiquetas se utilizan para cambiar el comportamiento del texto en el documento HTML. Vea algunas de estas etiquetas:

Etiquetas	Descripción
<a>	Representa un hiperenlace.
	Representa un texto enfatizado.
	Representa un texto en negrita.
<small>	Representa un comentario aparte, es decir, textos que no son esenciales para la compresión del documento, como puede ser los derechos de autor.
<s>	Muestra un texto tachado.
<cite>	Muestra un texto en cursiva.
<q>	Muestra un texto entre comillas.
<abbr>	Representa una abreviación o un acrónimo.
<time>	Representa un valor de fecha y hora.
	Representa un texto sin significado específico.
 	Representa un salto de línea.

Etiquetas de contenido incrustado. Estas etiquetas se utilizan para incrustar contenido interactivo en su web. Por ejemplo, vídeo, audio, imágenes, etc. Vea, a continuación, algunas de ellas:

Etiquetas	Descripción
	Representa una imagen.
<iframe>	Representa un contexto anidado de navegación, es decir, un documento HTML embebido. Esta etiqueta se suele utilizar para incrustar contenido de otros sitios webs.
<video>	Representa un vídeo con la interfaz necesaria para reproducirlo.
<audio>	Representa un audio o stream de audio.
<map>	En conjunto con <área> define un mapa de imagen.
<svg>	Define una imagen vectorial embebida.
<canvas>	Representa un mapa de bits en el que se pueden utilizar <i>scripts</i> para renderizar gráficos.

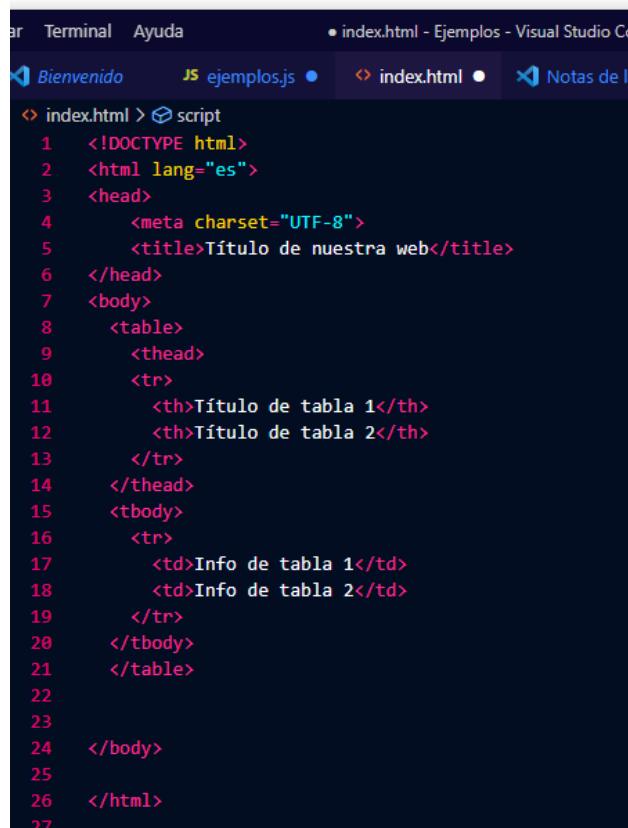
Etiquetas para tablas. Las etiquetas para tablas permiten crear tablas en su página web.

Etiquetas	Descripción
<table>	Representa una tabla.
<caption>	Representa el título de una tabla.
<colgroup>	Representa un conjunto de una o más columnas de una tabla.
<col>	Representa una columna de una tabla.
<tbody>	Represneta un bloque de filas que describen los datos concretos de una tabla.
<thead>	Representa el bloque de filas que describen las etiquetas de columna de una tabla.
<tfoot>	Representa los bloques de fila que describen los resúmenes de columna de una tabla.
<tr>	Representa una fila de celdas de una tabla.
<td>	Representa una celda de datos de una tabla.
<th>	Representa una celda de encabezado en una tabla.

Ejemplo de creación de tablas en HTML:

Titulo de tabla 1 **Titulo de tabla 2**

Info de tabla 1 Info de tabla 2



The screenshot shows a Visual Studio Code interface with a dark theme. The top bar has tabs for 'index.html - Ejemplos - Visual Studio Code', 'Bienvenido', 'ejemplos.js', 'index.html', and 'Notas de la clase'. The main editor area contains the following HTML code:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Título de nuestra web</title>
</head>
<body>
    <table>
        <thead>
            <tr>
                <th>Título de tabla 1</th>
                <th>Título de tabla 2</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Info de tabla 1</td>
                <td>Info de tabla 2</td>
            </tr>
        </tbody>
    </table>
</body>
</html>
```

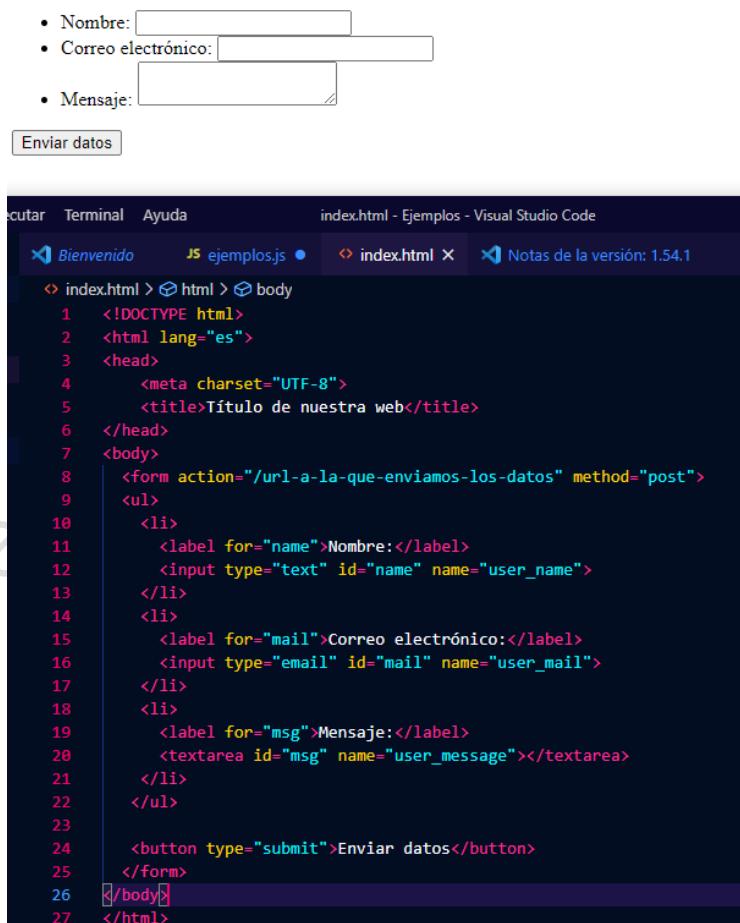
Etiquetas de script. Etiquetas de script tan solo hay dos:

Etiquetas	Descripción
<script>	Se utiliza para insertar <i>scripts</i> dentro de su documento HTML
<noscript>	Se utiliza para poder dar respuesta a los navegadores que no soportan <i>scripts</i> .

Etiquetas para formularios. Estas etiquetas permiten crear formularios en su web:

Etiquetas	Descripción
<form>	Representa un formulario permitiendo controles para enviar datos a un servidor.
<fieldset>	Representa un conjunto de controles.
<legend>	Representa un título de un <fieldset>.
<input>	Representa un campo de datos escrito que permite al usuario editar datos.
<label>	Representa el título de un control de formulario.
<button>	Representa un botón.
<select>	Representa un control que permite la selección entre un conjunto de opciones.
<datalist>	Representa un conjunto de opciones predefinidas para otros controles.
<optgroup>	Representa un conjunto de opciones agrupadas lógicamente.
<option>	Representa una opción en un elemento <select>, o una sugerencia de un elemento <datalist>
<textarea>	Representa un control de edición de texto multilínea.
<progress>	Representa el progreso de finalización de una tarea.
<meter>	Representa una medida dentro de un rango conocido.

Ejemplo de creación de formulario en HTML:



The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows files: "index.html - Ejemplos - Visual Studio Code", "Bienvenido", "ejemplos.js", and "Notas de la versión: 1.54.1".
- Code Editor:** Displays the HTML code for a form. The code includes a form with three input fields (text, email, and textarea) and a submit button. The code is numbered from 1 to 27.
- Output:** Shows the rendered HTML output with three input fields and a submit button labeled "Enviar datos".

```

1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <title>Título de nuestra web</title>
6  </head>
7  <body>
8      <form action="/url-a-la-que-enviamos-los-datos" method="post">
9          <ul>
10         <li>
11             <label for="name">Nombre:</label>
12             <input type="text" id="name" name="user_name">
13         </li>
14         <li>
15             <label for="mail">Correo electrónico:</label>
16             <input type="email" id="mail" name="user_mail">
17         </li>
18         <li>
19             <label for="msg">Mensaje:</label>
20             <textarea id="msg" name="user_message"></textarea>
21         </li>
22     </ul>
23
24     <button type="submit">Enviar datos</button>
25
26 </form>
27 </body>

```

2.3.2. Especificaciones y características de las instrucciones

En este apartado va a ver los atributos de las etiquetas más importantes y básicas para componer su documento HTML. No obstante, siempre es bueno buscar en Internet atributos y etiquetas de HTML, ya que, es imposible conocer todas las que hay.

Entre los atributos más comunes se encuentran los siguientes:

Etiqueta	Atributo	Descripción
<a>	Href	Se utiliza para redirigir el hiperenlace.
<a>	Target	Este atributo se usa para definir cómo se abrirá el hiperenlace, si en la misma ventana o en otra.
Múltiples etiquetas	Name	Este atributo permite a un <i>script</i> acceder a su contenido.
<body>	Bgcolor	Permite establecer un color de fondo
	Size	Permite modificar el tamaño de la fuente.
	Color	Permite modificar el color de la fuente.
<form>	Method	Permite elegir con qué método se enviarán los datos del formulario si GET o POST.
<form>	Action	Indica a dónde va a enviar la información el formulario.
	Border	Establece si una imagen tiene borde o no.
	Height	Establece la altura de una imagen.
	Width	Establece la anchura de una imagen.
	Src	Se establece la dirección donde se encuentra la imagen a insertar.
<table>	Bgcolor	Permite modificar el color de fondo de una tabla.
<table>	Align	Modifica la alineación de una tabla.
<table>	Border	Define los bordes a una tabla.
<td>	Colspan	Extiende la celda sobre varias columnas.
<td>	Rowspan	Extiende la celda sobre varias filas.
<td>	Valing	Establece la alineación vertical de los datos que contienen una celda.

2.3.3. Elementos del lenguaje de guion

En este apartado, va a ver los elementos que componen el lenguaje de guion. Entre ellos tiene:

- **Variables.** Que es donde se almacenan los datos que serán o han sido procesados.
- **Operaciones.** Se usan junto con las variables para poder procesar esos datos.
- **Comparaciones.** Permiten hacer comparaciones entre los datos, también usando las variables.
- **Asignaciones.** Permiten asignar datos a las variables antes o mientras trabaja con ellas.

A continuación, va a profundizar el aprendizaje de estos elementos.

2.3.3.1. Variables

Una variable está formada por un espacio en el sistema de almacenaje del ordenador, es decir, en la memoria principal del equipo. A cada variable le da un nombre simbólico que suele describir la información que guarda en ella. A su vez, ese espacio que forma la variable tendrá una información conocida o desconocida. Los datos que se procesan en una variable, son traducidos a binario para poder operar con ellos. Vea un ejemplo de variable:

```
Var color;
```

A esta variable la ha llamado color, como su nombre indica, en ella se almacenan colores, por lo tanto, esta variable almacenará caracteres. Por ejemplo:

```
Var color = "Amarillo";
```

En JavaScript, como ha visto anteriormente, no es necesario definir el tipo de variable, es decir, yo puedo crear la variable color y almacenar en ella un número o un dato booleano. Sin embargo, en otros lenguajes de programación esto no sucede así, por lo que debe asignar a la variable un tipo de dato. Vea un ejemplo de cómo crearía la misma variable color en C:

```
Char color;
```

La palabra char está indicando en la variable color se van a almacenar datos de tipo texto. Lo mismo pasa si quisiera crear variables de otros tipos, por ejemplo, de números enteros:

```
Int numero;
```

Las variables, además de albergar diferentes tipos de datos, tienen un ámbito donde funcionan, es decir, hasta donde son accesibles. Para ello, se dividen en dos tipos:

- **Variables globales.** Son accesibles desde cualquier parte de su programa.
- **Variables locales.** Solo son accesibles en el ámbito donde se declaran.

Para declarar variables globales en JavaScript, hay que hacerlo fuera de la o las funciones. Por ejemplo, vea el caso de una variable que contiene un mensaje y una función que accede a la variable para mostrar el mensaje contenido en ella:

```
JS ejemplos.js > ⚡ mostrarMensaje
1   var mensaje= "Soy un mensaje";
2
3   function mostrarMensaje(){
4       alert (mensaje);
5   }
```

Por el contrario, para crear variables locales se introducirá la variable en la función, por lo que, si intenta llamar a esa variable desde otra función, esta no será accesible. Ejemplo:

```
1
2
3   <function contenedoraVariable(){
4       var mensaje= "Soy un mensaje";
5
6   }
7
8   <function mostrarMensaje(){
9       alert (mensaje);
10  }
```

Para dar un nombre identificativo a una variable, existen varias reglas que debe tener en cuenta:

- El primer carácter del nombre no puede ser un dígito.
- El resto de caracteres pueden ser combinaciones de dígitos o letras, incluido el guion bajo.
- No puede usar palabras que estén reservadas por el lenguaje que está usando. Por ejemplo, no puede llamar a una variable var.
- Las variables deben tener un nombre identificativo acorde a los datos que se van a almacenar en ella. Esto nos facilitará la tarea como programadores.



En un programa puede definir cuantas variables necesite, pero hay que tener en cuenta que cada variable ocupa espacio en la memoria RAM del ordenador, por lo que definir más variables de la cuenta puede interferir en el rendimiento de su programa.

TOME NOTA

2.3.3.2. Operaciones

Las operaciones se realizan sobre las variables para obtener un resultado concreto. Por ejemplo, si quiere crear un programa que calcule el área de un rombo en base a la diagonal mayor y diagonal menor que introduzca el usuario. Para ello, debe pedir dos datos al usuario la diagonal mayor, que se almacenará, a su vez, en una variable y la diagonal menor, que se almacenará en otra variable. Cuando haya recogido ambos datos, debe aplicar una serie de operaciones para calcular el área del rombo. Esta serie de operaciones dará otro dato que será el área del rombo, el cual debe almacenar en otra variable. Vea un ejemplo:

Var d; //diagonal menor

Var D; //diagonal mayor

Var área=(d*D)/2;

Como ha visto, las operaciones que se realizarán serán dos. La primera será la multiplicación de la diagonal mayor por la menor y después se dividen por dos. Al realizar estas operaciones, el resultado se asignará a la variable área.

Los tipos de operaciones que se pueden realizar sobre los datos de las variables serán:

- **Aritméticas.** Las operaciones aritméticas básicas son la suma, la resta, la multiplicación, la división, el resto, el incremento, el decremento y los compuestos.
- **Lógicas.** Las operaciones lógicas se componen por operaciones de mayor, menor, mayor o igual que, menor o igual que, iguales, estrictamente igual, NOT, AND, OR.
- **De bits.** Las operaciones con bits permiten manejar números en binario y se componen por operadores como AND, OR, XOR, desplazamientos a la izquierda, desplazamientos a la derecha u complementación.
- **Otras.** Por ejemplo, para instanciar un objeto utiliza new o para borrar datos de un objeto utiliza delete.

2.3.3.3. Comparaciones

Las comparaciones son un tipo de operaciones que necesita utilizar para que el programa tome ciertas decisiones según el resultado de la comparación. Las comparaciones se pueden hacer sobre dos o más variables. Las comparaciones son muy útiles a la hora de comparar datos de usuario y contraseña. Por ejemplo, cuando ingresa a su cuenta de Facebook y se escribe su usuario y contraseña, al darle clic al botón Entrar, lo que está haciendo es enviar los datos que ha escrito al servidor. Una vez que lleguen los datos al servidor, allí se compararán y si la comparación nos devuelve un dato true, abrirá su cuenta. En cambio, si el dato es false, no dejará entrar en la cuenta.

Las comparaciones son tipos de operaciones que se usan muy asiduamente y son muy importantes a la hora de programar. Para ello, dispone de varios operadores de comparación:

- **Igual que**, se representa con doble signo igual (==).
- **Estrictamente igual que**, se representa con triple signo igual (====).
- **Distinto de**, se representa con admiración de cierre y signo igual (!=).
- **Mayor que**, se representa con >.
- **Menor que**, se representa con <.
- **Mayor o igual que**, se representa con >=.
- **Menor o igual que**, se representa con <=.g

2.3.3.4. Asignaciones

Las asignaciones se utilizan para establecer valores a las variables. Por ejemplo, si tiene una variable denominada cliente y le quiere asignar un nombre, tan solo debe utilizar el símbolo de igualdad (=) para asignarle ese valor. Vea el siguiente ejemplo:

Var cliente = "Fernando";

Además, las asignaciones también se utilizan para recoger datos que quiera pasar a los parámetros de una función. Por ejemplo, si crea una función que calcule el área de un rombo, puede crear una variable que pase los parámetros a la función CalculoRombo y, a su vez, que muestre el resultado de la operación. Vea el siguiente ejemplo:

```
function CalculoRombo(d,D){  
    var d;  
    var D;  
    return area=d*D/2;  
  
}  
  
var resultado = CalculoRombo (10,15);  
  
console.log(resultado);
```

También, si dispone de dos objetos que pertenezcan a la misma clase, puede hacer una asignación de objetos. Como verá en el siguiente ejemplo, si tiene la clase Animal e instancia animal1 y animal2, puede asignarle los valores de animal 2 a animal1 y viceversa. Para ello, utilice el método Object.assign(); y como ve a continuación el objeto animal1 tiene los parámetros vacíos. Sin embargo, cuando ordena que aparezca en consola el objeto animal1, devuelve los parámetros que tiene animal2. A continuación, puede ver el ejemplo:

```
▶ Animal {especie: "perro", raza: "Pastor Alemán", color: "negro"}  
Terminal Ayuda ejemplos.js - Ejemplos - Visual Studio Code  
Bienvenido JS ejemplos.js X index.html C# csharp_ejemplo.cs ●  
ejemplos.js > ...  
1 class Animal{  
2     constructor (especie,raza,color){  
3         this.especie=especie;  
4         this.raza=raza;  
5         this.color=color;  
6     }  
7 }  
8  
9 const animal1 = new Animal(' ', ' ', ' ');  
10 const animal2 = new Animal ('perro','Pastor Alemán','negro');  
11  
12 Object.assign (animal1,animal2);  
13  
14 console.log(animal1);
```



TOME NOTA

Nunca debe confundir la comparación (==) de la asignación (=). Esto suele ser un error muy típico cuando se comienza a programar.

2.3.4. *Objetos del lenguaje de guion*

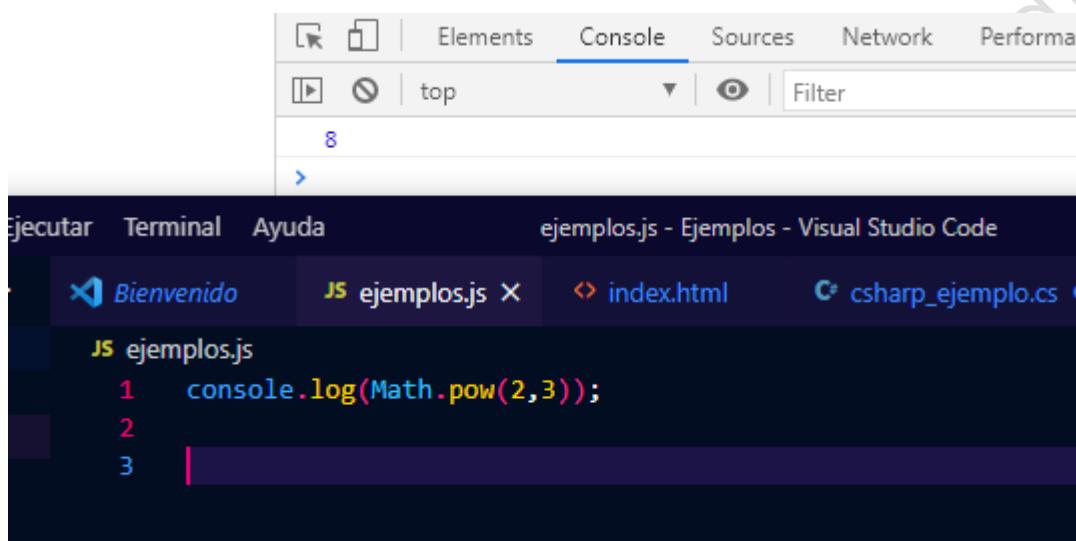
Los objetos son representaciones de algo concreto en el mundo real. Los objetos constan de atributos y métodos. Crear un objeto en JavaScript, es tan sencillo como especificar sus propiedades y sus métodos. Como se ha visto anteriormente, en JavaScript se pueden crear objetos directamente o se puede crear una clase de donde sacar los objetos. Una vez haya creado esa clase, tan solo debe instanciar el objeto mediante una variable.

A continuación, va a ver los pilares básicos sobre los que se sustentan los objetos:

a) Métodos

Decir método es lo mismo que decir función, la única diferencia es que cuando Se habla de métodos, está hablando de funciones que pertenecen a una clase. Los métodos permiten dividir el trabajo que hace un programa en tareas más pequeñas. Para poder utilizar los métodos de una clase, primero debe instanciar un objeto para llamar al método, por lo que utilice la siguiente sintaxis: `objeto.nombre_método();`

En JavaScript, también tiene objetos predefinidos con sus determinados métodos que nos permiten realizar diversas tareas. Por ejemplo, el objeto Math posee métodos como pow que se utiliza para calcular la potencia de un número. Vea un ejemplo:



```
ejemplos.js - Ejemplos - Visual Studio Code
Bienvenido  ejemplos.js  index.html  csharp_ejemplo.cs
JS ejemplos.js
1   console.log(Math.pow(2,3));
2
3
```

Como puede ver, para usar el objeto Math y su método pow, tan solo debe acceder al método y pasarle dos números por parámetro. Si quiere ver todos los métodos y propiedades del objeto Math, tan solo debe escribir Math. Y se nos abrirá una lista donde puede ver los métodos del objeto Math.

Véalo en el siguiente ejemplo:



```
1 console.log(Math.pow(2,3));
2
3 Math.~
```

The screenshot shows a code editor with the following code:

```
1 console.log(Math.pow(2,3));
2
3 Math.~
```

A dropdown menu is open over the word "Math.", listing various properties and methods. The "pow" method is highlighted with a blue background and white text, indicating it is selected or being typed. Other listed methods include log10, log1p, log2, max, min, random, round, sign, sin, sinh, and sqrt.

Al igual que el objeto Math, tiene el objeto Date, Array, Boolean, String, etc. Y con ellos sus métodos.

b) Eventos

Gracias a los eventos, los usuarios pueden interactuar con los elementos de la página web. Un evento es una acción del usuario en su documento web al que tiene que dar respuesta. Como programadores se debe dar respuesta a ese evento. Para ello, el manejador de eventos aporta los siguientes eventos:

- **Onclick.** Se utiliza para codificar un evento cuando el usuario clica un elemento.
- **onDoubleClick.** Hace lo mismo que el anterior pero cuando el usuario hace doble clic.
- **onMouseOver.** Salta el evento cuando el usuario pasa el ratón por encima del elemento.
- **onKeyDown.** El evento ocurre cuando se oprime una tecla.
- **onError.** Se activa cuando se produce un error durante la carga de un archivo externo.

Como estos eventos, hay muchísimos más que puede encontrar aquí. A continuación, va a ver dos maneras de cómo programar un evento onclick:

The screenshot shows a Visual Studio Code interface. On the left, there are two buttons labeled "Pulsar botón 1" and "Pulsar botón 2". The "Console" tab is active, displaying the following logs:

```
El segundo botón se ha pulsado ejemplos.js:16
El botón 1 se ha pulsado ejemplos.js:8
```

The "Terminal" tab shows the command "tar Terminal Ayuda ejemplos.js - Ejemplos - Visual Studio Code". The "File" tab shows "Bienvenido" and "ejemplos.js x". The "Sources" tab shows "index.html" and "csharp_ejemplo.cs".

The code editor displays the following JavaScript code:

```
var boton1 = document.createElement("input");
// Atributos o Propiedades:
boton1.id = "boton_2";
boton1.type = "button";
boton1.value = "Pulsar botón 1";
// Evento:
boton1.onclick = function(){
    console.log("El botón 1 se ha pulsado");
}
// Insertar el botón al final de 'div1':
document.getElementById("div1").appendChild(boton1);
// Crear un elemento botón
var btn = document.createElement("button"); // Insertar el texto del botón
btn.innerHTML = "Pulsar botón 2";
btn.onclick = function(){
    console.log ("El segundo botón se ha pulsado"); //dar funcionalidad al botón al clicar
}
document.body.appendChild(btn); // posicionar el botón en el body
```

c) Atributos

Los objetos se componen de atributos y métodos. Los atributos son los que definen las características individuales de cada objeto y los diferencia uno de otros. Los atributos determinan su apariencia, estado u otras cualidades. Por ejemplo, para el objeto taza podría definir los atributos como color, forma y material. Dependiendo de la taza, tendrá unos atributos u otros.

La idea fundamental es que pueda operar con estos atributos a través de los métodos implementados en dicho objeto.

d) Funciones

Las funciones son un conjunto de instrucciones que se agrupa bajo un mismo nombre. Este conjunto de instrucciones solo llega a ejecutarse si se llama a la función desde dentro del código.

Las funciones se caracterizan por:

- Permitir estructurar mejor su código.
- Permitir repetir una serie de instrucciones de una manera sencilla.
- Permitir reutilizar el código.

Debe distinguir entre los términos llamar a la función y definir la función. Cuando llame a la función, simplemente, está haciendo que esta se ejecute, es decir, el flujo del programa da un salto, ejecuta la función y luego vuelve al flujo natural del programa. Por otra parte, cuando se habla de definir una función, está hablando de crear las instrucciones necesarias para que la función tenga la funcionalidad que busca.

Las funciones constan de tres partes principales:

Definición o nombre de función. A las funciones puede darle el nombre que quiera, aunque lo más recomendable es darle un nombre que defina a la función. Por ejemplo, si crea una función que resta dos números introducidos por el usuario, lo lógico es que la función se llame Resta. Por lo tanto, la función quedaría de esta manera: function Resta ();

Parámetros que recibe la función. Los parámetros son los datos que recibe una función para operar con ellos. Las funciones, además, de recibir datos de entrada, también puede devolver datos de salida, tan solo debe usar la orden return. Tome el ejemplo anterior de la función resta y haga que esta devuelva el resultado de la resta:

```
Function Resta (x,y){
```

```
    Return x-y;
```

```
}
```

Una vez tiene la función definida, tan solo, tiene que llamarla para que se ejecute. El código quedaría de la siguiente manera:

```
Function Restar (x,y){
```

```
    Return x-y;
```

```
}
```

```
Console.log(Restar(2,5));
```

Como puede ver, ha tenido que pasarle dos parámetros a la función, ya que, si no le pasa los parámetros, la función no podrá hacer la resta, pues no tendrá ningún dato para poder hacerlo.

Cuerpo principal de la función. El cuerpo principal de la función, tan solo es el trozo de código que compone a la función. En el caso de la función Resta que ha visto anteriormente, el cuerpo sería el código que va entre llaves {}.

RECURSO MULTIMEDIA



2.4. Tipos de *scripts*: inmediatos, diferidos e híbridos

En JavaScript, encuentra tres tipos *de scripts*:

- **Scripts inmediatos.** Son *scripts* que se ejecutan al cargar la página. Estos *scripts* se sitúan en el propio código HTML dentro de la etiqueta <body>.
- **Scripts diferidos.** Estos *scripts* se cargan con la página, pero no se ejecutan hasta que el usuario realiza una acción que desencadena un determinado evento. Como, por ejemplo, pulsar un botón. Este tipo de *script*, también, se sitúa dentro de la etiqueta <body>.
- **Scripts híbridos.** Los *scripts* híbridos son una combinación de los dos scripts anteriores. Estos *scripts* se pueden situar tanto dentro de la etiqueta <body>, como dentro de la etiqueta <head>.

2.4.1. Script dentro del cuerpo del lenguaje de marcas

Los *scripts* se pueden escribir dentro de la etiqueta <body> utilizando, a su vez, las etiquetas <script> y </script>. Aunque, esto no es una práctica muy recomendada, ya que, hace que el código sea difícil de leer. Para insertar el código JavaScript en el <body> en su HTML debe hacerlo de la siguiente manera:

```
<html>  
<head>  
<title></title>  
</head>  
<body>  
<script> Instrucciones en JavaScript </script>  
</body>  
</html>
```

Ejecutables al abrir la página

Cuando necesita que se ejecuten acciones justo cuando se han terminado de cargar todos los elementos de la página, debe modificar la etiqueta <body> de HTML y utilizar el evento onload. Este tipo de eventos es muy utilizado, por ejemplo, en las tiendas online cuando salta el banner para suscribirse en el newsletter nada más entrar en la web. Para crear este tipo de eventos lo hará de la siguiente manera:

```
<html>
```

```
<head>  
<title></title>  
</head>  
  
<body onload="función()">  
  
<script> Instrucciones en JavaScript </script>  
  
</body>  
  
</html>
```

Ejecutables por un evento

Hay eventos que se ejecutan como respuesta a una acción del usuario hacia un elemento. Por ejemplo, el botón de un cuestionario que envíe información a la empresa. Vea un ejemplo para programar este tipo de eventos:

```
<html>  
<head>  
<title></title>  
</head>  
  
<body>  
  
<script> function PulsarBoton (){  
  
    Alert("Ha pulsado el botón");  
  
} </script>  
  
<button type="button" onclick="PulsarBoton()">Botón Prueba</button>  
  
</body>  
  
</html>
```

2.4.2. Script dentro del encabezado del lenguaje de marcas

Si quiere crear un *script* en el encabezado de su web, tan solo debe insertar las etiquetas `<script>` dentro del `<head>` de su documento. La diferencia entre declarar un *script* dentro del `<head>` o del `<body>`, es que los *scripts* del `<head>` se ejecutarán antes que los del `<body>`. Aunque, es más recomendable insertar los *scripts* justo antes de la etiqueta `</body>`, a veces debe crear los scripts dentro del `<head>`. Sobre todo, si los *scripts* que está insertando modifican el contenido de la página. Vea un ejemplo:

```
<html>  
<head>  
<title></title>  
  
<script> Instrucciones en JavaScript </script>  
  
</head>  
  
<body>  
  
</body>  
  
</html>
```

2.4.3. Ejecución de un script

Hasta ahora ha tenido la ocasión de probar algunos *scripts* sencillos, aunque, todavía tiene que aprender una de las bases para poder trabajar con el lenguaje. Existen dos maneras fundamentales de ejecutar *scripts* en la página. La primera se trata de la ejecución directa de *scripts*, en cambio, la segunda sucede como respuesta a la acción de un usuario.

a) Ejecución al cargar la página

Como ha visto anteriormente, puede hacer script que se ejecuten cuando la página se haya cargado. Para ello, tan solo debe indicar en el `<body>` con el atributo `onload` el nombre de la función que quiere que se ejecute al cargar la página. Por ejemplo, esto es muy útil si quiere que, al abrir su web, el usuario vea algún contenido importante en un pop up.

b) Ejecución después de producirse un evento

Los eventos son acciones que realiza el usuario. Los programas como JavaScript están preparados para atrapar estos eventos y realizar acciones como respuesta. De este modo se pueden programar programas y páginas interactivas, ya que controlará los movimientos del usuario y responderá a ellos.

Existen muchos tipos de eventos distintos, por ejemplo, la pulsación de un botón, el movimiento del ratón o la selección de texto en la página.

Las acciones que quiere realizar como respuesta a un evento se pueden indicar de muchas maneras distintas, por ejemplo, dentro del propio código HTML, en atributos que se colocan dentro de la etiqueta que quiere que responda a las acciones del usuario. Antes, ha visto cómo se puede crear un evento añadiendo onclick dentro de la etiqueta <button>. Ahora, va a ver cómo realizar lo mismo, pero llamando al botón desde JavaScript. Vea el ejemplo:

```
<html>
<head>
<title>Mi primera web</title>
</head>
<body>
<h1>Ejemplo de ejecución de script directa</h1>
<p>A continuación se realizará una suma y se mostrará el resultado</p>
<button id="boton1">Botón Prueba</button>
<script>
Function PulsarBoton(){
    Alert("Ha pulsado el botón");
}
Document.getElementById('boton1').addEventListener('click',PulsarBoton);
</script>
</body>
</html>
```

Como puede observar en el código, ha creado un botón en HTML y luego lo ha llamado en JavaScript para darle su funcionalidad. Para ello, ha utilizado el id dentro de la etiqueta botón.



TOME NOTA

Recuerde que el flujo de un programa siempre va de arriba hacia abajo, por lo que, para que funcione este código debe haber creado el botón antes que el script, si no, saltará un error.

c) Ejecución del procedimiento dentro de la página

Este es el método más básico de ejecutar *scripts*. En este caso se incluyen instrucciones dentro de la etiqueta <script>, tal como ha comentado en apartados anteriores. Cuando el navegador lee la página y encuentra un script, va interpretando las líneas de código y las va ejecutando una tras otra. A este tipo de ejecución de scripts, se le llama ejecución directa, ya que, se ejecuta de manera natural al flujo de código. Un ejemplo de ejecución directa de script es el siguiente:

```
<html>  
<head>  
<title>Mi primera web</title>  
</head>  
<body>  
<h1>Ejemplo de ejecución de script directa</h1>  
<p>A continuación se realizará una suma y se mostrará el resultado</p>  
<script>  
var num1=5;  
var num2=7;  
var resultado=num1+num2;  
  
alert('El resultado de la suma es '+resultado);  
</script>  
</body>  
</html>
```

En este ejemplo, el código primero cargará el `<head>` con el título correspondiente y después, comenzará a cargar lo que hay dentro del `<body>`. Cuando llegue al script, el navegador ejecutará la acción línea por línea.

d) Tiempos de ejecución

Los tiempos de ejecución los utilice para calcular la eficiencia de sus scripts. Obviamente, cuanto menos tiempo tarde su *script* en realizar una tarea más eficiente será.

Para poder medir el tiempo de ejecución de un *script*, va a utilizar unos scripts que coloca justo antes y después de iniciar su código. Veálo en el siguiente ejemplo:

```
<html>  
<head>  
</head>  
<body>  
<script>  
    Inicio = new Date();  
  
    //Código a medir  
  
    Fin = new Date();  
  
    Document.write ((fin-inicio)+ " milisegundos");  
  
</script>  
</body>  
</html>
```

e) Errores de ejecución

Cuando trabaje con *scripts* embutidos dentro de un documento HTML, puede experimentar, principalmente, tres tipos de errores:

- **Errores en tiempo de carga.** Estos errores suceden cuando la página se carga. Normalmente, estos errores ocurren debido a errores en la sintaxis del *script*.
- **Errores en tiempo de ejecución.** Puede ocurrir que la página en sí se cargue sin errores, aunque en ocasiones puede que no haya utilizado la sintaxis de manera totalmente. Por

ejemplo, al equivocarse con las minúsculas y mayúsculas experimentará un error en tiempo de ejecución.

- **Errores lógicos.** Estos errores son los más complejos a la hora de detectar un error y, también, suelen ser los errores más comunes que experimente. Más adelante, aprenderá cómo manejar este tipo de errores y corregirlos.

Cuando se produzcan este tipo de errores, será informado a través de la consola del navegador. Los errores más comunes a la hora de programar con JavaScript son:

- Utilización de cadenas de texto sin encerrar entre comillas "".
- Utilización de un solo signo de igualdad en las comparaciones, en vez de utilizar un doble signo de igualdad ==.
- Usar un objeto o variable que no ha sido previamente declarada.
- Uso incorrecto de las palabras reservadas.
- Mala colocación de las llaves {}, y paréntesis () en funciones y estructuras de control.

Antiguamente cuando no existían editores de texto como Sublime o Visual Code, estos errores eran más habituales, e incluso eran difíciles de ver. Sin embargo, si utiliza este tipo de editores, dirá la línea donde ha fallado el código, e incluso advertirá de por qué ha fallado. Vea un ejemplo de error de cierre de llaves {} y de no definición de una función en Visual Code:

The screenshot shows a code editor window with a dark theme. A file named 'ejemplo2.js' is open, containing the following code:

```
1 function (){  
2     alert("Mensaje de error");  
3 }
```

The cursor is at the end of the third line. Below the editor, the 'PROBLEMAS' tab is selected, showing two errors:

- Se esperaba un identificador. ts(1003) [1, 10]
- Se esperaba '}'. ts(1005) [3, 31]

A tooltip at the bottom right indicates: 'ejemplo2.js[1, 12]: El analizador esperaba encontrar un elemento {" que coincidiera con el del token {" aquí.'

3. ELEMENTOS BÁSICOS DEL LENGUAJE DE GUIÓN

Este tema se centra en el lenguaje JavaScript, ya que, es el lenguaje principal para el desarrollo de páginas web. Como se ha visto en el capítulo anterior, JavaScript es un lenguaje de programación interpretado, es orientado a objetos, débilmente tipado y dinámico. Actualmente, a JavaScript se le denomina como EcmaScript 6.

JavaScript se usa como lenguaje de programación del lado del cliente implementado con el HTML para mejorar tanto la interfaz de usuario como el dinamismo de la página web.

3.1. Variables e identificadores

Las variables son espacios en la memoria principal del ordenador que están listos para almacenar información y poder trabajar con ella. Para trabajar con las variables, lo que hace es referenciarlas por el nombre que le ha dado como identificador. Por ejemplo, var número; la referenciaría como "número".

Para darle nombre a las variables, debe seguir unas reglas:

- El nombre identificador solo puede contener letras, números y caracteres. No se permiten los espacios en blanco, sin embargo, sí puede usar el guion bajo (_).
- Es aconsejable que el primer carácter por el que empieza una variable sea una letra, ya que, cuando se escribe una variable con guion bajo, está haciendo referencia a variables privadas. Aunque en JavaScript no existan modificadores de visibilidad, puede crear variables privadas.
- JavaScript es un lenguaje de programación case sensitive, esto quiere decir que hay diferencia entre mayúsculas y minúsculas, por lo que, la variable Coche y coche, se interpretarán como dos variables diferentes.
- No puede nombrar a una variable con cualquier palabra reservada del lenguaje.

Vea, a continuación, algunas de las palabras reservadas del lenguaje JavaScript:

Abstract	Do	If	Package	Throw
Boolean	Double	Implements	Private	Throws
Break	Else	Import	Protected	Transient
Byte	Extends	In	Public	True
Case	False	Instanceof	Return	Try
Catch	Final	Int	Short	Var
Char	Finally	Interface	Static	Void
Class	Float	Long	Super	While
Const	For	Native	Switch	With
Continue	Function	New	Synchronized	Default
Goto	Null	This	Typeof	let



TOME NOTA

La memoria principal del ordenador es la memoria RAM, esta memoria se la denominada memoria volátil, ya que, al apagar el equipo se pierde toda la información que hubiera guardada en ella. Por lo tanto, todas las variables que usa un programa se pierden al apagar o reiniciar el equipo.



Más Info



RECURSO MULTIMEDIA



3.1.1. Declaración de variables

Para trabajar con variables en JavaScript, antes, se deben declarar. Sin embargo, no es obligatorio inicializarlas. Tan solo, debe darles un nombre identificador válido para poder referenciarlas en sus programas.

Para crear una variable, normalmente, se utiliza la siguiente sintaxis:

Var coche;

Var edad;

Por otra parte, en la versión EcmaScript 6 tiene otra manera de crear una variable. Para ello, se usa la palabra reservada let. La sintaxis sería la misma que con la palabra reservada var. La única diferencia entre var y let, es que las variables let solo alcanzan el bloque de código donde se ha definido. Vea el siguiente ejemplo:

```
js ejemplo2.js > ...
1  function varTest() {
2      var x = 31;
3      if (true) {
4          var x = 71; // ¡misma variable!
5          console.log(x); // 71
6      }
7      console.log(x); // 71
8  }

9
10 function letTest() {
11     let x = 31;
12     if (true) {
13         let x = 71; // variable diferente
14         console.log(x); // 71
15     }
16     console.log(x); // 31
17 }
18 }
```

Como puede observar, en la función varTest(), tiene dos variables var que tanto dentro del if, como fuera, son la misma variable y su valor cambia de 31 a 71. Sin embargo, en la función letTest(), la variable let que está fuera del if, es una variable diferente a la variable let que está dentro del if. Es por ello, que al hacer el console.log(x) dentro de la función letTest(), el resultado es 31 y no 71, porque se está referenciando a la variable let externa del if. También, hay que tener en cuenta que puede hacer uso de la variable var desde cualquier parte del código, sin embargo, esto no sucede con la variable let.

Las variables en JavaScript no necesitan tener un tipo de datos establecidos, es decir, en la misma variable se puede guardar información numérica, de texto, booleana, etc.

Dependiendo del alcance de la variable, puede describir dos tipos de variables:

- **Variables locales.** Este tipo de variables solo son visibles dentro del trozo de código donde se crean. Por ejemplo, si crea una variable de tipo local dentro de una función, no puede recurrir a ella desde fuera de la función. Para este tipo de variables, también, podría usar las variables let.
- **Variables globales.** Estas variables se crean en el programa principal para que sean accesibles desde cualquier parte del programa. Cabe destacar que, si crea una variable var fuera de una función y, después, crea una función donde nombrar a una variable sin escribir var delante, esta variable será global. Vea un ejemplo:

```
JS ejemplo2.js > ...
1  var numero = 2
2  function miFuncion (){
3      numero = 19
4      document.write(numero) //imprime 19
5  }
6  document.write(numero) //imprime 2
7 //llamamos a la función
8 miFuncion()
9 document.write(numero) //imprime 19 |
```

Como puede ver en la siguiente imagen, la variable que está dentro de la función es global, ya que, se puede acceder a ella desde fuera de la función. Sin embargo, si en la variable numero que está dentro de la función pusiera var obtendría el siguiente resultado:

```
1  var numero = 2
2  function miFuncion (){
3      var numero = 19
4      document.write(numero)
5  }
6  document.write(numero) //imprime 2
7 //llamamos a la función
8 miFuncion()           //imprime 19
9 document.write(numero) //imprime 2|
```



TOME NOTA

En JavaScript no es necesario declarar una variable antes de utilizarla. Y tampoco es necesario establecer el tipo de dato de la variable.

3.1.2. Operaciones con variables

Las variables, al fin y al cabo, son almacenes de datos que utiliza para operar con ellos y obtener como resultado otros datos. Gracias al uso de las operaciones puede operar con estos “almacenes” y obtener los resultados que necesita para que su programa funcione. Al operar con variables, los datos que contienen se modifican según el programa vaya operando con ellas.

Los ámbitos más comunes donde se opera con variables son:

- Al mostrar datos.
- Al pedir datos.
- Al enviar datos.
- Al recibir datos.
- En el uso de expresiones.
- En funciones.
- En llamadas a las funciones.
- En objetos.

Vea un ejemplo de operaciones con variables donde piden introducir su nombre y su apellido y devuelven una frase que dice cómo se llama y cuál es su apellido:

```
function concatenarNombre(){  
    var nombre = prompt("Introduzca su nombre ");  
    var apellido = prompt ("Introduzca su apellido ");  
    return `Se llama ${nombre} y su apellido es ${apellido}`;  
}  
  
console.log(concatenarNombre());
```

Las variables solo permiten operar con un solo dato, pero hay situaciones en las que necesita operar con más de un dato. En estas situaciones, utilice otro tipo de almacenaje de datos que verá más adelante.

3.2. Tipos de datos

En JavaScript, como ha visto anteriormente, no es necesario indicar los tipos de datos que almacena una variable. Pero, es importante saber qué tipo de datos está manejando para poder operar correctamente con ellos. En este apartado, aprenderá los tipos de datos que puede utilizar en JavaScript.

RECURSO MULTIMEDIA



3.2.1. Datos booleanos

Los datos booleanos son datos que representan la lógica binaria, ya que, solo pueden ser 0 o 1, o lo que es lo mismo, verdadero o falso. Este tipo de dato es muy útil cuando está utilizando estructuras de control en su programa. Por ejemplo, imagine que quiere crear una plataforma donde necesita saber cuánto tiempo pasa el usuario conectado. Para ello, debe utilizar este tipo de dato indicando que cuando sea false o 0, es que el usuario no está conectado y cuando sea 1 o verdadero es cuando está conectado y debe contar el tiempo desde que está a 1 hasta que vuelva a estar a 0.

3.2.2. Datos numéricos

En JavaScript no hay distinción entre números enteros o con decimales. Todos los datos numéricos se consideran de tipo number. Aunque, es cierto que cuando se opera con números decimales, el tiempo de cálculo y los recursos requeridos por la computadora aumentan.

Por otro lado, tiene el valor especial NaN (Not a number) o lo que es lo mismo, no es un número. Este valor se devuelve cuando intenta hacer cálculos numéricos y su variable no contiene datos numéricos, sino de otro tipo.

En otros lenguajes de programación, sí debe especificar el tipo de dato numérico que almacena en la variable, y por ello, es bueno conocer los datos numéricos existentes en otros lenguajes. Generalmente los datos numéricos con los que se topa si utiliza otros lenguajes de programación serán los siguientes:

- **Entero.** Los números enteros son los que no tienen decimales y en programación se representan como Int. A veces puede encontrar tipos como BigInt, que se usan para números enteros enormes y el TinyInt, que se usa para números enteros pequeños.
- **Reales.** Los números reales son aquellos que utilizan decimales. Este tipo datos numéricos puede verlos representados con float o double. La diferencia entre estos dos tipos de datos es que double es dos veces más preciso que float, por lo que, la mayoría de las veces, refleja los números reales como double.

3.2.3. Datos de texto

Los datos de texto se representan como String, es decir, como cadenas de texto. Por ejemplo: "Mi nombre es Francisco" sería una cadena de 20 caracteres, ya que, los espacios en blanco también cuentan como caracteres.

En JavaScript, los datos String se delimitan entre comillas, por lo que, para crear una variable de tipo String, tan solo debe hacer lo siguiente:

```
Var nombre="Francisco"
```

Por el contrario, si el tipo de dato String se encuentra anidado dentro de una instrucción que utilice comillas dobles, este deberá escribirse bajo comillas simples (""). Por ejemplo:

Nombre="Buscar nombre"; onClick="buscar('Francisco')"



TOME NOTA

Cualquier dato que esté encerrado bajo comillas dobles, será tratado como texto. Por ejemplo, un número de teléfono se puede escribir como texto, ya que, no va a operar matemáticamente con él.

Ejemplo: var miTelefono="852566965"

3.2.4. Valores nulos

En JavaScript hay dos tipos de valores nulos que funcionan de forma diferente. El primero, es el valor null. Una variable con valor null, JavaScript la toma como una variable existente y puede operar con ella. Sin embargo, undefined para JavaScript es una variable o que no ha sido declarada, o que jamás se le ha asignado un valor. Es decir, undefined realmente no es un valor para JavaScript y, por lo tanto, no puede operar con él. Ejemplo:

```
Var num= 1 + null;
```

```
Var num1= 1 + undefined;
```

Si hace la prueba con estas dos variables en JavaScript, verá que la primera variable da como resultado 1, ya que opera null como si fuera un 0. Sin embargo, con la variable num1 no ocurre esto, ya que, undefined no es ningún valor y, por lo tanto, no es operable. El resultado de la variable num1 será NaN.



TOME NOTA

Todas las variables en todos los lenguajes de programación llevan asociadas un tipado de datos , aunque en JavaScript no hace falta definirlo explícitamente.

3.3. Operadores y expresiones

Los operadores son los símbolos que se utilizan al operar con variables. Por ejemplo, una operación es una multiplicación entre dos variables como pueden ser $x*y$. En JavaScript, puede operar con variables sin tener en cuenta el tipo de datos que se contienen en ellas. En este apartado, va a ver los tipos de operaciones que pueden realizar en JavaScript, como también, las expresiones.

Una expresión es un conjunto de variables y operadores que se evalúan con un único valor, es decir, da como resultado un único tipo de dato. Existen dos tipos de expresiones:

- Las que asignan un valor a una variable. Por ejemplo, `a=12`;
- Las que poseen un valor. Por ejemplo: `5^2`

A continuación, va a ver la jerarquía de las operaciones. Esta jerarquía es la que se usa, de igual manera, en matemáticas:

- Los paréntesis (), los corchetes [] y la regla del punto `objeto.atributo`
- Negación !, negativo -, incremento ++ y decremento --.
- Multiplicación *, división / y módulo %
- Suma + y resta -
- Operadores condicionales
- Condicionales de igualdad == y desigualdad !=
- Lógicos y booleanos
- Asignación

Vea un ejemplo de la jerarquía de las operaciones:

$5 * (5+2) - 10[2^2 + (4-3)]$

Esta operación, daría como resultado 30.

Más Info▼

RECURSO MULTIMEDIA



3.3.1. Operadores de asignación

Los operadores de asignación se utilizan para dar un valor definido a una variable, en este caso. Si tiene una variable llamada num y quiere que desde el primer momento tenga un valor asignado. Tan solo, debe asignar la variable num a ese valor, utilizando un signo de asignación. Por ejemplo: `num = 2;`

Aparte de este tipo de asignación simple, en JavaScript puede utilizar otros tipos de asignaciones:

- **Asignación con suma.** Este tipo de asignación está representada con los símbolos `+=` y sirve para sumarle a la función un valor mientras se le asigna. Por ejemplo, coja el ejemplo anterior:

`Num = 2;`

`Num += 5;`

En este caso num pasaría de valor 2 a valer 7. Escribir `num += 5`, es lo mismo que escribir `num = num+5;`

- **Asignación con resta.** La asignación con resta se representa con los símbolos `-=` y sirve para restar un valor a la variable a la que se le asigna. Por ejemplo, `num -= 1`. Esta asignación funcionaría igual que la anterior, pero, en este caso, restando.
- **Asignación con multiplicación.** Esta asignación se representa con los símbolos `*=` y se operaría de la siguiente manera: `num *= 3`. Por lo que num, pasaría de valer 2 a valer 6.
- **Asignación con división.** Está representada con los símbolos `/=`. Con este tipo de asignación, el contenido de la variable es dividido por el valor de la asignación. Por ejemplo: `num /= 2`. Su resultado sería 1, ya que la variable en su origen valía 2.
- **Asignación con módulo.** Está representada con los símbolos `%=`. Esta asignación devolverá el valor del resto al dividir la variable por el valor con el que se le asigna. Por ejemplo:

`Num = 5;`

`Num% = 2;`

El resultado sería 1, ya que, al dividir 5 entre 2 da como resto 1.

3.3.2. Operadores de comparación

Los operadores de comparación se usan para comparar datos entre dos o más variables y, así, hacer que el programa sea capaz de tomar decisiones en base al resultado de estas comparaciones. Para esta tarea, puede utilizar los siguientes operadores de comparación:

- **Igualdad.** Se escribe con doble signo igual `==` y devuelve como resultado verdadero si ambos valores son iguales si no, devolverá falso. Ejemplo: `x=2; y=2; x==y;` devuelve verdadero.
- **Distinto.** Utiliza el símbolo `!=` y devuelve verdadero si los valores contenido en ambas variables son distintos si no, devolverá falso. Ejemplo: `x = 3; y = 0; x!=y;` devuelve verdadero.
- **Mayor que.** Se representa con el símbolo `>` y devolverá verdadero si el valor de la primera variable es mayor que el de la segunda. En caso contrario, devolverá falso. Ejemplo: `x = 10; y = 5; x>y;` devuelve verdadero.
- **Menor que.** Se escribe con el símbolo `<`, y devolverá verdadero si el valor de la primera variable es menor que el de la segunda. Ejemplo: `x = 2; y = 5; x<y;` devuelve verdadero.

- **Mayor o igual que.** Este operador usa el símbolo \geq , en este caso devolverá verdadero si la primera variable tiene un valor mayor o igual que la segunda. Ejemplo: $x = 5; y = 5; x \geq y$; devuelve verdadero, ya que, ambas variables tienen el mismo valor.
- **Menor o igual que.** Se representa con el símbolo \leq , en el caso que la primera variable sea menor o igual a la segunda, el resultado que obtendrá será verdadero. Ejemplo: $x = 2; y = 3; x \leq y$; devuelve verdadero.



Es importante comprender las diferencias entre menor o mayor y menor o igual que o mayor o igual que, ya que, cuando usa mayor o igual que o menor o igual que se está incluyendo un rango de números pudiendo ser verdadero que ambas variables tengan el mismo valor.

TOME NOTA

3.3.3. Operadores aritméticos

Los operadores aritméticos son los más utilizados diariamente, son los operadores que se utilizan para realizar operaciones matemáticas simples. A continuación, va a ver los tipos de operadores aritméticos que puede utilizar:

- **Suma.** Se representa con el símbolo $+$ y puede usarlo para sumar tantos valores o variables necesite.
- **Resta.** Se representa con el símbolo $-$ y puede usarlo para restar tantos valores como variables necesite.
- **Multiplicación.** Se representa con el símbolo $*$ y lo utiliza para multiplicar valores y variables.
- **División.** Se representa con el símbolo $/$, se utiliza para dividir valores y variables.
- **Resto o módulo de una división.** Se representa con el símbolo $\%$ y permite obtener el resto de una división.

3.3.4. Operadores sobre bits

Este tipo de operadores permiten tratar los datos en forma de cadena binaria, es decir, cadena de ceros y unos. Si quiere trabajar con bits, tiene que entender muy bien la lógica binaria.

A continuación, va a ver cómo funcionan los operadores de bits:

- **Operador AND.** El operador AND toma dos números enteros y realiza la operación AND lógica en cada par correspondiente de bits. El resultado en cada posición es 1 si el bit correspondiente de los dos operandos es 1, y 0 de lo contrario. Vea un ejemplo:

Valor A	Valor B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Operador OR. Una operación OR de bit a bit, toma dos números enteros y realiza la operación OR inclusivo en cada par correspondiente de bits. El resultado en cada posición es 1 si el bit correspondiente de cualquiera de los dos operandos es 1, y 0 si ambos bits son 0. Vea un ejemplo:

Valor A	Valor B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Operador Exclusive XOR. El operador XOR toma dos números enteros y realiza la operación OR exclusivo en cada par correspondiente de bits. El resultado en cada posición es 1 si el par de bits son diferentes y 0 si el par de bits son iguales. A continuación, verá un ejemplo:

Valor A	Valor B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Operador de negación NOT. A este operador, también se le denomina operador de complemento. Es una operación unaria que realiza la negación lógica de cada bit invirtiendo los bits del número, de tal manera que los ceros se convierten en unos y viceversa. Vea un ejemplo:

Valor A	NOT A
0	1
1	0

Desplazamiento lógico de bits. El desplazamiento lógico se utiliza para mover bits hacia la izquierda o hacia la derecha para, así, colocarlos en la posición adecuada. En programación, el primer operando representa el binario que quiere modificar, mientras que el segundo indica las posiciones a desplazar. Vea un ejemplo:

Desplazamiento a la izquierda	Desplazamiento a la derecha
$00011100 \ll 1 = 00111000$	$00011100 \gg 1 = 00001110$
$00011100 \ll 2 = 01110000$	$00011100 \gg 2 = 00000111$



En lógica binaria los ceros a la izquierda no tienen valor alguno, mientras que los ceros a la derecha sí lo tienen.

TOME NOTA

3.3.5. Operadores lógicos

Los operadores lógicos principales son:

Operador Y lógico o, lo que es lo mismo, AND (&&): Este operador lógico devuelve un valor verdadero (true), si ambos operandos son ciertos, o falso (false) si tan solo uno de los valores no es cierto. Este operador lógico funciona de la siguiente manera, los datos del primer operando, que es el de la izquierda, son convertidos a datos booleanos, es decir, verdadero o falso. Si el dato que se ha convertido a dato booleano es falso, automáticamente, se detiene la operación. Por el contrario, si el primer valor es verdadero (true), la operación continua. A continuación, verá el ejemplo de una expresión verdadera:

a=a && b=b

Tiene dos expresiones, a=a y b=b, con el operador AND, simplemente está juntando ambas expresiones. Está diciendo que *a* es igual a *a* y que *b* es igual a *b*. El primer término (a=a) se pasa a valor booleano, es decir, verdadero o falso. Como *a* es igual a *a*, el resultado que obtendrá es verdadero (true), por lo que la operación continua hacia la segunda expresión. La segunda expresión está diciendo que *b* es igual a *b*, por lo que esta segunda expresión también va a dar como resultado un valor verdadero (true). Por lo tanto, tendrá: true && true = true. Como ambos valores son verdaderos, la operación da como resultado verdadero (true).

Por otra parte, va a ver un ejemplo de una expresión falsa:

a=b && b=b

En esta operación tiene dos expresiones, una que dice que *a* es igual a *b* y la segunda que dice que *b* es igual a *b*. En esta operación, la primera expresión dará como resultado un valor falso (false), ya que, *a* no es igual a *b* (false && true). Y, aquí se detendrá la operación, ya que, como ha visto anteriormente, si un valor es falso, tendrá como resultado un valor falso (false) y no importará los resultados que obtenga posteriormente. En el caso que (b=b && a=b) la operación hubiera continuado pero el resultado que habría obtenido sería el mismo.

Operador O lógico o, lo que es lo mismo, OR (||): Este operador funciona de manera similar al anterior, el valor del primer operando, que es el de la izquierda, se convierte a valor booleano (true o false) pero al contrario que el operador anterior, cuando la primera expresión da como resultado

falso (false), este continúa operando. Cuando el primer operando da como resultado el valor verdadero (true) es cuando el operador se detiene. A continuación, va a ver ambos ejemplos:

var= número

número=5

número=1 || número=5

Primeramente, ve que tiene una variable. Las variables son elementos que se utilizan para almacenar valores, ya sean numéricos o de texto. A su variable la ha llamado número, porque va a almacenar, en este caso, un número. Imagine que en esta variable se van a almacenar un número del 0 al 100 de forma aleatoria. La siguiente expresión *OR* es la que va a tomar los valores de *número* y va a procesarlos para obtener un valor booleano. Por lo tanto, si el número que está almacenado en la variable *número* es igual a 5 (número=5) entonces obtendrá como resultado de la primera expresión falso (false) y el operador pasará a procesar la siguiente expresión comparando si *número* es igual a 5. Como *número* es igual a 5 (número=5) la segunda expresión dará como resultado verdadero (true).

Var=número

número=1

número=1 || número=5

En esta ocasión la variable *número* equivale a 1. El operador comenzará comparando el primer operando (número=1) y lo procesará como valor booleano, que en este caso es verdadero (true). En este momento, la operación se detendrá, ya que, ha encontrado uno de los dos valores válidos para que la operación sea verdadera.

Operador NO lógico o, lo que es lo mismo, NOT (!): El operador NOT (!) es un operador más complejo que los dos anteriores y no debe usarse hasta que se tenga cierta destreza programando. Si *a* equivale a un número y este es distinto a 0, entonces (*a=true*) y (*!a=false*). Por el contrario, si (*a=0*), entonces (*a=false*) y (*!a=true*). Tenga en cuenta que *!a* es la negación de *a* y, por lo tanto, siempre será lo contrario a lo que equivalga *a*. Para las cadenas de texto se considera que la cadena de texto vacía es un valor falso (false) y las cadenas de texto con texto en su interior es verdadero (true), por lo que si (*a='casa'*), entonces *a* equivale a verdadero y *!a* a falso. Vea los siguientes ejemplos:

a=10 al procesar este dato a booleano obtendría que *a=true*

b=0 al procesar este dato a booleano obtendría que *b=false*

entonces:

!a=0 el valor booleano es *!a=false*

!b= (un número distinto de 0) el valor booleano de *!b=true*

Vea un ejemplo con cadenas texto:

a='casa' al procesar este dato a booleano obtendrá que a=true

b="" al procesar este dato a booleano obtendrá que b=false

entonces:

!a=false

!b=true

El operador lógico de negación también puede combinarse con el operador relacional (=) dando lugar a la expresión (!=), es decir, 'es distinto a' y también se puede combinar con los otros operadores lógicos. Vea un ejemplo:

```
var numero1=10;
```

```
var numero2=0;
```

```
numero1!=0 && numero2!=3
```

El resultado de esta operación es resultado=true, ya que el número 1 es distinto a 0 y el número 2 es distinto a 3.

3.3.6. Operadores de cadenas de caracteres

Además de los operadores de comparación, que se pueden usar en valores de cadena, el operador de concatenación, representado por el símbolo +, concatena valores de cadena devolviendo otra cadena que es la unión de las dos o más cadenas concatenadas. Vea un ejemplo:

```
Var cadena1= "Me llamo ";
```

```
Var cadena 2 = "Ernesto";
```

```
Cadena3 = cadena1 + cadena2;
```

El resultado de la cadena3 sería Me llamo Ernesto.

Además, con las cadenas de caracteres, también, puede utilizar el signo += para añadir una cadena de texto dentro de una variable en la que ya haya una cadena de texto y, así, unir ambas cadenas. Por ejemplo.

```
Var cadena = "Esto es ";
```

```
cadena += "un ejemplo";
```

Como resultado obtendrá la cadena: Esto es un ejemplo.

Aparte de estas formas clásicas de unir cadenas de texto, en EcmaScript 6, tiene la posibilidad de utilizar plantillas de cadena. Las plantillas de texto, o template strings, son cadenas literales de texto incrustadas en el código fuente que permiten su interpolación mediante expresiones.

Para crear estas cadenas, debe utilizar las comillas invertidas `` para delimitar las cadenas. Y, para añadir marcadores de posición, utilice el signo del dólar \$. Vea un ejemplo:

Var nombre = "José";

```
Document.write(`Mi nombre es ${nombre}`);
```

Como puede observar, la cadena de texto que hay en la variable nombre se posiciona dentro del marcador de posición marcado con \$, mientras que la cadena de texto literal se escribe directamente dentro de la expresión.

Como resultado obtendrá la siguiente cadena de texto: Mi nombre es José.

Este tipo de cadenas son muy útiles, por ejemplo, a la hora de hacer programas que tengan que coger un dato que va cambiando. Imagine una aplicación de banca electrónica en la que cada vez que entra con su usuario salude diciendo Hola + (su nombre). Pues en este caso, usaría las plantillas de cadena.



TOME NOTA

Es muy importante no confundir las tildes invertidas `` con el símbolo de apostrofe '' , pues si intenta crear plantillas de cadena con el apostrofe, dará error.

3.3.7. Operadores especiales

Los operadores especiales de JavaScript son:

- **Operador condicional ternario.** Este operador es el único de JavaScript que tiene tres operandos. Este operador se usa con frecuencia como un atajo para la instrucción IF. La sintaxis de este operador es la siguiente: condición ? expr1 : expr2

La condición se evalúa como true o false. Si la condición es true, entonces el operador retorna el valor de la expresión 1, de lo contrario, devuelve el valor de la expresión 2. Vea un ejemplo:

Var velocidad;

Velocímetro = velocidad <= 120 ? "Velocidad permitida" : "Velocidad no permitida";

Dependiendo del valor que tome la variable velocidad, si es menor o igual a 120 verá un mensaje en pantalla que no dirá que la velocidad está permitida, si no dirá que la velocidad no está permitida.

- **Operador coma.** El operador coma tiene como objetivo separar múltiples expresiones para evaluarlas una a una, de izquierda a derecha, devolviendo el valor del último operando. Su sintaxis es la siguiente:

Expr1, expr2, expr3...

Esto quiere decir que el intérprete recibe varias expresiones encadenadas para pasar a evaluar cada una de las mismas en estricto orden para, finalmente, retornar el último valor obtenido.



TOME NOTA

Hay que distinguir entre el operador coma que está tratando en este apartado y la coma que utiliza para separar variables con var, pues no son lo mismo. Con el operador coma puede utilizar funciones como operandos

Vea un ejemplo:

```
Var doble = function (x) {
```

```
    Return x*2;
```

```
};
```

```
Var triple = function (x){
```

```
    Return x*3;
```

```
}
```

```
Doble (4), triple (3); // El resultado es 9, pues coge el último valor.
```

Además, el uso del operador coma está muy ligado al trabajo con bucles y resulta relativamente frecuente verlo dentro de los parámetros de una instrucción for. Por ejemplo en el siguiente ejemplo:

```
Var log = function () {
```

```
Document.write ('El valor de x es: ', x);  
};  
  
For (var x = 0; log(), x< 5; x++){  
  
    Console.info(x);  
  
}
```

El resultado que dará este bucle es el siguiente:

El valor de x es: 0

0

El valor de x es: 1

1

El valor de x es: 2

2

El valor de x es: 3

3

El valor de x es: 4

4

El valor de x es: 5

Como puede observar en el resultado, la función se ejecuta 6 veces, mientras que la expresión que hay dentro del bucle for, tan solo, se ejecuta 5 veces. Esto se debe a que, al poner la función antes de que el intérprete de JavaScript evalúe la expresión `x<5`, ya se ha ejecutado la función.

El operador coma también se puede usar como ampliación del operador condicional ternario y, así, ahorrarse crear una estructura con IF. Vea un ejemplo:

```
Var jugadorPierde = () => {  
  
    Vidas ? (vidas--, continuar()): (finJuego(), salirJuego());  
  
}
```

En este ejemplo si el jugador de una partida vive, se le reduce el número de vidas y continúa jugando, si no, el juego se termina.

- **Operador Delete.** Este operador se utiliza para borrar un objeto, una propiedad de un objeto, una variable, o un elemento de un array. Este operador devuelve true si la operación se borra con éxito, y devuelve false en caso contrario. Su sintaxis varía dependiendo de qué es lo que quiere eliminar. Vea varios ejemplos:
 - Delete nombre_objeto; // Con esta orden, elimina un objeto.
 - Delete nombre_objeto.atributo; // Con esta orden elimina un atributo de un objeto.
 - Delete nombre_array[índice] // Con esta orden elimina los datos que haya que en índice que indique de un array.
- **Operador In.** Este operador devuelve true si la propiedad especificada está en el objeto indicado, por otro lado, devolverá false en caso contrario. La sintaxis es la siguiente:

Prop in object

Prop es una cadena o expresión numérica que representa el nombre de una propiedad o el índice de un array.

Object es el objeto sobre el que se comprueba si contiene la propiedad con el nombre especificado. Vea un breve ejemplo:

```
Var coche ={marca: "BMW", modelo:"Serie 1" año:2021};
```

```
"marca" in coche //Devuelve true
```

```
"modelo" in coche // Devuelve true
```

```
"color" in coche // Devuelve false.
```

- **Operador Instanceof.** Este operador permite comprobar si un objeto pertenece a una clase concreta. Este operador se utiliza cuando necesita comprobar el tipo de objeto en tiempo de ejecución. Por ejemplo, cuando aprenda a controlar excepciones, puede recurrir a diferentes códigos de manipulación de excepciones dependiendo del tipo de excepción tomada. Vea un ejemplo:

```
Color1=new String ("azul");
```

```
Color1 instanceof String // Devuelve verdadero
```

```
Color2 = "rojo";
```

```
Color2 instanceof String // Devuelve falso.
```

- **Operador New.** Este operador se usa para crear instancias de objetos, ya sean creados por nosotros o predefinidos por JavaScript. La sintaxis de este operador es:

```
nombreObjeto = new tipoObjeto();
```

Vea un ejemplo con un objeto predefinido por JavaScript:

```
Dia1= new Date(2021,03,17);
```

Ahora Dia1 sería un objeto de tipo Date().

- **Operador This.** Este operador se utiliza para hacer referencia al propietario de la función que la está invocando o en su defecto, al objeto donde dicha función es un método.

Cuando no está dentro de una estructura definida, esto es un objeto con métodos, el propietario de una función siempre es el contexto global. En el caso de los navegadores web, dicho objeto será window.

Vea un ejemplo donde tiene los atributos de un objeto y quiere introducir un nuevo atributo:

```
Var miPerfil = {
```

```
Nombre: "José",
```

```
Apellido: "García",
```

```
nombreCompleto: this.nombre + this.apellido
```

```
}
```

En este caso el operador this, está referenciando al propio objeto y está accediendo a un atributo del mismo.

Operador Typeof. Este operador devuelve una cadena y dice el tipo de valor del operando, es decir, si es string, number, boolean, etc. Vea un ejemplo:

```
Var nombre = "José";
```

```
Typeof nombre; //devuelve string.
```

Operador Void. El operador void especifica una expresión que se evalúa sin devolver un valor. Por ejemplo, se puede crear un enlace tipo void para enviar un formulario cuando el usuario haga clic en el botón. Vea el ejemplo:

```
<a href="javascript:void(document.forms["miFormulario"].submit())">Pulse para enviar</a>.
```

3.3.8. Expresiones de cadena

Las expresiones de cadena permiten comprobar en cadenas de datos una serie de condiciones. Por ejemplo, si quiere escribir un campo que sea de contraseña, puede indicar qué caracteres quiere que conformen la contraseña. O, por ejemplo, si quiere crear un campo tipo email, se comprobará que el email contenga determinados caracteres como una @ y un punto.

Al igual que las cadenas de string van encerradas entre comilla dobles "", las expresiones de cadena van encerradas entre dos barras //, y se le asignan a una variable. Vea un ejemplo:

```
Var cadena=/ejemplo/;
```

Imagine que quiere saber si un texto contiene letras minúsculas, lo que debe hacer es lo siguiente:

```
Var buscarMinuscula=/[a-z]/;
```

```
Var palabra = "COcHE";
```

```
Alert(buscarMinuscula.test(palabra)); // Esta expresión devolverá true.
```

Del mismo modo, puede establecer diferentes patrones, como por ejemplo [aeiou] para buscar vocales o, [0-9] para buscar dígitos. A continuación, va a ver algunos caracteres especiales que puede usar para buscar datos:

Caracteres	Significado
x y	Coincide con x o y. Por ejemplo /azul rosa/ reconoce el "azul" en "chaqueta azul" y "rosa" en "chaqueta rosa".
\w	Busca caracteres alfanuméricos
\W	Devuelve falso si solo contiene caracteres alfanuméricos
\d	Busca dígitos
\D	Devuelve falso si contiene solo dígitos
\s	Contiene alguno de los caracteres que definen espacios en blanco
\S	Devuelve falso si solo contiene caracteres que definen espacios en blanco
^	Se utiliza para indicar que se busca al principio de la cadena
\$	Se utiliza para indicar que se busca en el final de la cadena

Existen muchos más caracteres especiales como los que ha visto. Puede encontrarlos en la siguiente web

RECURSO MULTIMEDIA



3.3.9. Expresiones aritméticas

Al igual que en las expresiones de cadena, en las expresiones numéricas puede utilizar las mismas reglas para validar un campo. Imagine que quiere crear un campo que pida un número de teléfono. Como sabe, un número de teléfono tiene 9 dígitos, por lo tanto, puede crear una expresión que valide este campo de la siguiente manera:

Var teléfono =/\d{3}-\d{3}-\d{3}/// La expresión dará verdadero si número introducido tiene el siguiente formato 111-111-111

Otro ejemplo que puede ver es que, si quiere configurar un campo para el DNI, tendría que configurar un campo que pida 8 dígitos, un guion y una letra mayúscula al final. Para ello, realizaría la siguiente expresión:

Var dni = /^\d{8}-[A-Z]\$/;

Vea, a continuación, una tabla con algunos caracteres útiles para usar en este tipo de expresiones:

Caracteres	Descripción
[a-z]	Todos los caracteres en minúscula
[^abc]	Todos los caracteres, excepto a, b y c
A*	Aparición de a cero o varias veces
A+	Aparición de a una o más veces
A?	Aparición o no de a
A{m,n}	Aparición de a de n a m veces
()	Agrupar caracteres

3.3.10. Expresiones lógicas

Para construir las expresiones lógicas, utilice los operadores lógicos. Mediante estos operadores se unirán las diferentes expresiones. Recuerde que los operadores lógicos son AND, OR y NOT. Las expresiones lógicas las puede combinar con el resto de expresiones y de operadores e, incluso, con las variables. Vea un ejemplo.

Var hora = 9;

```
Var finDeSemana = true;  
  
If (hora < 10 || hora > 18 || finDeSemana) {  
  
    Alert ("La oficina está cerrada");  
  
}
```

El resultado de esta expresión, hará saltar la alerta y el programa dirá que la oficina está cerrada, ya que, la variable finDeSemana es true, y como está usando un operador lógico OR escogerá la primera expresión que sea verdadera para ejecutar lo que hay dentro del if.

3.3.11. Expresiones de objeto

En JavaScript tiene el objeto RegExp, que sirve para hacer coincidir texto con un patrón. Hay dos maneras de crear un objeto RegExp, una notación literal y un constructor. Los parámetros de la notación literal se encierran entre barras y no utilizan comillas. En cambio, los parámetros de la función constructora no se encierran entre barras, y utilizan comillas. Vea ambos ejemplos:

Var er = /ab+c/i; // Notación literal.

Var er = new RegExp ('ab+c','i') //Constructor con patrón de cadena como primer argumento.

Var er = new RegExp (/ab+c/, 'i') // constructor con expresión regular literal como primer argumento.

Esta última expresión está disponible desde la versión EcmaScript 6.

Además, el objeto RegExp dispone de las siguientes propiedades:

- **lastIndex.** Devuelve la posición del carácter donde empieza la siguiente coincidencia en una cadena de búsqueda. Inicializado en -1.
- **\$1...\$9.** Devuelve las nueve partes memorizadas que se encontraron durante la coincidencia de patrones.
- **Flags.** Devuelve un string con los flags activados en la expresión regular.
- **Source.** Devuelve un string con la expresión regular original al crear el objeto.
- **Global.** Comprueba si el flag g está activo.
- **ignoreCase.** Comprueba si el flag i está activo.
- **Multiline.** Comprueba si el flag m está activo.
- **Unicode.** Comprueba si el flag u está activo.
- **Sticky.** Comprueba si el flag y está activo.

Las expresiones regulares tienen seis indicadores opcionales que permiten funciones como la búsqueda global y que no distinga entre mayúsculas o minúsculas. Estos indicadores se pueden usar por separado o juntos en cualquier orden y se incluyen como parte de la expresión regular. Vea los tipos de indicadores flags de los que dispone:

Flag	Descripción
G	Búsqueda global
I	Búsqueda que no distingue entre mayúsculas y minúsculas
M	Búsqueda multilínea
S	Permite que el . coincida con caracteres de nueva línea
U	Trata un patrón como una secuencia de puntos de código Unicode
Y	Realiza una búsqueda que coincida a partir de la posición actual en la cadena de destino.

La sintaxis para usar los flags es la siguiente:

Var ejemplo = /patrón flags;

Var ejemplo = new RegExp ('patrón','flags');

Por ejemplo, er = /\w+\s/g crea una expresión regular que busca uno o más caracteres seguidos de un espacio y busca esta combinación en toda la cadena.

3.4. Estructuras de control

Utiliza las estructuras de control para controlar el flujo de instrucciones de su programa. Gracias a ellas, puede redirigir el flujo para que se ejecuten las partes del programa que convenga en cada ocasión. Si no fuera por las estructuras de control, lo único que podría hacer es ejecutar una instrucción tras otra y no tendría forma de aplicar unas funciones u otras en según qué condiciones quisiera establecer.

Un ejemplo muy práctico para comprender la importancia de las estructuras de control es que cuando pida que un usuario introduzca su usuario y contraseña en la web, pueden ocurrir dos cosas. En primer lugar, que los datos sean correctos y, por lo tanto, se pueda loguear. En segundo lugar, podría ocurrir que el usuario o la contraseña sean incorrectos y, por lo tanto, la página volvería a pedirle las credenciales. Como puede observar, dependiendo de si las credenciales son correctas o no, el programa debe tomar unas acciones u otras. Esto es justamente lo que permiten hacer las estructuras de control.



Más Info



RECURSO MULTIMEDIA



3.4.1. Sentencia if

La sentencia IF permite ejecutar un bloque de instrucciones si se satisface la condición. En caso contrario, devolverá al flujo del programa. Sin embargo, la estructura IF se puede combinar con ELSE y, en este caso, podría ejecutar dos bloques de instrucciones dependiendo en si se satisface o no, la condición. La sintaxis de If es la siguiente:

```
If (condición){
```

Instrucciones a realizar en caso de que se cumpla la condición

```
}
```

Por otro lado, tiene la sintaxis de IF ELSE:

```
If (condición){
```

Instrucciones a realizar si se cumple la condición

```
} else {
```

Instrucciones a realizar si no se cumple la condición

```
}
```

A continuación, vea un ejemplo práctico:

```
If (num == 10){
```

Document.write("El número es 10");

```
}else {
```

Document.write("El número no es 10");

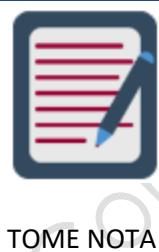
```
}
```

También puede concatenar varios valores de una variable haciendo uso de expresiones lógicas. Vea un ejemplo:

```
If (num == 1 || num == 2 || num == 3){  
  
    Document.write ("El número es un rango entre 1 y 3");  
  
}else {  
  
    Document.write("El número no puede ser ni 1, ni 2, ni 3");  
  
}
```

Además, en el caso que lo necesite, puede anidar varias estructuras if. Véalo en el siguiente ejemplo:

```
If (condición1){  
  
    If(condición2){  
  
        }else{  
  
            If (condición4){  
  
                }  
  
            }  
  
    }  
  
}
```



TOME NOTA

Es importante que al crear condiciones de evaluación en una estructura if, tenga en cuenta que debe utilizar el operador de comparación == y no confundirlo con el de asignación =, ya que, si asigna el primer valor a la variable, siempre se ejecutaría el primer bloque de código, pues la condición siempre sería verdadera.

Hay que distinguir entre el operador coma que está tratando en este apartado y la coma que utiliza para separar variables con var, pues no son lo mismo.

Con el operador coma, puede utilizar funciones como operandos

3.4.2. Sentencia while

El bucle WHILE se utiliza para realizar acciones mientras se cumpla la condición establecida. Cuando la condición deja de cumplirse, el programa sale del bucle y continúa su flujo normal. Su sintaxis es:

```
While (expresión booleana){
```

```
    Instrucciones a seguir
```

```
}
```

Una característica a tener en cuenta a la hora de decidirse a utilizar este tipo de bucles es que la condición es lo primero que se evalúa. A continuación, va a ver un ejemplo de uso del bucle WHILE:

```
Var i = 0;
```

```
While (i<9){
```

```
    Alert (i);
```

```
    i++;
```

```
}
```

Cuando i valga 8, el bucle evaluará la condición y parará, ya que i tiene que ser menor a 9.

Además, la estructura WHILE se puede combinar con DO para crear un bucle DO WHILE. Esta estructura es parecida a la anterior, la única diferencia es que la condición se comprueba al final. Este tipo de bucles son muy utilizados cuando crea un menú de opciones. Mientras el usuario no elija la opción de salir del programa, sigue trabajando con él. Por ello, necesita comprobar la opción, que el usuario ha elegido, al final del programa, ya que, al principio no sabe la opción que va a escoger. Vea la estructura DO WHILE en acción. Para ello, va a calcular el factorial de un número:

```
Var res = 1;
```

```
Var num = 10;
```

```
Do {
```

```
    Res = res * num;
```

```
    Num--;
```

```
} while (num>0);
```

3.4.3. Sentencia for

En el bucle WHILE y DO WHILE desconocía el número de iteraciones que se darían en el bucle, ya que continuar dentro del bucle o salir es algo que dependía de una condición. Pero, a veces, sabe exactamente el número de iteraciones que quiere hacer; no dependen de una condición. Para estos casos existe el bucle FOR cuya sintaxis es la siguiente:

For (inicio;test;incremento)

{

Sentencia;

}

Estos bucles son muy utilizados para recorrer listas de objetos. Si tiene una lista de 10 variables y quiere hacer algo sobre cada una de las variables, puede hacer un bucle FOR que hará las 10 iteraciones que necesita. En el siguiente ejemplo, va a reescribir el ejemplo anterior que hizo con el bucle WHILE, donde se mostraban los 10 primeros números:

For (i=0;i<10;i++)

{

Alert (i);

}

En programación, no hay bucles mejores ni peores, cada uno tiene sus características y se adaptan mejor o peor en cada caso. Es muy fácil convertir un bucle en otro y que el programa siga manteniendo intacto su comportamiento. Un buen programador sabrá en cada caso cuál de ellos utilizar dependiendo si se necesita comprobar una condición al inicio o al final, si sabe si el número de iteraciones que debe realizar es fijo o depende de una condición, etc.

3.4.4. Sentencia break

La sentencia BREAK permite manipular el comportamiento normal de los bucles. Esta sentencia permite terminar de forma automática un bucle, por lo que, si el programa llega a una instrucción de este tipo, saldrá inmediatamente del bucle y continuará ejecutando el resto de instrucciones que se encuentran fuera de este.



TOME NOTA

La utilidad de BREAK es terminar la ejecución del bucle cuando una variable toma un determinado valor o cuando se cumple alguna condición.

Aunque, es cierto que esta sentencia suele usarse para detener bucles, también, puede usarse para detener otros tipos de sentencias. La sentencia BREAK funcionaría de la siguiente manera:

```
For (i=0;i<10;i++){  
    Document.write(i);  
  
    If (i==50){  
  
        Break;  
  
    }  
  
}
```

Este bucle se recorrerá hasta que i tenga un valor de 50, en ese momento el bucle se parará automáticamente.

3.4.5. Sentencia continue

En contraste con la sentencia BREAK, CONTINUE no termina la ejecución del bucle por completo. En cambio, en un bucle WHILE, salta de regreso a la condición, mientras tanto, en un bucle FOR, salta a la expresión actualizada.

La sentencia CONTINUE puede incluir una etiqueta opcional que permite al programa saltar a la siguiente iteración del bucle etiquetado en vez del bucle actual. En este caso, la sentencia CONTINUE necesita estar anidada dentro de la sentencia etiquetada. Vea el siguiente ejemplo donde se muestra un bucle WHILE que tiene una sentencia CONTINUE que se ejecuta cuando el valor de i es 3. Así, no toma los valores 1,3,7 y 12:

```
I=0;  
  
N=0;  
  
While (i<5){  
  
    I++;
```

```
If(i==3)
```

```
    Continue;
```

```
    N+=i;
```

```
}
```

3.4.6. Sentencia switch

La estructura SWITCH permite tomar diversas opciones en función al valor que tome una determinada expresión. Este valor será comparado con una instancia CASE y, posteriormente, se ejecutarán las declaraciones asociadas a ese CASE. La sintaxis de esta estructura es la siguiente:

```
Switch (expresión){
```

```
Case valor1:
```

```
    Instrucciones;
```

```
    Break;
```

```
Case valor2:
```

```
    Instrucciones;
```

```
    Break;
```

```
Case valorn;
```

```
    Instrucciones;
```

```
    Break;
```



Puede anidar cualquier estructura de control dentro de otra, por ejemplo, un WHILE con un IF, un FOR con un WHILE, etc.

TOME NOTA

3.5. Funciones

Las funciones son uno de los bloques de construcción fundamentales en JavaScript. Una función en JavaScript es un conjunto de instrucciones que realiza una tarea o calcula un valor debiendo tomar alguna entrada y devolver una salida donde hay una relación obvia entre la entrada y la salida de datos.



TOME NOTA

Para usar una función, debe estar definida en algún lugar del ámbito desde el que se desea llamar.

3.5.1. Definición de funciones

Para definir una función debe escribir la palabra reservada function seguida de;

- El nombre de la función.
- Una lista de parámetros de la función, entre paréntesis y separados por comas.
- Las declaraciones de JavaScript que definen la función, encerrada entre llaves.

Por ejemplo, el siguiente código define una función simple llamada mostrar número:

```
Function mostrarNumero (num){  
    Document.write('Mostrar número ${num}');  
}
```

En esta función ve cómo al ejecutar la función pide el parámetro num que mostrará por pantalla.



TOME NOTA

Recuerde que las funciones son trozos de código que no se ejecutan hasta ser llamadas en el programa principal

RECURSO MULTIMEDIA



3.5.2. *Sentencia return*

La sentencia RETURN se utiliza para devolver datos desde la función al programa que ha llamado dicha función. El principal objetivo de las funciones es agrupar bajo un nombre una serie de instrucciones a seguir, que dan solución a un problema en particular. Cuando una instrucción de retorno se llama en una función, se detiene la ejecución de esta. Si se especifica un valor dado, este se devuelve a quien llama a la función. Si se omite la expresión, en su lugar, se devolverá undefined. Vea un ejemplo:

```
Function mostrarNumero(num) {  
    Return(`Mostrar número ${num}`);  
}
```

Esta función reescribe la función anterior y es más correcta que la anterior, ya que, la anterior además del parámetro devolvía undefined. Esta función devuelve una cadena de texto con el parámetro que le haya pasado.

3.5.3. *Propiedades de las funciones*

Cada vez que crea una función en JavaScript, por defecto, tiene asociadas diversas propiedades. Este apartado se centra en dos propiedades de la función arguments y calle, que van a permitir trabajar y gestionar cómodamente los parámetros opcionales y la identificación de la función que llama. A continuación, verá el funcionamiento de estas propiedades:

- **Arguments.** Arguments es un objeto accesible dentro de funciones que contiene los valores de los argumentos pasados a esa función. El objeto arguments es una variable local disponible en todas las funciones que no son funciones flecha. Se puede hacer referencia a los argumentos de una función dentro de esa función utilizando su objeto arguments. Tiene entradas para cada argumento con el que se llamó a la función, con el índice de la primera entrada en 0.

Por ejemplo, si a una función se le pasan 2 argumentos, se puede acceder a ellos de la siguiente manera:

Arguments[0]

Arguments[1]

Vea un ejemplo práctico. Tiene una función que puede sumar dos o más números:

```
Function suma (base){
```

```
    Base=Number(base);  
  
    For (var i = 1; i<arguments.length;i++){  
  
        Base += Number(arguments[i]);  
  
    }
```

```
    Return base;
```

En esta función, el bucle for recorrerá los parámetros que le pase a la función y los sumará. Aunque solo tenga un parámetro, puede pasar tantos como quiera. Por ejemplo:

```
Console.log (suma(2,3,2));
```

Dará como resultado 7, ya que, sumará los tres números.

- **Caller.** Esta propiedad se utiliza para mostrar el nombre de la función que llama, por lo tanto, devolverá una cadena de caracteres. Vea un ejemplo práctico de cómo funciona esta propiedad:

```
Function prueba(){
```

```
    If (prueba.caller == null){  
  
        Return ("La función ha sido llamada por sí misma");  
  
    }else {  
  
        Return ("Esta función ha sido llamada por" + prueba caller;  
  
    }  
  
}
```

Para probar esta función, debe crear otra función que llame a esta primera. Vea un ejemplo:

```
Function llamadora (){
```

```
    Return prueba();  
}  
  
Console.log(llamadora());
```

El resultado devolverá el else de la primera función, y por lo tanto, dirá que la función prueba () ha sido llamada por la función llamadora ().

3.5.4. Funciones predefinidas del lenguaje de guion

Por lo general, en todos los lenguajes de programación va a tener una serie de funciones predefinidas. En JavaScript, también, encuentra este tipo de funciones. A continuación, verá algunas de estas funciones con su respectiva sintaxis:

- **Eval(string).** Esta función predefinida recibe expresiones tipo string y las evalúa.

Vea un ejemplo:

```
Console.log(eval("2+2"));
```

El resultado esperado es 4.

- **ParseInt(string,base).** Esta función comprueba el primer argumento, que es un string, e intenta devolver un entero de la base especificada. Por ejemplo, una base 10 indica una conversión a un número decimal, una base 8 a un número octal, y así sucesivamente.

Vea un ejemplo:

```
ParseInt("F",16);
```

Esta función convierte la letra F a un valor hexadecimal.

En el caso de que el string no pueda pasarse a dato numérico, la función devolverá el valor NaN (Not a Number). Ejemplo:

```
parseInt ("hola"),8);
```

Devuelve NaN.

- **ParseFloat(string).** Esta función convierte su argumento, string, y devuelve un número de punto flotante, es decir, un número real. Si encuentra un carácter diferente al signo + o -, numerales de 0 al 9, un punto decimal o un exponente, devuelve el valor hasta ese punto e ignora ese carácter y todos los siguientes. Se permiten espacios anteriores y posteriores. Si el primer carácter no se puede convertir a número, esta función devolverá NaN.

Vea unos ejemplos donde todos devuelven el número 3,14;

```
parseFloat("3.14);  
parseFloat("314e-2");  
parseFloat("0.0314E+2");  
  
var cadena = "3.14"; parseFloat(cadena);  
  
parseFloat("3.14 caracteres");
```

- **isNaN(número).** Esta función intenta convertir el parámetro que le haya pasado a un número. Si el parámetro no se puede convertir, devuelve true. En caso contrario, devuelve false.

Vea unos ejemplos:

```
isNaN(NaN) //devuelve true, ya que, el parámetro no es un número.  
  
isNaN("string") //devuelve true.  
  
isNaN("12") // devuelve false.
```

- **isFinite(valor).** Esta función se utiliza para determinar si un número es un número finito. Para ello, la función examina el número de su argumento. Si el argumento es NaN, infinito positivo o infinito negativo, este método devuelve false, de otro modo devuelve true. Vea algunos ejemplos:

```
isfinite(NaN); //falso  
  
isFinite (Infinity) //falso  
  
isFinite (-Infinity) // falso  
  
isFinite(0)//verdadero  
  
isFinite(2e64)//verdadero
```

isfinite("0"); //verdadero. Aunque, si se hubiera utilizado Number.isFinite("0") habría dado falso.

- **Number(valor).** Los principales usos de esta función son convertir un string y otro valor a uno de tipo numérico. Si el argumento no puede ser convertido a un número, devuelve NaN. Vea los siguientes ejemplos:

Number("123") // retorna el número 123

Number("123") === 123 // retorna true

Number("Bienvenidos") //retorna NaN

- **String(valor).** Esta función convierte una cadena string en un objeto string. Vea un ejemplo:

```
Var s1="2+2";           //crea un string primitivo
```

```
Var s2 = new String("2+2"); //crea un objeto string
```

Por lo que si ahora aplica la función eval(), que ha visto anteriormente, obtendrá:

```
Console.log(eval(s1)) //devuelve el número 4
```

```
Console.log(eval(s2)); //devuelve la cadena "2+2", ya que, es un objeto.
```

3.5.5. Creación de funciones

Cuando crea funciones debe tener en cuenta varios aspectos, por ejemplo, si la función recibe o no parámetros, o si la función debe devolver algún dato.

Imagine que quiere crear una función que muestre un mensaje al entrar en una web. A esta función no se le deberá pasar ningún dato por parámetro, ni tampoco, devolverá ningún dato. Vea cómo quedaría la función:

```
Function bienvenida () {  
    Document.write ("Hola, gracias por visitar mi web");  
}
```

Los paréntesis de esta función se dejan vacíos, ya que, no hace falta pasarle ningún dato a la función para que pueda ejecutarse. Ahora, imagine que quiere personalizar este mensaje de bienvenida y quiere que en él aparezca el nombre del usuario que visita la página. En este caso, sí tendrá que pasarle información por parámetro a la función para que esta pueda ejecutarse con éxito. Vea cómo quedaría el código:

```
Function bienvenida (nombre_usuario){  
    Document.write(`Hola ${nombre_usuario}, gracias por visitar mi web);  
}
```

Si quisiera, además, incluir el apellido del usuario, tan solo, debería crear otro parámetro que pondría en el paréntesis y en la frase.

Para crear funciones que retornen datos, debe usar la instrucción return, que ya aprendió anteriormente, y el dato que quiere que devuelva la función. Vea una función con return como ejemplo:

```
Function nombreUsuario(){  
    Nombre=prompt("¿Cuál es su nombre?");  
    Return nombre;  
}
```

3.5.6. Particularidades de las funciones en el lenguaje de guion

Una de las particularidades de las funciones en JavaScript es que el paso de parámetros a las funciones se realiza por valor. Por lo que, si una función modifica el contenido de uno de sus argumentos pasado por parámetro, la modificación solo se contemplará dentro de la función, por lo tanto, no tendrá efecto en el programa principal.

En JavaScript puede crear funciones anidadas. Para anidar una función dentro de otra función, tendrá que tener en cuenta que la función anidada (inner) es privada a la función que la contiene (outer). También con la forma de cierre, aclosure.

Un cierre es una expresión que puede tener varias variables libres junto con un entorno que enlaza esas variables, es decir, que cierra la expresión.

Dado que una función anidada es un cierre, esto significa que una función anidada puede "heredar" los argumentos y las variables de su función contenedora (outer). En otras palabras, la función interna contiene el ámbito de la función externa.

La conclusión que se saca de esto es que:

- A la función interna se puede acceder solo a partir de sentencias en la función externa.
- La función interna forma un cierre, es decir, la función interna puede utilizar los argumentos y las variables de la función externa, mientras que la función externa no puede utilizar los argumentos, ni las variables de la función interna.

Vea un ejemplo donde se muestran funciones anidadas:

```
Function nCuadrado(a,b){
```

```
    Function cuadrado(x){
```

```
    Return x*x;  
}  
  
Return cuadrado (a) + cuadrado (b);  
}
```

El resultado de pasarle los valores (2,3) será 13.

3.6. Instrucciones de entrada / salida

Las instrucciones de entrada y salida se las denomina como instrucciones E/S. Estas instrucciones van a servir para obtener datos del exterior y enviar datos al exterior. Si estas instrucciones no existieran, los *scripts* carecerían totalmente de utilidad, ya que no podría tomar datos, por ejemplo, del teclado y procesarlos para mostrar, después, el resultado por pantalla.

Es decir, el usuario no podría interactuar con el programa.

3.6.1. Descripción y funcionamiento de las instrucciones de entrada y salida

Las instrucciones de entrada y salida de datos, además de ser conocidas como instrucciones de E/S, también puede denominarlas como instrucciones I/O. Estas instrucciones van a servir para recoger datos del exterior, por ejemplo, de un teclado o de un ratón, y poder usar esos datos en su programa. A esto se denomina entrada de datos. Por otra parte, tiene la salida de datos, que puede ser, por ejemplo, mostrar datos por pantalla.

JavaScript tiene dos funciones fundamentales para la entrada y salida de datos, son las siguientes:

- Prompt (entrada de información a través de una caja de texto).
- Document.write (salida de información por documento).
- Alert (Salida de datos por pantalla)
- Console.log (Salida e datos por la consola)

LECTURA DE TECLADO DE DATOS

En programación, leer datos significa capturar datos, normalmente introducidos por teclado y almacenarlos en una variable. Para leer un dato por teclado en JavaScript se usa la función prompt, cada vez que se necesite ingresar un dato con esta función, aparecerá una ventana donde se escribirá el valor que quiere ingresar.

Además, JavaScript, como ya sabe, es un lenguaje que se puede combinar con HTML, por lo que, también, podrá recoger datos que se escriban dentro de las cajas de texto de HTML. De esta manera es como crearía un formulario en una página web.

ALMACENAMIENTO EN VARIABLES

Las variables, como ha visto anteriormente, son espacios de memoria reservados para almacenar datos. Todos los datos que capture por teclado, Tendrá que almacenarlos en las variables para luego poder procesarlos. Cuando el dato es procesado, se devolverá a la variable para almacenarlo y mostrarlo, posteriormente, por pantalla.

IMPRESIÓN EN PANTALLA DEL RESULTADO

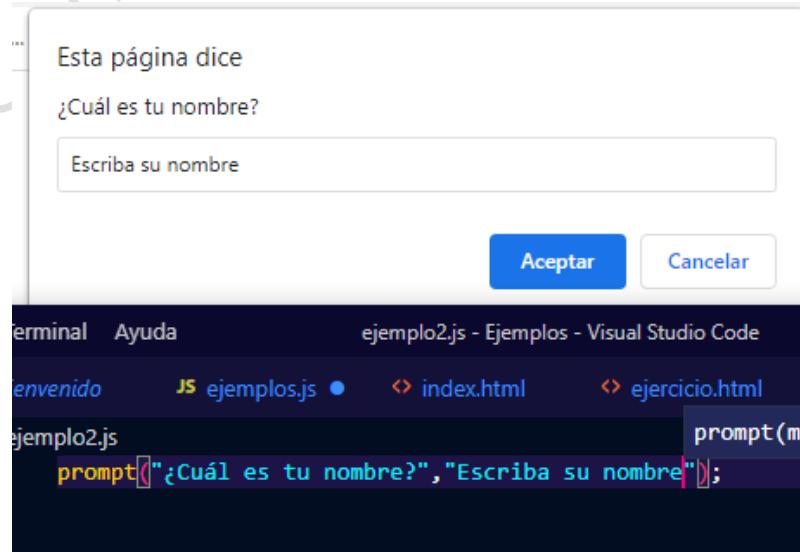
La mayoría de las veces, los programas hacen operaciones que son transparentes al usuario. En cambio, si necesita mostrar datos por pantalla, debe usar la sentencia document.write(). Como ya sabe, para crear un formato lo más dinámico posible, puede mezclar código JavaScript con HTML.



3.6.2. Sentencia prompt

La sentencia prompt muestra una caja de texto en la ventana del navegador, donde el usuario podrá introducir texto. Lo que hace esta sentencia es leer un valor que haya sido introducido por teclado en su caja de texto. Este texto, se almacena en la variable que le haya asignado, para posteriormente operar con él. La sintaxis de la sentencia prompt es la siguiente: prompt("texto descriptivo","valor por defecto");

El texto descriptivo es lo que informa al usuario de qué información es la que se le está pidiendo. Por otro lado, la parte de valor por defecto es opcional. El valor por defecto es un texto que viene escrito en la caja de texto para “aclararle” al usuario la información que debe escribir en ella. Vea un ejemplo:



3.6.3. Sentencia document.write

La sentencia document.write() se utiliza para poder mostrar información a través de la pantalla. La sintaxis de esta sentencia es la siguiente:

```
document.write("texto que se muestra")
```

Si lo que quiere es mostrar un texto junto con los datos que tenga una variable, Tendrá que utilizar las templates string. Su sintaxis es la siguiente:

```
Document.write(`Bienvenido, ${nombreDeLaVariable}`);
```



Recuerde que cuando usa templates string debe utilizar las tildes invertidas; si no lo usa, no obtendrá un error en la ejecución del programa.

Vea el siguiente ejemplo:

```
Bienvenido, Pablo
Terminal Ayuda ejemplo2.js - Ejemplos - Visual Studio Code
Bienvenido      JS ejemplos.js ●      index.html      ejercicio
JS ejemplo2.js > ...
1  var nombre= "Pablo";
2
3  document.write(`Bienvenido, ${nombre}`);
```

4. DESARROLLO DE SCRIPTS

A la hora de desarrollar programas de *script*, los programadores cometen errores a la hora de escribir el código. Estos errores pueden arreglarse con las herramientas adecuadas para ello. Pero, sobre todo, la primera cosa básica para evitar errores en sus programas es ser ordenado y tener el código bien estructurado, ya que, si posteriormente ve un error de código, será mucho más fácil detectarlo en un código limpio y ordenado. En informática, a los errores de un programa se les denomina **bugs**. Los bugs pueden ser errores de programación o problemas en otros sistemas con los que su programa interactúa, en el caso de JavaScript podría ser con los diferentes navegadores. La palabra depurar o debuggear significan lo mismo, y se utiliza para decir que está reparando los errores que pudiera haber en el programa. Estas palabras provienen de debugger que es una herramienta que permite la ejecución controlada de un programa para seguir cada instrucción ejecutada y localizar, así, los bugs o errores. A este proceso se le llama depuración.

Además, en la detección de errores en el código, la experiencia que tenga con el lenguaje de programación y con la programación en sí, juega un papel muy importante. Pues no hay una guía exacta y específica a seguir para resolver los diferentes errores que pueda encontrar en el programa. Sobre todo, en JavaScript, que al ser un lenguaje de programación poco tipado, va a ser más difícil detectar errores, ya que, va a ofrecer muy poca ayuda para encontrar los errores.

4.1. Herramientas de desarrollo, utilización

Los *scripts* son archivos básicos de texto con una serie de instrucciones que se van ejecutando de manera secuencial, una tras otra. Para desarrollar un *script*, tan solo, se necesita un editor básico de texto, es decir, podría desarrollar un *script* en un documento de texto enriquecido, guardándolo como .js. Además, Windows proporciona un editor de texto para este fin llamado Notepad. Aparte, puede encontrar otros tipos de editores de texto que aportarán diferentes herramientas como, por ejemplo, la herramienta de código más utilizada por los desarrolladores es Visual Studio Code que, además de ser una herramienta multiplataforma, aporta diversos *plug-ins* para hacer la tarea de programar más sencilla.

VSCode indica los errores de sintaxis y se puede sincronizar con GitHub. Además, la herramienta es de código abierto. Al igual que esta herramienta, puede encontrarse con herramientas como Visual Studio, Sublime, Atom, Vim, WebStorm, entre otras. Cabe destacar que los editores como Sublime y WebStorm son de pago. Cada una ofrece diferentes herramientas y ninguna es mejor que otra, simplemente, es el programador quien elige una u otra dependiendo de sus requerimientos a la hora de programar. Lo importante es probar diferentes herramientas de desarrollo y quedarse con la que más se adapte a su forma de programar.



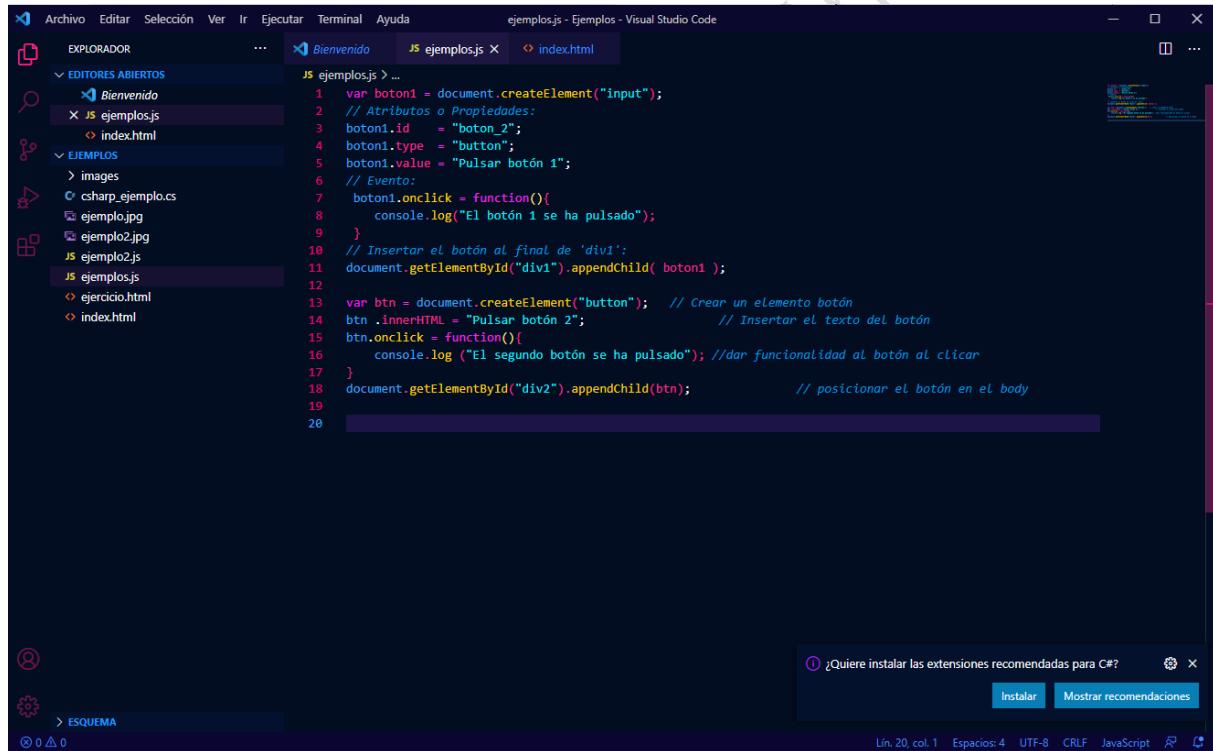
Más Info



4.1.1. Crear scripts con herramientas de texto

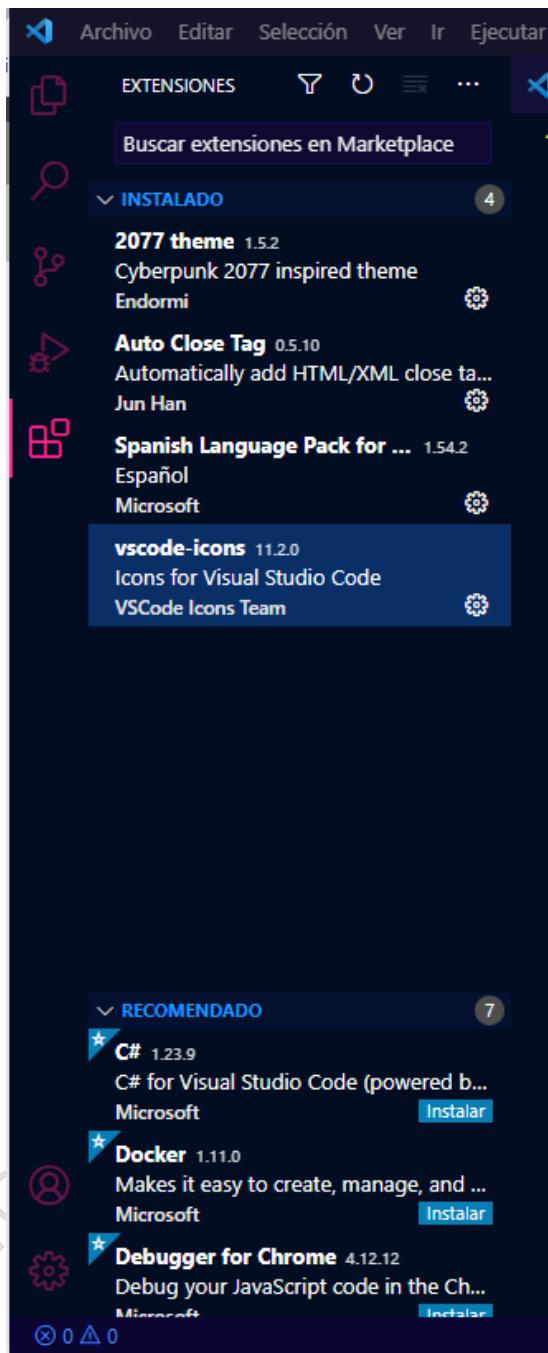
Hace algún tiempo, disponer de herramientas de texto para programar era algo impensable. Lo que se solía hacer es programar en un editor de texto donde se escribía el código HTML y el código de *script* en el mismo archivo y, posteriormente, se guardaba con extensión .html. Esta manera de crear código, como ya podrá haber detectado, no es muy ventajosa, ni práctica, ya que, si cometiera cualquier error a la hora de escribir su código, no habría forma de detectarlo de manera automática, ya que, esos editores de texto no tienen las herramientas necesarias para ese cometido.

Sin embargo, hoy en día hay cientos de aplicaciones muy prácticas y útiles para escribir código de programación. Incluso, algunas de ellas se adaptan al lenguaje de programación que esté escribiendo. Algunas de estas aplicaciones son de pago, pero hay muchísimas gratuitas y de código abierto. Por ejemplo, la herramienta que más se usa en programación es [VSCode](#) que es una herramienta de desarrollo gratuita y multiplataforma. Además, cuenta con un montón de *plug-ins* con los que puede amoldar la aplicación a sus necesidades. Puede ver la herramienta en la siguiente imagen:



Como puede observar este editor resalta las líneas de texto e, incluso, pone un color para cada elemento del código. Por ejemplo, las funciones y las variables están en rosa, los objetos en azul y sus atributos en amarillo.

Como ha visto anteriormente, en este tipo de herramientas puede instalar plug-ins para que la herramienta se amolde a sus necesidades. Para ello, tan solo, debe hacer clic en el último ícono de la barra lateral izquierda y puede ver los plugins que tiene instalados o buscar plug-ins que quiera instalar. Vea la siguiente imagen:



Para crear archivos con diferentes lenguajes de programación, tan solo, debe crear un nuevo archivo y guardarla con la extensión conveniente a ese lenguaje. Por ejemplo, si está programando en JavaScript, utilice la extensión .js.

Además, a medida que va escribiendo el código la propia aplicación va diciendo los errores de sintaxis que está cometiendo y en qué línea están ubicados. Véalo en la siguiente imagen:

The screenshot shows the Visual Studio Code (VS Code) interface. At the top, there are tabs for 'ejemplos.js', 'ejemplo2.js' (which is currently active), and 'index.html'. Below the tabs is a code editor window containing the following JavaScript code:

```
JS ejemplo2.js
1  function (){
2
3  }
4
5  documento.write()
6
7  }
8
9  var nombre = "Carlos";
```

At the bottom of the interface, the 'PROBLEMAS' (Problems) panel is open, showing three errors for the file 'ejemplo2.js':

- Se esperaba un identificador. ts(1003) [1, 10]
- Se esperaba ','. ts(1005) [5, 18]
- Se esperaba ','. ts(1005) [9, 6]

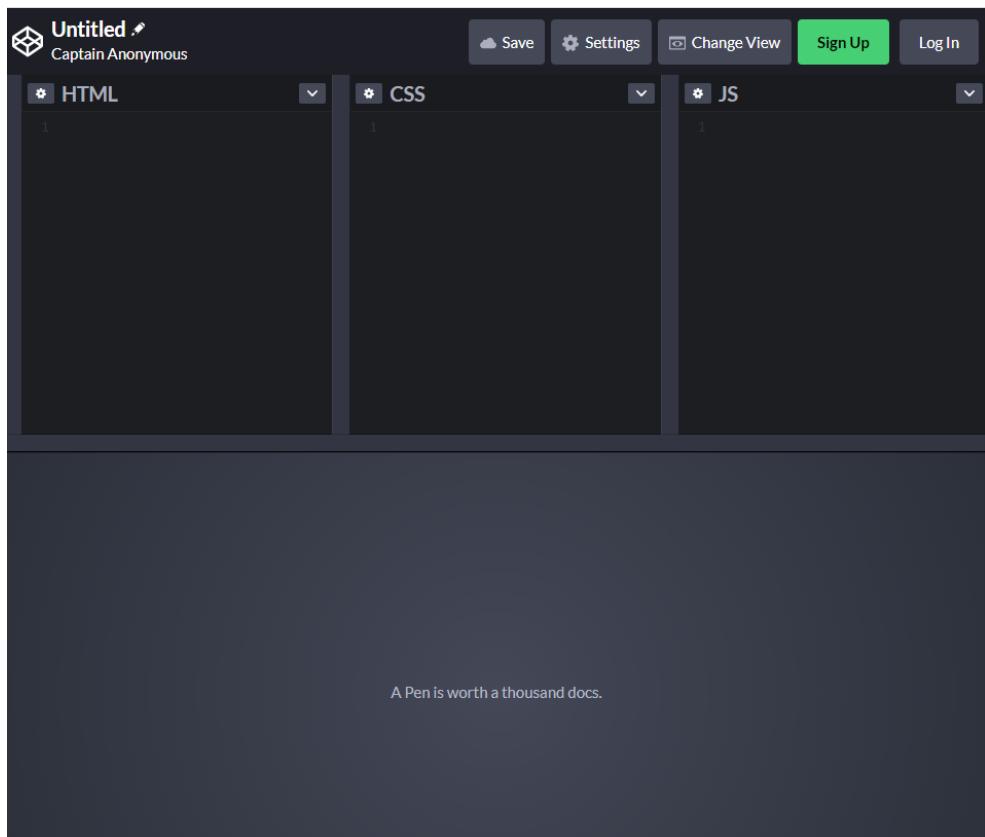
4.1.2. Crear scripts con aplicaciones web

Se entiende por aplicación web un conjunto de herramientas disponibles para el usuario y que estas, a su vez, están ofrecidas por un determinado servidor web. Las aplicaciones web son totalmente independientes del sistema operativo, por lo que, puede usarlas desde cualquier dispositivo que esté conectado a Internet. En la arquitectura de una aplicación web siempre va a tener tres componentes fundamentales:

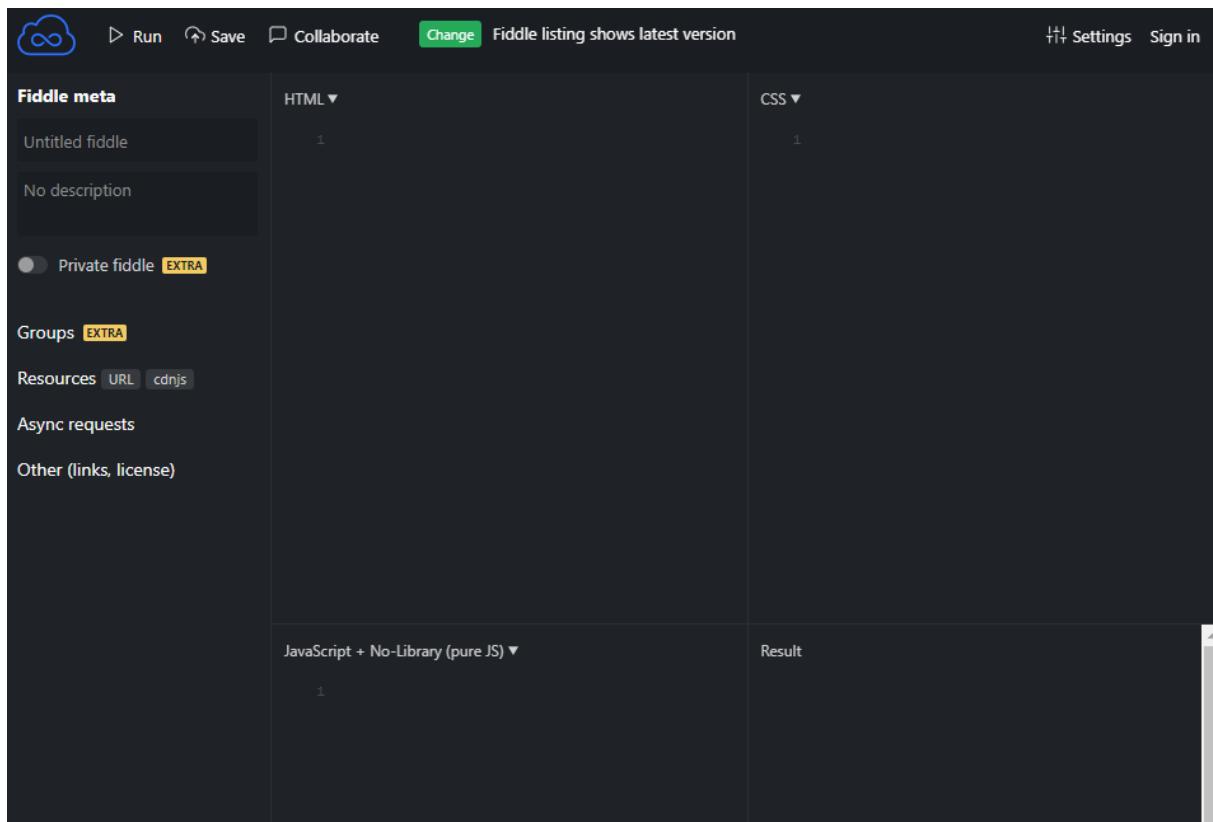
- **Servidor web.** Aloja la información en forma de archivos y la distribuye a los clientes que la solicitan.
- **Conexión de red.** Es el medio por el que se distribuyen los datos en la red.
- **Clientes web.** Es un software que accede al servidor y recupera los datos de él. Por ejemplo, los navegadores web son clientes.

Hoy por hoy, existen aplicaciones web que facilitan el desarrollo de su *script* y que, además, guardan la información en la nube, por lo que, puede editar su *script* desde cualquier equipo y en cualquier lugar. Este es el caso de Google Apps Script que permite vincular aplicaciones como Gmail, Google Forms, Sheets, etc. Y permite crear script para expandir las funcionalidades de estas aplicaciones.

Por otro lado, también tiene los editores de código online como, **Codepen** que es un editor online para HTML, CSS y JavaScript. Se carga rápidamente en sus navegadores y, también, se obtiene una vista previa del código escrito en el. Este editor de texto online es uno de los mejores editores online que hay por la red y, además, cuenta con un diseño minimalista. Véalo en la siguiente imagen:



Por otro lado, también tiene el editor **JS Fiddle** que está enfocado a diseñadores web. Este editor permite escribir código en lenguaje CoffeeScript, pero, también se puede trabajar con JavaScript puro o incluir librerías como Jquery, AngularJS, entre otros. Lo mejor de JS Fiddle es la opción de colaboración, la cual permite crear un link para enviar su código a otros usuarios para que puedan ayudar a desarrollar el código en tiempo real. Vea una imagen del editor JS Fiddle:



Por último, va a ver el editor online **Code AnyWhere**. Este editor es uno de los editores más populares entre los desarrolladores, ya que, este editor cuenta con un cliente FTP integrado, así como con el soporte para desarrollar en lenguaje HTML, PHP, JavaScript, CSS y XML. Gracias a los editores online puede hacer uso de ellos en cualquier ordenador que no disponga de programas necesarios para el desarrollo web. A diferencia de los dos editores anteriores, Code AnyWhere es un editor de pago, aunque, dejan usarlo durante un periodo de prueba de 7 días.

4.1.3. Recursos en web para la creación de scripts

En este apartado va a ver, entre otras cosas, las APIs. El término API es una abreviatura de Application Programming Interfaces, que en español significa interfaz de programación de aplicaciones. Se trata de un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de software a través de un conjunto de reglas.

Así pues, puede hablar de una API como una especificación formal que establece cómo un módulo de un software se comunica e interactúa con otro para cumplir una o muchas funciones. En JavaScript tiene muchas APIs del lado del cliente, estas no son parte del lenguaje en sí, sino que están construidas sobre el núcleo de este lenguaje de programación. Por lo general, las APIs se dividen en dos categorías:

- **APIs de navegador.** Están integradas en el navegador web y pueden exponer datos del navegador y del entorno informático circundante y hacer cosas complejas y útiles con él. Por ejemplo, la API de geolocalización proporciona algunas construcciones simples de JavaScript para obtener datos de ubicación con los que, por ejemplo, trazar tu ubicación en un mapa de Google. Realmente, el navegador está haciendo uso de códigos de bajo nivel complejos en segundo plano para comunicarse con el hardware GPS del dispositivo, recuperar datos de posición y devolverlos al entorno del navegador para su uso en su código.
- **APIs de terceros.** No están incluidas por defecto en el navegador y, por lo general, es necesario obtener el código e información desde algún lugar de la web. Por ejemplo, la API de Twitter permite hacer cosas como mostrar tus últimos tweets en un sitio web. Proporciona un conjunto especial de construcciones que se pueden usar para consultar el servicio de Twitter y devolver información específica.

Puede encontrar APIs en los siguientes sitios web [Google API](#), [GitHub](#) y [ProgrammableWeb](#). Aunque, siempre puede investigar más sitios donde encontrar APIs, por ejemplo, la API de Facebook, de YouTube, entre otras.

Además, aparte de las APIs como recursos web puede encontrar diferentes librerías para enriquecer su web. Las librerías son archivos con instrucciones para agregarle diversas funcionalidades y efectos a las páginas de Internet. Estos archivos son proporcionados por diferentes servicios en la red y se pueden descargar libremente o enlazarlos a las páginas.

Las librerías JavaScript más populares son:

- **JQuery.** Es la más popular de las librerías JavaScript de Internet. Es un archivo que contiene varias instrucciones que permiten que el navegador ejecute muchas funcionalidades adicionales. JQuery es muy empleada para hacer cambios de forma dinámica en el DOM (estructuras de las páginas) sin tener que recargarlas y agregarles gran cantidad de efectos y animaciones.
- **Bootstrap.** Es un framework web libre y de código abierto, muy popular usado para crear sitios web. Bootstrap ofrece plantillas HTML, hojas de estilo CSS, fuentes y una librería para añadir funcionalidades adicionales a las páginas, por ejemplo, botones, menús, barras de navegación, paneles, visores de imágenes, entre otros. Este framework se apoya en JQuery, por lo que es necesario cargar esta librería previamente. Para instalar Bootstrap en sus archivos puede hacerlo de varias maneras, una de ellas es descargar Bootstrap de su página oficial e instalarlo en su equipo. Pero, va a aprender una manera más sencilla. En la página oficial de **Bootstrap** encuentra una opción para instalar Bootstrap a partir de un CDN. Para ello, en su archivo HTML, debe insertar la siguiente línea de código en el <head> antes de cualquier hoja de estilo que tenga creada:

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0QLoh+rP48ckxlpbzKgwra6" crossorigin="anonymous">
```

Esta línea de código añade el archivo CSS que necesita Boostrap para funcionar.

A continuación, va a insertar una línea de código que añade JavaScript y Popper a su archivo. Esta línea de código debe de insertarse antes de la etiqueta </body>:

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js" integrity="sha384-JEW9xMcG8R+pH31jmWH6WWPOintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf" crossorigin="anonymous"></script>
```

Hecho esto, ya puede disponer de las ventajas de usar Bootstrap. Por ejemplo, dentro de la propia web oficial de Bootstrap, puede encontrar complementos, los cuales, con copiar el código que dan puede añadir a su web. Por ejemplo, vea una imagen de cómo puede añadir botones de colores a su web:

Examples

Bootstrap includes several predefined button styles, each serving its own semantic purpose, with a few extras thrown in for more control.

The screenshot shows a collection of Bootstrap buttons arranged in two rows. The top row contains seven buttons labeled 'Primary', 'Secondary', 'Success', 'Danger', 'Warning', 'Info', and 'Light'. The bottom row contains two buttons labeled 'Dark' and 'Link'. Below the buttons is a block of HTML code that generates them. A 'Copy' button is located in the top right corner of this code block.

```
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-secondary">Secondary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-light">Light</button>
<button type="button" class="btn btn-dark">Dark</button>

<button type="button" class="btn btn-link">Link</button>
```

- **Modernizr.** Es una librería que detecta funcionalidades HTML5 y CSS3 en los navegadores web. Esta librería se emplea en muchas aplicaciones web como una herramienta imprescindible para que puedan funcionar adecuadamente. Esto es, tan solo, un método de detección y no agrega ninguna funcionalidad adicional.

Antes de decidirse a insertar cualquier script externo en las páginas de un sitio web, debe de conocer que siempre existe el riesgo de que paralice por completo o bloquee la carga de la página web donde se encuentra. Si dicho *script* no está disponible o existe retraso en su entrega por algún motivo como lentitud en la red o no disponibilidad del servidor donde se aloja, su página puede bloquearse por completo. Para evitarlo tiene las siguientes alternativas:

- Enlazar los *scripts* o librerías desde un servicio rápido y estable que emplee un CDN o lo que es lo mismo, Content Delivery Network, que en español significa Red de Entrega de Contenido. Un CDN es un servicio que posee varios servidores o centros de datos repartidos geográficamente.
- Cargar el *script* de forma asíncrona. El modo asíncrono, opción que admiten todos los navegadores modernos, permite que el archivo o librería sea descargado de forma paralela a otros recursos, por lo que, si no está disponible o existen retrasos, no afectará a la carga de la página. Para hacer esto, es necesario agregar la propiedad `async` a la etiqueta de los scripts de la siguiente forma:
`<script async src="dirección-url-script.js"></script>`
- Alojar el *script* en su servidor en los casos en que lo permita. Esta puede ser la mejor opción, ya que, permite un mayor rendimiento. El contra de esta opción es que no está disponible para muchos casos en los que se usan servicios de alojamiento gratis.

4.2. Depuración de errores: errores de sintaxis y de ejecución

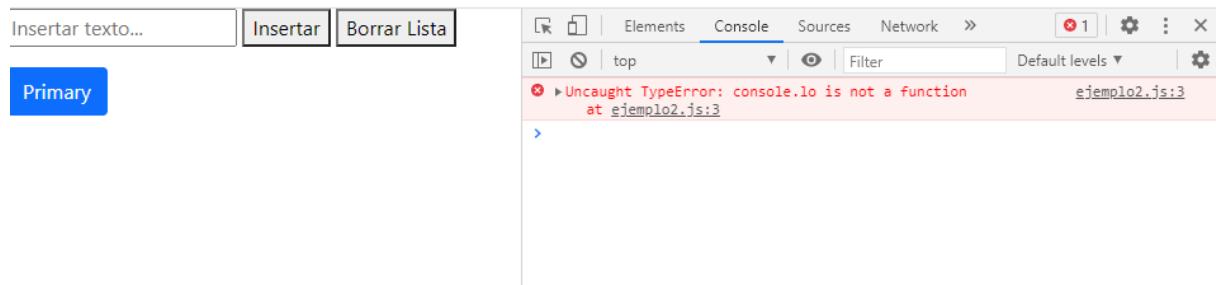
JavaScript resulta difícil de depurar, ya que se trata de un lenguaje interpretado y embebido. No existe un único método para depurar JavaScript y, tampoco, hay un método preferido por los programadores. Por ello, va a ver las diferentes alternativas para depurar JavaScript. Primeramente, va a agrupar las diferentes alternativas en tres grandes grupos que va a citar superficialmente para luego centrarse en una de estas alternativas:

Herramientas de validación. Existen validadores de código tanto online, como en aplicación de escritorio. Estas herramientas, en líneas generales, reciben un código JavaScript como entrada y devuelven una lista de los errores o problemas detectados en el código. Estos problemas pueden ser de sintaxis o de otro tipo. Como herramientas online puede citar [JSLint](#) y [JSHint](#). Entre las herramientas para instalar en el ordenador, tiene [Google Closure Linter](#) y [JavaScript Lint](#).

IDEs para desarrollo web. Un IDE, en inglés *Integrated Development Environment*, es un entorno de desarrollo integrado. En general se trata de entornos de desarrollo que proveen algunas herramientas o extensiones capaces de detectar errores en el código o, al menos, facilitar ayuda que debería dar lugar a que existan menos errores en el código. Son, en general, herramientas que proveen muchas funcionalidades y, por tanto, más pesadas que un simple editor de texto. Puede citar dentro de este grupo Eclipse, JSDT, VSCode, Microsoft Visual Studio, entre otros.

Herramientas nativas. Estas herramientas se integran en los navegadores web y permiten el análisis del código y la depuración de errores. Mozilla Firefox es uno de los navegadores que ha sido tradicionalmente preferido por los programadores por las herramientas de análisis y depuración que provee y por atenerse a los estándares. De hecho, Mozilla Firefox tiene una edición del propio navegador que está específicamente creada para desarrolladores, esta versión es Mozilla Firefox Developer. De cara a la depuración de código JavaScript dispone de herramientas nativas como la consola y de diferentes *plug-ins* para desarrolladores entre los que destaca Firebug. El navegador Chrome de Google provee herramientas nativas en lo que se denomina Chrome Developer Tools. Para abrir la consola en los distintos navegadores, tan solo, dar clic derecho en la página que quiera depurar y clicar la opción inspeccionar. Saldrá una ventana donde puede elegir diferentes opciones,

una de ellas será la opción console. Ahí es donde verá los errores y los mensajes de error que salta en su código. Vea un ejemplo:



Más Info

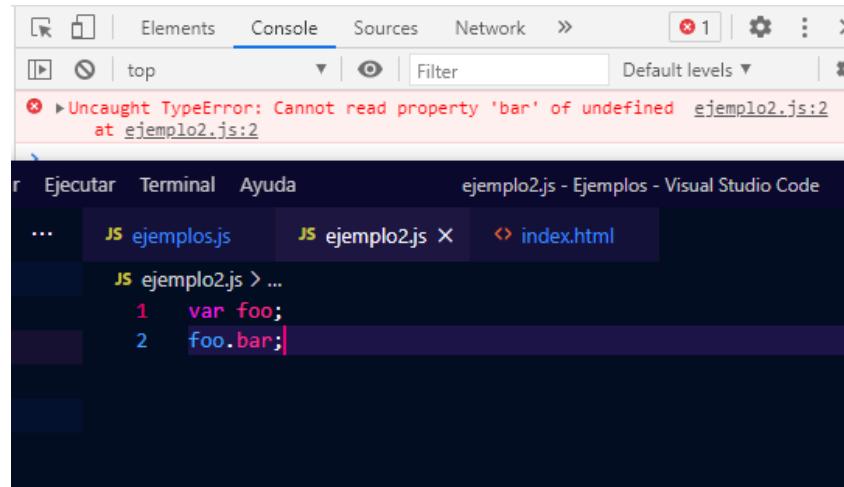
RECURSO MULTIMEDIA



4.2.1. Definición de los tipos de errores

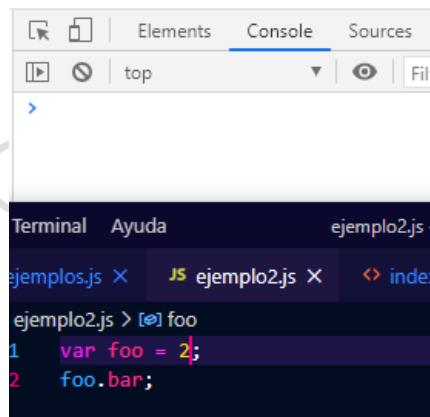
En este apartado va a ver los 10 errores que más se cometan al programar en JavaScript. Va a analizar cada uno de estos errores para saber bien porqué se producen y cómo solventarlos.

- **Uncaught TypeError: cannot read property.** Seguramente, mientras hacía algún ejercicio del curso, o probaba algún script, ya se haya encontrado con este tipo de error varias veces. Este error ocurre en Chrome cuando lee una propiedad o llama a un método de un objetivo indefinido. Vea un ejemplo:



En este caso, ha escrito un código de ejemplo para que salte ese tipo de error. Como ve, tiene una variable sin inicializar, es por ello que, el navegador no puede alcanzar el valor de dicha variable y, por lo tanto, no puede alcanzar su propiedad. Esto puede ocurrir por varias razones, pero lo común es que sea por culpa de la inicialización correcta del estado al representar los componentes de la interfaz de usuario.

Este tipo de error es fácil de solventar, tan solo, se debe inicializar el estado de los elementos con algún valor. Por ejemplo, si a la variable del ejemplo anterior le da un valor, el error desaparecerá. Vea el ejemplo:

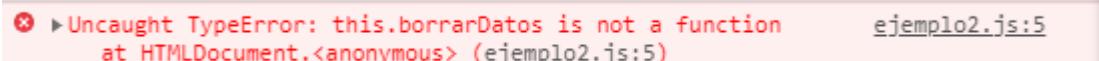


- **TypeError: 'undefined' is not an object (evaluating).** Este es un error que ocurre en Safari cuando se lee una propiedad o se llama a un método en un objeto indefinido. Este es el mismo error que el anterior, pero en vez de ocurrir en Chrome, ocurre en Safari.
- **TypeError: null is not an object (evaluating).** Este error, también, ocurre en Safari cuando se lee una propiedad o se llama a un método de un objeto nulo. Recuerde que, en JavaScript, null y undefined no es lo mismo, por lo que se ven dos mensajes de error diferentes. Undefined suele ser una variable que se ha asignado, mientras que null significa que el valor está en blanco.

- **Unknown: Script error.** Este error avisa cuando de origina un error de un archivo JavaScript servido desde un origen diferente, ya sea, desde un dominio, puerto o protocolo diferente. Es un error que no sabe de dónde viene ya que. No se sabe cuál es el error, ni desde qué parte del código se está originando. Por ejemplo, si su código está en un CDN, cualquier error no capturado se informará simplemente como *Script error*, en lugar de contener información útil. Esto es una medida de seguridad del navegador destinada a evitar el paso de datos a través de dominios que de otro modo no podrían comunicarse. Aunque hay métodos para acceder a este tipo de errores no va a verlos debido a su complejidad.
- **TypeError: ‘undefined’ is not a function.** Este error ocurre en Chrome cuando se llama a una función no definida. Como las técnicas de codificación de JavaScript cada vez se han vuelto más sofisticadas, ha habido un aumento correspondiente a la proliferación de ámbitos de autorreferencia dentro de las devoluciones de callbacks y closures, que son una fuente bastante común de esta confusión. Vea el siguiente ejemplo de código:

```
Function borrarDatos(){  
  
    Alert("Datos borrados");  
  
}  
  
Document.addEventListener ("click",function(){  
  
    This.borrarDatos();  
  
});
```

Si ejecuta este código y hace clic en la página, se producirá el siguiente error:

 **Uncaught TypeError: this.borrarDatos is not a function** [ejemplo2.js:5](#)
at HTMLDocument.<anonymous> ([ejemplo2.js:5](#))

Esto se debe a que la función anónima que se está ejecutando se encuentra en el contexto del documento, mientras que `borrarDatos()` se define en el objeto Window. Para solventar este problema puede tomar una solución que se utilizaba antes en los navegadores antiguos, simplemente, guarde la referencia `this` en una variable que luego puede ser heredada. Vea el ejemplo:

```
Var self = this;
```

Por lo tanto quedaría algo como:

```
Document.addEventListener ("click", function(){  
  
    Self.borrarDatos();  
  
});
```

});

Otra alternativa solo para navegadores recientes es que se puede usar el método bind() para pasar la referencia adecuada:

```
Document.addEventListener("click",this.borrarDatos.bind(this));
```

- **Uncaught RangeError: Maximum call stack.** Este es otro error que solo ocurre en Chrome en dos ocasiones. La primera es cuando se llama a una función recursiva que no tiene fin. La segunda, cuando se pasa un valor a una función que está fuera del rango establecido. Muchas funciones solo aceptan un rango específico de números para sus valores de entrada. Por ejemplo, Number.toExponential() tiene un rango del 0 al 20, en el caso de que ponga otro número que no esté en ese rango, saltará este error.
- **TypeError: Cannot read property 'length'.** Este error ocurre en Chrome debido a la lectura de la propiedad length para una variable indefinida. Normalmente, length está definido en un array, pero es posible que se encuentre este error si el array no se inicializa o si el nombre de la variable está oculto en otro contexto. Véalo con un ejemplo:

```
Var testArray =[“test”];  
  
Function testFunction(testArray){  
  
    For (var i=0;i<testArray.length;i++){  
  
        Console.log(testArray[i]);  
  
    }  
  
}  
  
testFunction();
```

Cuando se declara una función con parámetros, estos parámetros son locales. Esto significa que incluso si tiene variables con nombre testArray, los parámetros con los mismos nombres dentro de una función se tratarán como locales.

Hay dos maneras de resolver este problema. La primera es eliminar todos los parámetros en la declaración de la función. Vea un ejemplo:

```
Var testArray[“test”];  
  
Function testFunction(){  
  
    For(var i=0;i<testArray.length.i++){  
  
        Console.log(testArray[i]);  
  
    }  
  
}
```

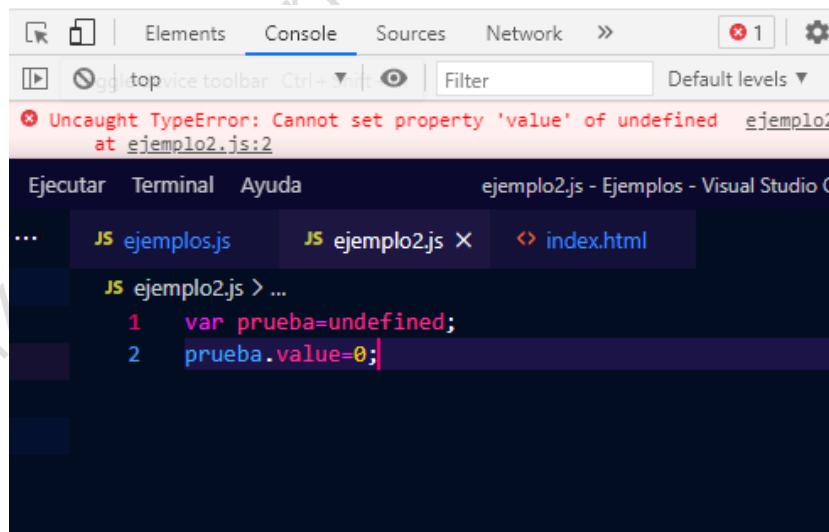
```
    }  
  
}  
  
testFunction();
```

La segunda es llamar a la función pasando el array que declara. Vea el código de ejemplo:

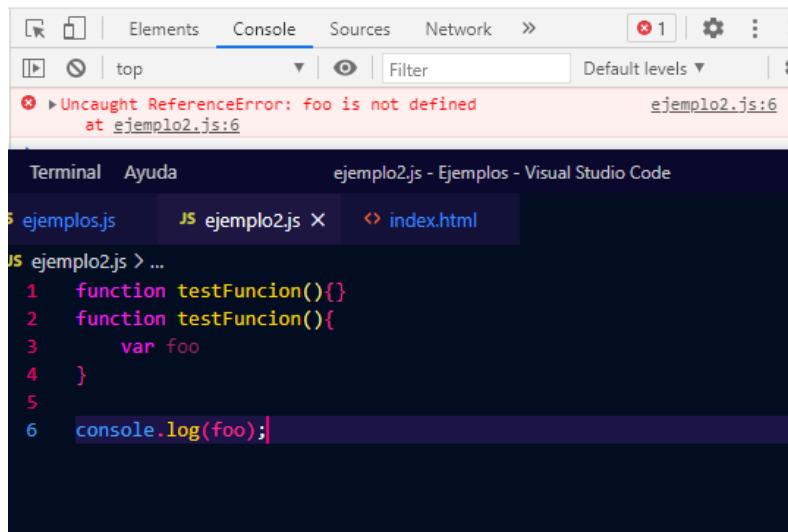
```
Var testArray=["test"];  
  
testFunction(testArray){  
  
    for(var i=0;i<testArray.length;i++){  
  
        console.log(testArray[i]);  
  
    }  
  
}
```

```
testFunction(testArray);
```

- **Uncaught TypeError: Cannot set property.** Cuando se trata de acceder a una variable no definida siempre retorna undefined y no puede obtener ni enviar ninguna propiedad de undefined. En este caso, dará este error. Vea un ejemplo donde saltará el error:



- **ReferenceError:** evento is not defined. Para finalizar, va a ver este error que ocurre cuando se trata de acceder a una variable que no está definida o no puede acceder a ella por que está fuera del ámbito establecido. Vea un trozo de código que replica este error:



A screenshot of the Visual Studio Code interface. At the top, the 'Console' tab is selected, showing a red error message: 'Uncaught ReferenceError: foo is not defined at ejemplo2.js:6'. Below the tabs, there's a terminal window with the command 'ejemplos.js' and an editor window with 'JS ejemplo2.js X'. The code in the editor is:

```
1 function testFuncion(){}
2 function testFuncion(){
3     var foo
4 }
5
6 console.log(foo);
```

RECURSO MULTIMEDIA



4.2.2. Escritura del programa fuente

Decir programa fuente es lo mismo que decir código fuente. El programa fuente es un conjunto de líneas de texto que forman parte esencial de un programa informático, siendo esto las instrucciones que debe seguir el ordenador para poder realizar la ejecución de una orden determinada. En este código se hace referencia al funcionamiento general de una aplicación o una herramienta en particular que lleva sus propias reglas y excepciones. Este código está creado en un idioma que es comprensible para el programador mediante la utilización de un lenguaje de programación determinado. Por ello, el código fuente se define como todo lo contrario al código objeto. Este último es el responsable de que el ordenador pueda interpretar las acciones que se ordenan y los comandos que han sido prefijados en este código para poder ser ejecutados, interpretados y retransmitidos por los componentes físicos de un ordenador.

Para poder realizar esta interpretación es necesario contar con sistemas de traducción que son conocidos bajo los nombres de compiladores, intérpretes y ensambladores. Entre otros, estos sistemas de traducción son los encargados de llevar estos lenguajes hacia un sistema binario de ceros y unos. Actualmente, también, es considerado como código fuente el lenguaje en el que está escrito un sitio web o alguna de sus herramientas web que estén presentes. En este caso, el navegador web es el intérprete de dichas codificaciones.

Teniendo en cuenta qué es y cómo funciona el código fuente, tiene que estar atentos a cómo lo organiza, ya que, si hay que arreglar algún fallo posteriormente, será mucho más sencillo hacerlo en un código bien organizado y legible. Es por ello que hay que utilizar comentarios siempre que lo necesite y así, saber para qué sirve las líneas de código que está escribiendo. A continuación, vea un código mal organizado:

```
JS ejemplo2.js > ⏺ saludo
1  function saludo (nombreUsuario){
2    while(nombreUsuario==""){
3      var nombreUsuario = prompt("Escriba su nombre de usuario");
4    }
5    if(nombreUsuario!=""){
6      document.write(`Hola, ${nombreUsuario}`);
7    }
8
9  }
10 var nombreUsuario = prompt("Escriba su nombre de usuario");
11 saludo(nombreUsuario);
```

En el código anterior, no se ve con claridad dónde comienzan las estructuras de control, para qué sirve la función o dónde empieza el programa principal. Vea a continuación el código de manera bien organizada:

```
//Esta función imprime por pantalla un saludo junto con el nombre de usuario
function saludo (nombreUsuario){
  /*Este while permite que el programa vuelva a pedir introducir nombre
  si el usuario no ha introducido nada*/
  while(nombreUsuario==""){
    var nombreUsuario = prompt("Escriba su nombre de usuario");
  }
  //este if hace que si el usuario introduce un nombre, se ejecute el saludo
  if(nombreUsuario!=""){
    document.write(`Hola, ${nombreUsuario}`);
  }
}
//Programa principal
var nombreUsuario = prompt("Escriba su nombre de usuario");
//Llamada a La función saludo()
saludo(nombreUsuario);
```

De esta manera, si en un futuro quiere cambiar el funcionamiento del programa o si cuando vaya a depurar el programa salta un error, será capaz de darle solución mucha más rápido y eficazmente,

4.2.3. Compilación del programa fuente

La tarea de compilar se refiere al proceso de traducción del código fuente, entendiéndose por código fuente las líneas de código que se han escrito en un lenguaje de programación, en este caso un

lenguaje de programación de alto nivel. Esta compilación del código fuente de un programa se realiza debido a que el código trabajado por el lenguaje de programación no es ejecutable directamente por la computadora, es por ella la necesidad de traducir las instrucciones contenidas en el texto al llamado lenguaje de máquina o código binario, la única manera posible de que la computadora entienda y, luego, ejecute las sentencias, las órdenes e instrucciones determinadas por el código fuente. Para realizar esta traducción, mejor llamada compilación, el desarrollador de software se vale de una herramienta llamada compilador.

Por otra parte, los compiladores son los programas encargados de hacer la traducción del código fuente de un programa, creado mediante un lenguaje de programación, a lenguaje de máquina, ya que, es el único lenguaje que entienden los procesadores de las computadoras. Los compiladores básicamente se dividen en dos partes. La primera es el Front End que es la parte del compilador encargada de analizar y comprobar la validez del código fuentes y en base a ella, crea los valores de la tabla de símbolos. Esta parte generalmente es independiente del sistema operativo para el cual se está compilando un programa. La segunda parte del compilador se denomina Back End, parte en la cual es generado el código máquina, el cual es creado de acuerdo a lo analizado en el Front End, para una plataforma específica, es decir, Windows, Linux, Mac, etc.

Cabe destacar que los resultados obtenidos por el Back End no pueden ser ejecutados en forma directa. Para ello, es necesaria la utilización de un proceso de enlazado llamado Linker, el cual básicamente es un software que recoge los objetos generados en la primera instancia de compilación incluyendo la información de las bibliotecas. Este software, primero, depura los objetos y luego enlaza el código objeto con sus respectivas bibliotecas para finalmente crear un archivo ejecutable.

Recorde que JavaScript es un lenguaje de guion y, por lo tanto, es un lenguaje interpretado. Esto quiere decir que es todo lo contrario a un lenguaje compilado. El intérprete de JavaScript es el propio navegador y, este es el que hace la función de “traducir” el código, línea por línea, a lenguaje de máquina. Esta acción se realiza en el momento de la ejecución del *script*. A pesar de ello, existen actualmente unas aplicaciones que simulan la compilación del código JavaScript y que se pueden integrar con el resto de desarrollo IDE. Estas aplicaciones ya han sido mencionadas anteriormente en otro apartado. Entre ellas destaca JavaScript Lint para Windows y CoffeeScript para Linus. Además, cabe destacar una herramienta online que también sirve para esta función, su nombre es **Ideone**.

4.2.4. Corrección de errores de sintaxis

Los errores de sintaxis son los errores de ortografía en el código que provocan que su programa no se ejecute en absoluto o que deje de funcionar a mitad del camino. Por lo general, estos errores también proporcionarán algunos mensajes de error. Normalmente, no es difícil corregirlos, siempre y cuando esté familiarizado con las herramientas adecuadas y sepa qué significan los mensajes de error.

Los errores de sintaxis más comunes son, por ejemplo, escribir una variable con tilde. Vea un ejemplo:

Var número;

Aunque, actualmente, sí pueda utilizar variables con tildes no es aconsejable, ya que, cualquier fallo ortográfico en la utilización de las variables dentro de las funciones, puede conllevar a que el programa no se ejecute adecuadamente. Además, puede que en otros lenguajes de programación no esté permitido el uso de tildes, por lo tanto, debería acostumbrarse a seguir unas normas de sintaxis comunes.

Otro de los errores comunes es, por ejemplo, el uso del operador de asignación =, cuando realmente quiere hacer una comparación == o una comparación estricta ===. Vea un ejemplo:

```
Var numero = 0;  
  
If (numero = 0){  
  
    Console.log("Cero");
```

Si intenta ejecutar este código, no va a saltar ningún error, pero tampoco se mostrará por consola la palabra cero como ha programado que se muestre. Esto ocurre porque realmente en la condición de if se está asignando al a variable el número 0 pero no la está igualando a 0, es decir, la expresión no puede dar un resultado booleano de true o false.

Otro de los errores comunes es poner los string sin comillas. Por ejemplo:

```
Console.log(Cero); en vez de console.log("Cero");
```

Por último, uno de los errores más comunes es escribir mal las instrucciones, por ejemplo, escribir console.lo(); en vez de console.log(). Aunque, si utiliza alguna herramienta como VSCode, el propio programa corregirá las palabras automáticamente.

Para concluir, la mejor forma de ir detectando este tipo de errores es ejecutar su programa poco a poco. Para ello, debe dividir el programa en varias partes e ir solucionando los errores parte por parte.

4.2.5. Corrección de errores de ejecución

Los errores de ejecución pueden deberse, por ejemplo, a que se intente acceder a una URL que no existe o a que se llame a una función con argumentos de tipo o valor incorrecto. En JavaScript, cuando se produce un error de ejecución, se lanza una excepción, como ya ha visto anteriormente.

Otro tipo de errores que existen en JavaScript son los errores lógicos. Estos son errores en los que la sintaxis, realmente, es correcta pero el código no hace lo que se pretendía que debía hacer, lo cual significa que el programa se ejecuta, pero da resultados incorrectos. A menudo, estos son más difíciles de arreglar que los errores sintácticos, ya que generalmente no hay un mensaje de error que pueda orientar hacia la fuente del error.

Imagine que quiere que una variable dé un número aleatorio del 1 al 100. Si escribe la variable let numeroAleatorio=Math.floor(Math.random())+1; siempre da el número 1. Para solucionar este error,

debe fijarse en la lógica de las operaciones. El método Math.random() genera un número decimal entre el 0 y el 1. Para que este método dé un número entero, debe pasarlo por el método Math.floor(), que redondea el número pasado al número entero más cercano, por lo que la expresión quedaría de la siguiente forma: Math.floor(Math.random())+1

Redondear un número decimal aleatorio entre 0 y 1 siempre devolverá 0, es por ello que siempre debe agregarle +1, ya que, con esto hará que devuelva 1. Pero, aquí es donde viene el error de esta sintaxis y es que, si quiere un número aleatorio del 1 al 100, necesita multiplicar el número por 100 antes de sumarle la unidad. Quedaría la siguiente expresión:

```
Math.floor(Math.random()*100)+1;
```

Ahora, siga la lógica con un ejemplo. Imagine que el método Math.random() da el número 0,132. Tendría el siguiente resultado:

```
Math.floor(0,132*100)+1;
```

Si hace las operaciones quedaría:

```
Math.floor(13,2)+1;
```

Ahora, el método Math.floor() redondeará el número al número entero más cercano que es 13 y, después, se sumaría la unidad. Así que, el número aleatorio que obtendría en su programa sería 14, que es lo que se mostraría en la consola.

Como ha visto, la clave para arreglar los errores lógicos es entender perfectamente cómo funciona el programa.

4.3. Mensajes de error

En JavaScript puede encontrar diversos tipos de errores. Estos errores los puede ver en la consola de cada navegador cuando ejecuta su programa. Vea los siguientes tipos de errores.

- **RangeError.** Errores generados por números fuera de rango, por ejemplo, con la siguiente sentencia:

```
Var rango=new array(10000000000000000000000000000000);
```

Este error ocurre porque los arrays solo pueden almacenar valores entre 0 y $2^{32}-1$.
- **ReferenceError.** Este error ya lo ha visto anteriormente y se produce cuando referencia variables que no existen.
- **SyntaxError.** Se lanza cuando no se cumplen las reglas del lenguaje, es decir, cuando tiene un error ortográfico.
- **TypeError.** Se produce cuando un valor no es de un tipo esperado, por ejemplo, al llamar a un método no existente de un objeto. Por ejemplo:

```
Var x={}; x.y();
```

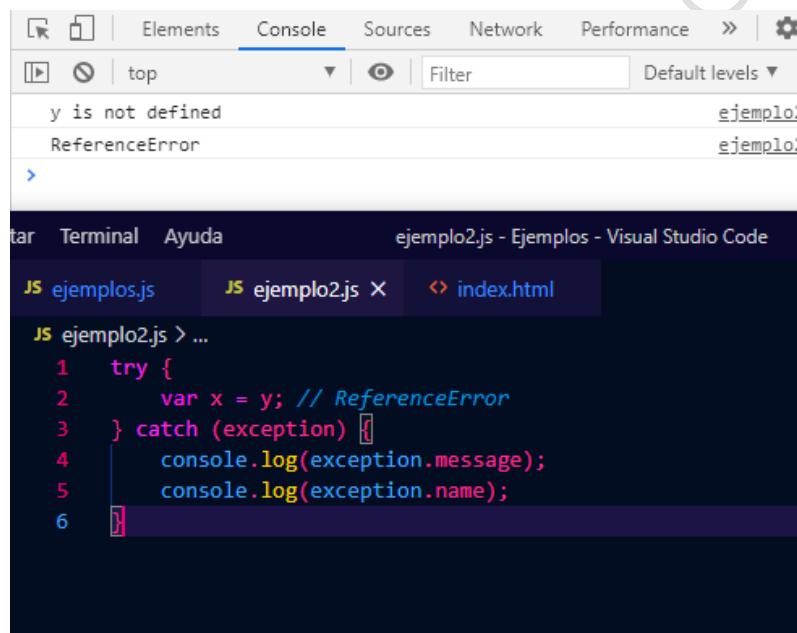
- **URIError.** Este error se lanza cuando métodos propios del lenguaje como encodeURI() o decodeURI() se invocan con URIs no válidas. Por ejemplo:

```
DecodeURIComponent(http://goo%%gle.com);
```

4.3.1. Funciones para controlar los errores

En JavaScript, como en otros lenguajes, existen métodos y eventos para controlar las excepciones y que, así, su programa no deje de funcionar.

El bloque que permite controlar las excepciones es try catch finally. En este bloque la única parte obligatoria es try, por lo que, puede ejecutar catch y finally de manera opcional. Catch solo se ejecuta si se produce una excepción en try y, aunque es opcional, si no lo incluye realmente no estará gestionando nada. En cuanto la excepción se produce en try, el control pasa a la sentencia catch y tendrá disponible la información acerca de la misma en la variable exception. Vea un ejemplo:



```
tar Terminal Ayuda ejemplo2.js - Ejemplos - Visual Studio Code
JS ejemplos.js JS ejemplo2.js X < index.html
JS ejemplo2.js > ...
1  try {
2      var x = y; // ReferenceError
3  } catch (exception) {
4      console.log(exception.message);
5      console.log(exception.name);
6 }
```

Gracias al bloque Try Catch, la consola muestra que la variable y no está definida y que el tipo de error que lanza es un ReferenceError.

Además, podría necesitar distinguir entre diferentes tipos de excepciones para gestionar los errores de diversas maneras. Para esto, podría utilizar instanceof dentro del bloque Catch de la siguiente manera:

```
Ejemplo2.js
1 if (exception instanceof TypeError) {
2   ...
3 } else if (exception instanceof ReferenceError) {
4   ...
5 } else {
6   ...
7 }
```

Por último, el bloque finally se ejecuta siempre haya o no haya excepción. Hay que tener en cuenta que incluso se ejecutará, aunque se haga un return en el bloque catch.

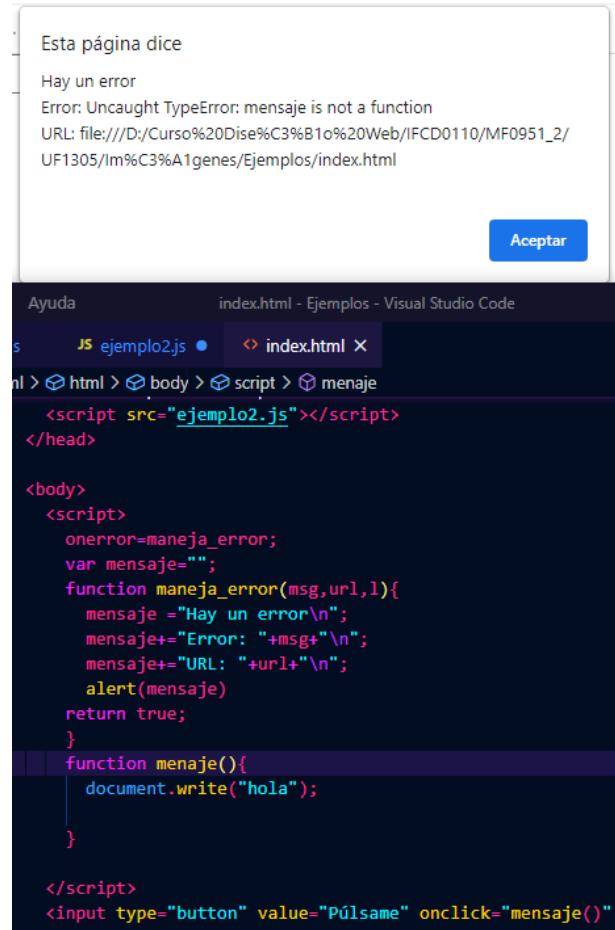
Por otro lado, tiene el evento onerror. Este evento se usa para capturar errores de la misma forma que usa try catch. Para usar este evento debe definir una función, que será la encargada de manejar los errores. A dicha función le debe pasar tres argumentos:

- **Msg.** Es el propio mensaje de error.
- **Url.** Es la página que causó el error.
- **L.** Es la línea de código donde ocurrió el error.

Esta función deberá retornar un valor booleano. Vea la sintaxis para usar este evento:

```
Onerror = función();  
  
Function función(msg,url,l){  
  
    Código para manejar el error.  
  
    Return true|false);  
  
}
```

A continuación, va a ver cómo se usa esta función en un caso práctico. Vea como ejemplo el código de la siguiente imagen:



Como puede observar, el botón está asociado a la función `mensaje()`, pero esta función tiene un error ortográfico; ya que, en vez de haberse llamado `mensaje()`, se llama `menaje()`. Entonces, lo que sucede es que al pulsar el botón que ha programado en HTML, en vez de ejecutar la función, que no se encuentra porque no tiene el mismo nombre, ejecuta la función que maneja el error. Es por ello que al pulsar el botón aparece el mensaje de error que ha programado en la función `maneja_error()`. De esta forma, será capaz de manejar los errores que pueda encontrar al programar, por ejemplo, botones.

5. GESTIÓN DE OBJETOS DEL LENGUAJE DE GUIÓN

Este tema se va a centrar en los objetos que puede encontrar en el lenguaje de guion y, sobre todo, en los que puede encontrar en JavaScript. Conocer los objetos, sus propiedades y sus métodos hace más fácil programar de manera mucho más clara, cómoda y sencilla.

Los objetos en JavaScript, como en otros muchos lenguajes de programación, se pueden comparar con objetos de la vida real. En JavaScript, un objeto es una entidad independiente con propiedades y tipos. Una propiedad de un objeto se puede explicar como una variable adjunta al objeto. Las propiedades de un objeto son lo mismo que las variables comunes de JavaScript, excepto por el nexo que tienen con el propio objeto. Estas definen las características del objeto.

A continuación, en este tema verá cómo gestionar y manejar de manera eficiente los objetos que proporciona el propio lenguaje. Esto es crucial para que programe de una manera profesional y eficiente.

5.1. Jerarquía de objetos

La jerarquía de los objetos es el tema más importante para aprender a manejar JavaScript y controlar lo que ocurre dentro del navegador con toda la potencia que ofrece el lenguaje. Se trata de aprender el DOM, es decir, Document Object Model o, lo que es lo mismo, modelo de objetos del documento. El DOM sirve para acceder a cualquiera de los componentes que hay dentro de una página. Por medio del DOM puede controlar el navegador y los distintos elementos que se encuentran en la página.

Cuando se carga una página, el navegador crea una jerarquía de objetos en memoria que sirven para controlar los distintos elementos de dicha página. Con JavaScript y la nomenclatura de objetos que ha aprendido, puede trabajar con esa jerarquía de objetos, acceder a sus propiedades e invocar sus métodos.

Cualquier elemento de la página se puede controlar de una manera u otra accediendo a esa jerarquía. Es crucial conocerla bien para poder controlar perfectamente las páginas web con JavaScript o cualquier otro lenguaje de programación del lado del cliente.



TOME NOTA

Antes de empezar a ver rigurosamente el DOM, va a ver un ejemplo típico de acceso a una propiedad de esta jerarquía para cambiar el aspecto de la página. Se trata de una propiedad de la página que almacena el color de fondo de la página web. Esta es la propiedad bgColor del objeto document.

Vea el ejemplo:

```
Document.bgColor = "blue";
```

Si ejecuta esta sentencia en cualquier momento, cambia el color de fondo de la página a azul. Hay que fijarse en que la propiedad bgColor tiene la C en mayúscula. Es un error típico equivocarse con las mayúsculas y las minúsculas en la jerarquía. Si no lo escribe bien, no funcionará y en algunos casos ni siquiera dará un mensaje de error.

En esta página definida con color de fondo blanco ha cambiado esa propiedad con JavaScript, por lo que saldrá un color de fondo azul. Vea el ejemplo:

```
1  <HTML>
2    <HEAD>
3      <TITLE>Prueba bgColor</TITLE>
4    </HEAD>
5    <BODY bgcolor=white>
6
7    <script>
8      document.bgColor = "blue"
9    </script>
10   </BODY>
11   </HTML>
```

En los ejemplos que ha visto hasta ahora, también, ha utilizado los objetos del DOM. En concreto, ha utilizado mucho el método `document.write()` para escribir texto en la página web.



Más Info

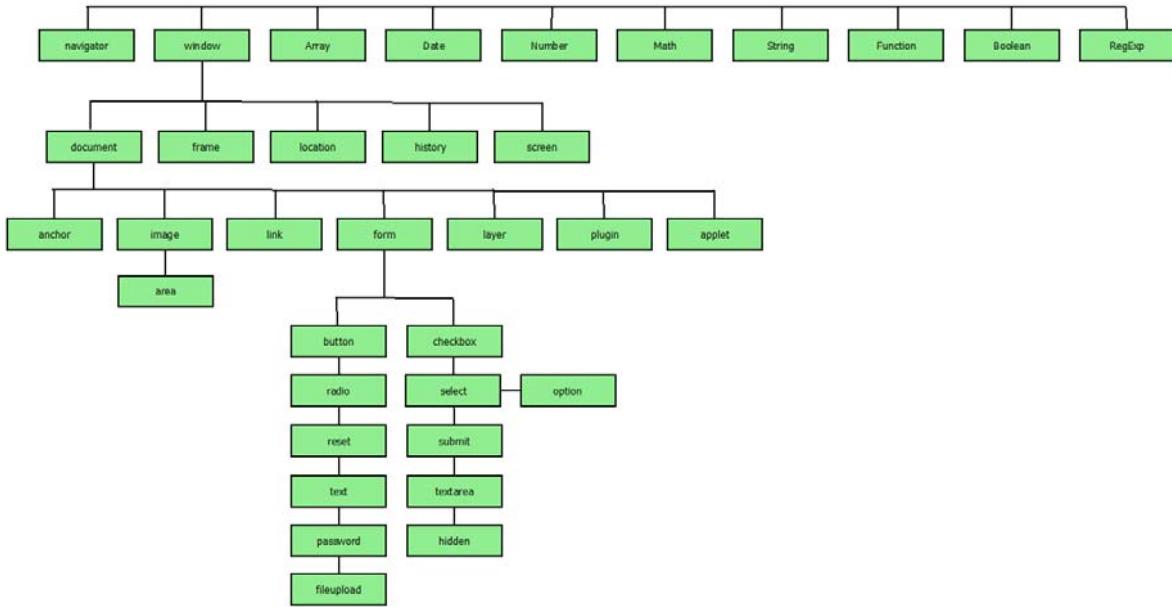


RECURSO MULTIMEDIA



5.1.1. Descripción de objetos de la jerarquía

En este apartado va a ver cómo está compuesta la jerarquía de objetos del navegador, más profesionalmente llamada DOM. Para ello, va a detallar alguno de sus elementos y verá una explicación sobre cómo se accede a ellos. Antes que nada, va a echar un vistazo al listado parcial de objetos que forman parte del DOM y en el cual aparecen los objetos generalmente más utilizados:



En esta imagen falta por recoger algunos objetos, pero sirve para hacerse una idea de cómo se organizan los objetos en el DOM. Como puede apreciar, todos los objetos comienzan en un objeto que se llama `window`. Este objeto ofrece una serie de métodos y propiedades para controlar el aspecto de la ventana, la barra de estado, abrir ventanas secundarias y otras cosas que verá más adelante cuando se explique el objeto en detalle.

Además de ofrecer control, el objeto `window` da acceso a otros objetos como el documento, que es la página web que se está visualizando, el historial de páginas visitadas o los distintos frames de la ventana. De modo que, para acceder a cualquier otro objeto de la jerarquía, debería empezar por el objeto `window`. Tanto es así que JavaScript entiende perfectamente que la jerarquía empieza en `window` aunque no lo señale. De manera que el ejemplo visto anteriormente para cambiar el color de fondo de la página, realmente, está empezando por `window`, lo que pasa es que JavaScript sobreentiende este concepto. Realmente estaría escribiendo:

```
Window.document.bgColor="blue";
```

O, por ejemplo:

```
Window.document.write();
```

Aunque podría utilizar estos objetos de esta forma, normalmente, se escribirán sin `window`, ya que, se sobreentenderá.

TOME NOTA

A continuación, va a ver una pequeña descripción de los objetos más importantes de la jerarquía, aunque será más adelante cuando se adentre en la explicación de cada objeto:

- **Window.** Hace referencia a la ventana en la que se encuentra y se puede observar cómo de este objeto dependen todos los demás. Es decir, para usar cualquier otro objeto, tiene que pasar por el objeto window.
- **Frame.** Se utiliza para dividir la ventana del documento en varias partes y mostrar en cada uno de ellas un contenido diferente.
- **Document.** Este objeto va a permitir controlar el documento y los elementos que están contenidos en él.
- **Location.** Este objeto contiene la URL actual.
- **History.** Este objeto es el que gestiona una lista con las páginas por las que ha navegado.
- **Navigator.** Este objeto da información sobre el navegador que está usando el usuario. Este objeto no depende de window.

5.1.2. Propiedades compartidas de los objetos

Muchas propiedades del objeto window son, a su vez, otros objetos. Es el caso de objetos como el historial de páginas web o el objeto documento, que tienen a su vez otras propiedades y métodos. Entre ellos destaca el objeto document, que contiene todas las propiedades y métodos necesarios para controlar muchos aspectos de la página. Ya ha visto alguna propiedad como bgColor, pero tiene muchas otras como el título de la página, las imágenes que hay incluidas, los formularios, entre otros.

Muchas propiedades de este objeto son, a su vez, otros objetos como los formularios. Todo esto lo verá cuando trate cada uno de los objetos por separado. Además, el objeto document tiene métodos para escribir en la página web y para manejar eventos de la página.

5.1.3. Navegar por la jerarquía de los objetos

El objetivo de este capítulo sobre la jerarquía de objetos es aprender a navegar por ella para acceder a cualquier elemento de la página. Esta no es una tarea difícil, pero puedo haber algún caso especial en el que acceder a los elementos de la página se haga de una manera que aún no se haya aprendido.

Como ha visto anteriormente, toda la jerarquía empieza en el objeto window, aunque no era necesario hacer referencia a window para acceder a cualquier objeto de la jerarquía. El siguiente objeto por orden de importancia es el objeto document, donde puede encontrar alguna propiedad especial que merece la pena comentar por separado debido a que su acceso es diferente a lo que ha aprendido. Esto se debe a que en document hay propiedades que se comportan como arrays, por ejemplo, la propiedad images es un array con todas las imágenes de la página web. También, se encuentran arrays para guardar los enlaces de la página, los applets, los formularios y las anclas.

Cuando una página se carga, el navegador construye, en memoria, la jerarquía de objetos. De manera adicional, construye también estos arrays de objetos. Por ejemplo, en el caso de las

imágenes va creando el array colocando en la posición 0 la primera imagen, en la posición 1 la segunda imagen y así hasta que las introduce todas. A continuación, va a crear un bucle que recorrerá todas las imágenes de la página e imprimirá su propiedad src que contiene la URL donde está situada la imagen:

```
For (i=0;i<document.images.length;i++){  
  
    Document.write(document.images[i].src)  
  
    Document.write("<br>")  
  
}
```

Se utiliza la propiedad length del array images para limitar el número de iteraciones del bucle. Luego, se utiliza el método write() del objeto document pasándole el valor de cada una de las propiedades src de cada imagen.

A continuación, va a ver el uso de otro array de objetos. En este caso va a introducir un poco más profundo de la jerarquía para llegar a la matriz elementos de los objetos formulario. Esta matriz contiene cada uno de los elementos que componen el formulario. Para ello, Tendrá que acceder a un formulario de la página al que puede acceder por el array de formularios y dentro de él, a la propiedad elements que es otro array de objetos. Para cada elemento del formulario va a escribir su propiedad value que corresponde con la propiedad value que se escribe en HTML. Vea un ejemplo:

```
For (i=0;i<document.forms[0].elements.length;i++){  
  
    Document.write(document.forms[0].elements[i].value)  
  
    Document.write("<br>")  
  
}
```

Este bucle es muy parecido al que tenía para recorrer las imágenes, con la diferencia de que ahora recorre el vector de elements al que accede por la jerarquía de objetos pasando por el array de formularios en su posición 0, la cual corresponde con el primer formulario de la página.

Teniendo en cuenta todo lo que ha visto en este apartado, ha aprendido a moverse por la jerarquía de objetos, por lo tanto, puede acceder a cualquier elemento del navegador o de la página.



Más Info



RECURSO MULTIMEDIA



5.2. Propiedades y métodos de los objetos del navegador

Para nombrar a los objetos de la jerarquía de objetos JavaScript debe tener en cuenta las siguientes reglas:

- El nombre de un objeto que desciende de otro en la jerarquía de JavaScript incluye el nombre de los objetos padre de la jerarquía. Por ejemplo, el objeto document puede nombrarlo como window.document. Un objeto hijo es, a su vez, un objeto y una propiedad del objeto padre. Document es a su vez un objeto y una propiedad de window.
- Dado que todos los objetos que puede usar en el código descienden de window, normalmente se omite el uso de window a la hora de nombrar un objeto. Por eso se escribe, por ejemplo, document.body en lugar de window.document.body, aunque ambas formas son válidas.
- JavaScript organiza de forma automática ciertos objetos de naturaleza “múltiple” en arrays. Por ejemplo, una página web puede contener varios formularios. JavaScript, de forma automática, crea un array de objetos cuyo nombre es forms, siendo el primer formulario el de índice 0, como ya ha visto anteriormente.

En este apartado, va a ver los primeros objetos de la jerarquía del DOM y se adentrará en entenderlos para, posteriormente, poder usarlos correctamente.

5.2.1. El objeto superior Windows#

Window es el objeto principal en la jerarquía, contiene las propiedades y métodos para controlar la ventana del navegador. De él dependen todos los demás objetos de la jerarquía.

A continuación, va a ver una lista con sus propiedades y métodos:

- **Closed.** Indica la posibilidad de que se haya cerrado la ventana.
- **defaultStatus.** Texto que se escribe por defecto en la barra de estado del navegador.
- **Document.** Objeto que contiene a la página web que se está mostrando.
- **Frame.** Se utiliza para crear ventanas dentro de la ventana principal y, así, dividirlas para mostrar diferentes contenidos.
- **Frames array.** Es el vector que contiene todos los frames de la página. Se accede por su índice a partir de 0.
- **History.** Objeto historial de páginas visitadas.

- **innerHeight.** Tamaño en píxeles en vertical del espacio donde se visualiza la página.
- **innerWidth.** Tamaño en píxeles en horizontal del espacio donde se visualiza la página.
- **Length.** Número de frames de la ventana.
- **Location.** Es la URL del documento que se está visualizando. Puede cambiar el valor de esta propiedad para moverse a otra página. Ver también la propiedad location del objeto document.
- **Locationbar.** Objeto de barra de direcciones de la ventana.
- **Menubar.** Objeto barra de menús de la ventana.
- **Name.** Nombre de la ventana. Lo asigna cuando abre una nueva ventana.
- **Opener.** Hace referencia a la ventana del navegador que abrió la ventana donde está trabajando.
- **outherHeight.** Tamaño en píxeles en vertical del espacio de toda la ventana. Esto incluye las barras de desplazamiento, botones, etc.
- **outherWidth.** Tamaño en píxeles en horizontal del espacio de toda la ventana. Esto incluye los mismos elementos que outherHeight.
- **Parent.** Hace referencia a la ventana donde está situada el frame donde está trabajando.
- **Personalbar.** Objeto barra personal del navegador.
- **Self.** Ventana o frame actual.
- **Scrollbars.** Objeto de las barras de desplazamiento de la ventana.
- **Status.** Texto de la barra de estado.
- **Statusbar.** Objeto barra de estado del navegador.
- **Toolbar.** Objeto barra de herramientas.
- **Top.** Hace referencia a la ventana donde está situado el frame donde está trabajando, como la propiedad parent.
- **Window.** Hace referencia a la ventana actual, igual que la propiedad self.

Vea un ejemplo de utilización de la propiedad status del objeto window. Esta propiedad sirve para escribir un texto en la barra de estado del navegador. Este ejemplo lo hará con el manejador de eventos onclick que ya ha visto anteriormente. Para usar esta propiedad, simplemente se asigna un texto a la propiedad status del objeto window. El texto que se coloca en la barra de estado debe estar escrito entre comillas simples porque está escribiendo dentro de unas comillas dobles. Vea el código:

```
<form>  
  
<input type="Button" value="Púlsame" onclick="window.status='Bienvenido'">  
  
</form>
```

5.2.2. El objeto navigator

Puede recuperar un objeto de tipo Navigator haciendo la invocación objetoNav=window.navigator; o dado que no es necesario especificar window, simplemente se escribe navigator.

El objeto navigator que cree dispondrá de propiedades y métodos que pueden ser muy útiles, aunque de momento muchas de las propiedades y los métodos son experimentales o no responden de la misma manera en los distintos navegadores. Va a citar aquí algunas de las propiedades.

- **userAgent.** Devuelve una cadena de texto representando el agente que se está empleando en la navegación. Puede identificar el navegador empleado, pero no es seguro porque puede configurarse para falsearlo. Vea un ejemplo de cómo usarlo:
`alert(window.navigator.userAgent);`
- **battery.** Devuelve un objeto BatteryManager que puede usarse para obtener información sobre el estado de la batería. No disponible para todos los navegadores. Vea un ejemplo:
`alert('¿Cargando?:'+window.navigator.battery.charging);`
Este código devolverá true si está cargando.
- **Geolocation.** Devuelve un objeto geolocation que puede usarse para obtener información sobre la ubicación (latitud, longitud) desde donde el usuario navega. Algunos navegadores no responden bien. Otros restringen o piden permiso al usuario por considerar que puede atentar contra su privacidad. A continuación, verá un ejemplo:
`Navigator.geolocation.getCurrentPosition(funcionExito,funcionFallo,arrayOpciones);`
- **Language.** Devuelve un string representativo del lenguaje del navegador. Algunos navegadores no disponen de language. Sin embargo, tienen disponible userLanguage. Otros navegadores no reconocen ni una ni otra forma. Vea el siguiente ejemplo:
`Var lenguaje = navigator.language || navigator.userLanguage;
Alert('Código lenguaje navegador:'+lenguaje);`
- **Online.** Devuelve un valor booleano que indica si se tiene o no conexión a Internet. Combinarlo con eventos window.ononline y window.onoffline, puede servir para mostrar un mensaje de alerta si se pierde o se recupera la conexión. Vea un ejemplo:
`Alert ('¿Está online?:'+window.navigator.onLine);`
Esta alerta devolverá true si está conectado a Internet.

Entre los métodos de los objetos navigator, únicamente citar a navigator.vibrate(), que generará una vibración en aquellos dispositivos que admiten la vibración, como por ejemplo, los smartphones.

RECURSO MULTIMEDIA



5.2.3. URL actual (location)

Este objeto, el cual puede usarse sin el prefijo window, sirve tanto para obtener la URL o parte de ella de la web donde se encuentra como para redireccionarse hacia otra página. Para recoger la URL de una página, tan solo, debe usar la instrucción window.location.href, la cual puede imprimir, guardar en una variable, introducir en un array, en una condición if o cualquier cosa que pueda necesitar.

Por otro lado, puede ser que no necesite toda la URL de la página, ya que puede que esté, por ejemplo, ejecutando el código en un resultado de búsqueda o por cualquier otra razón y aunque podría cortar la cadena y recoger solo la parte que necesitará, no es una solución muy elegante, sobre todo cuando JavaScript lo permite utilizando el siguiente código:

Window.location.hostname

De esta forma solo recibiría un resultado del hosting que sería, por ejemplo: miweb.com.

Además, puede ser que solo necesite el path, sin necesidad del hosting, para por ejemplo direcciones relativas y para ello, JavaScript ayuda con otra propiedad muy sencilla:

Window.location.pathname

Puede resultar interesante también, saber si la web se encuentra bajo un protocolo http o https, y la propiedad location permite consultarla con la siguiente línea de código:

Window.location.protocol.

En conclusión, para redireccionar con JavaScript debe hacerlo de la siguiente manera:

Window.location="dirección donde quiere redireccionar"

Por otra parte, vea los métodos que ofrece location:

- **Reload()**. Este método vuelve a cargar la URL que está especificada en la propiedad href del objeto "location".
- **Replace(cadena)**. Este método reemplaza el historial actual mientras carga la URL pasada en el parámetro cadena.

5.2.4. URL visitada por el usuario

El objeto history sale directamente del objeto window. El objeto history almacena todas las páginas que visita. Luego, con una serie de funciones, puede extraer de la memoria de la computadora las páginas ya visitadas sin tener que pedirlas nuevamente al servidor. La propiedad de solo lectura, window.history, devuelve una referencia al objeto history, quien provee una interfaz para manipular el historial de sesión del navegador, es decir, las páginas que han sido visitadas por el usuario. Por razones de seguridad, el objeto history no permite que el código sin privilegio acceda a las URLs de otras páginas en el historial de la sesión, pero permite navegar por el historial de la sesión.

A continuación, va a ver las funciones con las que cuenta el objeto history:

- **Window.history.back()**. Retrocede a la página anterior. Si llama a esta función obtendrá el mismo comportamiento que cuando presiona el botón “atrás” del navegador.
- **Window.history.forward()**. Avanza a la página siguiente que está almacenada en la caché del ordenador.
- **Window.history.go()**. Avanza o retrocede en la lista de páginas visitadas. Para avanzar por el historial debe escribir un número entero positivo y para retroceder, un número entero negativo. Vea un ejemplo:

```
Window.history.go(1);
```

```
Window.history.go(-2);
```

Además, algunas de las propiedades de las que dispone el objeto history son:

- **Current**. Propiedad que contiene una cadena con la URL completa de la entrada actual del historial.
- **Next**. Propiedad que contiene una cadena con la URL completa de la siguiente entrada del historial
- **Length**. Propiedad que contiene un número entero que representa el número de entradas del historial, es decir, el número de direcciones que han sido visitadas por el cliente.
- **Previous**. Propiedad que contiene una cadena con la URL completa de la entrada anterior en el historial.

Como nota, debe saber que HTML 5 ha introducido los métodos history.pushState() e history.replaceState(), los cuales permiten añadir y modificar entradas del historial, respectivamente. Ambos métodos trabajan en conjunto con el evento window.onpopstate.

Por un lado, el método pushState() toma tres parámetros un objeto estado, un título y, opcionalmente, una URL. El objeto estado es un objeto JavaScript el cual está asociado con la nueva entrada al historial creada por pushState(). Cada vez que el usuario navega hacia un nuevo estado, un evento popstate evento se dispara, la propiedad state del evento contiene una copia del historial de entradas del objeto estado.

Por otra parte, el título es algo que actualmente los navegadores ignoran, aunque a lo mejor en un futuro se le da utilidad. Para ello, lo más común es pasar una cadena de caracteres vacía o, como alternativa, podría pasar un título corto del estado hacia el cual se está moviendo.

Por último, la URL de la nueva entrada al historial estará dada por este parámetro. Recuerde que el navegador no intentará cargar la URL después de llamar a pushState(), pero podría intentar cargar la URL más tarde, por ejemplo, después de que el usuario reinicie su navegador.

En cambio, el método history.replaceState() trabaja exactamente igual a history.pushState excepto que replaceState() modifica la entrada al historial actual en lugar de crear una nueva. replaceState()

es particularmente útil si se desea actualizar el objeto estado o la URL de la actual entrada al historial en respuesta a alguna acción del usuario.

Para finalizar, el evento popstate es dirigido a la ventana cada vez que la entrada al historial cambia. Si la entrada al historial es activada y fue creada por un llamado a pushState o afectada por una llamada a replaceState, la propiedad state del evento popstate contiene una copia del historial de entradas del objeto de estado.

5.2.5. Contenido del documento actual (document)

El objeto document es el que tiene el contenido de toda la página que se está visualizando. Esto incluye el texto, las imágenes, los enlaces, los formularios, entre otros. Gracias a este objeto, va a poder añadir dinámicamente contenido a la página o hacer cambios según convenga.

Además, este objeto no solo sirve para cambiar la estética de la página también sirve para modificar aspectos como los links, los formularios, las URLs de las páginas, etc.

En el siguiente apartado, va a ver las propiedades más básicas que tiene el objeto window.document. Aunque sean las propiedades más básicas, son las tres propiedades más utilizadas el objeto window.

a) Título, color de fondo y formularios

Title. Esta propiedad contiene una cadena de texto que pertenece al título del documento, es decir, al título contenido en la etiqueta <title>. Vea un ejemplo:

```
<!DOCTYPE html>

<html>
  <head>
    <title>Bienvenidos a mi web</title>
  </head>
  <body>
    <script>
      Alert(document.title); // Esto mostrará el título que hay contenido en <title>
      Document.title = "Esta es mi web";
      Alert(document.title); // Esto sobreescibirá lo que hay en la etiqueta <title>
                            por Esta es mi web.
```

```
</script>  
  
</body>  
  
</html>
```

bgColor. Esta propiedad se utiliza para cambiar el color de fondo del documento HTML. Este color se expresa como una cadena de dígitos hexadecimales o como un nombre de color que esté dentro de los estándares de JavaScript. La forma hexadecimal de expresar el color es la siguiente “RRGGBB”. También, puede cambiar el color con el atributo bgcolor dentro de la etiqueta <body>. Vea unos ejemplos a continuación:

```
<html>  
  
<head>  
  
<title>Esta es mi web</title>  
  
</head>  
  
<body bgcolor="blue">  
  
<script>  
  
    document.bgColor="blue";  
  
    document.bgColor="#ff00ff";  
  
</script>  
  
</body>  
  
</html>
```

Como puede ver tiene tres ejemplos de cómo puede ponerle un color de fondo a su documento HTML. Todos estos ejemplos son igualmente válidos y correctos.

Forms. El objeto document.forms es el array que contiene todos los formularios de la página. Como se trata de un array, el acceso a cada formulario se realiza con la misma sintaxis de los arrays. La siguiente instrucción accede al primer formulario de la página:

```
Document.forms[0];
```

Además del array de formularios, el navegador crea automáticamente un array llamado elements por cada uno de los formularios de la página. Cada array elements contiene la referencia a todos los elementos cuadros de texto, botones, listas desplegables, entre otros; de ese formulario. Utilizando

la sintaxis de los arrays, la siguiente instrucción obtiene el primer elemento del primer formulario de la página:

```
Document.forms[0].elements[0];
```

La sintaxis de los arrays no siempre es tan concisa. El siguiente ejemplo muestra cómo obtener directamente el último elemento del primer formulario de la página:

```
Document.forms[0].elements[document.forms[0].elements.length-1];
```

5.3. Propiedades y métodos de los objetos del documento

En este apartado, va a ver, en profundidad, las propiedades y métodos del objeto `window.document`. También verá algunos ejemplos para entender mejor cómo funciona este objeto y sus propiedades.

Este objeto es esencial saber utilizarlo correctamente, ya que, es el que cambiará la estética a su web.

5.3.1. Propiedades del objeto document

Como ha visto anteriormente, el objeto `document` hace referencia al propio documento cargado en la ventana. En las primeras implementaciones de un modelo rudimentario de DOM, `document` contaba entre sus propiedades con una serie de colecciones que permitían acceder a parte de su contenido como:

- **Document.anchors.** Devolvía una matriz con todas las anclas del documento, es decir, todos los elementos ``
- **Document.applets.** Devolvía una matriz con todos los elementos applet.
- **Document embeds.** Devolvía una matriz con todos los elementos embed.
- **Document.forms.** Devolvía una matriz con todos los formularios.
- **Document.images.** Devolvía una matriz con todas las imágenes.
- **Document.links.** Devolvía una matriz con todos los vínculos.
- **Document.location.** Hacía referencia a la URL del documento.

Hoy en día, embeds ha sido despreciado y las demás propiedades son colecciones de nodos definidas en la interfaz `HTMLDocument` de DOM. No obstante, cuenta con métodos mucho más poderosos, sofisticados y estandarizados para acceder a lo mismo.

Por ello, se centrará en `location`. `Location` es un objeto dependiente de `document`, aunque también lo es de `window` y como tal posee propiedades y métodos.

Véalo a continuación:

- **Hash.** En una URL que apunta a un ancla del documento corresponde a la parte tras el almohadillado.
- **Host.** Es el nombre del servidor, es decir, en una URL se obviaría el protocolo http o https.
- **Hostname.** Ya se vió anteriormente y es lo mismo que host, pero eliminando www.
- **Href.** Es el URL completo. También lo ha visto anteriormente.
- **Pathname.** Sería la ruta tras el nombre del dominio.
- **Port.** Es el puerto especificado en la ruta.
- **Protocol.** Es el protocolo empleado para acceder al documento.
- **Search.** Es la cadena adjunta a la ruta de un documento enviada como búsqueda tras el signo de interrogación.

Además, tiene otros tipos de propiedades que están más relacionadas con la estética del documento en sí. Puede verlas a continuación:

- **alinkColor.** Esta propiedad tiene almacenado el color, en formato RGB, de los enlaces que estén activos.
- **Cookie.** Esta propiedad contiene una cadena con los valores de las cookies de la página web.
- **Domain.** Propiedad que sirve para almacenar el nombre del servidor que ha servido a la página web.
- **fgColor.** Propiedad que almacena el color en formato RGB del primer plano del documento.
- **lastModified.** Esta propiedad es una cadena que contiene la fecha de la última modificación de la página.
- **linkColor.** Esta propiedad almacena el color, en formato RGB, de los enlaces de la página.
- **Links.** Esta propiedad contiene un array que tiene los enlaces externos de la página web.
- **Referrer.** Propiedad que contiene una cadena con la URL de la página que llamó a la página web actual.
- **vlinkColor.** En esta propiedad puede almacenar el color, en formato RGB, de los enlaces visitados de la web.



Más Info



RECURSO MULTIMEDIA



5.3.2. Ejemplos de propiedades de document

A continuación, va a ver una serie de código de lenguaje de marcas e incrustación de lenguaje de guion con las propiedades más básicas del objeto document de JavaScript:

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

    Document.write(document.alinkColor);

    Document.write(document.gbColor);

    Document.write(document_fgColor);

    Var formularios="";

    For (i=0;i<document.forms.length;i++)

        Formularios+=document.forms"/n";

    If (formularios==0)

        Document.write("No hay formularios <br>");

    Else

        Document.write (document.forms.item(1));

        Document.write (document.lastModified);

        Document.write (document.linkColor);

        Document.write (document.location);

        Document.write (document.title);

        Document.write (document.vlinkColor);

        Document.write (document.referrer);

//Defina los enlaces

Var enlaces="";

For (i=0;i<document.links.length;i++)

    Enlaces+="document.link["+i+"]="+document.links[i]+<br>";
```

```
If (enlaces== "")  
    Document.write("No hay enlaces <br>");  
  
Else  
  
    Document.write (enlaces);  
  
</script>
```

Ahora, va a utilizar el evento onclick para colocar tres botones en la página que al pulsarlos cambie el color de fondo de la página. Para ello, verá el siguiente script:

```
<script>  
  
Function cambiarColor (colorear){  
  
    Document.bgColor=colorear  
  
}  
  
</script>  
  
<form>  
  
<input type="Button" value="Morado" onclick="cambiaColor('purple')">  
  
<input type="Button" value="Verde" onclick="cambiaColor('green')">  
  
<input type="Button" value="Amarillo" onclick="cambiaColor('yellow')">  
  
</form>
```

En este ejemplo, primeramente, define la función que será la encargada de cambiar el color de fondo y luego, crea tres botones que llamarán a la función cuando se pulsen pasándole el color como parámetro. Recuerde que el color se puede pasar tanto en hexadecimal como con un nombre del color predefinido por JavaScript.

Por último, verá un ejemplo individual de la propiedad title. La propiedad title guarda la cadena que conforma el título de su página web. Si accede a dicha propiedad, obtendrá el título y si la cambia, cambiará el título de la página web. Va a mostrar el título de la página en una caja de alerta y va a crear una función que modifique el título de la página asignándole el texto que le llega por

parámetro. Además, creará varios botones que llamen a la función pasándole distintos textos que se colocarán en el título de la página. Véalo a continuación:

```
<script>  
Alert (document.title);  
  
Function cambiaTitulo (texto){  
  
    Document.title=texto  
  
}  
  
</script>  
  
<form>  
  
<input type="Button" value="Titulo 1" onclick="cambiaTitulo('Bienvenidos a mi web')">  
  
<input type="Button" value="Titulo 2" onclick="cambiaTitulo('Gracias por visitar mi web')">  
  
<input type="Button" value="Titulo 3" onclick="cambiaTitulo('Hola a todos')">  
  
</form>
```



TOME NOTA

Recuerde que el título se puede ver en la barra de arriba de la ventana del navegador.

5.3.3. *Métodos de document*

Los métodos más importantes del objeto document y que están relacionados con el manejo del DOM son los siguientes:

- **Write (texto), writeln (texto).** Se utiliza para escribir texto HTML en el documento. El método writeln añade un salto de línea al final del texto que se haya pasado como argumento, solo si la ejecución se realiza entre las etiquetas <pre></pre> de HTML.

- **getElementById (identificador).** Devuelve el elemento del documento cuyo atributo Id coincide con el parámetro identificador. Si no existe el identificador devuelve un valor null.
- **getElementByName (nombre).** Retorna una colección de elementos cuyo atributo Name coincide con el indicado.
- **getElementByTagName (etiqueta).** Devuelve la lista de elementos cuya etiqueta sea la misma que la especificada por el parámetro. Si lo que se quiere es obtener la lista de todos los elementos hay que usar el valor especial asterisco *.
- **getElementByClassName (clase).** Localiza todos los elementos que tengan la clase indicada, es decir, el atributo Class. Es posible indicar más de una clase separadas por un espacio y sin importar el orden, para encontrar los elementos que contengan todas ellas.
- **querySelector (selector).** Permite encontrar el primer elemento que cumpla el selector CSS. Por ejemplo, las clases CSS deben indicarse con un punto delante del nombre .nombreClase y los identificadores con una almohadilla #nombrelId. Si no hay coincidencias, devuelve null.
- **querySelectorAll (selector).** Es similar al anterior, lo que los diferencia es que este devuelve todos los elementos que coinciden con el selector.
- **createElement (etiqueta).** Va a permitir crear nuevos elementos dinámicamente por código. Tendrá que indicar el nombre de la etiqueta que corresponde al elemento que quiere crear, por ejemplo: párrafo (p), capa (div), etc.
- **removeElement (etiqueta).** Permite eliminar elementos por código.

A continuación, va a ver un ejemplo de uso de write(texto) y writeln(texto):

```
<html>
<body>
<script>
    //Escritura de texto plano
    Document.writeln ("<pre> Una línea con 'writeln', hay salto de línea si el texto está entre las
etiquetas 'pre' de HTML </pre>");
    Document.write ("texto con write sin salto de línea");
    Document.write ("Otro texto con write sin salto de línea");
    // Si quiere escribir líneas de HTML bastará con usar document.write
    Document.write ("<hr>");
    Document.write ('<div id= "contenedor" style = "width:200px; height:400px; margin: auto;
padding: 4px; background: #fafafa; border: 2px solid black">');

```

```
Document.write (“<h2 style = ‘border-bottom: 2px solid black’> Este es el título </h2>”);

Document.write (“<h3> Un subtítulo </h3>”);

Document.write (“<p> Y comienza con su contenido.<br> Añadir un párrafo con el texto que
quiera añadir en el mismo. <br><b> Esto es solo un ejemplo</b></p>”);

Document.write(“</div>”);

</script>

</body>

</html>
```

Como ha podido ver, con JavaScript podría construir una página completa sin tocar siquiera el HTML. Tan solo necesitaría la estructura principal del documento HTML y el resto lo programaría con JavaScript. Vea a continuación un ejemplo de uso de getElementById (identificador):

```
<html>

<body>

<p id= “texto1”>Soy el primer párrafo</p>

<p id=”texto 2”>Soy el segundo párrafo</p>

<script>

//Se busca por identificador

Var elemento = document.getElementById(“texto 2”);

//Se comprueba si existe el elemento

If(elemento){

    Document.write (‘El párrafo “texto 2” existe<br’);

    Console.log (‘El párrafo “texto 2” existe’) //herramienta de desarrollo

}else{

    Document.write (‘El párrafo “texto 2” no existe<br’);

    Console.log (‘El párrafo “texto 2” no existe’) //herramienta de desarrollo
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

Ahora, vea un ejemplo de cómo usar getElementByName (nombre):

```
<html>
```

```
<body>
```

```
    <p name="coche">Toyota</p>
```

```
    <p name="animal">Caballo</p>
```

```
    <p name="coche">Seat</p>
```

```
    <script>
```

```
        //Se busca por nombre
```

```
        Var elemento = document.getElementByName ("coche");
```

```
        //Se muestra el resultado
```

```
        Document.write ("Hay "+elemento.length+" elementos 'coche'");
```

```
    </script>
```

```
</body>
```

```
</html>
```

Como puede ver, la diferencia entre este ejemplo y el anterior es que el id recoge un solo elemento, ya que, el identificador debe ser único para cada elemento. Sin embargo, cuando busca por nombre busca más de un elemento, ya que, el nombre sirve para nombrar grupos de cosas, por ejemplo, coches, prendas, etc. En definitiva, cosas más generales. Esto ayuda a que, si quiere cambiar el estilo, o quiere utilizar una función para un grupo específico de elementos, no tenga que estar añadiendo elemento por elemento, sino que esos elementos estén dentro de un grupo que se define con un nombre.

Vea a continuación cómo utilizar el método getElementsByTagName (etiqueta):

```
<html>

<body>

<div id="capa1">Esto es una capa HTML</div>

<p id="texto1">Esto es un párrafo</p>

<div id="capa2">Esto es una nueva capa</div>

<script>

//Busca por etiqueta

Var elementos = document.getElementsByTagName ("div");

//Muestra el resultado

Document.write ("Hay "+elementos.length+" elementos 'div'");

</script>

</body>

</html>
```

Este método es muy parecido al anterior, pero se diferencia de los otros dos, que ha visto anteriormente en que, al buscar por etiqueta, va a modificar todos aquellos elementos que estén bajo cierta etiqueta, por ejemplo, si quiere modificar todos los párrafos de su documento, utilice este método.

Por último, va a aprender a utilizar el método getElementsByClassName (clase). Para ello, vea el siguiente ejemplo:

```
<html>

<head>

<style>

.rojo {

    Color: red;

}
```

```
.azul {  
    color:blue;  
}  
  
.verde {  
    color: green;  
}  
  
</style>  
  
</head>  
  
<body>  
  
    <p class="color rojo">Párrafo rojo</p>  
  
    <p class= "color azul"> Párrafo azul</p>  
  
    <p class= "color verde"> Párrafo verde</p>  
  
<script>  
  
    //busca por una clase  
  
    Var elementos = document.getElementsByClassName("color");  
  
    //Se muestran valores  
  
    Document.write("Hay "+elementos.length+" elementos con clase 'color'  
    <br>");  
  
    //Se busca por varias clases  
  
    Elementos = document.getElementsByClassName ("azul color");  
  
    //Muestra valores  
  
    Document.write("Hay "+elementos.length+" elementos con clase 'color' y elementos con  
    clase 'azul'");  
  
</script>  
  
</body>
```

</html>

Como ha podido ver en este ejemplo, si busca elementos por nombre de clase, puede buscar más de una clase a la vez y, si le aplica un proceso a estas clases, tomaría todos los elementos que hay en ellas. Esto es lo que diferencia a este método de los que ha visto anteriormente.

5.3.4. Flujo de escritura del documento

Acerca del objeto document, también, es interesante hablar un poco sobre el flujo de escritura del documento web.

Cuando el navegador lee una página web, va interpretándola y escribiendo en su ventana. Al proceso en el que el navegador está escribiendo la página, lo Se denomina flujo de escritura del documento. El flujo comienza cuando se empieza a escribir la página y dura hasta que esta ha terminado de escribirse. Una vez terminada la escritura de la página, el flujo de escritura del documento se cierra automáticamente. Es aquí cuando se dice que se ha terminado de cargar la página.

Una vez cerrado el flujo, no se puede escribir en la web ni texto, ni imágenes, ni otros elementos.

En JavaScript tiene que respetar el flujo de escritura del documento forzosamente. Es por ello, que solo puede ejecutar sentencias `document.write()` cuando está abierto el flujo de escritura del documento, o lo que es lo mismo, mientras se está cargando la página.

Si recuerda las dos formas de ejecutar un script en JavaScript:

- Ejecución de los scripts mientras que carga la página. Aquí puede ejecutar `document.write()` y lo ha hecho habitualmente en los ejemplos anteriores.
- Ejecución de los scripts cuando la página ha sido cargada como respuesta a un evento del usuario. Aquí, no puede ejecutar un `document.write()` porque la página ha terminado de cargarse, de hecho, no lo ha hecho nunca durante el curso.

Hay un matiz que dar a que no se puede escribir en la página cuando ya está cerrado el flujo. En realidad, sí que se puede, pero necesita abrir el flujo otra vez para escribir en la página, tanto es así que, aunque no lo abra explícitamente, JavaScript se encargará de ello. Lo que tiene que quedar claro es que si hace un `document.write()` el flujo tiene que estar abierto si no lo está, se abrirá. El problema de abrir el flujo de escritura del documento una vez ya está escrita la página es que se borra todo su contenido.

Para que quede claro, va a hacer un *script* para escribir en la página una vez que se haya cargado. Simplemente, necesita un botón que al pulsarlo se ejecute el método `write()` del objeto `document`.

Vea un ejemplo:

<form>

```
<input type = button value = escribir onclick ="window.document.write('hola')">  
</form>
```

Si se fija, en el manejador de eventos onclick, ha colocado la jerarquía de objetos desde el principio, es decir, empezando por el objeto window. Esto es porque hay algunos navegadores que no sobreentienden el objeto window en las sentencias escritas en los manejadores de eventos.

El resultado de la ejecución puede variar de un navegador a otro, pero en cualquier caso, se borrará la página que se está ejecutando.



TOME NOTA

El flujo de escritura del documento es el proceso en el que el navegador escribe la página. Para escribir en la página desde JavaScript, el flujo del documento debe estar abierto.

5.3.5. Métodos open () y close () de document

Los métodos open () y close () del objeto document sirven para controlar el flujo del documento desde JavaScript. Permiten abrir y cerrar el documento explícitamente-

El ejemplo de escritura del epígrafe anterior se debería haber escrito con su correspondiente apertura y cierre del documento. Vea un ejemplo:

```
<script>  
  
Function escribe (){  
  
    Document.open()  
  
    Window.document.write('Hola')  
  
    Document.close()  
  
}  
  
</script>  
  
<form>  
  
<input type = button value = escribir onclick="escribe()">
```

</form>

Como puede ver, ahora, no se escriben las sentencias dentro del manejador porque cuando hay más de una sentencia, queda más claro poner una llamada a una función y en la función se colocan las sentencias que quiera.

5.4. Propiedades y métodos de los objetos del formulario

Toda creación de formulario necesita la inclusión en la página HTML de un objeto llamado Form, que contiene los campos de texto, las listas desplegables y otros controles como los botones de opción o las casillas de verificación.

El objeto Form es un subobjeto del objeto document en la jerarquía de los objetos JavaScript. Dispone de propiedades y de métodos que permiten manipularlo o asignarle acciones específicas como, por ejemplo, la eliminación del contenido de todos los campos del formulario, el envío por correo electrónico de la información escrita, entre otros.

En los epígrafes que verá a continuación, va a detallar algunas de las propiedades y métodos principales para la manipulación del objeto Form.

5.4.1. Propiedades principales del objeto form (name, action, method, target)

A continuación, vea las principales propiedades del objeto formulario:

- **Action.** Es la acción que quiere realizar cuando se envía un formulario. Se coloca generalmente una dirección de correo o la URL a la que le manda los datos. Corresponde con el atributo action del formulario
- **Encoding.** Define el tipo de codificación de los datos que se emplearán en el momento de enviar el formulario, por ejemplo, text/plain permite indicar que los datos de enviarán en formato de texto.
- **Length.** Corresponde al número de formularios en la página.
- **Method.** Especifica el modo en que los datos son enviados, ya sea mediante URL (post) o en variables ocultas (get).
- **Name.** Corresponde al nombre del formulario.
- **Target.** Indica la ventana meta de un conjunto de cuadros activa después del envío del formulario.
- **Elements array.** La matriz de elementos que contiene cada uno de los campos del formulario.

Con estas propiedades puede cambiar dinámicamente, con JavaScript, los valores de los atributos del formulario para hacer con él lo que se deseé dependiendo de las exigencias del momento.

Por ejemplo, podría cambiar la URL que recibiría la información del formulario con la instrucción. Vea el siguiente ejemplo:

```
Document.miFormulario.action = "miPágina.asp"
```

O cambiar el target para enviar un formulario en una posible ventana secundaria llamada miVentana. Lo puede ver en el siguiente ejemplo:

```
Document.miFormulario.target = "miVentana"
```

5.4.2. Métodos del objeto form (submit, reset, get, post)

En este epígrafe va a ver los principales métodos que puede invocar del objeto formulario:

- **Submit ()**. Este método se utiliza para hacer que el formulario se envíe, aunque no se haya pulsado el botón de enviar.
- **Reset ()**. Este método se utiliza para reinicializar todos los campos del formulario, como si se hubiese pulsado el botón reset.

Va a ver un ejemplo muy sencillo sobre cómo validar un formulario para enviarlo en caso de que esté lleno. Para ello, va a utilizar el método submit () del formulario.

El mecanismo es el siguiente: en vez de colocar un botón de submit, se coloca un botón normal, es decir, un <input type="button"> y hace que al pulsar ese botón se llame a una función que es la encargada de validar el formulario y, en caso de que esté correcto, enviarlo.

Vea un ejemplo:

```
<form name="miFormulario" action="correo@alqueseenvia.com" enctype="text/palin">

<input type="Text" name="campo1" value="" size="12">

<input type="button" value="Enviar" onclick="validaSubmite()">

</form>
```

No hay botón de submit, sino un botón normal con una llamada a una función que puede ver a continuación:

```
Function validaSubmite(){

    If (document.miFormulario.vampo1.value == "")

        Alert ("Debe llenar el formulario")

    Else
```

```
Document.miFormulario.submit ()  
}
```

En la función se comprueba si lo que hay escrito en el formulario es un string vacío. Si es así, se muestra un mensaje de alerta que informa que se debe llenar el formulario. En caso de que haya algo en el campo de texto envía el formulario utilizando el método submit del objeto form.

A continuación, va a ver cómo utilizar el método post y get. La diferencia entre los métodos get y post radica en la forma de enviar los datos a la página cuando se utiliza el botón Enviar. Mientras que el método get envía los datos usando la URL, el método post los envía de forma que no puede verlos, es decir, envía los datos en segundo plano u ocultos al usuario. Un ejemplo de cómo enviaría el método get los datos es el siguiente:

www.paginaweb.com/action.php?nombre=maria&apellidos1=gomez

Es por ello, que es muy importante saber utilizar bien ambos métodos para no poner al descubierto información comprometida.

Primeramente, va a aprender a utilizar el método post. Lo más cómodo es que, si quiere enviar datos por post, cree el formulario con los datos que deseé enviar. El formulario tendrá el atributo action dirigido a la página a la que quiere enviar los datos y el método de envío post. Vea el siguiente ejemplo:

```
<form action="pagina_destino.php" method=post name="formulario1">  
  
<input type="hidden" name="campo1" value="valor">  
  
<input type="hidden" name="campo2" value="otroValor">  
  
</form>
```

Como quiere que no se vea el formulario, solo quiere enviar sus datos por post, todos los campos del formulario son hidden, es decir, ocultos.

Para crear una función para enviar el formulario, lo siguiente que debe hacer es crear una función JavaScript que se ejecuta para enviar el formulario. Hace uso del método submit () asociado a los formularios. Vea el siguiente ejemplo:

```
<script>  
  
Function enviarFormulario () {  
  
    Document.formulario1.submit ()  
  
}
```

```
</script>
```

Si se fija, la función tiene una única sentencia que envía el formulario. Para ello, se accede primero al formulario por el nombre que le ha dado en el atributo name de la etiqueta <form> en el código HTML. El nombre del formulario era formulario1. Así que esa instrucción hace un submit () del formulario1, que a su vez es una propiedad del objeto document de la página.

Por otra parte, si quiere enviar datos post al hacer clic en un enlace, por ejemplo, a través del botón enviar, simplemente debe construir un enlace que llame a la función JavaScript que ha creado anteriormente. Vea el ejemplo:

```
<a href="javascript:enviarFormulario()">Enviar formulario</a>
```

Como ha podido ver, crear este enlace es muy sencillo, simplemente se indica con javascript: que se debe ejecutar un código JavaScript al pulsar el enlace. El código JavaScript es una simple llamada a la función enviarFormulario (). Para finalizar, vea el código completo terminado:

```
<html>
<head>
    <title> Enviar formulario al pulsar un enlace</title>
<script>
    Function enviarFormulario (){
        Document.formulario1.submit ()
    }
</script>
</head>
<body>
<form action="pagina_destino.php" method=post name="formulario1">
    <input type="hidden" name="campo1" value="valor">
    <input type="hidden" name="campo2" value="otroValor">
</form>
```

```
<a href="javascript:enviarFormulario()">Enviar formulario</a>  
</body>  
</html>
```

El paso de información de una página a otra puede hacerse de varias formas, ya ha visto cómo hacerlo con un formulario mediante el método post, y ahora va a hacerlo con un formulario usando get.

El uso de get en formularios, salvo que el desarrollador sepa muy bien lo que hace y los datos no sean comprometidos, no debe usarse para otras cosas diferentes a los formularios de búsqueda. El motivo es sencillo, el método get lo que hace es pasar variables y sus valores por la URL, es decir, no solo queda a la vista de cualquier usuario, sino que además la información puede quedar guardada en el historial del navegador. Imagine que se usa este método para identificar usuarios, y el nombre de usuario y la clave se queda guardada en el historial del navegador de un ordenador; por ejemplo, una biblioteca pública, el fallo en la seguridad sería catastrófico.

Va a ver un ejemplo del método get en una calculadora. Para este caso no sería un problema utilizar el método get, ya que unos valores numéricos que quiere sumar no tienen mucha importancia que aparezcan en la URL.

El método get, además de las diferencias que ha visto que tiene con post, tiene diferencias al recargar la página. Si se pasan las variables por URL, usando get, cuando se pulsa el botón volver atrás o se actualiza la página, se hace sin más problemas. No obstante, cuando se hace esto con el método post, sale un mensaje que dice algo así como: para recargar la página debe enviar la información que ya envió antes, y debe recargar la página de nuevo para ver el contenido. Otra diferencia entre get y post es que el primero no permite enviar más de 2048 caracteres.

A continuación, verá el ejemplo del uso de get en una calculadora:

```
<html>  
<head>  
<title>Formulario</title>  
</head>  
  
<body>  
  
<form action="sumar.php" method="get">  
  
<p>Valor 1: <input type="text" name="T1" size="20"></p>  
  
<p>Valor 2: <input type="text" name="T2" size="20"></p>
```

```
<p>Valor 3: <input type="text" name="T3" size="20"></p>

<p><input type="submit" name="B1" value="Sumar"></p>

</form>

</body>

</html>
```

En la siguiente imagen puede ver cómo get pasa los datos a través de la URL y son totalmente visibles para el usuario.



5.5. Propiedades y métodos de los objetos del lenguaje

Todos los objetos predefinidos por JavaScript tienen sus propios métodos. Los métodos, como ya ha visto anteriormente, sirven para hacer acciones como una ventana emergente, modificar las propiedades de un objeto, añadir elementos a una lista, quitarlos, transformar un string en un número entero o decimal, entre otros.

Los strings, los números, los arrays y las variables de tipo fecha son en realidad objetos, aunque los haya estado usando sin tenerlo en cuenta. Por lo tanto, también tienen métodos.

Además, puede crear sus objetos y sus métodos. Sin embargo, JavaScript es un lenguaje basado en el uso de objetos y no en la creación de los mismos. Vea algunos métodos de los objetos de predefinidos de JavaScript.

- **Métodos numéricos.** Los métodos numéricos más importantes son parseInt() y parseFloat().

parseInt () transforma un string en número entero. En caso de ser un texto como por ejemplo 5Kg, devolvería el número 5.

Por otro lado, parseFloat () hace lo mismo que parseInt () pero en este caso con un número decimal.

- **Métodos para strings.** Los métodos string que son indispensables conocer son indexOf (), lastIndexOf (), split (), slice() y substr ().

El método indexOf () devuelve el primer elemento de un string que sea igual al valor indicado dentro del paréntesis. En caso de no encontrarlo devuelve -1.

El método lastIndexOf () hace lo mismo que indexOf () pero empezando por el final de un string.

El método Split () crear una lista de elementos a partir de un string separando los elementos por el carácter indicado dentro del paréntesis.

Por último, el método slice (x, y) devuelve los y caracteres a partir de la posición x. Hay que tener en cuenta que se empieza a contar que el primer carácter siempre ocupa la posición 0.

- **Métodos para arrays.** Los métodos importantes de conocer para los arrays son length, join (), pop (), sort (), slice ().

Length es el método más importante de todos. Devuelve el número de elementos de una lista. Es muy útil para devolver el último elemento de una lista o añadir un nuevo elemento en la última posición. Hay que recordar que el primer elemento de una lista es 0 y que, por tanto, para acceder al último elemento de la lista tiene que acceder a length-1. join () junta todos los elementos de una lista separados por el carácter que indique dentro del paréntesis que escribe entre comillas, ya que, será un string.

Pop () elimina el último elemento de una lista y hace que la lista quede disminuida su longitud en una posición.

Sort () ordena alfabéticamente o numéricamente los elementos de una lista

Slice () tiene el mismo funcionamiento que el slice () de los strings pero aplicado a listas.

En definitiva, existen muchos más métodos, pero ha visto los más indispensables de estos grupos básicos.

5.5.1. Document (escribir texto, color fuente, color fondo, obtener elementos del documento actual HTML, título de página)

El objeto document en JavaScript tiene propiedades específicas para dar formato a su documento. Como ya sabe, con el objeto document se controla la página web y todos los elementos que contiene. El objeto document es la página actual que es está visualizando en ese momento. Depende del objeto window, pero también puede depender del objeto frame en caso de que la página se esté mostrando en un frame.

A continuación, va a ver las propiedades del objeto document que permiten cambiar la estética del documento:

- **Escribir texto.** Para escribir texto se utiliza Document.write y document.writeln, ambos permiten sacar información por el documento, información que se le pasa por parámetro. Por otro lado, lo que diferencia write de writeln es que writeln termina con un salto de página, como ya se ha visto anteriormente. Vea un ejemplo:

```
Document.write("Hola, esta es mi web");
Document.writeln ("Bienvenidos a mi web");
```

- **Color de fuente.** Puede modificar el color por defecto en el que se muestra el texto de una página de manera dinámica usando la propiedad document.fgColor. El valor asignado a la propiedad es un nombre de color o un código RGB, es decir, los tres pares de dígitos hexadecimales que especifican los valores del rojo, verde y azul que forman el color. En ambos casos el valor se le asigna como un string, entre comillas. Vea el siguiente ejemplo:

```
<script>  
Document.fgColor = "green";  
</script>
```

- **Color de fondo.** Puede modificar el color de fondo del documento utilizando la propiedad document.bgColor. El valor asignado a la propiedad deberá ser el nombre de un color o, en su caso, el código de color en RGB, es decir, en hexadecimal. Se utiliza de la misma manera que document.fgColor. Cabe destacar que esta propiedad está obsoleta, por lo que puede que haya navegadores que no la soporten y los que la soporten, terminarán no haciéndolo.

Para ello, hoy por hoy se utiliza CSS de la siguiente manera document.body.style.backgroundColor. Vea un ejemplo de bgColor:

```
<script>  
Document.bgColor ="darkblue";  
</script>
```

Para utilizar el nuevo estándar lo hará de la misma manera. Vea un ejemplo:

```
<script>  
  
Document.body.style.backgroundColor = "darkblue";  
  
</script>
```



TOME NOTA

Evite utilizar document.bgColor, ya que, está desactualizado y puede que haya navegadores que no soporten esta opción, y los navegadores que la soportan, terminarán sin soportarla. Para ello, utilice siempre que sea posible document.body.style.backgroundColor.

Obtener elementos del documento actual HTML. Para obtener elementos del documento HTML puede usar varios métodos, entre ellos tiene:

- **getElementById (“idElemento”).** Este elemento permite tomar un elemento del documento HTML por su id. El id de su elemento ha de ser único para ese elemento. Si el id del elemento no se encuentra, el método devolverá un valor null. Vea un ejemplo:

```
<html>
```

```
<head>

</head>

<body>

<p id="parrafo1">Aquí va su primer párrafo</p>

<script>

Function cambiarColor (nuevoColor) {

    Var elemento = document.getElementById('parrafo1');

    Elemento.style.color = nuevoColor;

}

</script>

</body>
```

Como puede ver en este ejemplo, ha creado una función que solo cambiará el color al párrafo 1, por lo que, si escribe más párrafos, mantendrán su color por defecto.

- **getElementsByName (“etiquetaElemento”)**. Este método es uno de los más utilizados para manipular el árbol DOM con JavaScript y como su nombre lo indica, getElementsByName permite selecciona una lista de nodos cuyo elemento es el especificado en el parámetro. A cada uno de los nodos se le asigna un índice de acuerdo con el orden en el que aparecen en el html del documento. Vea dos ejemplos:

```
document.getElementByTagName ('elemento');

document.getElementByTagName ('elemento') [índiceDelElemento]
```

Con la primera línea de código obtiene todos los elementos de un tipo, mientras que, con la segunda línea, elege uno de ellos en concreto. Por ejemplo:

```
Document.getElementsByTagName ('p');
```

Devuelve una lista con todos los párrafos del documento, mientras que:

```
Document.getElementsByTagName ('p') [3];
```

Devuelve el cuarto párrafo. Recuerde que la lista de elementos empieza en 0, por eso si quiere el cuarto párrafo, debe escribir un 3 y no un 4.

Algo que hay que recordar es que para seleccionar un elemento en concreto por medio de este método es necesario indicar el índice, incluso cuando solo existe un elemento de su tipo. Por ejemplo, todos sabe que un documento solo puede tener un body, pero no puede seleccionarlo por medio de `document.getElementsByTagName ('body')`, puesto que esto devuelve una lista de nodos, aunque solo contenga un único nodo. La forma correcta de seleccionarlo sería `document.getElementByTagName ('body') [0]`.

Por último, indicar que se pueden combinar los métodos vistos para seleccionar una lista de elementos dentro de un elemento concreto. Vea un ejemplo:

```
Document.getElementById ('parrafo1').getElementsByTagName ('li');
```

```
Document.getElementsByTagName ('div') [1].getElementsByTagName ('li');
```

La primera línea devolvería una lista con ítems de lista de documento, pero restringida a aquellos descendientes dentro del elemento con `id="parrafo1"`. La segunda, devolvería la lista de los li descendientes del segundo div del documento.

getElementsByName (“nombreElemento”). Este método devuelve un grupo de todos los elementos que hay en el documento que tenga el nombre especificado. Como pasa con el anterior elemento, también puede acceder a un elemento determinado añadiéndole el número del índice. Cabe destacar que este método está obsoleto en HTML 5 y que, en vez de usar este método, se usa directamente el método `getElementById`.

getElementsByClassName (“claseElemento”). Por último, este método retorna un objeto similar a un array de los elementos hijos que tengan todos los nombres de clase indicados. Cuando es llamado sobre el objeto `document` la búsqueda se realiza en todo el documento, incluido el nodo raíz. También se puede llamar a `getElementsByClassName ()` sobre cualquier elemento, en este caso devolverá solo los elementos hijos del elemento raíz indicado que contengan los nombres de clase indicados. La sintaxis de este método es la siguiente:

```
var elementos = document.getElementsByClassName (nombres);
```

```
var elementos = elementoRaiz.getElementsByClassName (nombres);
```

La variable `elementos` es una colección HTML de los elementos encontrados. Por otro lado, `nombres` es un string que representa la lista de nombres de clase a buscar. Los nombres de clase se separan por un espacio. `getElementsByClassName` se puede llamar sobre cualquier elemento, no solo sobre `document`. El elemento sobre el que se llama será usado como la raíz de la búsqueda. Vea algunos ejemplos de uso de este método:

Va a obtener todos los elementos de la clase coche. Para ello, escribe lo siguiente:

```
Document.getElementsByClassName ('coche');
```

Para obtener todos los elementos al mismo tiempo de dos clases, de la siguiente manera:

```
Document.getElementsByClassName ('coche moto');
```

Por último, para obtener todos los elementos que tengan la clase coche y que estén dentro de un elemento con id masRapidos, escribe lo siguiente:

```
Document.getElementById ('masRapidos').getElementsByClassName ('coche');
```

5.5.2. Windows (*open*)

En este epígrafe va a hablar de popups, o ventanas secundarias o emergentes. Es algo que se consigue desde JavaScript con el método `window.open ()`. Es una práctica habitual, pero de la que no debe abusar, porque este tipo de comportamiento se considera habitualmente como intrusivo para los usuarios.

En determinadas ocasiones es muy útil abrir un enlace en una ventana secundaria, es decir, una ventana aparte que se abre para mostrar una información específica, o para realizar algún tipo de diálogo con el usuario. Por ejemplo, una situación común es abrir la típica ventana para compartir un post en redes sociales.

Algunas ventajas de abrir un enlace en una ventana secundaria pueden ser que el usuario no se marche de la página donde estaba el enlace, que la ventana secundaria se pueda configurar libremente, por lo que se pueden hacer ventanas más grandes o pequeñas posicionadas en un lugar determinado y con más o menos menús de navegador. Y, en general, que el grado de control de la ventana secundaria utilizando JavaScript aumenta.

Para abrir una ventana secundaria puede hacerlo de dos maneras, con HTML y con JavaScript. Vea cada una de ellas:

- **Abrir una ventana con HTML.** Se puede abrir una ventana secundaria muy fácilmente con tan solo HTML. Para ello, puede utilizar el atributo `target` de las etiquetas `href`. Si pone `target="_blank"` en el enlace, la página se abrirá en una ventana secundaria. También, puede poner `target="xxx"` para que el enlace se presente en la ventana llamada `xxx` o en el frame `xxx`. El enlace quedaría de la siguiente manera:

```
<a href="mipagina.html" target="_blank">Abre el enlace en una nueva ventana</a>
```

El problema de abrir una página secundaria con HTML consiste en que no puede definir la forma de esta ni puede ejercer mayor control sobre ella, tal como se comentaba entre las ventajas de abrir una ventana secundaria con JavaScript. La ventana que se abre siempre será como el usuario tenga definido por defecto en su navegador.

Abrir una ventana secundaria con JavaScript. Para abrir una ventana con JavaScript, puede utilizar la instrucción `window.open ()`. Vea paso por paso cómo abrir una ventana secundaria utilizando JavaScript.

Primeramente, se escribe una sentencia JavaScript, la sentencia es simplemente la función `window.open ()`. Lo más complicado es saber cómo utilizar esta función, pero ahora verá que no tiene complicación alguna.

El método `window.open ()` recibe tres parámetros que se colocan dentro de los paréntesis. Véalo a continuación:

```
Window.open (URL, nombreDeLaVentana, formaDeLaVentana)
```

Va a analizar rápidamente cada uno de los parámetros por separado:

- **URL.** Representa la URL que desea abrir en la ventana secundaria, por ejemplo, <https://google.es>. Se puede abrir las URLs de las páginas que estén en su propio dominio o en dominios externos.
- **nombreDeLaVentana.** Es el nombre que se le asigna a esta ventana para dirigir enlaces con el atributo `target` del HTML.
- **formaDeLaVentana.** Se indica el aspecto que va a tener la ventana secundaria. Por ejemplo, se puede definir su altura, anchura, si tiene barras de desplazamiento, etc.

A continuación, vea un ejemplo de sentencia JavaScript completa para abrir una ventana secundaria:

```
Window.open (https://google.es, "ventana1", "width=120, height=300, scrollbars=NO")
```

Esta sentencia abrirá la página inicial de Google.es en una ventana secundaria a la que va a llamar `ventana1`. Además, la ventana será de 120 píxeles de ancho, 300 de alto y no tendrá barras de desplazamiento.

Cabe destacar que si después de abrir esa ventana coloca otro enlace en la página que abría la ventana cuyo atributo `target` está dirigido hacia el `nombreDeLaVentana`, en este caso `ventana1`, este enlace se mostrará en la ventana secundaria.

Función que abre una ventana. Lo más cómodo para abrir una ventana es colocar una función JavaScript que se encargue de las tareas de abrirla y que reciba por parámetro la URL que se desea abrir. Vea este sencillo script a continuación:

```
<script>
```

```
Function ventanaSecundaria (URL){
```

```
    Window.open (URL, "ventana1", "width=120, height=300, scrollbars=NO")
```

}

</script>

- **Colocación de un enlace.** Este enlace no debe estar dirigido directamente a la página que quiera abrir, sino a la sentencia JavaScript necesario para abrir la ventana secundaria. Para ejecutar una sentencia JavaScript con la pulsación de un enlace lo hará de la siguiente manera:

- **El enlace llama a la función que abre la ventana.** Ahora, vea cómo quedaría todo ese enlace en la página:

 Pinche en este enlace para abrir la ventana secundaria

En la página que vaya a colocar este enlace, debe tener el *script* que ha hecho anteriormente el cual contenía la función para abrir la ventana. Además, hay que fijarse que las comillas simples se colocan para definir la URL que se pasa como parámetro de la función ventanaSecundaria (). Son comillas simples porque el href del enlace ya tiene unas comillas dobles, y dentro de comillas dobles siempre se deben utilizar comillas simples.

A continuación, va a ver algunos parámetros para dar forma a una ventana emergente. Estos atributos se pueden utilizar en la función window.open () para definir la forma que deseé que tenga la ventana secundaria. Algunos parámetros son:

- **Width.** Ajusta el ancho de la ventana en píxeles.
- **Height.** Ajusta el alto de la ventana.
- **Top.** Indica la posición de la ventana. En concreto es la distancia en píxeles que existe entre el borde superior de la pantalla y el borde superior de la ventana.
- **Left.** Indica la posición de la ventana. En concreto, es la distancia en píxeles que existe entre el borde izquierdo de la pantalla y el borde izquierdo de la ventana.
- **Scrollbars.** Sirve para definir de forma exacta si salen o no las barras de desplazamiento. Scrollbars = no hace que nunca salgan. Scrollbars = yes hace que salgan.
- **Resizable.** Establece si se puede o no modificar el tamaño de la ventana. Con resizable = yes se puede modificar el tamaño y con resizable = no se consigue un tamaño fijo.

A partir de aquí, se enumeran otra serie de propiedades que sirven para mostrar, o no, un elemento de la barra de navegación que tienen los navegadores más populares, como podría ser la barra de menús o la barra de estado. Cuando pone al atributo =yes, está forzando a que ese elemento se vea. Cuando se pone el atributo = no, lo que hace es evitar que ese elemento se vea. Algunos de estos atributos son: **Directories**, o barra de directorios; **Location**, o barra de direcciones; **Menubar**, o barra de menús; **Status**, o barra de estado; **Titlebar**, o barra de título; y **Toolbar**, o barra de herramientas.

Por último, va a aprender a abrir una ventana sin un enlace, ya que, en ocasiones debe programar que se abra una ventana secundaria automáticamente, es decir, sin necesidad de que el usuario pulse sobre ningún enlace. En este caso, el código de la función ventanaSecundaria sirve también y habrá que añadir una línea de código JavaScript a continuación de la función ventanaSecundaria. Esta línea a añadir simplemente será una llamada a la función que se ejecutará según se está cargando la página. Vea como quedaría el código:

```
<script>

Function ventanaSecundaria (URL){

    Window.open(URL, "ventana1", "width=120, height=300, scrollbars=no")

}

ventanaSecundaria ("https://google.es");

</script>
```

La llamada a la función ventanaSecundaria se ejecuta según se está cargando la página, ya que, está fuera de la función.

5.5.3. History (go)

Dentro del objeto history, puede encontrarse, entre otros, con el método go. En concreto, la sintaxis para usar este método es history.go (posición) donde el parámetro posición indica la posición del historial al que se accede, normalmente este parámetro es de tipo entero, aunque puede ser también de tipo string.

Vea un ejemplo de uso de este método. Para ello, se abre un navegador web con la página de Google.es, luego, se visita la página Gmail.com y, a continuación, la de es.wikipedia.org. Así que, se crea a continuación el siguiente documento HTML:

```
<html>

<head>

<script>

Function retroceder (){

    Window.history.go (-2);

}

</script>
```

```
</head>

<body>

<input type="button" value="Retroceder 2 páginas" onclick="retroceder ()">

</body>

</html>
```

Si pulsa el botón que ha creado, llevará a la página de Gmail.com, ya que, al pulsar el botón retrocede dos páginas de las que se hayan guardado en el historial. Es decir, el método history.go () llevará a aquella posición que le indique en el historial.

5.5.4. Location (servidor)

La propiedad window.location () devuelve un objeto de tipo location con información relacionada con la URL del documento actual. Este objeto, tal y como puede ver en la jerarquía de objetos, es parte del objeto window y se accede a través de la propiedad window.location. No todos los navegadores soportan este objeto.

A continuación, va a ver las propiedades y métodos de este objeto. Las propiedades son las siguientes:

- **Href.** Devuelve un string con la URL completa. Window.location es equivalente a document.location, pero se recomienda usar window.location. Vea el siguiente ejemplo:
 - Alert ('URL actual: '+window.location.href);
 - Alert ('URL actual: '+location.href);
- **Protocol.** Devuelve el protocolo de la URL, normalmente, http o https. Si trabaja en local será file: Vea un ejemplo:
 - Alert ('protocolo actual: '+window.location.protocol);
- **Host.** Devuelve el nombre del servidor, normalmente como nombre de dominio, y si existe un puerto especificado también lo devuelve separado con :. Vea el siguiente ejemplo:
 - Alert ('El host actual es: '+window.location.host);
- **Hostname.** Devuelve el dominio o servidor. Vea un ejemplo:
 - Alert ('El hostname actual es: '+window.location.hostname);
- **Pathname.** Devuelve la ruta excluyendo el nombre de dominio o servidor. Por ejemplo:
 - Alert ('El pathname actual es: '+window.location.pathname);
- **Search.** Devuelve la cadena que representa los parámetros que existen en la URL. Ejemplo:
 - Alert ('Los parámetros en [URL: '+window.location.search\); En este ejemplo devolvería, por ejemplo, URL?name=jose&apellido1=gomez](#)
- **Hash.** Devuelve # seguido de un texto si en la URL existe un # seguido de un texto, es decir, un ancla de localización del fragmento html.
- **Origin.** Devuelve una cadena que representa la raíz del sitio web.

- A continuación, va a citar algunos métodos de los objetos location. Se debe tener en cuenta que algunos de los métodos no son soportados por algunos navegadores:
- **Assign (URLDestino).** Asigna una URL a la ventana actual, lo que genera que se cargue la URLDestino. El usuario puede volver atrás con el botón back del navegador.
- **Reload (opcion).** Da lugar a la recarga de la URL actual. Si se indica como parámetro true, la recarga se hará desde el servidor. Si no se especifica o es false, la recarga puede hacerse desde la caché.
- **Replace (URLDestino).** Asigna una URL a la ventana actual, lo que genera que se cargue la URLDestino. El usuario no puede volver atrás con el botón back del navegador.
- **toString().** Devuelve una cadena representativa de la URL. Window.location.toString() devuelve lo mismo que window.location.href.

Vea un ejemplo de lo aprendido. Suponga que tiene un botón que al ser presionado lleve al periódico Diario Sur, pero primero muestre una ventana de confirmación de si quiere, o no, ingresar al periódico. En caso de que el usuario presione el botón afirmativo, se redirecciona al sitio web de Diario Sur, en caso contrario se muestra un mensaje:

```
<!DOCTYPE html>

<html>

<head>

<title>Ejemplo con location</title>

</head>

<body>

<script>

Function verificarEntrada(){

    If(window.confirm('¿Desea salir del sitio e ingresar al periódico Diario Sur?'))

    {

        Window.location='https://diariosur.es';

    }

    Else{

        Window.alert('Usted no será redirigido al periódico');

    }

}
```

```
}
```

```
</script>
```

```
<button onclick="verificarEntrada()">Ingresar al periódico Diario Sur</button>
```

```
</body>
```

```
</html>
```

5.5.5. *Navigator (nombre, versión y detalles del navegador)*

El objeto navigator permite conocer datos como el navegador que se está utilizando, su versión, sistema operativo del usuario, etc. A continuación, vea sus propiedades:

- **appCodeName.** Devuelve el código del nombre del navegador.
- **appName.** Devuelve el nombre del navegador.
- **appVersion.** Devuelve la versión del navegador.
- **cookieEnable.** Determina si las cookies están habilitadas o no.
- **Platform.** Devuelve la plataforma sobre la cual está ejecutándose el navegador.
- **userAgent.** Devuelve una información completa sobre el agente de usuario, el cual es normalmente el navegador.
- **Plug-ins.** Propiedad que contiene un array con todos los *plug-ins* soportados por el cliente.

Vea un ejemplo en el que al pulsar un botón muestre la información del navegador:

```
<html>
```

```
<head>
```

```
<script>
```

```
Function informacionNavegador(){
```

```
    Document.write("Nombre de la aplicación: "+navigator.appName+"<br>");
```

```
    Document.write("Versión de la aplicación: "+navigator.appVersion+"<br>");
```

```
    Document.write("Plugins soportados: "+navigator.plugins+"<br>");
```

```
}
```

```
</script>
```

```
</head>
```

```
<body>
    <input type="button" value="Información" del navegador"
onlick="informaciónNavegador()">
</body>
</html>
```

Conzepto Comunicación Creativa

6. LOS EVENTOS DEL LENGUAJE DE GUION

Los eventos son la forma que tiene, como programador en JavaScript, de poder controlar las acciones que hace el usuario sobre su documento, además de poder definir un código programado para cuando se produzcan dichas acciones.

Gracias a los eventos y a su manejo, dota a sus documentos o páginas de interactividad, dado que puede responder a las acciones que hacen los usuarios en su página. Está más que habituados a controlar, normalmente, los eventos asociados a un botón como, por ejemplo, el método onclick. Pero en realidad puede aplicarlos a casi todos los elementos de una página.

JavaScript va a permitir asociar una función a cada uno de los eventos que se produzcan en su documento. De esta forma, cuando se produce un evento, JavaScript lo asocia con la función programada. Este tipo de funciones es lo que comúnmente se conoce por el nombre de manejadores de eventos. Como anécdota, debe saber que todos los eventos comienzan por la palabra on.

6.1. Utilización de eventos

Como ha visto anteriormente, un evento es algo que ocurre y puede ser detectado. Esta definición de partida es quizás demasiado simple. Podría decir que si un usuario hace clic sobre un botón en una página web se produce un evento, pero también que si el usuario estornuda se produce un evento, lo cual realmente no entra en el ámbito de la programación JavaScript, por lo que ha de acotar esta definición.

JavaScript es un lenguaje que puede realizar tareas propias de la programación tradicional como ejecuciones secuenciales de instrucciones o bucles iterativos, pero está especialmente preparado para ser un lenguaje de respuesta dinámica a los eventos que genera el usuario. Por ello, muchas veces se alude a JavaScript como un lenguaje dirigido por eventos o como un tipo de programación basada en eventos. Según este modelo de programación, la aplicación web está a la espera de que el usuario haga algo como, por ejemplo, pulsar un botón y es cuando existe una acción del usuario cuando se ejecuta un fragmento de código. Esto permite que las aplicaciones web sean interactivas en base al ciclo acción del usuario-reacción de la aplicación.

Podría hablar de distintos tipos de eventos:

Eventos interceptables y eventos no interceptables. Un evento interceptable es aquel que puede ser detectado por el sistema operativo o el navegador y generar la ejecución de un fragmento de código JavaScript como consecuencia de ello. Estos son los eventos JavaScript con los que trabaja. Por otro lado, un evento no interceptable sería aquel que no puede ser detectado como, por ejemplo, que el usuario se rasque la cabeza o que reciba un mensaje de correo electrónico en una aplicación local.

Eventos disparados por el usuario y eventos disparados por el sistema. Cada vez que el usuario realiza una acción interceptable como, por ejemplo, hacer clic o pulsar una tecla, dice que se dispara un evento generado por el usuario. Pero, además de eventos disparados por el usuario, existen eventos disparados por el sistema operativo o por el navegador. Por ejemplo, cuando termina la carga de una página web, en el navegador se dispara el evento de sistema onload. También puede incluir instrucciones específicas JavaScript para que se ejecuten cuando tenga lugar un evento de sistema.

6.1.1. Definición de eventos

Cada evento tiene un nombre, al que también se le llama tipo de evento. Por ejemplo, al evento correspondiente a hacer clic, se le identifica como onclick. Normalmente, los nombres se forman anteponiendo el prefijo on con el nombre, en inglés, de lo que ocurre. Así, por ejemplo, onload significa cuando termina la carga del documento HTML.

Cada evento está asociado a un objeto que es quien recibe o detecta el evento. A este objeto se le denomina target u objetivo del evento. Así, el evento onclck del ejemplo anterior tiene como objetivo un nodo HTML de imagen, y un nodo HTML div. El evento onload tiene como target el objeto window u objeto global.

Definir la estructura de un documento HTML es complejo y por ello se creó el DOM para poder describirla y manejarla.

El problema reside en que a lo largo del tiempo se crearon distintos modelos de eventos propuestos por distintas organizaciones. Distintos navegadores adoptaron distintos modelos de eventos, lo cual hacía difícil que el código programado y útil en un navegador, funcionara en otro que usaba un modelo de eventos distinto.

Un modelo de eventos implica como aspecto más importante qué eventos son detectados y cuáles son sus nombres. Pero esta no es la única cuestión relacionada con los eventos. Otra cuestión importante es en qué orden se ejecutan dos eventos que suceden simultáneamente. Por ejemplo, si tiene un evento onclick definido para un div, y otro evento onclick definido para una imagen dentro del div, al hacer clic sobre la imagen se producen dos eventos simultáneamente: el evento onclick correspondiente al div y el evento onclick correspondiente a la imagen. Entonces, ¿qué código de respuesta se debería ejecutar primero, el correspondiente al div o el correspondiente a la imagen? Al orden en que se van disparando los eventos se le denomina propagación de eventos. La propagación de eventos puede ser desde dentro hacia fuera, es decir, primero el evento de la imagen y después el div o desde fuera hacia dentro, primero el div y después la imagen. También, para casos más complejos, según otras formas de establecer el orden.

Cuestiones como esta también están comprendidas en el modelo de eventos. Al existir distintos modelos de eventos, el resultado era, y en cierta medida es, que el orden en que se ejecutaban los eventos dependía del navegador que estuviera utilizando.

Con el paso del tiempo se ha ido produciendo cierta estandarización entre navegadores, aunque esta todavía no es completa. En este curso no va a entrar a estudiar en profundidad ningún modelo de eventos en concreto, solo se hará referencia puntual a ellos cuando se considere necesario. Va a limitar a escribir la forma más estandarizada y aceptada por la industria y la comunidad de programadores para trabajar con eventos.

Pero se debe tener en cuenta lo siguiente:

Distintos navegadores pueden responder de distinta manera a un mismo código.

- Algunos navegadores pueden no reconocer el código que sí es reconocido por otros navegadores.
- Algunos navegadores pueden considerar como eventos algo que otros navegadores simplemente ignoran y no son capaces de capturar. No debe preocuparse por eso, ya que, eso no nos aportará ninguna solución. La estrategia que recomienda es tratar de usar el código más inter-compatible que pueda y sepa, siendo conscientes de que hoy por hoy es imposible conseguir que una página web se muestre exactamente igual y responda exactamente igual en todos los navegadores. Conociendo que no puede cambiar esta situación, su objetivo será crear páginas web cuyo funcionamiento sea correcto en el mayor número de navegadores posible.



Más Info



RECURSO MULTIMEDIA



6.1.2. Acciones asociadas a los eventos

La forma más tradicional, para indicar en el código que una serie de instrucciones deben ejecutarse cuando tiene lugar un evento, sería la consistente en añadir un atributo a la etiqueta HTML que recibe el evento, también llamado captura de eventos en línea. La sintaxis sería la siguiente:

```
<etiqueta nombreEvento="acción a ejecutar">...</etiqueta>
```

Donde la acción a ejecutar es un fragmento de código JavaScript, que en caso de comprender varias sentencias deben ir separadas por punto y coma. Vea el ejemplo siguiente:

```
<a onclick="alert('Cambia la url')" href="https://www.google.es">Esto es Google</a>
```

En este ejemplo puede decir que onclick es el nombre del evento, aunque a veces se prefiere decir que el nombre del evento es click. También puede decir que es un atributo de la etiqueta <a>, y puede decir que es el manejador del evento, es decir, el elemento que dice qué ha de ejecutarse cuando se produzca el evento.

Una posibilidad interesante de este tipo de manejadores de eventos es que puede pasar a una función el elemento HTML que recibe el evento usando la palabra clave this. Vea el ejemplo:

```
<a onclick="llamarFuncion(this)" href="https://www.google.es"> Esto es Google </a>
```

Al invocar a this está pasando el elemento HTML, nodo del DOM <a>, lo cual puede ser de interés en numerosas ocasiones en que interese ejecutar una serie de acciones sobre el elemento que recibe la acción.

Con esta forma de manejo puede plantear una pregunta, si hay una acción predeterminada cuando se pulsa el link, es decir, dirigirse a la url de destino, ¿qué se ejecuta primero, el código de respuesta o la acción predeterminada? Se estandarizó que en primer lugar se ejecuta el *script* de respuesta al evento y luego la acción predeterminada. Pero como en programación todo es posible, se planteó la cuestión de por qué no idear una forma de hacer que la ejecución predeterminada se produzca solo cuando quiera. La solución que se le dio a esto fue permitir que el manejador del evento devolviera un valor booleano true, por defecto, si se debía ejecutar la acción predeterminada, o false para evitar que se ejecutara la acción predeterminada. Vea la sintaxis general y un ejemplo:

```
<etiqueta nombreEvento="acción a ejecutar; return valorBooleano">...</etiqueta>
```

```
<a onclick="alert('link desactivado'); return confirm('Va a acceder a Google.es')"  
href="https://www.google.es"> Esto es Google </a>
```

La ejecución de la acción predeterminada se podía dejar en manos del usuario a través de un mensaje de confirmación usando la función global de JavaScript confirm, de modo que el manejador devuelve true si el usuario pulsa Aceptar o false si el usuario pulsa en Cancelar. Vea el siguiente ejemplo:

```
<a onclick="return confirm('Va a acceder a Google.es ¿Está seguro?')"  
href="https://www.google.es">Esto es Google</a>
```

Algunos eventos, como el cierre de una ventana, pueden generar la ejecución de un código JavaScript, pero no se puede impedir en este caso la acción predeterminada porque sería ir contra los deseos del usuario. Quizás se pudiera impedir, pero no tendría lógica. Esta forma de introducir manejadores de evento, es la más antigua de todas, es reconocida por todos los navegadores, al igual que los cuatro eventos “mágicos”, en el sentido de que cuando se introdujeron fueron un gran avance en el desarrollo web, que son onclick, onload, onmouseover y onmouseout.

Desde la época del navegador NetScape en que se introdujeron los manejadores de eventos básicos y los eventos básicos, hasta hoy en día, ha habido una larga y complicada evolución. Hoy en día puede decir que la captura de eventos en línea, aunque es compatible con todos los navegadores, tiene un serio inconveniente, obligar a mezclar aspectos de comportamiento o respuesta dinámica con aspectos de estructura del documento HTML. Como sabe, hoy día hay consenso en que es ventajoso separar la estructura HTML de la presentación CSS y del comportamiento JavaScript.

 Más Info

RECURSO MULTIMEDIA



6.1.3. Jerarquía de los eventos desde el objeto Windows

Como ya se ha visto en el tema anterior, JavaScript tiene una determinada jerarquía denominada DOM que tiene que cumplir si quiere trabajar con objetos y propiedades predefinidas. Cuando utiliza manejadores de eventos como objeto, debe saber todo lo relativo a eventos está incluido en el objeto window. En este epígrafe va a aprender a utilizar manejadores de eventos como propiedades de objetos.

Puede independizar los manejadores de eventos usando código JavaScript para crearlos manejando la propiedad de nodos HTML del DOM que representa el atributo de la etiqueta HTML. Vea un ejemplo:

```
<!DOCTYPE html>

<html><head><title>Usando manejadores de eventos</title><meta charset="utf-8">
</head>

<body>

<div id="cabecera"><h2>Eventos DOM</h2><h3>Ejemplos JavaScript</h3></div>

<div style="color:blue;" id="pulsador" onclick="ejemplo()">Probar</div>

<a href="https://www.google.es">Google.es</a>
```

```
<a href="https://www.google.com">Google.com</a>

</body>

</html>

<script>

Var elemsA=document.getElementsByTagName('a');

For (var i=0; i<elemsA.length; i++){

elemsA[i].onclick= function(){return confirm ('Va a ser transferido hacia' +this.href);}

}

</script>
```

Sobre este ejemplo, debe repasar algunas cuestiones:

- **El código JavaScript está incluido detrás del código HTML.** Esto se debe a que, si el código JavaScript se incluye antes que el HTML, JavaScript se ejecuta antes de que se haya cargado la página y al no haberse cargado los elementos HTML no sería posible establecer la propiedad onclick de los mismos, es decir, getElementsByTagName ('a') devolvería cero elementos.
- **A cada nodo tipo link ('a') se le asigna como propiedad onclick una función,** en este caso haciendo uso de return y confirm, de modo que el usuario puede elegir entre aceptar ser transferido al link elegido o cancelar.
- **This dentro de la función anónima a ejecutar ante un evento onclick** sobre un elemento a hace referencia al elemento propietario de la función, en este caso el propio nodo de tipo link a, de modo que puede rescatar la url de destino escribiendo this.href.

Una forma alternativa de escribir el *script* sería:

```
<script>

Var elemsA=document.getElementsByTagName('a');

For (var i=0; i<elemsA.length; i++){

elemsA[i].onclick = transferir;

function transferir(){

return confirm ('Va a ser transferido hacia' +this.href);

}
```

```
}
```

```
</script>
```

En este caso se usa una asignación a la función. Hay que tener en cuenta que no se escribe elemsA[i].onclick transferir();} porque esto daría lugar a que se ejecutara la función, es decir, estaría invocando a la función. Al escribir la asignación sin paréntesis la función queda asignada pero no se ejecuta.

Finalmente, tiene una alternativa más lógica que permite no tener que poner el *script* después del código HTML. La idea es la siguiente: el evento onload informa de cuándo se ha completado la carga de la página web. En ese momento disparará una función que se encargará de introducir los manejadores de eventos que quiera. Como en el momento en que se dispara la función la carga de la página ya se ha realizado, no es necesario que el código JavaScript para realizar este proceso quede debajo del código HTML.

Vea un ejemplo:

```
<!DOCTYPE html>

<html><title>Ejemplo de manejadores</title><meta charset="utf-8">

<script>

Window.onload=function (){

Var elemsA=document.getElementsByTagName('a');

For (var i=0; i<elemsA.length; i++){

elemsA[i].onclick=transferir;

function transferir() {return confirm ('Va a ser transferido hacia '+this.href);}

}

</script>

</head>

<body><div id="cabecera"><h2>Manejadores de eventos</h2></div>

<div style ="color:blue"; id="pulsador" onclick="ejemplo()">Probar</div>

<a href="https://www.google.es">Google.es</a>
```

```
<a href="https://www.google.com">Google.com</a>  
</body>  
</html>
```

Se debe recordar que al escribir `window.onload=algo();` la función `algo()` se ejecutará porque mediante esta instrucción está indicando que esa función debe ejecutarse cuando se produzca el evento `onload` sobre el objeto `window`. Por ejemplo, `window.onload =alert('La página ha terminado de cargar');` permite mostrar un mensaje de alerta advirtiendo de que la página ha terminado de cargar.

6.2. Eventos en elementos de formulario

Los eventos definen lo que hace el usuario en la página o en el elemento de la página al que se le aplica. Cualquier acción que haga el usuario es un evento como: mover el ratón, hacer clic o doble clic, pulsar una tecla, entrar con el ratón en un elemento, seleccionarlo con el ratón o con la tecla TAB, etc. Incluso el hecho de acabar de cargarse la página es un evento.

En JavaScript, se puede detectar cuando se produce un determinado evento y aplicarle un código para que al producirse el evento se ejecute ese código. De esta forma se interactúa con el usuario, el cual al provocar el evento desencadena la ejecución del código JavaScript asociado.

En concreto, los elementos de formulario son aquellos en los que cambia el estado de algún elemento de un formulario. Y, en este epígrafe va a centrar en dos elementos específicos: `onselect ()` y `onchange ()`;

6.2.1. Onselect (al seleccionar un elemento de un formulario)

El evento `onselect ()` hace que se ejecute un *script* cuando se selecciona texto dentro de un elemento perteneciente a un formulario, ya sea una casilla de texto o un área de texto. Su sintaxis es la siguiente:

```
<input type="tipoElemento" onSelect="función">
```

A continuación, vea un ejemplo de uso de este evento:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<script>
```

```
Function funcionEjemplo ()
```

```
{  
    Alert ("Se ha seleccionado un texto");  
}  
  
</script>  
  
<title> Ejemplo evento onselect</title>  
  
</head>  
  
<body>  
  
<form>  
  
<textarea rows=3 cols=40 onSelect="funcionEjemplo()"/>  
  
</form>  
  
</body>  
  
<html>
```

En el ejemplo anterior, si se selecciona un texto del elemento textarea, se verá cómo aparece un mensaje de alerta de que se ha seleccionado texto.

En el siguiente ejemplo, puede ver cómo definir una caja de texto con un cierto texto predefinido. Si este texto se selecciona se disparará el evento onSelect e informará de que ha seleccionado el texto. Para entender mejor esto, vea el siguiente ejemplo:

```
<!DOCTYPE html>  
  
<html>  
  
<head>  
  
<script>  
  
Function funcionEjemplo (nombre)  
  
{  
    Alert ("Ha seleccionado el texto"+nombre);  
  
}
```

```
</script>

<title> Ejemplo evento onselect</title>

</head>

<body>

Texto: <input type="text" value="Texto predefinido" onSelect="funcionEjemplo(this.value)">

</body>

<html>
```

Además, se puede ver cómo hace uso en este ejemplo del operador this. Este operador sirve para pasar datos a JavaScript del elemento HTML actual. En su caso, del elemento input escoge su value para pasárselo a la función y que lo muestre en el cuadro de texto. Este evento está definido para los objetos text y textarea.

6.2.2. Onchange (al cambiar el estado de un elemento de un formulario)

Este evento ocurre cuando se cambia el valor de un elemento de formulario. Este evento ocurre, por ejemplo, cuando se cambia el estado de los botones de radio y casillas de verificación. Su sintaxis en HTML es la siguiente:

```
<elemento onchange="instruccionesJavaScript">
```

Por otro lado, la sintaxis en JavaScript es la siguiente:

```
Object.onchange=function(){instrucciones};
```

Vea un ejemplo en el que cuando introduce un texto en una caja de texto y la abandona, pasa el texto a mayúsculas:

```
<!DOCTYPE html>

<html>

<head>

<script>

Function miFuncion(){

Var x=document.getElementById("fnombre");
```

```
x.value=x.value.toUpperCase();  
}  
  
<title> Ejemplo evento onchange</title>  
  
</script>  
  
</head>  
  
<body>
```

Introduce tu nombre:

```
<input type="text" id="fnombre" onchange="miFuncion()">  
  
<p>Cuando abandone la caja de texto, podrá comprobar cómo ha cambiado el texto a mayúsculas<p>  
  
</body>  
  
</html>
```

A continuación, se puede ver un ejemplo de un documento en el que define un texto con un valor predefinido de 30. Cuando el usuario cambie este valor es cuando saltará el evento onchange informándole de que se ha producido un cambio en este elemento de texto. Vea el ejemplo:

```
<!DOCTYPE html>  
  
<html>  
  
<head>  
  
<script>  
  
Function valida (dato)  
  
{  
  
    Alert('Cambio el valor de 30 a '+ dato);  
  
}  
  
</script>  
  
<title> Ejemplo evento onchange</title>
```

```
</head>
```

```
<body>
```

Cambie a cualquier valor distinto de 30:


```
<form name='formulario1' method='post'>  
  
<input type='text' value='30' onchange='valida(this.value)'>  
  
</form>  
  
</body>  
  
</html>
```

Este evento está definido para los objetos FileUpload, Select, Text y Textarea.

6.3. Eventos de ratón. Eventos de teclado

Gracias a los eventos de ratón y teclado, va a poder programar cierta interactividad con estos dos elementos los cuales el usuario podrá usar para cambiar los estados de los elementos que configuran una web. Los eventos de ratón son aquellos en los que el usuario utiliza el ratón para provocarlos. En cambio, los eventos de teclado son aquellos en los que el usuario utiliza el teclado.

A continuación, va a ver ambos elementos de manera más detallada y, también, verá cómo puede implementarlos en su web para así aportarle más funcionalidad y una mejor experiencia de usuario.



Más Info

RECURSO MULTIMEDIA



6.3.1. Eventos de ratón

Los eventos de ratón son acciones que realiza el usuario en una interfaz de usuario utilizando el ratón. La interpretación de estas acciones mediante software desarrollado para ello, permite ejecutar una función asociada a dicha acción.

La programación de los eventos de ratón se lleva a cabo mediante llamada a rutinas que se ejecutan cuando se produce una acción. Además, la acción que se llevará a cabo será diferente dependiendo de en qué parte de la pantalla se sitúe, por ejemplo, si se pulsa el botón izquierdo y el puntero está en una parte externa a una aplicación, no ocurrirá nada, pero si, por el contrario, el puntero se encuentra sobre un área que contiene un botón, al pulsar sobre el ratón se deberá ejecutar la rutina asociada a ese botón.

Algunos ejemplos de eventos de ratón son:

a) Onmousedown (al pulsar sobre un elemento de la página)

Este evento se produce cuando el usuario pulsa un botón del ratón sobre un elemento. Su sintaxis en HTML es la siguiente:

```
<elemento onmousedown="instruccionesJavaScript">
```

La sintaxis en JavaScript es la siguiente:

```
Object.onmousedown=function(){instrucciones}
```

A continuación, vea un ejemplo que dice qué botón del ratón se ha pulsado, si el derecho o el izquierdo:

```
<!DOCTYPE html>

<html>

<head>

<script>

Function cualBoton (event)

{

If (evento.button=="0")

{

Alert("Se ha pulsado el botón izquierdo del ratón");

}

Else

{
```

```
Alert ("Se ha pulsado el botón derecho del ratón");  
}  
}  
  
<title> Ejemplo evento onmousedown</title>  
  
</script>  
  
</head>  
  
<body>  
  
<div onMouseDown="cualBoton (event);">Esto es un ejemplo<br><br>  
Puede pulsar con cualquiera de los botones del ratón <br>  
En cualquier sitio de este texto <br>  
  
</div>  
  
</body>  
  
</html>
```

Como se puede observar, a la función cualBoton se le pasa por parámetro el evento que se ha generado para obtener información de él. Este evento está definido para los objetos button, document y link.

b) Onmousemove (al mover el ratón por la página)

Este evento se produce cuando el puntero se mueve mientras está sobre un elemento. Su sintaxis en HTML es la siguiente:

```
<element onmousemove="instruccionesJavaScript">
```

Si lo programa con JavaScript su sintaxis será la siguiente:

```
Object.onmousemove=function(){instruccionesJavaScript};
```

Vea el siguiente ejemplo de uso de onmousemove:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>

<script>

Function miFuncion (e) {

X=e.clientX;

Y=e.clientY;

Coor="Coordenadas; ("+x+","+y+")";

Document.getElementById("rec").innerHTML=coor

}

Function limpiarCoordenada ()

{

Document.getElementById("rec").innerHTML="";

}

</script>

<title> Ejemplo evento onmousemove</title>

</head>

<body style="margin:0px;">

<div id="coordiv" style="width:199px; height:99px; border:1px solid" Onmousemove="miFuncion (evento)" onmouseout="limpiarCoordenada()">

</div>

<p> Mueva el ratón sobre el rectángulo y podrá obtener las coordenadas de su ratón</p>

<p id="rec">

</p>

</body>
```

</html>

Este evento no está definido para los objetos, por eso, es necesario asociarlos a ellos. Para asociar este elemento a cualquier objeto tiene que utilizar la siguiente sintaxis:

nombreObjeto.nombreEvento

c) Onmouseout (al salir del área ocupada por un elemento de la página)

Este evento se produce cuando el usuario mueve su puntero del ratón hacia fuera de un elemento. Este evento es el contrario a onmouseover, y surte efecto cuando el puntero sale de la zona donde está programado este evento. Su sintaxis es la siguiente:

<elemento onmouseout="instruccionesJavaScript">

Y, si quiere utilizarlo en JavaScript debe escribir la siguiente sintaxis:

Object.onmouseout=funciton(){instrucciones};

Va a ver un ejemplo para aprender cómo utilizar este evento:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<script>
```

```
Function bigImg(x){
```

```
    x.style.height="64px";
```

```
    x.style.width="64px";
```

```
}
```

```
</script>
```

```
</head>
```

```
<body>
```

```

```

<p>La función bigImg() ocurre cuando el puntero del ratón entra y sale de la imagen.</p>

</body>

</html>

Este evento está definido para los objetos layer, link y img.

d) Onmouseover (al entrar el puntero del ratón en el área ocupada por un elemento de la página)

Este método se aplica cuando quiere que al mover el puntero del ratón sobre un elemento HTML, este cambie su estado, ya sea su color, su forma, etc. Su sintaxis en HTML es la siguiente:

<elemento onmouseover="instruccionesJavaScript">

Por el contrario, si quiere usarlo en JavaScript debe aplicar la siguiente sintaxis.

Object.onmouseover=function (){instrucciones};

Vea un ejemplo, a continuación, en el que cuando entra con el puntero del ratón sobre una imagen, esta se amplíe y cuando salga, se reduzca:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<script>
```

```
Function agrandarImg (x)
```

```
{
```

```
x.style.height="64px";
```

```
x.style.width="64px";
```

```
}
```

```
Function normalImg (x)
```

```
{
```

```
x.style.height="32px";
```

```
x.style.width="32px";  
}  
</script>  
  
<title> Ejemplo evento onmouseover</title>  
  
</head>  
  
<body>  
  
<img onmouseover="agrandarImg(this)" onmouseout="normalImg(this) border='0'"  
Src="XXX" alt="XXX" width="32" height="32">  
  
<p>La función agrandarImg() ocurre cuando el puntero del ratón entra en la imagen.</p>  
  
<p> La función normalImg() ocurre cuando el puntero del ratón sale de la imagen. </p>  
  
</body>  
  
</html>
```

Este evento está definido para los objetos layer, link e img.

e) **Onmouseup** (al soltar el usuario el botón del ratón que anteriormente había pulsado)

Este evento, puede decir que, es el contrario al evento onmousedown. Por lo tanto, ocurre cuando el usuario suelta el botón del ratón sobre un elemento. Su sintaxis en HTML es la siguiente:

```
<elemento onmouseup="instruccionesJavaScript">
```

Para poder manejar este evento con JavaScript, usa la siguiente sintaxis:

```
Objet.onmouseup=function(){instrucciones};
```

En el ejemplo que va a ver a continuación, si pulsa con el botón del ratón y mantiene el clic, puede ver cómo cambia de color el texto y, cuando deje libre el botón del ratón, el texto volverá a su color original. Vea el ejemplo:

```
<!DOCTYPE html>  
  
<html>  
  
<head>
```

```
<script>

Function cambiaColor(elmnt,clr)

{

Elmnt.style.color=clr;

}

</script>
```

```
<title> Ejemplo evento onmouseup</title>

</head>

<body>

<p onmousedown="cambiaColor(this,'purple')"

Onmouseup="cambiaColor(this,'green')">
```

Haga clic sobre el texto para cambiar el color. Para ello, haga clic y mantenga el clic presionado. Cuando suelte el botón, verá que el color del texto vuelve a su estado normal</p>

```
</body>

</html>
```

Este evento está definido para los objetos button, document y link.

Para finalizar con los eventos de ratón, va a ver un ejemplo de código en el que usa los eventos onmouseup, onmousedown y onmousemove. Se define un documento y crea un botón. Vea el siguiente ejemplo:

```
<!DOCTYPE html>

<html>

<head>

<script>

Function miFuncion_onmouseup (){

Document.form1.miBoton.value="El botón del ratón no está pulsado"
```

}

Function miFuncion_onmousedown(){

Document.form1.miBoton.value="El botón del ratón está pulsado"

}

Function miFuncion_onmousemove(e){

X=e.clientX;

Y=e.clientY;

Coor="Coordenadas: ("+x+","+y+");

Document.getElementById("rec").innerHTML=coor;

}

<title> Ejemplo evento onmouseup</title>

</script>

<title> Ejemplo evento onmouseup</title>

</head>

<body style=margin:0px">

<form name='form1' method='post'>

Mantenga el clic del ratón sobre este botón, a continuación, suéltelo:

<input type="button" name="miBoton" value="El botón del ratón no está pulsado" onmouseup="miFuncion_onmouseup()" onmousedown="miFuncion_onmousedown()">

Pase el ratón por encima del rectángulo:

<div id=coordiv" style="width:199px; height:99px; border:1px solid"

Onmousemove="miFuncion_onmousemove (event)">

</div>

<p id="rec">

```
</p>  
</form>  
</body>  
</html>
```

6.3.2. Eventos de teclado

Al igual que en el apartado anterior se vieron los eventos de ratón, también dispone de una serie de eventos asociados al teclado. Estos eventos van a recoger la información derivada de la acción que tenga lugar entre el usuario y el teclado.

Los eventos de teclado son los más difíciles de manejar, ya que hay que tener en cuenta que a veces son incompatibles entre diferentes navegadores. Tiene, además, varios tipos de teclas; las teclas normales, que pueden ser números y letras y las teclas especiales, que son las teclas de función, tabulador, teclas Alt, Shift, etc. Tiene, también, tres tipos de eventos de teclado onkeyup, onkeypress y onkeydown. También, hay que tener en cuenta que la misma propiedad da diferentes resultados para la misma tecla dependiendo del evento al que se ha llamado.

Para mayor dificultad, no todos los teclados son iguales, aunque la mayoría son de un modelo estándar, ni utilizan la misma información. Puede haber diferencias entre distintos idiomas o distintas configuraciones.

a. Onkeydown (al pulsar una tecla el usuario)

Este evento ocurre cuando el usuario presiona una tecla del teclado. Vea el siguiente ejemplo que no avisará de si ha pulsado una tecla en el teclado:

```
<!DOCTYPE html>  
  
<html>  
  <head>  
    <script>  
  
      Funciton miFuncion(){  
  
        Alert ("Ha presionado una tecla en la caja de texto")  
  
      }  
  
    </script>
```

```
<title> Ejemplo evento onkeydown</title>

</head>

<body>

<p>Presione una tecla dentro de la caja de texto.</p>

<input type="text" onkeydown="miFuncion()">

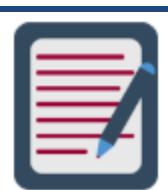
</body>

</html>
```

Este evento está definido para los objetos document, image, link y textarea.

b. Onkeypress (al dejar pulsada una tecla un tiempo determinado)

Este evento tiene lugar cuando el usuario mantiene presionada una tecla. La diferencia entre onkeypress y onkeydown es que onkeydown, además de leer las teclas que son números y caracteres, es capaz de leer teclas como suprimir, Bloq Mayús, etc. Y aparte keypress tiene como valor el atributo charCode, mientras que keydown viene en keyCode. Esta diferencia hay que tenerla en cuenta, ya que, hay teclas que no devuelven charCode y, por lo tanto, puede hacer fallar su código.



El evento onkeypress está ya obsoleto y puede no funcionar en todos los navegadores, por lo que se deberá usar, siempre que sea posible, el evento onkeydown o beforeinput.

TOME NOTA

Vea el siguiente ejemplo donde la caja de texto cambiará de color en cuanto pulse una tecla:

```
<!DOCTYPE html>

<html>

<head>

<script>
```

```
Function miFuncion(){  
Document.getElementById("Caja").style.backgroundColor="green";  
}  
  
</script>  
  
<title>Ejemplo de evento onkeypress</title>  
  
</head>  
  
<body>  
  
<p>Apriete una tecla estando dentro de la caja de texto</p>  
  
<input id="Caja" type="text" onkeypress="miFuncion()">  
  
</body>  
  
</html>
```

Este evento está definido para los objetos document, image, link y textarea.

Onkeyup (al liberar la tecla apretada)

Este evento se dispara cuando el usuario libera una tecla que antes estaba presionada. Vea un ejemplo donde el programa toma los caracteres que se introducen en la caja de texto y los convierte en mayúsculas:

```
<!DOCTYPE html>  
  
<html>  
  
<head>  
  
<script>  
  
Function miFuncion (){  
  
Var x=document.getElementById("fnombre");  
  
x.value=x.value.toUpperCase();  
  
}  
  
</script>
```

```
<title>Ejemplo de evento onkeyup</title>
```

```
</head>
```

```
<body>
```

```
<p> La función se disparará cuando introduzca un carácter en la caja de texto. Entonces, transformará los caracteres a mayúscula</p>
```

Introduzca una frase:

```
<input type="text" id="fnombre"
```

```
Onkeyup="miFuncion()>
```

```
</body>
```

```
</html>
```

Este evento está definido para los objetos document, image, link y textarea.

6.4. Eventos de enfoque

Tiene básicamente dos eventos de enfoque disponibles. Cuando entra a un objeto seleccionándolo, le da el foco de entrada a dicho objeto y, cuando abandona el objeto, se pierde el foco de entrada. Normalmente, al cargarse una página es ella misma la que tiene por defecto el foco de entrada. Sobre este tipo de eventos va a ver específicamente dos de ellos a continuación, uno es onBlur y el otro onFocus.

Básicamente, el evento onFocus se dispara cuando el objeto toma foco y el evento onBlur, cuando el objeto pierde el foco.

RECURSO MULTIMEDIA



6.4.1. Onblur (cuando un elemento pierde el foco de la aplicación)

La propiedad onblur devuelve un manejador de eventos para el evento onBlur en el elemento actual. El evento onblur ocurre cuando un elemento se sale de un elemento HTML como puede ser input, select y a. A continuación, vea la sintaxis de este evento en HTML:

```
<elemento onblur="instruccionesJavaScript">
```

Por otro lado, va a ver su sintaxis en JavaScript:

```
Object.onblur=function(){instrucciones};
```

A continuación, verá un ejemplo en el que el color de fondo de la página cambiará a beige cuando esta pierda el foco, es decir, cuando seleccione otra cosa que no sea la propia página y cuando vuelva a seleccionar la página, esta se tornará en negro. Vea el ejemplo:

```
<!DOCTYPE html>

<html>

<head>

<title>Evento onBlur</title>

</head>

<body onblur="document.body.style.backgroundColor='black'" onfocus="document.body.style.backgroundColor='beige'">

</body>

</html>
```

Este evento está definido para los objetos button, checkbox, fileUpload, layer, password, radio, reset, select, submit, text, textarea y window.

6.4.2. Onfocus (cuando un elemento de la página o la ventana gana el foco de la aplicación)

Este evento se ejecuta cuando el usuario sitúa el foco de entrada en un elemento del documento HTML. Su sintaxis en HTML es la siguiente:

```
<elemento onfocus="instruccionesJavaScript">
```

En cambio, su sintaxis en JavaScript es la siguiente:

```
Object.onfocus=function(){instrucciones};
```

A continuación, vea un ejemplo en el que va a hacer que un objeto de textarea cuando reciba el foco, es decir, cuando el usuario clique para escribir en él, alerte con un mensaje de que dicho elemento tiene el foco. Véalo a continuación:

```
<!DOCTYPE html>
```

```
<html>
<head>
<title> Ejemplo evento onfocus</title>
</head>
<body>
<textarea name="x" cols="40" rows="30" onfocus="alert('El bloque de texto tiene el foco en este momento')">
Pinche aquí para darle el foco a este elemento
</textarea>
</body>
</html>
```

Este evento está definido para los objetos button, checkbox, fileupload, layer, password, radio, reset, select, submit, text, textarea y window.

6.5. Eventos de formulario

Como ha visto anteriormente, los formularios se utilizan para pedir una serie de datos que tiene que llenar el usuario el cual interactúa con la página web. Este formulario se envía al servidor para que los datos sean procesados. Antes de enviar un formulario al servidor, debe comprobar que esté libre de errores. Para ello, es posible notificar los errores que comete el usuario de forma inmediata utilizando JavaScript. Esto ayuda a obtener una mejor experiencia de usuario y a reducir la carga de trabajo del servidor.

Por lo tanto, en esto consisten los eventos de formulario, en ser capaces de manejar los datos que se introducen en él. Los eventos que verá, a continuación, en este epígrafe son onreset y onsubmit.

6.5.1. Onreset (al hacer clic en el botón de reset de un formulario)

Este evento se ejecuta cuando se pulsa el botón reset del formulario. La finalidad de este evento es resetear todos los campos que se encuentren asociados al formulario. La sintaxis de onreset en HTML es la siguiente:

```
<elemento onreset="instruccionesJavaScript">
```

También puede usarlo en JavaScript con la siguiente sintaxis:

```
Object.onreset=function(){instrucciones};
```

Vea un ejemplo de uso de este evento donde puede ver cómo el formulario consta de dos botones, uno es el botón enviar y otro, el botón borrar el contenido. Si pulsa enviar, el contenido del formulario se enviará a ejemplo@ejemplo.com, por el contrario, si pulsa borrar, el contenido que ha rellenado en el formulario desaparecerá. Vea el ejemplo a continuación:

```
<!DOCTYPE html>

<html>

<head>

<title>Ejemplo de evento onreset</title>

</head>

<body>

<form action="mailto:ejemplo@ejemplo.com" method="post" enctype="text/plain"
onreset="return confirm ('¿Desea borrar los datos del formulario?')"
onsubmit="return confirm ('¿Desea enviar la información?')">
```

Dirección de correo:

```
<input type="text" name="email" onselect="alert ('Si pulsa control+C podrá copiar la información')">

<input type="reset" value="Borrar datos">

<input type="submit" value="Enviar">

</form>

</body>

</html>
```

Este evento está definido solo para el objeto form.

6.5.2. Onsubmit (al pulsar el botón de enviar el formulario)

El método onsubmit permite al usuario, una vez haya terminado de llenar la información del formulario, enviar la información al servidor para ser procesada por este. Este método utiliza la siguiente sintaxis en HTML:

```
<element onsubmit="instruccionesJavaScript">
```

Además, también puede utilizarlo con la siguiente sintaxis en JavaScript:

```
Object.onsubmit=funciton(){instrucciones};
```

También es importante comprobar los datos de entrada del formulario que el usuario ha introducido para, por ejemplo, comprobar que no hay entradas incorrectas como edad:000. Para ello, puede hacer uso de la función test(), que comprueba la correcta entrada de datos y devuelve false en caso de que encuentre algún error o incoherencia. Vea el ejemplo en el siguiente código en el que solamente se envía el formulario si la función test () devuelve true:

```
<form action="mailto:ejemplo@ejemplo.com" method="post" enctype="text/plain" onsubmit="return test()">
```

Ahora, imagine que necesita comprobar los campos de un formulario uno por uno para ver si el usuario ha introducido correctamente los campos. Lo ideal sería definir una función llamada, por ejemplo, validar en el elemento onsubmit. Véalo en el siguiente ejemplo:

```
<!DOCTYPE html>

<html>

<head>

<title> Ejemplo evento onsubmit</title>

</head>

<body>

<form action="mailto:ejemplo@ejemplo.com" methos="post" enctype="text/plain" onsubmit="return validar()">

</body>

</html>
```

Por otro lado, debería escribir la función validar () en la parte de JavaScript utilizando la función test () a su antojo.

Para finalizar, vea un ejemplo de uso del evento onsubmit:

```
<!DOCTYPE html>

<html>
```

```
<head>

<title> Ejemplo de uso del evento onsubmit</title>

</head>

<body>

<form name="miFormulario" action="mailto:ejemplo@ejemplo.com" onsubmit="alert ('Ha pulsado el botón enviar'); return false;">

<input type="submit" value="Enviar" name="Enviar">

</form>

</body>

</html>
```

Este objeto está solamente definido para el objeto form.

6.6. Eventos de ventana

Como y se ha visto, el objeto window representa la ventana que contiene un documento DOM. En un navegador que tenga pestañas, cada pestaña contiene su propio objeto window. Esto significa que el objeto window no se comparte entre diferentes pestañas de la misma ventana del navegador. Algunos métodos como los que verá más adelante se aplican sobre toda la ventana del navegador y no sobre una pestaña específica a la que pertenece el objeto window. Generalmente, cualquier cosa que razonablemente no pueda pertenecer a una pestaña, pertenece a la ventana.

Los eventos que va a ver en este apartado, al ser eventos de ventana, pertenecerán exclusivamente al objeto window. Además de los dos eventos que verá a continuación, hay muchos más eventos que son exclusivos del objeto window.

Este tipo de eventos permiten reconocer cómo interactúa el usuario con la ventana donde está cargado el documento web. En este epígrafe va a ver dos de los eventos asociados al elemento ventana, uno de ellos es onmove y otro, onresize.

6.6.1. Onmove (al mover la ventana del navegador)

Este evento se genera cuando el usuario mueve una ventana o un frame que componga a la ventana principal del navegador. La sintaxis de este elemento en HTML es la siguiente:

```
<elemento onmove="instruccionesJavaScript">
```

Este elemento también se puede usar en JavaScript directamente aplicando la siguiente sintaxis:

```
Object.onmove=function(){instrucciones};
```

A continuación, va a ver un ejemplo del uso del evento de ventana onmove:

```
<!DOCTYPE html>

<html>

<head>

<title> Ejemplo evento onmove</title>

</head>

<body onmove="alert ('La ventana se está moviendo')">

</body>

</html>
```

En este ejemplo que acaba de ver, se puede ver cómo ha definido para el documento el evento onmove. Cuando el usuario interactúa con la ventana y la mueve, este evento recogerá la información y informará de que la ventana se está moviendo. Este evento está únicamente definido para el objeto window.

6.6.2. Onresize (al redimensionar la ventana del navegador)

Este evento se genera cuando el usuario redimensiona la ventana del navegador. Su sintaxis en HTML es la siguiente:

```
<elemento onresize="instruccionesJavaScript">
```

También, puede aplicar este evento en JavaScript utilizando la siguiente sintaxis:

```
window.onresize=function(){instrucciones};
```

A continuación, va a ver un ejemplo en el que se muestra una ventana a pantalla completa con sus coordenadas y cuando el usuario la hace más pequeña, podrá ver qué coordenadas va teniendo. Vea el ejemplo a continuación:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>

<script>

Function miFuncion()

{

Var w=window.outerWidth;

Var h=window.outerHeight;

Var txt="El tamaño de la ventana es: ancho='"+w+"', alto='"+h+';

Document.getElementById("demo").innerHTML=txt;

}

<title>Ejemplo de evento onResize</title>

</script>

</head>

<body onresize="miFuncion()">

<p>Cambio el tamaño de esta ventana.</p>

<p id="demo"></p>

</body>

</html>
```

Este evento está definido únicamente para el objeto window.

6.7. Otros eventos

Como ya sabe, los eventos son la manera que tiene en JavaScript de controlar las acciones de los usuarios y definir un comportamiento de la página cuando se produzcan. Cuando un usuario visita su web e interactúa con ella se producen los eventos y con JavaScript puede definir qué quiere que ocurra cuando se produzcan.

Con JavaScript puede definir qué es lo que pasa cuando se produce un evento como podría ser que un usuario pulse sobre un botón o edite un campo de texto, cosa que ha visto en epígrafes anteriores. Sin embargo, también hay eventos en los que el usuario, por ejemplo, abandona la

página, o el usuario acaba de entrar y quiere que se dispare un evento al cargar la página, o quiere que el usuario pueda arrastrar un archivo y soltarlo en su web, etc. Todos estos eventos son los que verá en este epígrafe.

En definitiva, el manejo de eventos es el caballo de batalla para hacer las páginas interactivas, porque con ellos puede responder a las acciones de los usuarios. Hasta ahora, en este tema ha podido ver muchos ejemplos de manejo de eventos en JavaScript. Pero a continuación, verá algunos eventos más que pueden resultar útiles y curiosos para crear su propia web.

Los eventos que va a ver en este epígrafe son eventos que no pertenecen a un solo grupo de eventos y son los siguientes: onunload, onload, onclick, ondragdrop, onerror y onabort.

6.7.1. Onunload (al abandonar una página)

Este evento se genera cuando la página se ha descargado o la ventana del navegador se ha cerrado. Este evento también ocurre cuando se recarga la página. Si la página tuviera además un evento onload, este también se generaría. La sintaxis de onunload en HTML es la siguiente:

```
<elemento onunload="instruccionesJavaScript">
```

Por otro lado, su sintaxis en JavaScript es la siguiente:

```
Window.onunload=function(){instrucciones};
```

A continuación, verá un ejemplo en el que cuando un usuario actualiza la página o la cierra aparece un mensaje de saludo. Vea el código siguiente:

```
<!DOCTYPE html>

<html>

<head>

<meta charset=UTF-8>

<script>

Function miFuncion()

{

    Alert ("Muchas gracias por visitar mi página");

}

</script>
```

```
<title> Ejemplo evento onunload</title>

</head>

<body onunload="miFuncion()">

<h1>Bienvenidos a mi web</h1>

<p>Cierre esta ventana o pulse F5 para recargar la página. </p>

</body>

</html>
```

Este evento únicamente está definido para el objeto window.

6.7.2. Onload (al terminar de cargarse la página o imágenes)

Este evento se dispara cuando un objeto o página ha terminado de cargarse. Normalmente donde más suele utilizarse es en la etiqueta body. La sintaxis de este evento en HTML es la siguiente:

```
<elemento onload="instruccionesJavaScript">
```

Si quiere utilizar el evento onload en JavaScript, debe usar la siguiente sintaxis:

```
Window.onload=function(){instrucciones};
```

A continuación, va a ver un ejemplo en el que se avisa al usuario de que la página se ha cargado totalmente antes de mostrar su contenido. Véalo en el siguiente código:

```
<!DOCTYPE html>

<html>

<head>

<title>Ejemplo de evento onload</title>

<script>

Function miFuncion ()

{

    Alert ("la página ha terminado de cargarse");

}
```

```
}
```

```
</script>
```

```
</head>
```

```
<body onload="miFuncion ()">
```

```
<h1>Va a ver cómo se carga una página</h1>
```

```
</body>
```

```
</html>
```

A continuación, vea otro ejemplo donde construir un reloj que se visualizará una vez se haya cargado la página. Véalo en el siguiente código:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<script>
```

```
Function mueveReloj()
```

```
{
```

```
momentoActual= new Date();
```

```
hora=momentoActual.getHours()
```

```
minuto=momentoActual.getMinutes()
```

```
segundo=momentoActual.getSeconds()
```

```
horalimprimible=hora+":"+minuto+":"+segundo
```

```
document.form_reloj.reloj.value=horalimprimible
```

```
setTimeout("mueveRelog()",1000)
```

```
}
```

```
</script>
```

```
<title>Ejemplo 2 del evento onload</title>

</head>

<body onload="muestraReloj()">

<form name="form_reloj">

<input type="text" name="reloj" size="10">

</form>

</body>

</html>
```

Este evento está definido para los objetos image, layer y window.

6.7.3. *Onclick (al hacer clic en el botón del ratón sobre un elemento de la página)*

Este evento se dispara cuando el usuario hace clic en un elemento de su documento. La sintaxis del evento onclick en HTML es la siguiente:

```
<elemento onclick="instruccionesJavaScript">
```

Además, este evento lo puede utilizar en JavaScript usando la siguiente sintaxis:

```
Object.onclick=function(){instrucciones};
```

A continuación, verá un ejemplo en el que si hace clic sobre un botón, añade información al documento:

```
<!DOCTYPE html>

<html>
<head>

<script>

Function miFuncion()

{
    Document.getElementById("demo").innerHTML="Un texto más";
}
```

```
}
```

```
</script>
```

```
<title> Ejemplo evento onclick</title>
```

```
</head>
```

```
<body>
```

```
<p>Pulsa el botón</p>
```

```
<button onclick="miFuncion()">Púlsame</button>
```

```
<p id="demo"></p>
```

```
</body>
```

```
</html>
```

Ahora, va a ver otro ejemplo de uso del evento onclick. En este caso va a crear un nuevo documento al que le inserta un botón con el que, cuando el usuario lo pulsa, se actualiza la página. Vea el código a continuación:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<script>
```

```
Function actualizar ()
```

```
{
```

```
Window.location.reload();
```

```
}
```

```
</script>
```

```
<title> Ejemplo evento onclick</title>
```

```
</head>
```

```
<body>  
<form>  
  <input type="button" value="Refrescar página" onclick="actualizar()">  
</form>  
</body>  
</html>
```

Este evento está definido para los objetos button, document, checkbox, link, radio, reset y submit.

6.7.4. Ondragdrop (al soltar algo que se ha arrastrado sobre la página)

Este evento se genera cuando el usuario suelta algún objeto sobre la ventana del navegador, por ejemplo, una imagen, un fichero, etc. Actualmente, este evento está obsoleto y se utilizan los estándares del HTML5 para conseguir el resultado drag and drop. En este epígrafe no va a explicar los estándares del HTML5 para este evento porque es un tema muy complejo y profundo, pero en la siguiente [web](#) se le puede echar un vistazo.

A continuación, verá un ejemplo de código utilizando este evento:

```
<!DOCTYPE html>  
  
<html>  
  <head>  
    </head>  
  
  <body ondragdrop='return(confirm("¿Está usted seguro de que desea continuar?"))'>  
  
    Intente pegar un elemento en esta página.  
  
  </body>  
  
</html>
```

Este evento está únicamente definido para el objeto window.

6.7.5. Onerror (al no poderse cargar un documento o una imagen)

Este evento se produce cuando la carga de un documento, página o imagen produce un error. Por ejemplo, en el siguiente código se muestra una imagen que produce un error, es decir, la imagen no se encuentra disponible en el directorio para su visualización. En este caso el evento onerror se encargará de manejar este error. Vea el código a continuación:

```
<!DOCTYPE html>

<html>

<head>

</head>

<body>



</body>

</html>
```

Además, va a ver a continuación otro ejemplo de este mismo ejemplo, pero en este caso lo usa cuando el documento aún se está cargando. Vea el ejemplo siguiente:

```
<!DOCTYPE html>

<html>

<head>

<script>

Window.onerror=function()

{

Alert('Ha ocurrido un error')

}

</script>

</head>
```

```
<body>  
<script>  
Document.write('Hola'  
</script>  
</body>  
</html>
```

En el código anterior puede ver que ha dejado un paréntesis sin poner. Se puede observar, también, cómo define el evento onerror a través del objeto window y su función correspondiente para que informe de que se ha producido un error al cargar el documento. Por lo que, al cargar la página, se disparará el evento alert e informará de que ha ocurrido un error durante la carga de la página.

Este evento está definido para los objetos image y window.

6.7.6. Onabort (al detenerse la carga de una imagen de la página o irse de la página)

Este evento se activa al cancelarse la carga de una página, ya sea porque se pulsa el botón de cancelar o porque el usuario se marcha de la página por otro enlace.

Este ejemplo contiene una imagen que tiene el evento onabort asignado para que se ejecute una función en caso de que la imagen no llegue a cargarse. La función informa al usuario de que la imagen no se ha llegado a cargar y le pregunta si desea cargarla otra vez. Si el usuario contesta afirmativamente, entonces se pone a descargar la imagen otra vez. Si, contesta negativamente, no hace nada. La pregunta se hace con una caja confirm de JavaScript. A continuación, verá el código:

```
<!DOCTYPE html>  
  
<html>  
<head>  
<title>Evento onabort</title>  
  
<script>  
  
Function preguntarSeguir(){
```

 Respuesta=confirm("Ha detenido la carga de la página y hay una imagen que no está viendo. ¿Desea cargar la imagen?")

If(respuesta)

```
Document.img1.src="https://ejemplo.com/imagenes/ejemplo.jpg" width=500 height=458  
alt="Imagen de prueba" border="0" onabort="preguntarSeguir()">>
```

```
<br/>
```

Pulse el botón para parar la carga de la página y se pondrá en marcha el evento onabort.

```
</body>
```

```
</html>
```

Este ejemplo solo tendrá lugar si siempre se detuviese la carga por pulsar el botón de cancelar, pero si lo que pasa es que el usuario ha cancelado por irse a otra página a través de un enlace, aparecerá la caja de confirmación, pero no pasará nada y el usuario, simplemente, se machará.

7. BÚSQUEDA Y ANÁLISIS DE *SCRIPTS*

En este tema va a aprender a cómo buscar *scripts* correctos y adecuados para el tipo de programa que está realizando. Hoy en día, gracias a las nuevas tecnologías, tiene a su alcance cientos de sitios en Internet donde puede encontrar diversos tipos de *scripts* en diversos lenguajes de programación. Sin embargo, no todos los *scripts* que ofrecen estos sitios van a ser totalmente válidos. Por lo tanto, para conseguir scripts adaptados a sus necesidades, debe seguir las siguientes pautas:

- **Cumplir con sus expectativas.** Normalmente, es muy complicado encontrar un código que se adapte al cien por cien al tipo de programa que está desarrollando. Lo ideal es ver múltiples ejemplos, ver diferentes tipos de códigos y, sobre todo, cómo hacen los *scripts* otros programadores, esto ayuda a familiarizarse con el lenguaje que está utilizando y, cada vez, puede ir escribiendo código mejor y más eficiente. Por lo tanto, es muy importante fijarse en el código de otros programadores expertos.
- **Funcionar de forma correcta.** A veces, se topará con determinados códigos que aparentan solventar determinado problema. Sin embargo, al copiar este código en su documento, se dará cuenta que este no funciona. Esto puede ser por diversas razones como, por ejemplo, que ese código pertenezca a otra versión de JavaScript más antigua y por eso haya determinadas instrucciones que los navegadores ya no reconozcan. En este caso, debe depurar el script y cambiar esas instrucciones por las sintaxis actuales para la misma instrucción. Otro de los problemas puede ser que el código incluya erratas que el autor no se haya dado cuenta al escribir el código. En este caso, tan solo debe dar con ellas y corregirlas. También, es importante que ese código lo pruebe en distintos navegadores, ya que, puede ocurrir que ese código no sea funcional en el navegador que use normalmente.
- **Ser lo más eficiente desde el punto de vista computacional.** Este punto es uno de los más importantes. Imagine que está buscando un código para ordenar por fecha los pedidos que han hecho sus clientes en su web. Si el código que ha implementado para esta tarea tarda una media de dos horas en hacerla, este código no es nada eficiente y, por lo tanto, debería desecharlo y volver a crear un *script* nuevo y eficiente para la tarea que lo crea.

Lo más normal, es que seguramente no encuentre un código que solucione su problema completamente. Pero, si se basa en su experiencia como programador y trabaja con el código que ha obtenido, lo puede modificar a su antojo para obtener el resultado deseado. Esto, lo hacen los programadores constantemente, ya que, si existe una base de código predefinida no tiene sentido volver a programar otra vez lo mismo. Por lo tanto, lo lógico es partir de un código predeterminado y cambiarlo para que dé el resultado que quiere obtener.

7.1. Búsqueda en sitios especializados

Si busca en Internet verá miles y miles de páginas que ofrecen información relativa a JavaScript. Muchas de ellas pueden estar desfasadas y, por lo tanto, ofrece código que ya no sea útil para los navegadores actuales. Por esta razón, lo mejor es mirar varios ejemplos del mismo código y, sobre todo, mirar en las páginas oficiales en las cuales indican qué código está obsoleto y cuál es la nueva sintaxis para la misma instrucción. Incluso, hay páginas oficiales que indican en cuáles navegadores funciona determinada instrucción y en cuáles no.

En este apartado va a ver en profundidad las páginas oficiales más actualizadas y más fiables, los tutoriales mejor explicados y más profesionales, foros en los que puede preguntar dudas que no sepa solventar y las múltiples bibliotecas que puede usar para mejorar JavaScript.

7.1.1. Páginas oficiales

Actualmente, JavaScript no dispone de ninguna página oficial que dé soporte a dicho lenguaje de programación. Lo que sí va a encontrar son ciertas páginas que profundizan mucho en su utilización.

Algunas de las páginas oficiales de JavaScript que verá en este epígrafe están en inglés. Estas páginas serán muy útiles a la hora de desarrollar su web, ya que, en ellas puede encontrar toda la sintaxis actualizada, los atributos, los métodos y todo muy bien explicado.

La primera página que va a ver es la de [MDN](#), esta página enseña a usar JavaScript desde su inicio. Además de ser una página principal para consultar la sintaxis correcta de JavaScript o si está o no desactualizada alguna instrucción, ofrece un tutorial para aprender a programar con JavaScript. La web es la siguiente:

The screenshot shows the MDN Web Docs homepage with the following details:

- Header:** MDN Web Docs (with Mozilla logo), Technologies, References & Guides, Feedback, Search MDN (with magnifying glass icon), Sign in.
- Breadcrumbs:** Tecnología para desarrolladores web > JavaScript
- Language Options:** Change language | View in English
- Table of contents:** Tutoriales
- Related Topics:** JavaScript Guide, Intermediate, Advanced, References, Built-in objects, Expressions & operators, Statements & declarations, Functions.
- Main Content (JavaScript page):**
 - Section:** JavaScript
 - Text:** JavaScript (JS) es un lenguaje de programación ligero, interpretado, o compilado just-in-time (just-in-time) con funciones de primera clase. Si bien es más conocido como un lenguaje de scripting (secuencias de comandos) para páginas web, y es usado en muchos entornos fuera del navegador, tal como Node.js, Apache CouchDB y Adobe Acrobat. JavaScript es un lenguaje de programación basada en prototipos, multiparadigma, de un solo hilo, dinámico, con soporte para programación orientada a objetos, imperativa y declarativa (por ejemplo programación funcional). Leer más en [acercade JavaScript](#).
 - Text:** Esta sección está dedicada al lenguaje JavaScript en sí, y no a las partes que son específicas de las páginas web u otros entornos host. Para información acerca de APIs específicas para páginas Web, consulta [APIs Web](#) y [DOM](#).
 - Text:** El estándar para JavaScript es ECMAScript. A partir del 2012, todos los navegadores modernos soportan completamente ECMAScript 5.1. Los navegadores viejos soportan al menos ECMAScript 3. Desde Junio 17, 2015, ECMA International publicó la sexta versión principal de ECMAScript, que oficialmente se llama ECMAScript 2015, y que inicialmente se denominó ECMAScript 6 o ES6. Desde entonces, los estándares ECMAScript están en ciclos de lanzamiento anuales. Esta documentación hace referencia a la última versión preliminar, que actualmente es [ECMAScript 2020](#).
 - Text:** No confundas JavaScript con el lenguaje de programación Java. Ambos "Java" y "JavaScript" son marcas o marcas registradas de Oracle en los Estados Unidos y otros países. Sin embargo, los dos lenguajes de programación tienen sintaxis, semántica y usos muy diferentes.

La siguiente web que va a ver se llama [desarrolloweb.com](#) esta web ofrece contenido totalmente en español. En ella puede encontrar manuales sobre JavaScript y sobre las librerías que puede utilizar con JavaScript. Una cosa muy interesante es que ofrece manuales para desarrollo de JavaScript en dispositivos móviles. Su web es la siguiente:



Otra web que puede interesar es la de [w3school.com](https://www.w3schools.com), en esta web además de poder encontrar documentación de JavaScript, también puede encontrar información sobre HTML y CSS que ayudará a adornar su web y a tener un diseño más limpio y profesional de cara al cliente. Esta web es un poco escueta en comparación con las dos anteriores, pero, como ha visto antes, siempre hay que mirar varios sitios para sacar la información que necesita para el desarrollo de una web. La web de w3school es la siguiente:

A screenshot of the w3schools.com website. It features two main sections: "JavaScript" (top half) and "PYTHON" (bottom half). Each section has a title, a brief description, and two buttons: "Learn [language]" and "JavaScript Reference". On the right side of each section, there is a "Try it Yourself" button. The JavaScript section contains a code example:

```
<button onclick="myFunction()">Click Me!</button>
<script>
function myFunction() {
  var x = document.getElementById("demo");
  x.style.fontSize = "25px";
  x.style.color = "red";
}
</script>
```

 and the Python section contains:

```
if 5 > 2:
    print("Five is greater than two!")
```

La siguiente web que verá se llama javascript.info es una web hecha por la comunidad de JavaScript, donde puede encontrar muchísima documentación. La web es la siguiente:

Search in the tutorial

SEARCH

Table of contents

Main course contains 2 parts which cover JavaScript as a programming language and working with a browser. There are also additional series of thematic articles.

PART 1

The JavaScript language

Here we learn JavaScript, starting from scratch and go on to advanced concepts like OOP. We concentrate on the language itself here, with the minimum of environment-specific notes.

An introduction

- | | |
|-----------------------------------|-----------------------|
| 1.1 An Introduction to JavaScript | 1.3 Code editors |
| 1.2 Manuals and specifications | 1.4 Developer console |

JavaScript Fundamentals

- | | | |
|---|--------------------------------------|----------------------------------|
| 2.1 Hello, world! | 2.7 Type Conversions | 2.13 Loops: while and for |
| 2.2 Code structure | 2.8 Basic operators, maths | 2.14 The "switch" statement |
| 2.3 The modern mode, "use strict" | 2.9 Comparisons | 2.15 Functions |
| 2.4 Variables | 2.10 Conditional branching: if, ?' | 2.16 Function expressions |
| 2.5 Data types | 2.11 Logical operators | 2.17 Arrow functions, the basics |
| 2.6 Interaction: alert, prompt, confirm | 2.12 Nullish coalescing operator ??' | 2.18 JavaScript specials |

Code quality

- | | | |
|-------------------------|----------------|----------------------------------|
| 3.1 Debugging in Chrome | 3.3 Comments | 3.5 Automated testing with Mocha |
| 3.2 Coding Style | 3.4 Ninja code | 3.6 Polyfills and transpilers |

7.1.2. Foros

Los foros son muy útiles para encontrar información referente a JavaScript. La característica fundamental de los foros es que puede retroalimentarse de su propia información, es decir, expone la duda que tiene y seguramente obtendrá muchas respuestas, todas ellas distintas entre sí. Esta herramienta siempre va a ser muy útil, ya que, algunos de los errores que encuentre al desarrollar cualquier web, puede que no los haya visto nunca, pero al exponerlo en este canal está a la vista de muchísima gente que sabrá cómo ayudarle. Como programadores, debe encajar cada respuesta de manera que ayude a solucionar el problema.

A continuación, va a nombrar algunos foros importantes donde puede encontrar ayuda con JavaScript:

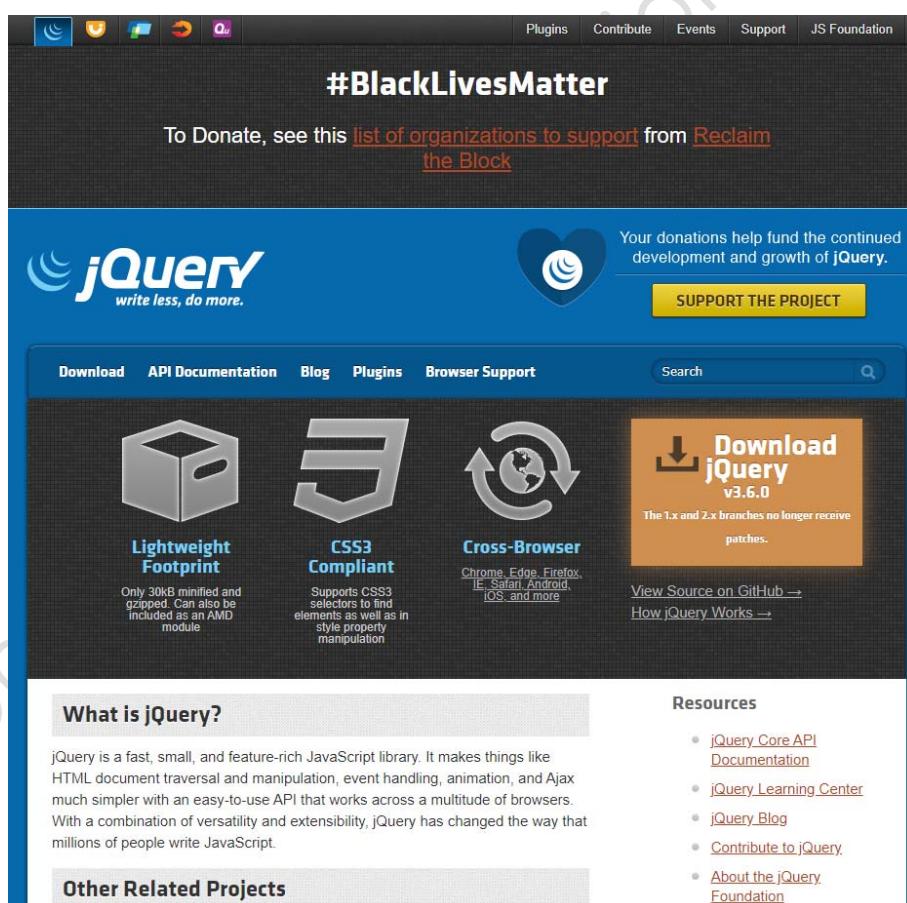
- Lawebdelprogramador.com
- Aprendeaprogramar.com
- Pildorasinformaticas.es
- Forosdelweb.com
- [Programación.net](http://Programacion.net)

7.1.3. Bibliotecas

En general, las librerías JavaScript son un código reutilizable que a menudo tiene un caso de uso principal. Las librerías proporcionan muchas funcionalidades estándar para que los desarrolladores no tengan que preocuparse por muchas funciones. Así, pueden usar estas para crear páginas web fácilmente utilizando componentes de la interfaz de usuario, utilidades de lenguaje, funciones matemáticas y más. Una librería consta de varias funciones, objetos y métodos según el lenguaje. Además, se pueden incluir en un proyecto sin depender de una estructura en particular. Es decir, será libres de utilizar una, dos o tantas librerías JavaScript como necesite.

A continuación, va a ver algunas de las librerías JavaScript más útiles donde varias de ellas no son muy conocidas, pero que se usan para el desarrollo web:

jQuery. Es una librería de JavaScript rápida e incluida en un solo archivo. Proporciona muchas funciones integradas mediante las cuales se pueden realizar diversas tareas de manera fácil y rápida, como selección o manipulación DOM. Permite agregar interactividad y efectos visuales en un sitio web. Su [portal](#) donde puede descargarla es el siguiente:



Moment.js. Moment.js ayuda a trabajar con las fechas. Lo que permite, por ejemplo, es que en vez de mostrar la fecha en formato “publicado el 07 de julio del 2020 a las 18:50 a.m.”, como lo devuelve

JavaScript, Moment.js lo estandariza y simplifica poniendo “hace 5 minutos”. Da la posibilidad de poner varios formatos de fechas. Como puede ver a continuación los ejemplos:

Format Dates

```
moment().format('MMMM Do YYYY, h:mm:ss a'); // May 5th 2021, 9:39:13 pm  
moment().format('dddd'); // Wednesday  
moment().format("MMM Do YY"); // May 5th 21  
moment().format('YYYY [escaped] YYYY'); // 2021 escaped 2021  
moment().format(); // 2021-05-05T21:39:13+02:00
```

Relative Time

```
moment("20111031", "YYYYMMDD").fromNow(); // 10 years ago  
moment("20120620", "YYYYMMDD").fromNow(); // 9 years ago  
moment().startOf('day').fromNow(); // a day ago  
moment().endOf('day').fromNow(); // in 2 hours  
moment().startOf('hour').fromNow(); // 39 minutes ago
```

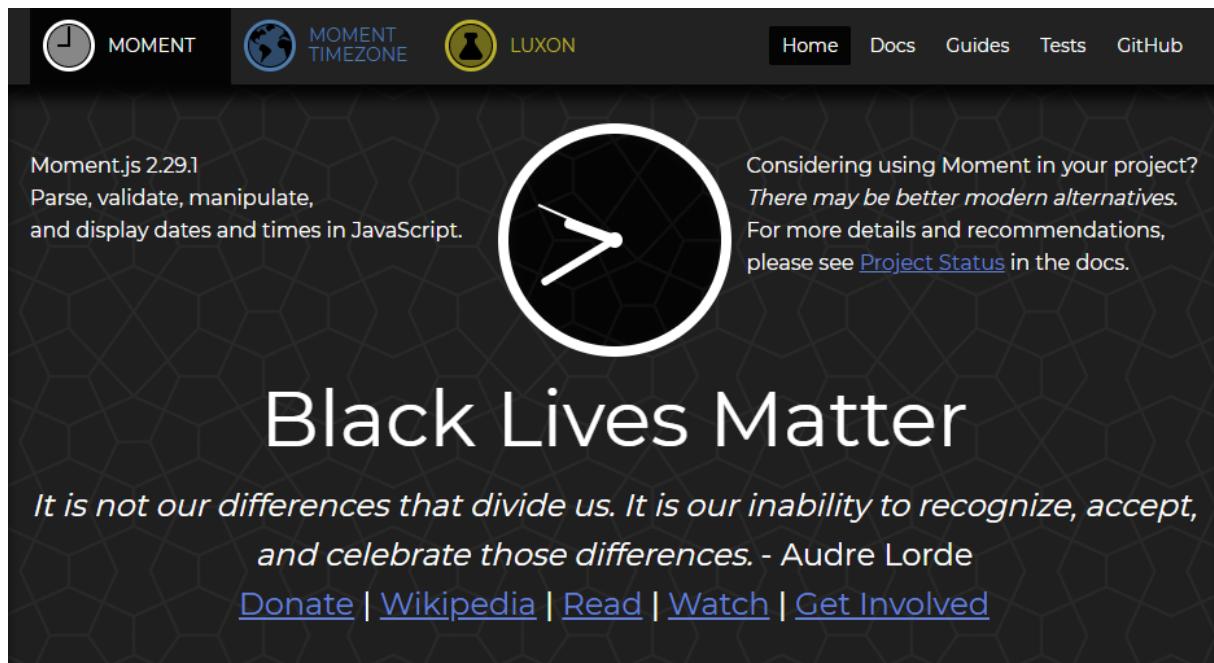
Calendar Time

```
moment().subtract(10, 'days').calendar(); // 04/25/2021  
moment().subtract(6, 'days').calendar(); // Last Thursday at 9:39 PM  
moment().subtract(3, 'days').calendar(); // Last Sunday at 9:39 PM  
moment().subtract(1, 'days').calendar(); // Yesterday at 9:39 PM  
moment().calendar(); // Today at 9:39 PM  
moment().add(1, 'days').calendar(); // Tomorrow at 9:39 PM  
moment().add(3, 'days').calendar(); // Saturday at 9:39 PM  
moment().add(10, 'days').calendar(); // 05/15/2021
```

Multiple Locale Support

```
moment.locale(); // en  
moment().format('LT'); // 9:39 PM  
moment().format('LTS'); // 9:39:13 PM  
moment().format('L'); // 05/05/2021
```

El [portal](#) donde se puede descargar es el siguiente:



Download

moment.js

moment.min.js 18.2k gz

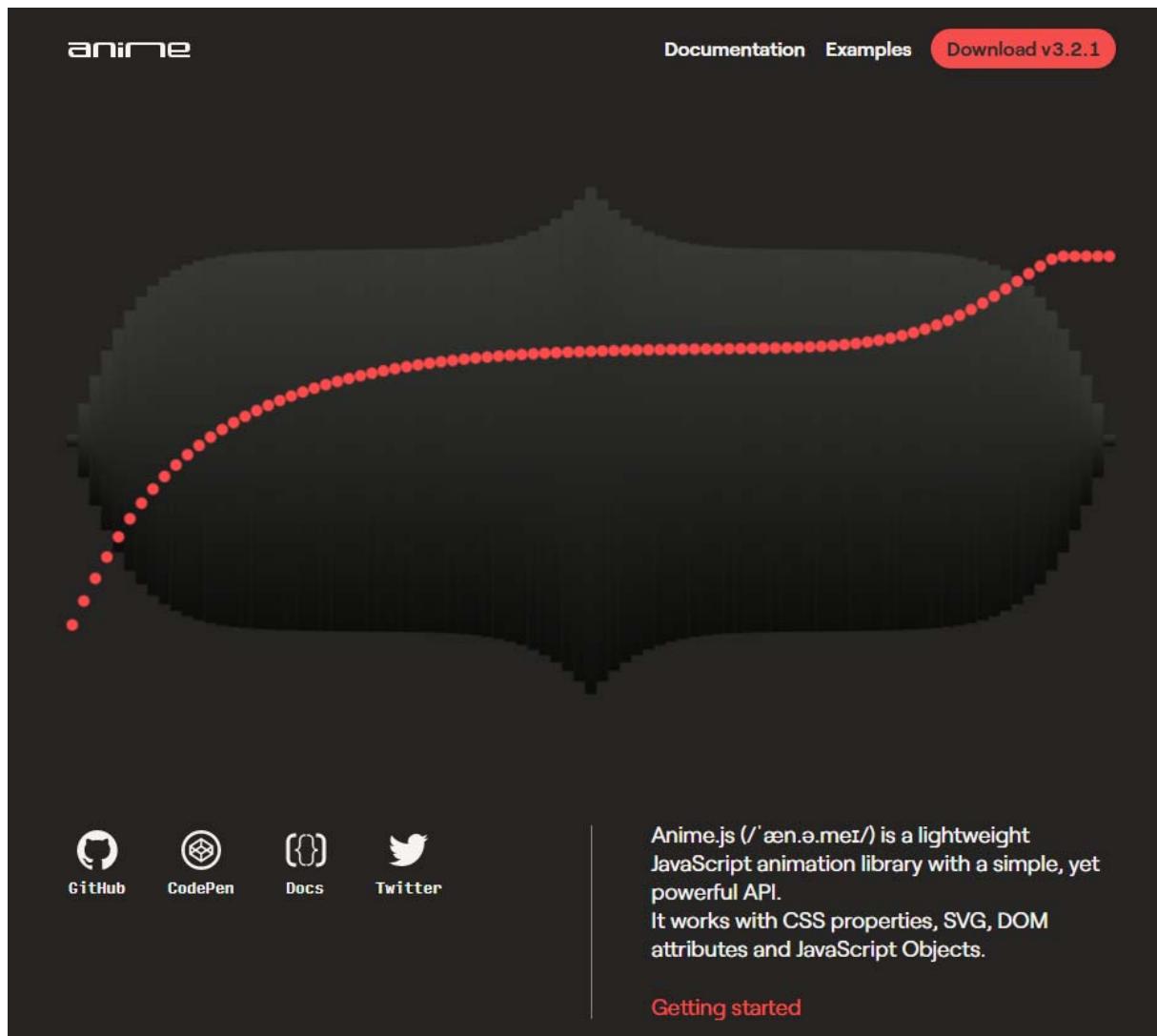
moment-with-locales.js

moment-with-locales.min.js 73.5k gz

Install

```
npm install moment --save      # npm
yarn add moment                # Yarn
Install-Package Moment.js     # NuGet
spm install moment --save      # spm
meteor add momentjs:moment    # meteor
bower install moment --save   # bower (deprecated)
```

Anime.js. Se trata de una librería para animar diferentes propiedades CSS, SVG o atributos DOM en una página web. Permite controlar todos los aspectos de la animación y proporciona muchas maneras de especificar las propiedades que quiere animar o los elementos que quiere utilizar. La [web](#) donde puede descargar esta librería es la siguiente:



Ramda. Es una librería funcional y práctica para programadores de JavaScript. Se centra en la inmutabilidad y las funciones libres de efectos secundarios. Las funciones de Ramda se ejecutan automáticamente, lo que permite crear nuevas funciones a partir de las antiguas simplemente al no proporcionar los parámetros finales. La [web](#) donde puede descargar Ramda es la siguiente:

Ramda v0.27.0 Home Documentation Try Ramda GitHub Discuss

Ramda

A practical functional library for JavaScript programmers.

[build passing](#) [npm package](#) [0.27.1](#) [dependencies](#) [gitter](#) [join chat](#)

Why Ramda?

There are already several excellent libraries with a functional flavor. Typically, they are meant to be general-purpose toolkits, suitable for working in multiple paradigms. Ramda has a more focused goal. We wanted a library designed specifically for a functional programming style, one that makes it easy to create functional pipelines, one that never mutates user data.



What's Different?

The primary distinguishing features of Ramda are:

- Ramda emphasizes a purer functional style. Immutability and side-effect free functions are at the heart of its design philosophy. This can help you get the job done with simple, elegant code.
- Ramda functions are automatically curried. This allows you to easily build up new functions from old ones simply by not supplying the final parameters.
- The parameters to Ramda functions are arranged to make it convenient for currying. The data to be operated on is generally supplied last.

The last two points together make it very easy to build functions as sequences of simpler functions, each of which transforms the data and passes it along to the next. Ramda is designed to support this style of coding.

D3.js. Es una librería de JavaScript para manipular documentos basados en datos. Proporciona una gran facilidad y flexibilidad para crear visualizaciones de datos, infogramas dinámicos e interactivos en navegadores web. La [web](#) donde puede encontrar esta librería es la siguiente:

D3 Data-Driven Documents



Like visualization and creative coding? Try interactive JavaScript notebooks in [Observable!](#)

D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS. D3's emphasis on web standards gives you the full capabilities of

[See more examples](#)
[Chat with the community](#)

7.2. Operadores booleanos

Los operadores booleanos, también conocidos como operadores lógicos, son palabras o símbolos que permiten conectar de forma lógica conceptos o grupos de términos para así ampliar, limitar o definir sus búsquedas rápidamente. Son muy sencillos de usar y pueden incrementar considerablemente la eficacia de sus búsquedas bibliográficas. En general, existen tres tipos de operadores que puede utilizar en sus búsquedas. En concreto, los operadores puramente booleanos, los de posición y los relacionales. En este apartado va a centrar en los operadores booleanos.

Los operadores booleanos son los más conocidos y son el punto clave de este epígrafe. Quizás el que menos presencia tenga actualmente en los sistemas de búsqueda sea el operador XOR. Sin embargo, el resto será muy útil incluso para realizar búsquedas rutinarias en Google.

En temas anteriores ya se vio cómo funcionaban este tipo de operadores dentro de la programación, en este epígrafe, sin embargo, verá cómo se utilizan en los navegadores.

RECURSO MULTIMEDIA



7.2.1. Funcionamiento de los operadores booleanos

En este apartado va a ver cómo funcionan y se utilizan los tres tipos básicos de operadores booleanos:

- **AND (Y).** Este operador se usa para poder unir términos de búsqueda y que los documentos que busca contengan ambos términos. Un ejemplo sería “Gauss y matemáticas”, el cual si se lo da a un buscador buscará documentos que contentan los dos términos de búsqueda unidos por el operador booleano. En algunos buscadores puede usar el símbolo + para indicar una AND. En el ejemplo anterior, quedaría “Gauss + matemáticas”.
- **OR (O).** Este operador se usa para poder unir términos de búsqueda y que los documentos que busca contengan uno u otro término o ambos. Por ejemplo, si pone “ordenadores o consolas”, buscará documentos que contengan alguno de los dos términos o ambos. Para escribir un OR también puede utilizar el símbolo |, por lo tanto, el ejemplo anterior quedaría de la siguiente manera “ordenadores | consolas”.
- **NOT (negación).** Este operador se usa para buscar documentos que no contengan los términos que se pasen. Por ejemplo, la búsqueda “ordenadores NOT consolas” buscarán páginas que contengan la palabra ordenadores pero que no contenga la palabra consolas. Este operador también se puede utilizar con el símbolo – o con AND NOT. Vea cómo quedaría el ejemplo anterior en ambos casos. Con el símbolo – quedaría ordenadores – consolas y con AND NOT, ordenadores AND NOT consolas.
- **XOR (O exclusivo).** Este operador se usa para buscar dos términos excluyentes uno al otro. Es decir, si busca ordenadores XOR consolas, el buscador mostrará documentos con ordenadores o con consolas, pero nunca documentos donde vengan ambos términos a la vez.

7.2.2. Utilización en distintos buscadores

Normalmente, va a usar los operadores booleanos en los diferentes buscadores de Internet con el fin de acotar o restringir sus búsquedas para que lo que busque que se ajuste a los resultados que obtiene en el buscador.

El buscador que se usa a día de hoy por excelencia es Google, tanto en su versión .com como en la .es. Es por ello, que se centrará en este navegador para explicar cómo usar los diferentes operadores booleanos en sus búsquedas.

Por defecto, Google considera que, si no da un valor booleano a sus búsquedas entre término y término, añadirá por defecto un AND. De esta forma, si busca Ordenador potente y no da un operador booleano, Google lo sustituirá por ordenador AND potente.

A continuación, verá algunos ejemplos de búsquedas en Google y el resultado que da para comprenderlas mejor:

- **Si busca Historia del arte.** Todas las palabras serán consultadas como historia-del-arte y Google mostrará las páginas con mayores probabilidades de contener la información deseada. En general, las páginas que contengan estas tres palabras en un orden cualquiera. Las palabras como las preposiciones y los artículos son tan frecuentes que no influirán en el resultado de la búsqueda. Sería prácticamente igual si hubiese buscado historia arte.
- **Si busca Historia OR Arte.** Se mostrarán páginas que contengan cualquiera de las dos palabras Historia o Arte.
- **Si busca "Historia del Arte".** Google mostrará páginas que contenga esta frase exacta, ya que, Google interpreta el uso de comillas como que quiere buscar el contenido que introduzca entre las comillas de manera exacta.
- **Si busca películas -románticas -acción.** Google mostrará todas las páginas que contengan la palabra película, pero que no contenga ni la palabra acción, ni la palabra romántica.
- **Si busca películas +documental.** La búsqueda se llevará a cabo por el término películas, y se incluirán los resultados que contengan el término documental.
- **Si busca ~divertido.** Se buscarán páginas en cuyos resultados aparezca un término parecido a divertido. Podría aparecer divertir, divertirse, etc.
- **Si busca Define: ordenador.** Permitirá conocer el significado de una palabra. En el ejemplo, mostrará el significado de la palabra ordenador.
- **Si busca secretos * Google.** Devolverá páginas que contengan una frase con los términos secretos y Google y además un, y sólo un, término cualquiera entre las dos palabras. Esto se debe al uso del asterisco entre los términos.
- **Si busca Go*.** Devolverá páginas que contengan al menos el principio de la palabra Go, más cualquier combinación posible como puede ser Google, Gol, Goles, Gonzalo, Gota, etc.
- **Botón Voy a tener suerte.** Con este botón se visita automáticamente la primera página que devolvería la consulta efectuada, es decir, no se garantiza que la búsqueda sea lo más correcta posible.

Aunque en este epígrafe se centre exclusivamente en el buscador Google, estos operadores pueden usarse en la mayoría de navegadores que tiene disponibles actualmente como, por ejemplo, los siguientes:

- Edge
 - Mocilla Firefox
 - Opera
 - Google Chrome
 - Brave
-

7.3. Técnicas de búsqueda

Antes de aprender sobre las técnicas de búsqueda que se aplican a los buscadores, tiene que saber que los buscadores son sistemas informáticos que trabajan recopilando información en Internet con el objetivo principal de mostrar la información previamente solicitada a los usuarios. Entre los diferentes buscadores que hay en Internet, destaca el buscador de Google que es el más utilizado a nivel global.

Un buscador web o motor de búsqueda es un sistema informático que busca todo tipo de información, como pueden ser imágenes, textos, vídeos, documentos, etc. En la World Wide Web, almacenándola en una enorme base de datos para arrojar la información solicitada. Es decir, los buscadores dan a los usuarios la oportunidad de encontrar en Internet la información que necesitan de una forma rápida y sencilla mediante consultas de búsqueda.

Los buscadores web funcionan mediante la orden de búsqueda con palabras clave, imágenes o voz. Primero el usuario introduce los datos, acto seguido, los motores de búsqueda responden con un listado de páginas web relacionado con el contenido a buscar. Para ello, la herramienta utiliza los comúnmente denominados robots o spiders, que rastrean todas las páginas web para crear una gran base de datos con la que proporcionar toda la información al usuario. En definitiva, el propósito general de un motor de búsqueda es poder ofrecer resultados de calidad a los usuarios que realizan una búsqueda en la red.

No todos los motores de búsqueda presentan del mismo modo los resultados de búsqueda, a continuación, se recogen los tres principales tipos de buscadores que hay en Internet según cómo recopila y arroja resultados:

Jerárquicos. Son organizados y clasifican los resultados de la búsqueda según la relevancia que tiene el sitio en el buscador web. Cuentan con una interfaz de interrogación textual y revisan las páginas web a través de sus arañas. Con ellas recopilan toda la información de los contenidos que tienen relación con la búsqueda que realiza el usuario. Toman el historial del usuario como guía para mostrar los resultados.

Metabuscadores. Recopilan la información de varios motores de búsqueda para ofrecer un resultado general de la consulta realizada, es decir, permiten buscar en varios buscadores al mismo tiempo. Una vez reciben la respuesta, se la remiten al usuario tras realizar un filtrado de los resultados que depura los repetidos y ordena los enlaces como, por ejemplo, MetaCrawler o Buscopio.

Directorios. Los directorios o índices son listas de recursos organizados por temas o categorías generales, se estructuran jerárquicamente ofreciendo enlaces directos a otras páginas o recursos de Internet. Los resultados de las solicitudes se organizan basándose en la fecha de publicación. Requieren de intervención humana para su correcto funcionamiento. Un ejemplo de este tipo de buscador es Dmoz o páginas amarillas.

7.3.1. Expresiones

La habilidad para encontrar la información que busca en Internet va a depender, en gran medida, de la precisión y efectividad con la que utilice los motores de búsqueda de los buscadores. Un motor de búsqueda es un gran índice con la mayoría de las páginas existentes en Internet. Es en este índice donde hace las búsquedas por medio de palabras o frases, obteniendo como resultado una lista de las páginas que contienen dichos datos. Aparte de las expresiones vistas anteriormente, tiene otras, como son:

(...) Es equivalente a "...". Por ejemplo (ordenadores potentes).

- **NEAR.** Lo que puede utilizar entre dos o más términos de búsqueda. Su objetivo es localizar documentos con ambos términos próximos entre sí. Por ejemplo, ordenador NEAR potente, darán páginas que contengan ordenador continuado de la palabra potente, o bien ordenador NEAR/2 potente, que dará páginas con la palabra ordenador y dos palabras más adelante, la palabra potente.
- **Fields.** Lo puede utilizar para especificar dónde localizar la búsqueda en cada documento. Así, potencia la relevancia de la búsqueda. Por ejemplo, "body: ordenador" buscará páginas que en su cuerpo tengan dicha palabra.
- **Cache.** Muestra la página almacenada en la caché en vez de recurrir al servidor de Internet que la aloja.
- **Link.** Esta etiqueta va a permitir obtener una lista de páginas que están vinculadas a la página pasada por el link.
- **Related.** Esta etiqueta permite ampliar la búsqueda a páginas similares a la indicada, y es por esto por lo que debe ir acompañada de un determinado sitio web.
- **Info.** Esta etiqueta permite conocer si la página indicada es conocida por el buscador o no.
- **Define.** Esta etiqueta permite obtener diversas definiciones desde distintos glosarios.
- **Stocks.** Esta etiqueta permite recuperar información financiera.
- **Site.** Esta etiqueta permite restringir la búsqueda a un sitio, dominio o subdominio determinado.
- **Allintitle.** Esta etiqueta va a permitir restringir la búsqueda a una o varias palabras dentro del título de la página.
- **Intitle.** Esta etiqueta permite restringir la búsqueda de una palabra o frase dentro del título de la página.
- **Intext.** Esta etiqueta permite restringir la búsqueda de una palabra determinada en el texto principal de las páginas.
- **Allintext.** Esta etiqueta permite restringir la búsqueda a diversas palabras determinadas en el texto principal de las páginas.
- **Inanchor.** Esta etiqueta permite restringir la búsqueda a páginas en las cuales el texto subrayado de enlace coincide con la palabra que le ha pasado de búsqueda, es decir, con el término de la búsqueda.
- **Allinurl.** Esta etiqueta permite restringir la búsqueda a las páginas que contienen varias palabras en la dirección del sitio web.

- **Inurl.** Esta etiqueta permite restringir la búsqueda a las páginas que contienen una determinada palabra en la dirección de su sitio web.
- **Daterange.** Esta etiqueta permite restringir la búsqueda entre un rango de fechas determinadas.
- **Filetype.** Esta etiqueta permite restringir la búsqueda a un determinado tipo de documento, por ejemplo, pdf, gif, jpg, etc. Con solo añadir la extensión del archivo a buscar en la búsqueda.

Lo ideal sería combinar todos los operadores anteriores para formar expresiones de búsqueda mucho más concisas.

7.3.2. Definiciones de búsquedas

Para realizar buenas definiciones de búsquedas va a seguir una serie de pasos:

- El tema a buscar debe ser claro y conciso.
- La definición del tema debe ser breve y concisa, es decir, utilice pocas palabras y utilice palabras clave.
- Debe evitar buscar temas generales.
- Debe dividir un tema general en varios temas concretos.

Debe contrastar la información que está buscando en varios sitios distintos, por lo tanto, debe limitar los años de búsqueda, acotar al máximo los temas buscados, conocer a ser posible los temas exactos y hacer uso de los operadores booleanos y comandos facilitados por el buscador en sí.

A continuación, va a ver una serie de problemas que se presentarán en las búsquedas y va a proponer una lista de posibles soluciones:

Problema	Solución
No encuentra información sobre el tema buscado	Possible exceso de palabras en la búsqueda de la información. Definir mejor las palabras clave de la búsqueda. Intentar probar con una o dos palabras clave de la búsqueda. Contrastar resultados con otro buscador.
Aparecen muchos documentos sobre el tema buscado	Dividir la información de la búsqueda en temas más simples. Localizar temas concretos que respondan al tópico general de la búsqueda
Aparece escasa información sobre el tema buscado	Precisar mejor el tema de búsqueda. Localizar otra palabra clave. Intentar localizar la palabra clave que es el eje sobre el que gira la búsqueda.
No entiende la información que está buscando	Intentar a partir de conocimientos previos y no de un tema en concreto.
No encuentra información comprensible para la búsqueda lanzada.	Precisar mejor el tema de búsqueda. Contrarrestar la información con otros buscadores.
No se explica el tema de búsqueda	Intentar recurrir a otros buscadores para obtener más información

7.3.3. Especificaciones

Va a citar una serie de especificaciones que debe tener presente siempre que realice una técnica de búsqueda. Dichas especificaciones son:

- Identificar los conceptos importantes del problema a investigar.
- Identificar las palabras claves que describen los conceptos anteriores.
- Determinar si existen sinónimos y términos relacionados a los conceptos básicos de investigación.
- Ingresar las palabras en minúsculas, salvo que sean nombres propios.
- Si usa términos en inglés, por regla obTendrá más resultados. En español, la cantidad de referencias que obtenga será mucho menor.
- Usar el plural en vez del singular.

7.4. Técnicas de refinamiento de búsquedas

En este epígrafe, va a ver consejos con los que puede realizar búsquedas de una manera más sencilla y eficiente. A continuación, va a ver unos consejos que facilitarán las búsquedas en Internet:

Por convenio, el sistema de búsqueda de un buscador intentará localizar las páginas que contienen todas las palabras exactas que se introducen para buscarlas. Si este procedimiento de búsqueda falla o no tiene éxito, entonces el buscador intenta localizar las páginas que contienen alguna de las palabras exactas que han sido introducidas para la búsqueda. Si esto ocurriera, lo normal es que se informe a la hora de presentar la búsqueda por pantalla.

Se pueden introducir término de búsqueda con palabras acentuadas y con el carácter “ñ”. Las búsquedas son sensibles a los acentos en las vocales. Buscar “está” produce resultados distintos a buscar “esta”. Para concluir, los resultados de las búsquedas serán iguales si introduce los términos de dicha búsqueda con mayúsculas o con minúsculas. Buscar “coche” produce los mismos resultados que buscar “COCHE”.

RECURSO MULTIMEDIA



7.4.1. Utilización de separadores

Muchas veces, se utilizan separadores para dar formato o estilo al título de su página web. Para ello, se usan los siguientes separadores |, /, -, _, etc. Por ejemplo, imagine el siguiente título en su tienda online:

“Los mejores productos para su vehículo – www.mejoresproductos.com – Productos de calidad para su vehículo”

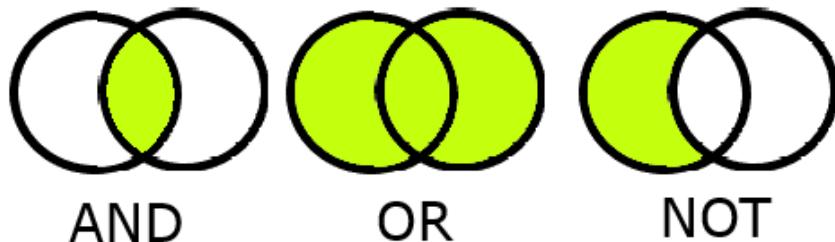
Estos separadores no afectarán de que el navegador busque por determinadas palabras, ya que, estos separadores los interpreta como delimitadores de palabras, por lo que, no los tendrá en cuenta durante la búsqueda.

Cuanta más técnicas de refinamiento se usen en las búsquedas de información, obtendrá más resultados ajustados a lo que está buscando, y no perderá el tiempo consultando páginas que no ofrecen la información que está buscando.

7.4.2 Utilización de elementos de unión

En este apartado va a ver los conjuntos de unión, que vio anteriormente, de manera gráfica. Los conjuntos de unión son AND, OR y NOT. Véalo en la siguiente imagen:

Conjuntos de unión



Como puede observar en la imagen, si utiliza una búsqueda con AND donde tuviéra dos términos a buscar, daría como resultado la intersección de ambos conjuntos.

Sin embargo, si la misma búsqueda la hiciera con OR, daría como resultado el total del conjunto de ambos términos. Por último, si la búsqueda la hiciera con NOT, solamente daría como resultado el conjunto del primer término a buscar y de ese conjunto daría el resultado que no comparte con el segundo término.

7.5. Reutilización de scripts

La reutilización de *scripts* es una manera de ganar tiempo a la hora de desarrollar una web, ya que, hay muchísimo código escrito para cualquier efecto o elemento que puede incluir en una web. También, a la hora de escribir su propio código, puede reutilizarlo, de hecho, esto es una buena práctica en programación, ya que, así ahorra recursos tanto de su máquina como en el procesamiento de datos, es decir, hace que su web sea más rápida cargando.

En este epígrafe va a ver algunos sitios donde puede buscar *scripts* gratuitos para implementar en su web para, así, ser más eficientes y rápidos a la hora de crear una web.

7.5.1. Scripts gratuitos

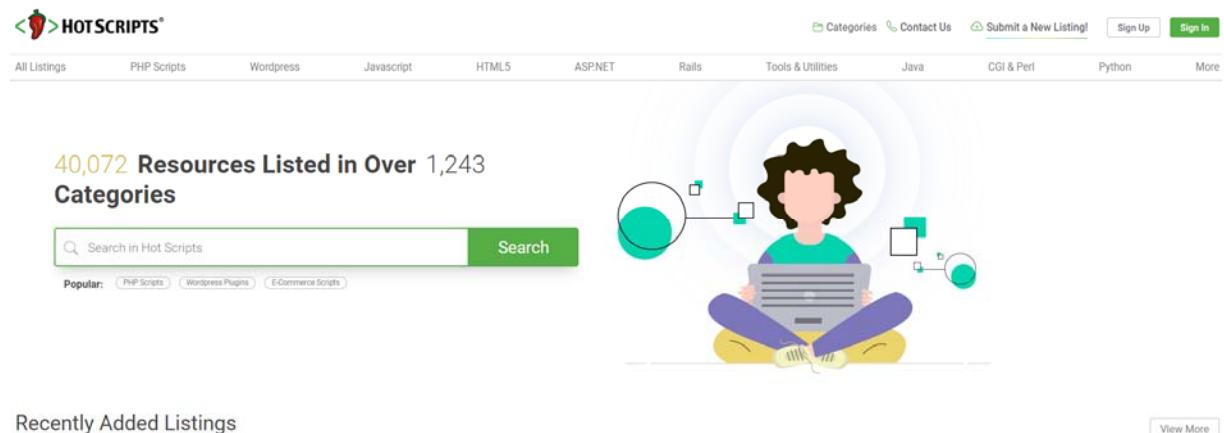
En este epígrafe va a ver una serie de sitios web en donde puede obtener *scripts* gratuitos, no solo para JavaScript, sino para cualquier lenguaje *script*. A continuación, verá un listado con páginas interesante donde puede descargar sus *scripts*:

Hscripts.com. Esta página está totalmente en inglés, pero cuenta con gran variedad de *scripts* gratuitos directos para copiar en sus documentos. Además, los scripts están divididos por categorías tales como validación de formularios, efectos de imagen, efectos de ratón, etc. Su [web](#) es la siguiente:

The screenshot shows the homepage of Hscripts.com. At the top, there's a navigation bar with links for Home, Scripts, Tutorials, Tools, Snippets, FAQ, Statistics, Paid Scripts, Web Icons, and PDF. A search bar is also at the top. On the left, there's a sidebar with categories like Free Scripts (PHP, JavaScript, JQuery, CSS3, JSP), Online Tutorials (HTML, HTML5, CSS, C++, Excel2007 Functions), and Web Tools (Menu Builder, Button Creator, Color Picker, Image Sprites Generator). Below this, there's a section for Web Icons (Cliparts, Logos, Country Maps, National Emblems, Background). In the center, there's a "Most Popular Scripts" sidebar with links to Banner Ad, Star Rating, Banner Rotate, Slideshow Javascript, Multiple Star Rating, Advanced Newsticker, Add Comments, Petty Cash Management, User Login & Registration, and Review System. There's also a "Latest Snippets" sidebar with links to What is hashchange event in jquery?, Time to ping, Hide Console Window, GetUptime() .net, and Toggle a service. Finally, there's a "Latest News" sidebar with links to Facebook Login, URL Shortner, Perl New Year, and Php Weather Report.

Hscripts.com

Hotscripts.com. Esta página es similar a la anterior. Está completamente en inglés, pero también es muy fácil buscar scripts en ella, ya que, también está dividida por categorías. Su [web](#) es la siguiente:



Efectosjavascript.com. Esta página, además de contener código para scripts, también contiene tutoriales, manuales y ejemplos explicados con el código que ofrece. Esta web se dedica totalmente a los *scripts* desarrollados con JavaScript y en ella puede ver las diferentes categorías donde puede encontrar diversos tipos de *scripts*. Su [web](#) es la siguiente:

Efectos Javascript

Cerrar Menu

MEJORADO POR Google



< Volver

Efectos JavaScript

Alertas

Colección de Efectos y trucos en Javascript para copiar y pegar

Botones

Código fuente para el webmaster. Todos los scripts están con sus respectivos ejemplos para hacerlos más fácil de entender. Free Javascript.

Popups

Algunos de los Efectos presentados no funcionan bien con Mozilla, la totalidad de ellos si funcionan con internet explorer.

Relojes

Puedes Utilizarlos en el desarrollo de tu sitio web, los más utilizados son los popups o ventanas emergentes, puedes crearlas con javascript, cargando automáticamente al entrar o salir de una web o haciendo click en algún hipervínculo.

Punteros

Todos los efectos javascript pueden ser modificados en su formato color, tamaño, ubicación, velocidad de carga o descarga.

Barra título

Consejos: Al copiar un javascript como los presentados en este sitio, modifícalo en block de notas o wordpad antes de copiarlo a tu sitio. Si no estás seguro donde pegarlos puedes verlo en la sección "como se pega un script".

Barra estado

Fondos

Botones Flash Paypal

Uterra.com. Esta página ofrece una amplia información sobre validaciones y expresiones regulares en JavaScript, además, ofrece muchos *scripts* diversos, código HTML, estilos CSS, múltiples herramientas, etc. Su [web](#) es la siguiente:

The screenshot shows the homepage of Notasweb.com. At the top, there's a banner with the text "DISEÑO Y PROGRAMACIÓN WEB. TODO LO NECESARIO PARA TU SITIO WEB." over a background image of a bridge at night. Below the banner is a navigation bar with links: INICIO (highlighted in grey), HTML, CSS, JAVASCRIPT, PHP, MYSQL, HERRAMIENTAS, and JUEGOS. On the left, there's a "BUSCADOR" (Search) section with a search input field containing "En Uterra.Com" and a "Buscar" button. A "CONTENIDOS" (Contents) sidebar lists links: TU WEB DESDE CERO, CÓDIGO HTML, CÓDIGO CSS, CÓDIGO JAVASCRIPT, CÓDIGO PHP, and BASES DE DATOS. The main content area features a section titled "APLICACIONES WEB A MEDIDA." with the subtext "Desarrollamos aplicaciones Web para necesidades específicas." It includes a bulleted list of services: Soporte e-learning, Sistemas de recogida de datos, and Encuestas y cuestionarios. There are also links for "+ info" and "AQUÍ". On the right, a vertical column titled "CÓDIGO" lists numbered items from 1 to 13, each with a corresponding link: 1.- Como datos, 2.- Regis, 3.- Lectu una base, 4.- Pagin MySQL, 5.- Compr datos, 6.- Compr MySQL, 7.- Compr una base, 8.- Borr datos, 9.- Pagin, 10.- Borr contend, 11.- Extr datos pa, 12.- Extr base de, 13.- Con.

Notasweb.com. En esta página, además de encontrar código en diferentes lenguajes de programación, también tiene grupos donde puede participar y recibir ayuda. Además, algo muy interesante es que dentro de estos grupos hay personas que buscan diseñadores, desarrolladores web, etc. Así, que es una página bastante interesante para dar a conocer como desarrolladores web. Su [web](#) es la siguiente:

Acceder | Registrarse | Contacta con nosotros

 NotasWeb.com
La red social de los desarrolladores web.

Buscador: Buscar

PHP CSS / Diseño JavaScript AJAX SQL .mobi .NET Flash SEO Optimización Servidores

[desarrollo de aplicaciones móviles iOS y Android.](#)
[desarrollo de videojuegos para móviles, PC y consolas.](#)
[expertos en desarrollo web con Symfony y Wordpress.](#)
[desarrollo de aplicaciones de Realidad Virtual y Realidad Aumentada.](#)

Últimos artículos

 [Gradiente lineal](#)
Tags: gradiente degradado fondo
background-color: #bbb; background-image: -moz-linear-gradient(top, #aaa, #ccc); background-image: ...

 [Codigo para dejar visible un Div específico](#)
Tags: div-ocultar-mostrar
<head> <script type="text/javascript" ...>

 [Limitar número de palabras de un texto.](#)
Tags: cadena-deTexto palabras limitar
function limitarPalabras(\$cadena, \$longitud) ...

 [Exportar Base de datos a Excel](#)
Tags: excel php exportar base-de-datos tablas
Gente, hoy les traigo algo muy facil de usar y muy necesario cuando trabajamos con reportes desde una base de ...

Grupos destacados

[Ver todos los grupos](#)

 **Busca Colaboración**
43 miembros
21 mensajes

 **Principiantes**
35 miembros
18 mensajes

 **PHP**
44 miembros
9 mensajes

 **Aplicaciones Facebook**
11 miembros
2 mensajes

Últimos temas en los grupos

 [Busco socio para desarrollar app para IOS y Android](#)
Grupo: Busca Colaboración
Busco socio para desarrollar una aplicación para iphone y plataforma android. Conocimientos sobre aplicar

 [Busco diseñador](#)
Grupo: Busca Colaboración
Hola, busco diseñador HTML/CSS para proyecto de red social. Yo pongo la programación. La explotación ...

 [Busco Diseñador HTML simple y sencillo](#)
Grupo: Busca Colaboración
Buenas mi nombre es jose, y estoy en la creacion de una pequeña comunidad tengo hecho el nucleo listo (PHP, ...)

 [Busco desarrollador y diseñador para web corporativo](#)
Grupo: Busca Colaboración

Artículos más vistos

7.5.2. Generalización de códigos

Lo bueno de programar con JavaScript es que puede crear el código *script* dentro de un archivo con su extensión .js. Es decir, utilice un editor de texto cualquiera y, simplemente, comenzará a escribir el código en JavaScript sin tener que utilizar las etiquetas *<script>* y *</script>*. Dicho archivo lo guarda con el nombre que desee y con la extensión .js. A continuación, va a ver cómo insertar ese código en HTML teniendo su script dentro de su archivo .js:

```
<!DOCTYPE html>

<html>

<head>

<script type="text/JavaScript" src="nombreArchivo.js">

</script>

</head>
```

```
<body>  
  
<script type="text/JavaScript" src="nombreArchivo.js">  
  
</script>  
  
</body>  
  
</html>
```

Esta es la manera de incrustar el código JavaScript desde un archivo externo a su documento HTML. Si observa el ejemplo anterior, puede hacer uso de esta técnica tanto en la parte del *head* del documento HTML, como en la parte del *body*. Como principal ventaja de esta técnica, tiene que citar que puede reutilizar el código en las páginas que lo necesite, simplemente lo llama de la forma anterior y lo Tendrá disponible. Gracias a este método, solo Tendrá que programar una vez dicho código para poder utilizarlo cuantas veces quiera.

RESUMEN

En esta Unidad Formativa ha aprendido, la lógica de la programación, cómo interpreta un ordenador las instrucciones que le da mediante un lenguaje de programación.

Para facilitar el trabajo de programar, ha aprendido a hacer ordinogramas donde puede ver de manera gráfica las instrucciones a seguir. Además, ha aprendido junto con los ordinogramas a utilizar pseudocódigo, que es un paso más allá de un ordinograma, pero sin ser un código de programación real, sino que está más bien creado para que sea fácil para la comprensión humana.

Por otro lado, ha visto qué son los objetos, cómo funcionan, qué atributos tienen y cómo crearlos.

Ha visto cómo crear las mismas instrucciones con diferentes códigos de programación. Por otra parte, ha profundizado en el lenguaje JavaScript donde ha visto cada una de sus características, cómo relaciones JavaScript con HTML y cómo utilizar su sintaxis.

Ahora, ya sabe lo que son los atributos, métodos, eventos y funciones. También la diferencia que hay entre los tipos de script diferidos, híbridos e inmediatos. Además, ha aprendido lo que son las variables e identificadores y los tipos que hay de datos. Para qué se utilizan cada uno de ellos y cómo.

Ha aprendido los tipos de operadores que existen y cómo utilizarlos para crear las diferentes instrucciones en el lenguaje de programación.

Otra de las cosas que ha aprendido es a desarrollar scripts de manera que hagan lo que quiere que haga el mismo. Junto con el desarrollo de script, ha aprendido a solucionar errores que pueden ocurrir durante la programación del lenguaje o en la ejecución del programa, en este caso, en la interpretación por parte del navegador.

Además, ha aprendido lo que es la jerarquía de los objetos y a manipular el DOM. Gracias a los eventos que ha aprendido, puede crear páginas mucho más dinámicas e interactivas. Ha visto los tipos de eventos que existen y cómo crearlos de diferentes maneras.

Por último, ha aprendido a utilizar scripts de terceros que puede encontrar en páginas de scripts oficiales y muy conocidas. También, ha refinado la manera de buscar en Internet gracias a las diferentes expresiones de búsqueda.

GLOSARIO

API: Una API es un conjunto de definiciones y protocolos que se utilizan para desarrollar e integrar el software de las aplicaciones. API significa interfaz de programación de aplicaciones. Las API permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados.

Atributo: Un atributo es una propiedad o característica que se puede asignar a un objeto o elemento. Mediante el uso de atributos se pueden asignar valores específicos a ciertos elementos. Por ejemplo, un atributo sería el color, el tamaño, o el material de un objeto.

Backend: Es la parte del desarrollo web que se encarga de que toda la lógica de una página web funcione. Se trata del conjunto de acciones que pasan en una web, pero no ve cómo, por ejemplo, la comunicación con el servidor.

Booleano: Los datos y expresiones booleanas son datos o expresiones que solo pueden tener dos valores verdadero o falso. Este tipo de datos y expresiones se suelen utilizar como estructuras de control dentro de un programa. Por ejemplo, la estructura if es una expresión booleana que da como resultado un dato booleano.

Condición: Es toda sentencia de la cual se puede determinar su verdad, true, o falsedad, false. En su gran mayoría, son comparaciones. Por ejemplo, $4 > 5$; esta sentencia es una condición porque tiene resultado verdadero o falso; en este caso es falso, ya que, 4 no es mayor que 5.

Framework: Un framework es una estructura conceptual y tecnológica de soporte definido, normalmente con módulos de software concretos, que puede servir de base para la organización y desarrollo de software. Como ejemplo de frameworks puede encontrar .NET, Django, Angular, React, etc.

Frontend: Es la parte del software que interactúa con los usuarios, es decir, lo que comúnmente llama lado del cliente. El frontend se desarrolla principalmente con los lenguajes HTML, CSS y JavaScript.

Función: Una función es lo mismo que un método, la única diferencia es que el método es una función que pertenece a una clase y la función es un trozo de código libre que se ejecuta cuando es invocada. Por ejemplo, una función sería abrirVentana ()�.

Hijo: La clase hijo es la clase que hereda de la clase padre. En la clase hijo, también, se pueden añadir atributos y funciones diferentes a los de la clase padre. A esta clase también puede denominarla subclase.

Instrucciones E/S: A estas instrucciones también se las denomina como instrucciones I/O, o instrucciones de entrada y salida de datos. Una instrucción de entrada consiste en asignar a una o más variables, uno o más valores recibidos desde el exterior. En cambio, una instrucción de salida consiste en llevar hacia el exterior los valores obtenidos de la evaluación de una lista de expresiones.

Iteración: La iteración es el acto de repetir un proceso para generar una secuencia de resultados, tanto ilimitada como limitada, con el objetivo de acercarse a un propósito o resultado deseado. En el contexto de la informática, la iteración es un bloque de construcción estándar de algoritmos. Por ejemplo, el bucle for es un bucle iterativo.

Método: Es una subrutina cuyo código es definido en una clase y puede pertenecer tanto a una clase, como es el caso de los métodos de clase o estáticos, como a un objeto, como es el caso de los métodos de instancia.

Objeto: Un objeto es una unidad dentro de un programa informático que tiene un estado y un comportamiento. Es decir, tiene una serie de datos almacenados y tareas que realiza con esos datos en el tiempo de ejecución. Como ejemplo, puede pensar en una botella, es un objeto con sus características y sus funciones.

Padre: La clase padre es la que transmite, de manera automática, su código a las clases hijas. Esto es el concepto de la herencia en programación. A la clase padre también se le denomina superclase.

Plug-in: Son pequeños programas complementarios que amplían las funciones de aplicaciones web y programas de escritorio. Por lo general, cuando instala un *plug-in*, el software en cuestión adquiere una nueva función. Por ejemplo, puede encontrar diversos *plug-ins* para navegadores web como Adblocker que sirve para que no salten anuncios en ventanas emergentes.

Propiedad: Una propiedad es una variable que existe en una clase y que puede contener cualquier tipo de dato. Puede usar propiedad o atributo de manera indistinta en el ámbito de la programación.

Script: Un script es un documento que contiene instrucciones escritas en códigos de programación. El script es un lenguaje de programación que ejecuta diversas funciones en el interior de un programa. Por ejemplo, JavaScript es un lenguaje de script y un documento ejemplo.js es un script.

Secuencia: Término también conocido como estructura secuencial, es aquella en la que una instrucción o acción sigue a otra en secuencia. En este tipo de programación se presentan operaciones de inicio a fin, inicialización de variables, operaciones de asignación, cálculo, summarización, entre otras.

URL: Son siglas en inglés de Uniform Resource Locator, que en español significa localizador uniforme de recursos. Como tal, el URL es la dirección específica que se asigna a cada uno de los recursos disponibles en la red con la finalidad de que estos puedan ser localizados o identificados. Por ejemplo, una url sería <https://www.google.es>.

Variable: Una variable es donde se guardan y se recuperan datos que se utilizan en un programa. Cada variable ocupa un espacio de memoria, es decir, puede imaginar este concepto como si cada variable fuera un cajón de donde puede meter y sacar elementos para almacenarlos o usarlos.