# Tipos de datos

Por el momento en JavaScript existen siete datos primitivos:

* Number → Número entero o punto flotante.
* BigInt → Entero con precisión.
* String → Secuencia de caracteres que representan texto.
* Boolean → Valores booleanos (true, false).
* Null → Denota valor nulo.
* Undefined → Indica que no tiene un valor definido.
* Symbol → …
* Object (no cuenta como dato primitivo).

JavaScript es dinámico en cuanto al uso de tipos de datos. *No es necesitas especificar el tipo de dato de una variable* cuando ésta es declarada. Automáticamente se convierte durante la ejecución del script.

## Literales

Los literales representan valores en JavaScript. Son valores fijos que literalmente proveemos al script.

## Number

Los números tienen tres valores simbólicos:

**+Infinity**: Hace referencia al número más grande de los positivos.

**-Infinity**: Hace referencia al número más pequeño de los negativos.

**NaN**: Hace referencia a que es no es un número.

## BigInt

Nos permiten representar números enormes, pero no pueden ser utilizados para operaciones con valores del tipo de dato **number**, así mismo no se puede utilizar el objeto **Math** con este tipo de dato.

## Strings

Se puede declarar strings usando comillas simples o dobles.

Strings avanzados pueden ser creados con secuencias de escape:

**\x**:es interpretado como un número hexadecimal.

**\u**: secuencia de escape **Unicode**, requiere al menos 4 dígitos hexadecimales.

**\u{}**: secuencia de escape **Unicode**, cuando tiene más de cuatro dígitos.

Los strings pueden ser concatenados con el operador **+** retornando la unión de dos strings.

## Booleans

El tipo de dato boolean tiene dos valores literales: **true** y **false**. Los valores primitivos son diferentes de los valores que provee el objeto **Boolean**

## Null

Así mismo cuando evaluamos una variable con el valor de nulo, ésta se comporta como un **0** en contextos numéricos y como un **false** en contextos booleanos,

## Undefined

Todas las variables sin inicializar tendrán el valor de undefined. Podemos utilizar undefined para determinar si una variable tiene un valor o no. Así mismo undefined se comporta como un **false** en contextos booleanos.

# Variables

En JavaScript existen tres maneras para declarar una variable:

* var → Declara una variable.
* let → Declara una variable local al bloque en que trabajemos.
* const → Declara una constante local al bloque en que trabajemos.

Se le llama **identificador** al nombre que le asignaremos a la variable, dichos identificadores conforman ciertas reglas.

Deben de empezar con un carácter, guion bajo (\_), signo de dólar ($), seguido de más caracteres y/o dígitos. Se pueden crear identificadores con caracteres de Unicode.

*Todas las variables sin inicializar tendrán el valor de* ***undefined****.*

# Operadores

## Operadores de asignación

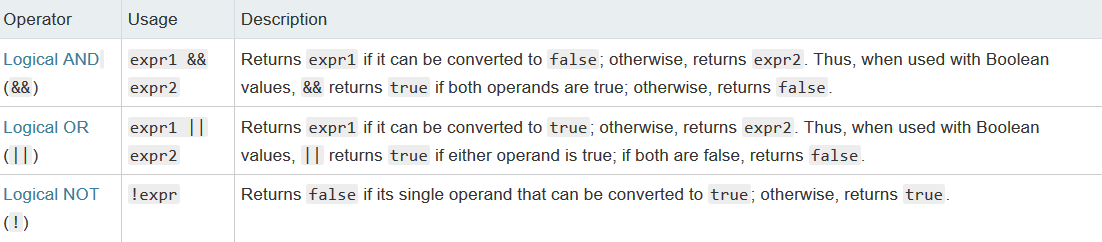
El operador de asignación más simple es el igual (=), el cual asigna el valor de la derecha a la izquierda.

Es común ver **operadores de asignación compuestos**, los cuales sirven como atajos.

Algunos ejemplos son:

En JavaScript existen asignaciones más complejas como la **destructuring assignement**, la cual nos permite extraer múltiples valores para asignarlos a múltiples variables.

## Operadores lógicos

Los operadores lógicos son comúnmente utilizados con valores booleano, así mismo retornan un valor booleano. Si son utilizados con valores no-booleanos probablemente retornaran un valor no-booleano.

## Operadores de comparación

Estos operadores regresan un valor lógico, son comúnmente utilizados en condicionales.

Algunos ejemplos son:

# Colecciones

## Arreglos

Un arreglo es un conjunto ordenado de valores, a los que podemos acceder mediante índices, en donde el primer elemento tendrá el índice de 0 y el ultimo tendrá el índice de longitud-1.

Para poder saber la longitud de un arreglo utilizaremos el método **lenght**.

Un arreglo puede ser inicializado de manera literal con **corchetes** (**[]**). Si deseamos *inicializar* un arreglo de *un solo elemento* es indispensable el uso de corchetes. Así mismo si deseamos establecer una longitud a nuestro arreglo podemos asignarle el objeto **Array(N)** en donde N es u número entero y será la longitud.

En JavaScript podemos asignarle valores a un índice en específico. De igual manera para referirnos a los elementos de un arreglo escribiremos el índice entre corchetes para acceder a él.

En JavaScript podemos hacer **arreglos multidimensionales**, anidando arreglos dentro de arreglos.

## Maps

Un mapa es similar a un **Object** solo que trabaja con **valores** que son relacionados con un **key** *la cual nosotros proporcionamos.* Dicha llave puede ser llamada, borrada, y detectar si hay un valor asignado en ella.

Podemos obtener el tamaño de un Map con el método **size**. La iteración de los mapas es en el orden que fueron insertados los elementos.

## Sets

Los sets son colecciones con elementos únicos, es decir no puede haber elementos repetidos. Solo se podrán iterar los elementos en el orden que fueron insertados

## Spread operator (…)

El spread operator nos permite traer todos los elementos de algún **iterable** para poder adjuntarlos a un nuevo arreglo.

# Ciclos e iteraciones

Los ciclos ofrecen una manera rápida de hacer algo repetidamente.

## For

Un ciclo flor se repite hasta que una condición especifica obtiene **false**. El ciclo **for** suele repetirse un número *determinado* de veces.

La sintaxis del ciclo for es similar a Java y C.

## Do-While

Esta declaración se repite hasta que una condición especifica obtiene **false**. El ciclo do-while suele repetirse un número indeterminado de veces. El bloque del ciclo *siempre se ejecutará una vez antes de ser evaluada la condición*.

## While

La declaración while se ejecuta cada que una condición especifica obtiene **true**. A diferencia del do-while, primero evalúa y después ejecuta su bloque. El ciclo while suele repetirse un número indeterminado de veces, siempre siendo dependiente de la condición.

## Labeled statement

JavaScript nos permite colocar un identificador a cualquier declaración, de manera que un ciclo puede tener un identificador y poder referirnos a dicho ciclo.

## Break

Utilizaremos **break** para terminar un ciclo, así mismo podemos utilizarlo con un *labeled statement.*

## Continue

Utilizaremos **continue** para saltarnos la iteración en la que estemos y pasar a la siguiente (en caso de poder). Puede ser utilizado en cualquier ciclo o *label*.

## For-in

Ciclo que itera una variable (la cual especificamos) sobre todas las propiedades enumeradas de un objeto.

Utilizar el for-in en arreglos nos retornará los índices. Siempre será mejor utilizar el **for** tradicional o for-of.

## For-of

Esta declaración crea un ciclo que recorre objetos iterables

# Controladores de flujo

## Bloques

Es la más básica de las declaraciones, es utilizada para agrupar declaraciones. Es delimitada por un par de llaves.

## Condicionales

Es un conjunto de comandos que se ejecutan si (**if**) una condición especifica es **true**. Así mismo tiene su contraparte (**else**) la cual se ejecuta si dicha condición es **false**.

De ser necesario la contraparte (**else**) puede ser ejecutado con otra condición.

*Una* ***buena práctica*** *es siempre utilizar la declaración de bloque en los condicionales.*

Si es necesario hacer una asignación dentro de una condicional, dicha asignación deberá ir entre paréntesis.

Los siguientes *valores* son *evaluados* *como falsos*:

* false
* undefined
* null
* 0
* NaN
* String vacío ("")

El resto de los valores será evaluado como **true**, pasen a un condicional.

## Operador ternario

JavaScript cuenta con el operador ternario, el cual sirve para evaluar condiciones de manera más reducida, así mismo nos permite poder hacer condiciones al momento de asignar algún valor a una variable.

## Switch

Permite evaluar una expresión e intentar compararla con los posibles **casos**. Si se encuentra coincidencia el programa ejecuta la declaración asociada.

Si no se encuentra un caso el programa buscará el caso **default**. Si no se encuentra dicho caso, terminará con la ejecución del switch.

Por convención el caso **default**, se escribe al final, pero éste es opcional.

De manera opcional se puede agregar un **break** al terminar el caso sin embargo éste es opcional. De no contar con algún break el switch pasará al siguiente caso (si lo hay).

## Exceptions

En JavaScript se pueden tirar (**throw**) excepciones y manejarlos utilizando las declaraciones **try…catch**.

Consiste en un bloque **try** el cual contiene una o más declaraciones y el bloque **catch** contiene las declaraciones que especifican que hacer si una excepción fue tirada en el bloque **try.**

El bloque **catch** recibe un parámetro que aloja el valor de la declaración arrojada. Se puede utilizar este parámetro para obtener información de dicho error.

El bloque **finally** se ejecutará sin importar que se haya capturado el error o no (aun cuando ya se trató dicho error).

*Se pueden anidar más de una declaración* ***try…catch****.*

# Funciones

## ¿Qué es una función?

Una función es un bloque de código al cual se le asignan tareas y se ejecutan una vez es invocada la función. De manera opcional pueden retornar un valor de cualquier tipo.

Una función en JavaScript consiste en la palabra reservada **function** seguida del nombre de la función, después los de parámetros que tendrá entre paréntesis por último sus declaraciones deberán ir encerradas en llaves.

Los valores primitivos (como un número) enviados a una función no pueden ser modificados globalmente. En cambio, los valores no-primitivos (como un arreglo), sus cambios serán visibles afuera de la función.

## Funciones anónimas

En JavaScript se pueden crear funciones anónimas; estas no tienen nombre. Y pueden ser asignadas a variables.

## Expresiones de función

Una función de expresión es alojada en una variable, y el nombre de la función puede ser omitido. Solo se ejecuta una vez.

Una función puede ser definida en base a un condicional. También en JavaScript las funciones pueden ser anidadas.

Si llamamos la función sin paréntesis estamos pasando el objeto función en sí mismo (una referencia de la función). Si llamamos la función con paréntesis pasamos el resultado de su ejecución.

## Closures

Esta característica de JavaScript nos permite que al anidar funciones las funciones internas tengan acceso a todas las variables establecidas en las funciones de afuera.

## Manejo de argumentos

Los argumentos de una función se mantienen en un arreglo. Para acceder a ellos llamaremos a **arguments** el cual es un objeto y cuenta con métodos para saber la cantidad de argumentos, entre otras cosas.

## Funciones con parámetros default

Podemos asignarle un valor default a los parámetros con los que trabajemos, de esta manera en caso de no enviar un parámetro esperado, tendrá dicho valor.

## Rest parameters

Nos permite representar un número indefinido de argumentos en forma de un arreglo.

## Funciones flecha

Si se tiene una función con una sola expresión por retornar podemos sustituirla por una función flecha. En las funciones comunes la palabra reservada **this** representa al objeto que invocó dicha función, sin embargo, en las funciones flecha **this** representa al objeto que definió la función flecha.

# Objetos

## ¿Qué es un objeto?

Los objetos son un tipo de dato no-primitivo que contienen múltiples valores. Un objeto puede contener **propiedades** y **métodos.**

Una *propiedad* es cualquier valor que se le asigna a un objeto, este valor tendrá un nombre que servirá como identificador.

Los *métodos* son acciones que actúan en los objetos, estos pueden alterar o leer los valores de las propiedades.

Existen múltiples maneras de crear un objeto en JavaScript:

* Se puede crear un objeto con valores literales.
* Se puede crear un objeto con la palabra reservada **new**.
* Se puede definir un **constructor** y crear objetos en base al constructor.

Los objetos trabajan de manera dinámica, es decir durante tiempo de ejecución pueden ser modificados constantemente.

## Propiedades

Puedes definir nuevas propiedades asignándoles un valor. Todas las propiedades sin valores asignados en un objeto tendrán el valor de **undefined**.

Todas las propiedades tienen una **key** y un **value**. Podemos acceder a dichas propiedades con un punto o con corchetes.

Las *keys* con corchetes, pueden usar cualquier identificador sin importar el uso de espacios, números al inicio, etc.

Las propiedades pueden ser eliminadas con la keyword **delete**.

## Métodos

En la definición de un método utilizamos la keyword **this** para referirnos al objeto de la función (dueño).

Los métodos de un objeto pueden ser llamados mediante un punto. Así mismo podemos invocar dicho método para que se ejecute como una función utilizando paréntesis **()** al final.

Para añadir nuevos métodos solo es necesario referirnos al nuevo método con un punto **(.)** y escribir la función.

## Get y Set

Existe un formato para modificar y leer una propiedad, es mediante un get o un set.

**get →** será una función la cual retorne el valor de una propiedad de nuestro objeto.

**set →** será una función que reciba un nuevo valor y modifique una propiedad con dicho valor.

El uso de *getters* y *setters* nos permite tener un código seguro y de mejor calidad.

## ECMAScript 2015

En esta actualización de JavaScript incluyeron la keyword **class**, el método **constructor()** entre otras cosas. Resultando más fácil la creación de objetos.

El método *constructor()* es llamado cada vez que se crea una instancia de dicha *clase.*

Así mismo la sintaxis de los métodos dentro de una clase, cambió. Ahora solo agregaremos una función (sin el uso de **function**).

Se añadió las keywords **static** y **extends**. Así mismo el método **super()** el cual nos sirve para referirnos a la clase padre.

# Programación asíncrona

Las declaraciones en JavaScript son ejecutadas de manera síncrona, sin embargo, en ocasiones es necesario hacer una tarea (la cual puede demorar) y seguir ejecutando tareas en lo que se termina, a esto se le conoce como programación asíncrona.

JavaScript tiene varias soluciones cuando se trata de programación asíncrona:

## Callbacks

Un callback es una función que se pasa como argumento a otra función; que luego se invoca dentro de la función externa para completar algún tipo de rutina.

Los Callbacks pueden llegar a dar problemas en ocasiones, formando líneas de código extensas y poco legibles, para esos casos podemos utilizar las promesas.

## Async and await

La palabra reservada **async** nos ayuda a definir funciones asíncronas. Estas funciones retornan una promesa como resultado.

La palabra reservada **await** la cual solo es válida dentro de funciones asíncronas, nos permite pausar la ejecución para esperar a que termine la promesa.

## Promesa

Es un objeto el cual representa la terminación o el fracaso de una operación asíncrona. Una promesa acepta dos callbacks como parámetros, el primero cuando se resuelve y el segundo en caso de ser rechazado.

# Comentarios

Existen tres maneras de documentar nuestro código en JavaScript, estos comentarios serán ignorados al momento de ejecutar los programas.

## Comentarios de una sola línea

Los comentarios de una sola línea comienzan con **//**. Todo lo que esté hasta el fin de la línea será ignorado al ejecutar el script.

## Comentarios de múltiple línea

Los comentarios de múltiple línea empiezan con **/\***¨y terminan con **\*/**. Todo lo que esté entre **/\*** y **\*/** será ignorado al ejecutar el script.