EVIDENCIA DE CONOCIMIENTO: GA3-220501093-AA3-EV01 BASES TEÓRICAS DE ESTRUCTURAS DE ALMACENAMIENTO EN MEMORIA

Sor Junny Londoño Rivera

Aprendiz

Donaldo Andrés Beltrán Prieto

Instructor

Servicio Nacional de Aprendizaje-SENA

ANALISIS Y DESARROLLO DE SOFTWARE (2627038)

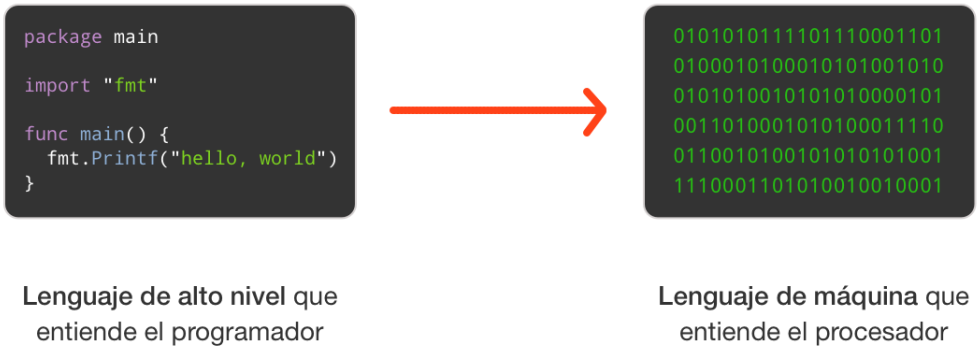
Regional Quindío.

2023

TALLER.

Para su desarrollo es importante la lectura del componente formativo. Elaborar un documento en el cual se registren los siguientes elementos:

1.Principales diferencias entre los lenguajes compilados e interpretados.



Lenguajes Compilados	Lenguaje Interpretados
Un lenguaje compilado genera una fila binario no modificable.	Un lenguaje interpretado es escrito en un lenguaje de programación definido y modificable en cada momento.
Las instrucciones vienen enviadas directamente al procesador.	Las instrucciones deben traducirse antes de llegar al procesador.
Se requieren dos pasos separados para ejecutar el programa desde el código fuente.	El código fuente se ejecuta a través de un solo comando.
Dado que el programa ya se ha traducido, la ejecución es más rápida.	El programa debe traducirse cada vez aumentando el tiempo de ejecución.
El programa solo se puede ejecutar en ciertas máquinas y sistemas operativos.	El programa funciona en todas las máquinas y sistemas.
Los errores de compilación, impiden que se compile el código.	Los errores de compilación son visibles solo si se inicia el programa.
Ejemplos de lenguaje compilados son: C, C++, Delphi.	Ejemplos de lenguaje interpretados son: Python, JavaScript, Perl, PHP

2. Características principales de JavaScript

Sus Principales características son:

Simplicidad. Posee una estructura sencilla que lo vuelve más fácil de aprender e implementar.

Velocidad. Se ejecuta más rápido que otros lenguajes y favorece la detección de los errores.

Versatilidad. Es compatible con otros lenguajes, como: PHP, Perl y Java. Además, hace que la ciencia de datos y el aprendizaje automático sean accesibles.

Popularidad. Existen numerosos recursos y foros disponibles para ayudar a los principiantes con habilidades y conocimientos limitados.

Carga del servidor. La validación de datos puede realizarse a través del navegador web y las actualizaciones solo se aplican a ciertas secciones de la página web.

Actualizaciones. Se actualiza de forma continua con nuevos frameworks y librerías, esto le asegura relevancia dentro del sector.

Tomado de: https://ceeivalencia.emprenemjunts.es/?op=8&n=28660

Otras Características de JavaScript:

Lenguaje del lado del cliente**:** Cuando se dice que un lenguaje es del lado del cliente, nos referimos a que se ejecuta en la máquina del propio cliente a través de un navegador. Algunos de estos lenguajes son el propio JavaScript, HTML, CSS o Java.

Esta categoría de lenguajes se diferencia de la otra gran categoría: los lenguajes del lado del servidor. Estos lenguajes se ejecutan e interpretan por el propio servidor y necesitan un tratamiento antes de mostrarlos al usuario final. Algunos de los lenguajes de programación del lado del servidor más conocidos son PHP, ASP o PERL.

Lenguaje orientado a objetos: JavaScript es un lenguaje orientado a objetos. Que un lenguaje esté orientado a objetos quiere decir que utiliza clases y objetos como estructuras que permiten organizarse de forma simple y son reutilizables durante todo el desarrollo. Otros lenguajes orientados a objetos son Java, Python o C++.

De tipado débil o no tipado: Que un lenguaje sea de tipado débil quiere decir que no es necesario especificar el tipo de dato al declarar una variable. Esta característica supone una gran ventaja a la hora de ganar rapidez programando, pero puede provocar que cometamos más errores que si tuviéramos esa restricción que poseen los lenguajes de tipado fuerte como C++ o Java.

De alto nivel: Que JavaScript sea un lenguaje de alto nivel significa que su sintaxis es fácilmente comprensible por su similitud al lenguaje de las personas. Se le llama de "alto nivel" porque su sintaxis se encuentra alejada del nivel máquina**,** es decir, del código que procesa una computadora para ejecutar lo que nosotros programamos.

Un lenguaje de alto nivel como JavaScript permite que su barrera de entrada y su curva de aprendizaje se acorte drásticamente. Un ejemplo podría ser que la sentencia condicional empiece por "IF" que significa "si…" en inglés, permitiendo asociar rápidamente su funcionamiento y significado. Otro lenguaje de alto nivel muy utilizado y uno de los mejores para iniciarse en programación por esta característica es Python.

Tomada de: https://www.miteris.com/blog/que-es-javascript-caracteristicas-librerias/

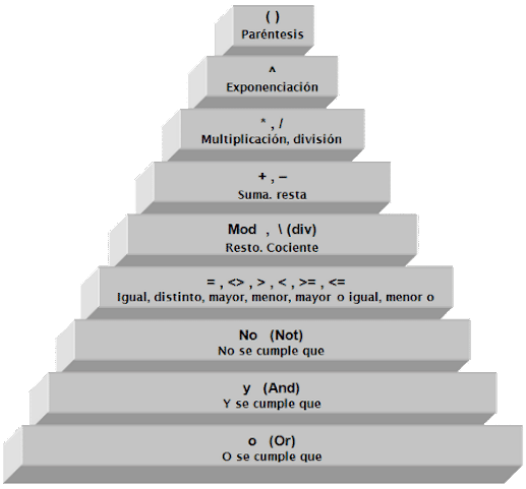
3. Tipos de datos primitivos y uso en JavaScript

TIPO DE DATOS PRIMITIVOS	USO EN JAVASCRIPT
Undefined	Indeterminado o indefinido
Boolean	Tipo booleano, los valores posibles son true o false
Number	Números enteros o decimales
String	Cadena de texto
BigInt	Números enteros grandes
Symbol	Referencia a otros datos

Otros tipos de datos más complejos o abstractos de datos:	
Null	Valor nulo o ausencia de un valor
Object	Tipo estructural especial que no es de datos, pero para cualquier instancia de objeto construido que también se utiliza como estructuras de datos (new Object, new Array, new Map, new Set, new WeakMap, new WeakSet, new Date y casi todo lo hecho con la palabra clave new).
Function	Una estructura sin datos, aunque también responde al operador t.
Cuando se declaran variables se debe considerar los siguientes tipos, ya se han usado dos (2) de ellas:	
Var	Declara una variable, opcionalmente la inicia a un valor.
Let	Declara una variable local con ámbito de bloque, opcionalmente la inicia a un valor.
Const	Declara un nombre de constante de solo lectura y ámbito de bloque.

Tomado de https://sena.territorio.la/content/index.php/institucion/Titulada/institution/SENA/Tecnologia/228118/Contenido/OVA/CF14/index.html#/curso/tema3

4. Operadores en JavaScript



Al trabajar con Javascript (o con cualquier lenguaje de programación), es muy habitual hacer uso de los llamados operadores. Se trata de unos símbolos que nos permitirán hacer una serie de operaciones con uno o más operadores (generalmente, números). Sin embargo, esto se entiende mejor con ejemplos, por lo que vamos a desglosar los operadores básicos en una serie de bloques y a explicarlos para que se entienda lo mejor posible:

Aritméticos: Operadores para realizar operaciones matemáticas.
Asignación: Operadores para guardar información en variables.
Unarios: Operadores que se utilizan con un sólo operando.
Comparación: Operadores para realizar comprobaciones.
Binarios: Operadores a bajo nivel (a nivel de bits).
Operadores aritméticos
En primer lugar, vamos a centrarnos en los operadores aritméticos, que son los operadores que utilizamos para realizar operaciones matemáticas básicas. Los más sencillos son los cuatro primeros, que forman parte de las operaciones matemáticas básicas habituales:

Nombre	Operador	Descripción
Suma	`a + b`	Suma el valor de **a** al valor de **b**.
Resta	`a - b`	Resta el valor de **b** al valor de **a**.
Multiplicación	`a * b`	Multiplica el valor de **a** por el valor de **b**.
División	`a / b`	Divide el valor de **a** entre el valor de **b**.
Módulo	`a % b`	Devuelve el resto de la división de **a** entre **b**.
Exponenciación	`a ** b`	Eleva **a** a la potencia de **b**, es decir, a^b. Equivalente a `Math.pow(a, b)`.

Menos frecuentes suelen ser el módulo o la exponenciación, sin embargo, muy útiles en algunas situaciones.

Operador módulo

Observa el siguiente ejemplo donde utilizamos la operación módulo para limitar el índice:

```
const numbers = [10, 20, 30, 40, 50, 60, 70, 80];

for (let i = 0; i < numbers.length; i++) {
  const mod = i % 2;
  console.log(numbers[i], numbers[mod]);
}
```

Observa que en el **console.log()** estamos mostrando **numbers[i]** y luego **numbers[mod]**. Si ejecutas este código, comprobarás que en el primer caso, se van mostrando los valores del array **numbers**, es decir, **10, 20, 30...** y así hasta **80**. Sin embargo, en el segundo caso del **console.log()**, donde utilizamos **mod** como índice, se repiten los dos primeros: **10, 20, 10, 20, 10, 20...**.

Esto ocurre porque en la línea **const mod = i % 2** hemos hecho el módulo sobre 2 y no estamos dejando que ese índice crezca más de **2**, los valores que va a tomar **mod** en el bucle serán **0, 1, 0, 1, 0, 1...**, puedes comprobarlo cambiando el **console.log()** y mostrando los valores **i** y **mod**.

Operador de exponenciación

En el caso de la exponenciación, simplemente podemos utilizar el operador ******. Antiguamente, la exponenciación se hacía a través del método **Math.pow()**, sin embargo, ahora podemos hacerlo a través de este operador, con idéntico resultado:

```
const a = 2;
const b = 5;

console.log(Math.pow(a, b));    // 32
console.log(a ** b);            // 32
console.log(a * a * a * a * a); // 32
```

En este caso, estamos haciendo la operación 2^5, es decir, **2 * 2 * 2 * 2 * 2**.

Operadores de asignación

Al margen de los anteriores, también tenemos los **operadores de asignación**. Estos operadores nos permiten asignar información a diferentes constantes o variables a través del símbolo =, lo cuál es bastante lógico pues así lo hacemos en matemáticas.
No obstante, también existen ciertas **contracciones** relacionadas con la asignación que nos permiten realizar operaciones de una forma más compacta. Son las siguientes:

Nombre	Operador	Descripción
Asignación	`c = a + b`	Asigna el valor de la parte derecha (*en este ejemplo, una suma*) a `c`.
Suma y asignación	`a += b`	Es equivalente a `a = a + b`.
Resta y asignación	`a -= b`	Es equivalente a `a = a - b`.
Multiplicación y asignación	`a *= b`	Es equivalente a `a = a * b`.
División y asignación	`a /= b`	Es equivalente a `a = a / b`.
Módulo y asignación	`a %= b`	Es equivalente a `a = a % b`.
Exponenciación y asignación	`a **= b`	Es equivalente a `a = a ** b`.

Por ejemplo, realizar la asignación **a = a + b** sería exactamente lo mismo que escribir **a += b**, sólo que esta última está escrita de una forma más resumida.

Operadores unarios

Los operadores unarios son aquellos que en lugar de tener **dos operandos**, como los anteriores, sólo tienen **uno**. Es decir, se realizan sobre un sólo valor almacenado en una variable.

Nombre	Operador	Descripción
Incremento	a++	Usa el valor de a y luego lo incrementa. También llamado **postincremento**.
Decremento	a--	Usa el valor de a y luego lo decrementa. También llamado **postdecremento**.
Incremento previo	++a	Incrementa el valor de a y luego lo usa. También llamado **preincremento**.
Decremento previo	--a	Decrementa el valor de a y luego lo usa. También llamado **predecremento**.
Resta unaria	-a	Cambia de signo (niega) a a.

Una tarea muy común en programación es la de incrementar variables. Por ejemplo, si hacemos un **a = a + 1**, lo que estamos haciendo es incrementar el valor de **a** un número más grande del que ya poseía. Para hacer esto mucho más fácil y cómodo, podemos escribir **a++** que es equivalente a lo anterior. Sin embargo, hay un pequeño matiz, que es lo que se denomina **preincremento** y **postincremento**.

Para ver las diferencias clave del **preincremento** y el **postincremento** observa el siguiente ejemplo:

```
let a = 0;

while (a < 5) {
  console.log(a, a++, a);
}
```

Observa que en el **console.log()** primero usamos la **a**, luego hacemos el **postincremento** (*es decir, la usamos y luego incrementamos*), y finalmente la volvemos a mostrar por consola. Prueba a reemplazar el **a++** por **++a** y hacer el mismo ejemplo. Esto nos dará el siguiente resultado:

```
a a++ a
- --- -
0  0  1
1  1  2
2  2  3
3  3  4
4  4  5
```

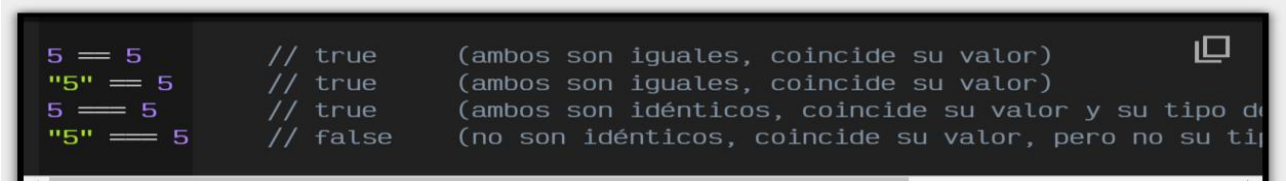
```
a ++a a
- --- -
0  1  1
1  2  2
2  3  3
3  4  4
4  5  5
```

Operadores de comparación

Los operadores de comparación son aquellos que utilizamos en nuestro código (*generalmente, en el interior de un **if**, aunque no es el único sitio donde podemos utilizarlos*) para realizar comprobaciones. Estas expresiones de comparación devuelven un booleano con un valor de **true** o **false**.

Nombre	Operador	Descripción
Operador de igualdad ==	a == b	Comprueba si el **valor** de a es igual al de b. **No comprueba tipo de dato**.
Operador de desigualdad !=	a != b	Comprueba si el **valor** de a no es igual al de b. **No comprueba tipo de dato**.
Operador mayor que >	a > b	Comprueba si el valor de a es mayor que el de b.
Operador mayor/igual que >=	a >= b	Comprueba si el valor de a es mayor o igual que el de b.
Operador menor que <	a < b	Comprueba si el valor de a es menor que el de b.
Operador menor/igual que <=	a <= b	Comprueba si el valor de a es menor o igual que el de b.
Operador de identidad ===	a === b	Comprueba si el **valor y el tipo de dato** de a es igual al de b.
Operador no idéntico !==	a !== b	Comprueba si el **valor y el tipo de dato** de a no es igual al de b.

Recuerda que en Javascript, hay diferencia entre == (*igualdad*) y === (*identidad*). Mientras que el primero sólo comprueba el valor de la comparación, el segundo comprueba **el valor y el tipo de dato** de la comparación. La diferencia se entiende muy fácil con el siguiente ejemplo:

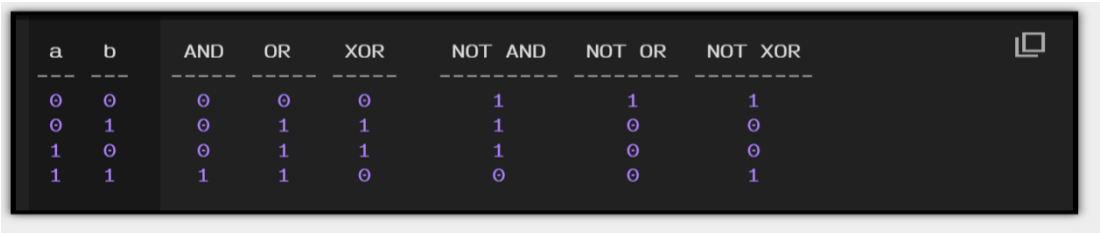


Operadores binarios

Aunque en Javascript no es algo que se utilice demasiado, existen los denominados **operadores a nivel de bit**. Se trata de una serie de operadores que nos permiten realizar operaciones básicas trabajando a nivel binario, donde los operandos solo pueden tomar valores de **0** o **1**:

Nombre	Operador	Descripción
Operador **AND**	`a & b`	Devuelve `1` si ambos operandos son `1`.
Operador **OR**	`a | b`	Devuelve `1` si al menos un operando es `1`.
Operador **XOR** (OR exclusivo)	`a ^ b`	Devuelve `1` si ambos operandos son diferentes.
Operador **NOT** (unario)	`~a`	Invierte los bits del operando (por ejemplo, 000101 pasa a 111010). Trunca a 32 bits.
Operador **LEFT SHIFT**	`a << b`	Desplazamiento de bits hacia la izquierda. Ej: `11` (3) pasa a `110` (6).
Operador **RIGHT SHIFT**	`a >> b`	Desplazamiento de bits hacia la derecha. Ej: `11` (3) pasa a `1` (1).
Operador **RIGHT SHIFT** sin signo	`a >>> b`	Desplazamiento de bits hacia la derecha, como un operador sin signo.

Por ejemplo, los tres primeros suelen ser los más habituales, donde podríamos crear las llamadas **tablas de verdad**, sin embargo, también podemos combinarlo con el NOT y conseguir variaciones:



a	b	AND	OR	XOR	NOT AND	NOT OR	NOT XOR
0	0	0	0	0	1	1	1
0	1	0	1	1	1	0	0
1	0	0	1	1	1	0	0
1	1	1	1	0	0	0	1

5. Apoye todas las argumentaciones con imágenes ilustrativas y no olvide referenciar las fuentes de información utilizadas.

https://lenguajejs.com/javascript/introduccion/operadores-basicos/

https://sena.territorio.la/content/index.php/institucion/Titulada/institution/SENA/Tecnologia/228118/Contenido/OVA/CF14/index.html#/curso/tema3

https://www.miteris.com/blog/que-es-javascript-caracteristicas-librerias/