**33 conceptos**

**JAVASCRIPT**

**Resumen por: MM**

Contenido

[General 2](#_Toc90023698)

[Stack 2](#_Toc90023699)

[Heap 2](#_Toc90023700)

[Event loop 2](#_Toc90023701)

[Tipos de datos primitivos 2](#_Toc90023702)

[Def 2](#_Toc90023703)

[Objetos 3](#_Toc90023704)

[Referencia vs Valores 3](#_Toc90023705)

[Sistemas de tipos de javascript 3](#_Toc90023706)

[Coerción de tipos 3](#_Toc90023707)

[Equivalencia y compatibilidad de tipos 4](#_Toc90023708)

[== vs === 4](#_Toc90023709)

General  
Java script se ejecuta en un solo hilo, esto quiere decir que solo se ejecuta una instrucción a la vez.

Stack: Esta compuesta de una pila de frames, los cuales se crean al momento de que se ejecute una función.  
  
Frames: El frame encapsula información del contexto y las variables locales donde se ejecuta una función.

Heap: Es el que almacena los objetos, es una regio de memoria, sin estructura ni orden.

Event loop:

Tipos de datos primitivos:

Def: Valores básicos inmutables sin propiedades y métodos.

.String  
.Number  
.Boolean

Symbol (agregado en 2015)

Crea un atributo único sin importar que este tenga el mismo nombre, la misma descripción.  
Los símbolos son propiedades ocultas y no aparecen en el listado de propiedades de los objetos.  
Para acceder a los símbolos de un objeto se puede hacer mediante Object.getOwnPropertySymbols(/\*El objeto\*/);  
Se puede crear un symbol con una descripción mediante var s1 = Symbol.for(‘desc’); y podemos acceder a esta descripción mediante var desc = Symbol.keyFor(s1);

BigInt (agregado en 2019)

Se utiliza agregando una ‘n’ al final del número, solo permite operaciones entre big ints y al momento de convertirlo a int, puede surgir errores de aproximación.

Objetos:

En javascript cualquier cosa que no sea un tipo primitivo es un objeto. Por lo que una función, un objeto primitivo o un array es considerado un objeto. Estos objetos se almacenan en HEAP.

# Referencia vs Valores

|  |  |
| --- | --- |
| var fruta = “manzana”;  var frutaCopia = fruta; | Genera una copia del valor de fruta y se lo asigna a fruta copia. Siempre y cuando sea un tipo de dato primitivo. |
| var objeto = { prop1:”hola”, prop2: 23}  var objetoCopia = objeto; | Para los objetos funciona diferente, ejecutar el código de la izquierda genera en el objetoCopia una referencia hacia objeto, eso es porque objeto contiene el puntero a memoria que contiene los datos prop1 y prop2. |
| funcion primitivos(fruta){  fruta+=” roja”;}  funcion objetos(objeto){  objeto.prop1+= “ ,como estas?”;} | Para los tipos primitivos, cuando lo pasamos a una función, esta mantendrá su valor original al finalizar el scope. Esto es porque primitivos recibe una copia de fruta, por lo que no modifica la variable original.  Diferente sucede con las funciones que reciben objetos, al tener una copia de la referencia están modificando el objeto a nivel global. |

# Sistemas de tipos de javascript

Si el lenguaje usa compilador este chequea los tipos de datos durante la compilación, y se llama chequeo de tipos estático. Evita que sucedan errores por tipos de datos durante la ejecución. Lenguajes que realizan esto son, por ejemplo, Java o C#.

Los lenguajes que no son compilados, o sea interpretados, por ejemplo: javascript, hacen la comprobación de tipos durante la ejecución. Esto se debe existe un programa, por ejemplo: v8 para Chrome, que sabe leer y ejecutar el código javascript que le pasamos.

La manera en que lo hace es: por cada línea de que lee, verifica que cumpla con los requerimientos para ser ejecutada, entre ellos el chequeo de tipos y si todo está bien, la ejecuta y continua con la siguiente.

A esto se lo llama como chequeo dinámico de tipos.

## Coerción de tipos

Es la conversión implícita de tipo que realiza el motor de javascript para concretar una operación.

Frente a 40+’60’ = “4060”  
Frente a 40-’60’ = -20

Para evitar que el lenguaje haga la conversión por si mismo, nosotros podemos convertir estos valores.

Se puede pasar entre estos 3 tipos

Boolean  
String  
Number

Se puede realizar de múltiples formas

Usando el nombre del tipo y lo que queremos convertir

string(123);  
number(‘55’);  
boolean(10);

también se puede convertir

de numero a string 2021 + ‘ ’;

de boolean a string true + ‘ ’;

de string a number +’123’;

de boolean a number +false;

de string a boolean !!’hola’;

## Equivalencia y compatibilidad de tipos

Es como el lenguaje determina cuando un tipo des compatible o equivalente con otro tipo.

Los tipados a continuación entran dentro del chequeo estático de tipos:   
Tipado nominal: dos tipos son compatibles cuando tienen el mismo nombre o cuando uno es un subtipo del otro(herencia);

Tipado estructural: para que dos tipos sean compatibles es suficiente que compartan la estructura que nos interesa.  
  
Dentro del chequeo dinámico de tipos:

Se utiliza algo llamado DUCK TYPING, esto significa que no importa de que tipo es un objeto siempre y cuanto tenga los atributos y métodos a los cuales queremos acceder.

# == vs ===

## Para tipos primitivos

=== compara el contenido de la variable y el tipo de la misma, por ejemplo:

’2’ === 2 da como resultado false

‘hola’ === ‘chau’ da como resultado false

2 === 2 da como resultado true

A este operador se le llama igualdad estricta o strict equality.

Para == java script utiliza la coerción de tipos, por lo que cuando hacemos

’2’ == 2 java internamente convierte el ‘2’ en 2 para realizar la comparación (da true)

A este operador se le conoce como operador de igualdad débil o loose equality

## Para objetos

Como se guarda la referencia a la memoria donde se guarda la información del objeto

Si tengo:

Var obj1 = { nombre:’hola’ }

Var obj2 = { nombre:’hola’ }

No importa si uso == o ===, el resultado de esta comparación es false.

La única manera de que den iguales es que tengan la misma referencia.  
Es por esta razón que al operador === se le conoce como operador de identidad, ya que estamos preguntando si tienen la misma referencia en memoria.

En javascript comparar objetos se vuelve complicado, para realizarlo, se puede convertir el objeto a string y compararlos, pero esto tiene el problema de que si las propiedades vienen en distinto orden va a dar false.  
La otra opción es utilizar una librería por ejemplo lodge, la cual contiene un comparador de objetos, el cual itera por todas las propiedades y las compara.

# Scope