Aprendiz: Rafael Dario Escalante Sandoval

Instructor: Andrés Rubiano Cucarían

Bases teóricas de estructura de almacenamiento en memoria

Análisis y desarrollo de software ficha 2977466

Contenido

Introducción

Programar es el proceso de crear software escribiendo, probando, depurando y dando mantenimiento a las instrucciones del computador en un lenguaje de programación. A este conjunto de instrucciones se les denomina código fuente del software creado.

Existen centenares de lenguajes de programación, muchos de ellos fueron creados para dar respuesta a problemas particulares o máquinas específicas. No es de interés ahora discutir todos los tipos de lenguajes existentes, sino, más bien, conocer aquellos que pueden ser interpretados por computadores, smartphones o que pueden ser empleados para proveer servicios informáticos a través de internet y su clasificación más general.

Un lenguaje de programación es diferente al lenguaje de códigos que puede entender la máquina (lenguaje de máquina). Los lenguajes de programación pueden dividirse en dos categorías: lenguajes interpretados o lenguajes compilados.

Principales diferencias entre los lenguajes compilados e interpretados

ASPECTO	LENGUAJES COMPILADOS	LENGUAJES INTERPRETADOS
Proceso de ejecución	El código fuente se traduce completamente a código máquina antes de ejecutarse.	El código fuente se ejecuta línea por línea por un intérprete.
Velocidad de ejecución	Más rápido, ya que el código ya está traducido a código máquina antes de la ejecución	Generalmente más lento porque la traducción ocurre en tiempo real.
Revisión de errores	Los errores se detectan antes de la ejecución durante la fase de compilación.	Los errores se detectan en tiempo de ejecución, mientras se ejecuta el código.
Generación de código	Produce un archivo binario o ejecutable después de la compilación (por ejemplo, .exe).	No genera un archivo binario; el código fuente es ejecutado directamente.

Dependencia del Sistema	El código compilado es específico para la plataforma en la que fue compilado.	El código es más portátil, ya que el intérprete puede adaptarse a diferentes plataformas.
Modificación del Código	Para aplicar cambios en el código, es necesario recompilar todo el programa.	Los cambios pueden ejecutarse inmediatamente sin recompilar, solo reiniciando el intérprete.
Uso de Recursos	Mayor uso de recursos en la fase de compilación, pero menor en ejecución.	Usa más recursos durante la ejecución, ya que la interpretación ocurre en tiempo real.
Ejemplos de lenguajes	C, C++, Rust, Go	Phyton, javascript, Ruby, PHP

Características principales de JavaScript

¿Qué es JavaScript?

JavaScript es un lenguaje de programación, de secuencias de comandos, capaz de aportar soluciones eficaces en la mayoría de los ámbitos de la tecnología. Te permite crear contenido de actualización dinámica, controlar multimedia, animar imágenes, etc.

¿Para qué se utiliza JavaScript?

Los usos más importantes de JavaScript son los siguientes:

- Desarrollo de sitios web del lado del cliente (front end, en el navegador).
- Desarrollo de aplicaciones para dispositivos móviles, híbridas o que compilan a nativo.
- Construcción de servidores web y aplicaciones de servidor.
- Desarrollo de aplicaciones de escritorio para sistemas Windows, Linux y Mac.
- Desarrollo de juegos.

JavaScript: características y beneficios

- **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 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.

¿Cuáles son las desventajas de JavaScript?

- **Compatibilidad con los navegadores**. Los diferentes navegadores web interpretan el código JavaScript de forma distinta. Por lo tanto, necesitarás probarlo en todos los navegadores populares, incluyendo las versiones más antiguas.
- **Depuración**. Aunque algunos editores de HTML admiten la depuración, son menos eficaces que otros editores. Encontrar el problema puede ser un reto, ya que los navegadores no muestran ninguna advertencia sobre los errores.

Tipos de datos primitivos y uso en JavaScript

¿Qué son los Tipos de Datos Primitivos?

Los tipos de datos primitivos en JavaScript **son aquellos que no poseen métodos ni propiedades**. Además, los valores asignados con estos tipos de datos **son inmutables**, lo que quiere decir que después de asignar una variable a un valor primitivo, si deseas cambiar su valor necesitaras reasignarle un valor nuevo, ya que su valor inicial no puede ser modificado, simplemente se substituye con el nuevo valor.

Como se menciona anteriormente, tenemos siete (7) tipos primitivos en JavaScript por el momento: **string, number, booleano null, undefined, Symbol y bigint**.

JavaScript nos permite saber el tipo de una variable o valor con el método **typeof**. Éste método siempre te devolverá **string** con el nombre del tipo de valor que le hayas pasado:

string

Te sirve para describir cualquier cadena de texto. Tienes tres diferentes maneras de representarlo:

- con comillas simples (**'**)
- con comillas dobles (**"**)
- con "backticks" (`)

```
const nombre = "Horacio"
const nombre = "Horacio"
const nombre = `Horacio`
```

number

A diferencia de otros lenguajes de programación, en JavaScript solo hay una manera de representar cualquier tipo de número, tanto como números enteros como números decimales (mas adelante tocaremos el tema de **bigInt**)

```
typeof 42 // 'number'
typeof 12.2 // 'number'
typeof -24 // 'number'
```

Numero Especiales

Además de **Number.MAX_SAFE_INTEGER**, JavaScript también tiene otros números especiales:

- **Number.MAX_SAFE_INTEGER**
- **Number.MIN_SAFE_INTEGER**
- **Number.MAX_VALUE**
- **Number.MIN_VALUE**
- **Infinity**
- **Infinity**
- **0**
- **-0**
- **NaN**: Sí, **NaN** es de tipo numero

Tenemos estos números disponibles porque JavaScript necesita una manera de representar cualquier operación matemática que quieras hacer, incluso aunque sea "imposible" o "errónea", como por ejemplo **1 / 0** (el resultado es **Infinity**).

boolean

Este tipo de dato solo permite dos valores: **true** o **false**. Estos valores son habituales usarlos cuando hacemos comparaciones o expresiones en nuestros programas. Dentro de cualquier evaluación (por ejemplo, dentro de un **if**) JavaScript convierte el resultado de la evaluación a **boolean.**

null

Este tipo de valor nos ayuda a representar **la ausencia de valor**. Pero con **null** tenemos un problema: **¿Por qué la ejecución de typeof null devuelve "object"?**

null nos engaña. Así es, es un mentiroso.

Pues eso, **null** sigue siendo un tipo primitivo, pero recibimos "object" por un error histórico en el lenguaje, que seguramente no va a arreglarse jamás (se intentó solucionar, pero decidieron que era mejor no hacer nada porque rompería muchas páginas webs en el intento 🤷)

undefined

Se considera como valor de un dato o variable desconocido. Solo hay un valor con este tipo: **undefined**.

Siempre que creamos una variable, el primer valor que se le asigna a esa variable es **undefined** (recuerdas el hoisting?)

Symbol

Este tipo de dato se usa para **crear valores únicos, irrepetibles.**

Registro global de Símbolos

Existe un registro global de símbolos, en el que podemos crear y recibir el mismo símbolo a partir de la descripción. Además, que este registro es compartido en nuestra página, incluso entre los iframes y serviceworkers que estén en ella.

bigInt

Este tipo de dato nos permite usar cualquier número entero **sin límite de tamaño**. Es decir que ya no tenemos la limitación que encontramos con el tipo "number" anteriormente mencionado.

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 rápidas con uno o más operandos (generalmente números, aunque también pueden ser otros tipos de datos).

Sin embargo, esto se entiende mejor con ejemplos, por lo que vamos a hacer un desglose de los operadores que utilizaremos:

Operadores basicos

Operadores aritméticos

Vamos a centrarnos en primer lugar 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)`.

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
```

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`.

Operadores de comparación

Una tarea habitual y frecuente en programación es la de realizar comparaciones. Es necesario realizar comprobaciones continuamente para saber si debemos hacer una acción u otra diferente. Existe una serie de **operadores de comparación** para realizar estas comprobaciones de forma fácil y rápida.

Operadores de igualdad

Los **operadores de comparación de igualdad** 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.

Son muy similares en otros lenguajes, por lo que, si has programado alguna vez, probablemente no te resulten desconocidos:

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.

Operadores de identidad

A parte de los **operadores de igualdad**, Javascript es un lenguaje que también tiene **operadores de identidad**. Estos operadores se diferencian a los dos primeros anteriores en que en lugar de dos símbolos de igual ==, utiliza tres ===:

Nombre	Operador	Descripción
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.

```
5 == 5     // true   (ambos son iguales, coincide su valor: 5)
"5" == 5   // true   (ambos son iguales, coincide su valor: 5)
5 === 5    // true   (ambos son idénticos, coincide valor y tipo de da
"5" === 5  // false  (no son idénticos, coincide valor pero no tipo de
```

Por esta razón, en JavaScript se suele generalizar que es mucho mejor utilizar === en lugar de ==, ya que comprueba ambas cosas, valor y tipo de dato, y por lo tanto la comprobación es mucho más estricta.

Operadores lógicos

Existen algunos operadores menos conocidos a nivel general, pero que son muy útiles para ciertas tareas. Se denominan **operadores lógicos** y se suelen utilizar para realizar comprobaciones rápidas:

	Operador	Descripción
&&	Operador lógico AND	Devuelve true si ambos valores son verdaderos. Sino, false.
||	Operador lógico OR	Devuelve true si alguno de los dos valores es verdadero. Sino, devuelve false.
!	Operador lógico NOT	Devuelve el valor opuesto (se dice que niega el valor).

Fuentes Bibliográficas

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Grammar_and_types#tipos_de_datos

https://zajuna.sena.edu.co/Repositorio/Titulada/institution/SENA/Tecnologia/228118/Contenido/OVA/CF14/index.html#/curso/tema1

https://www.freecodecamp.org/espanol/news/lenguajes-compilados-vs-interpretados/

https://www.mytaskpanel.com/javascript-caracteristicas-beneficios-casos/

https://www.escuelafrontend.com/tipos-primitivos-en-javascript

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