Chuleta: JavaScript + Ejemplo

Lenguaje JavaScript

Clases: `Array` `String` `Math` `Date` `Map` `Set` `Number` `Object` `Function`

Sintaxis: `Función` `Objeto` `Clase`

Definición de variables: `let` `const` `var`

Tipos de datos: `string` `number` `boolean` `undefined` `object` `function`

Estructuras condicionales: `if/else` `switch`

Estructuras repetitivas: `while` `do/while` `for` `for-of` `for-in`

Operadores relacionales: `>` `>=` `==` `!=` `<` `<=` `===` `!==`

Operadores matemáticos: `+` `-` `*` `/` `%` `**` `++` `--`

Operadores lógicos: `&&` `||` `!`

Operadores de asignación: `=` `+=` `-=` `*=` `/=` `%=` `**=` `<<=` `>>=` `>>>=` `&=` `|=`

Operadores de desplazamiento de bits: `<<` `>>` `>>>`

Operadores bit a bit: `&` `|` `^` `~`

Otros operadores: `delete` `typeof` `instanceof` `in` `??` `,` `()` `new` `?:` `void` `...`

Desestructuración: `Arreglos` `Objetos` `Parámetros`

JSON: `JSON.stringify(objeto)` `JSON.parse(cadena)`

Tutoriales Programación Ya

Clase: Array

Jugar

`length` Retorna el número de elementos del array.

`concat(array1, [array2], ...)` Retorna un nuevo array que es la concatenación de los arrays que se pasan como parámetros.

`copyWithin(target, start, [end])` Copia los elementos del array en una posición específica del array. El primer parámetro es el índice de destino, el segundo es el índice de inicio y el tercero es el índice de fin.

`entries()` Retorna un nuevo objeto Array Iterator que contiene los pares clave/valor para cada índice de la matriz

`every(función)` Retorna true si todos los elementos del array cumplen con la función que se pasa como parámetro.

`fill(valor, [comienzo], [fin])` Rellena el array con el valor que se pasa como parámetro. El segundo parámetro es opcional y es el índice de inicio y el tercero es el índice de fin.

`filter(función)` Retorna un nuevo array con los elementos que cumplen con la coidición de la función que se pasa como parámetro.

`find(función)` Retorna el primer elemento del array que cumpla con la condición de la función que se pasa como parámetro.

`findIndex(función)` Retorna el índice del primer elemento del array que cumpla con la condición de la función que se pasa como parámetro.

`flat([profundidad])` Crea un nuevo arreglo con todos los elementos de subarreglo concatenados recursivamente hasta la profundidad especificada, uno por defecto.

`flatMap(funcion)` Primero mapea cada elemento usando una función la función map y luego aplana el resultado con profundidad 1. Es idéntico a a map() seguido de a flat() de profundidad 1, pero un poco más eficiente.

`forEach(funcion)` Ejecuta la función para cada elemento del array en forma ascendente. La función recibe el valor, el índice y el arreglo.

`Array.from()` Es un método estático que crea un nuevo array a partir de un objeto iterable (son objetos iterables en JavaScript los String, array, Map, Set)

`includes(elemento,[desde])` Retorna true si el elemento se encuentra en el array. El segundo parámetro es opcional y es el índice de inicio donde debemos comenzar a buscar.

`indexOf(elemento,[desde])` Retorna el índice del elemento en el array. Si no se encuentra el elemento, retorna -1.

`Array.isArray(dato)` Es un método estático que retorna true si el objeto pasado como parámetro es un array.

`join([separador])` Retorna una cadena con todos los elementos del arreglo separados por el separador que se pasa como parámetro. Si no se pasa ningún parámetro, el separador es una coma.

`keys()` Retorna un nuevo objeto Array Iterator que contiene los índices del arreglo.

`lastIndexOf(elemento,[desde])` Retorna el último índice del elemento encontrado en el array, el arreglo es recorrido en forma inversa. Si no se encuentra el elemento, retorna -1.

`map(funcion)` Retorna un nuevo array con los resultados de aplicar la función que se pasa como parámetro a cada elemento del array.

`Array.of(elemento0, elemento1,...)` Es un método estático que crea un nuevo array con los elementos que se pasan como parámetros.

`pop()` Elimina el último elemento del array y lo retorna. Si el array esta vacío luego retorna undefined.

`push(elemento 1, elemento2, ...)` Añade uno o más elementos al final del array y retorna el número de elementos del array.

`reduce(funcion, [valor inicial])` Reduce el array a un único valor aplicando la función que se pasa como parámetro.

`reduceRight(funcion, [valor inicial])` Reduce el array a un único valor aplicando la función que se pasa como parámetro, recorriendo los elementos del arreglo partiendo desde el final.

`reverse()` Invierte el orden de los elementos del arreglo. Retorna la referencia del arreglo invertido.

`shift()` Elimina el primer elemento del array y lo retorna. Si el array esta vacío luego retorna undefined.

`slice(inicio,[fin])` Retorna una copia del array desde el índice inicio hasta el índice fin (no incluido).

`some(funcion)` Retorna true si alguno de los elementos del array cumple con la condición que se pasa como parámetro.

`sort([funcion])` Ordena los elementos del array. Si se pasa una función como parámetro, se ordena de acuerdo a la función.

`splice(inicio, cant. borrar, item1, item2, itemN)` Permite borrar y añadir elementos en un arreglo.

`toLocaleString([local], [opciones])` Convierte el array en una cadena de texto con el formato especificado por el local.

`toString()` Convierte el array en una cadena de texto separadando los elementos con comas.

`unShift(item1, item2, itemN)` Añade uno o más elementos al inicio del array y retorna el número de elementos del array.

`values()` Retorna un nuevo array Iterator que contiene los valores del arreglo.

Clase: String

Jugar

`length` La propiedad 'length' de un objeto String almacena la cantidad de caracteres.

`at(índice)` Retorna el caracter de la posición indicada en índice, comienza a numerarse en cero. En el caso de pasar un valor negativo se comienza desde el final del String.

`charAt(índice)` Retorna el caracter de la posición indicada en índice, comienza a numerarse a partir de cero.

`charCodeAt(índice)` Retorna un valor entero comprendido entre 0 y 65535 que representa el caracter respectivo.

`codePointAt(índice)` Retorna un valor entero para cualquier caracter Unicode. Remplaza al método charCodeAt

`concat(string1, string2, ...)` Retorna un String con la concatenación de todos sus parámetros, más el contenido del objeto que llama al método concat.

`endsWith(string, [length])` Retorna true si el String finaliza con el String que le pasamos como primer parámetro. El segundo parámetro es opcional y es el número de caracteres del String principal a tener en cuenta en la comparación.

Es un método estático que devuelve un String que se crea utilizando la secuencia

`String.fromCharCode(num1, num2, ..., numN)`	de códigos enviados (se pueden enviar valores comprendidos entre 0 y 65535).
`String.fromCodePoint(num1, num2, ..., numN)`	Es un método estático que devuelve un String que se crea utilizando la secuencia de códigos enviados.
`includes(string,[posición])`	Retorna true si el String contiene el String a buscar que le pasamos como primer parámetro. El segundo parámetro es opcional y es la posición de inicio de la búsqueda.
`indexOf(string,[posición])`	Retorna la posición de la primera ocurrencia del String que le pasamos como primer parámetro. El segundo parámetro es opcional y es la posición de inicio de la búsqueda.
`match(expresión regular)`	Retorna un array con todas las coincidencias de una expresión regular en el String.
`matchAll(expresión regular)`	Retorna un iterador de todos los resultados de ocurrencia en una cadena de texto contra una expresión regular.
`normalize([formato])`	Retorna la forma de normalización Unicode del string (si el valor no es una cadena, primero será convertido a ese tipo)
`padEnd(numero, [string])`	Rellena el string actual con una cadena determinada que le pasamos eventualmente en el segundo parámetro, sino rellena con espacios en blanco.
`padStart(numero, [string])`	Rellena en el principio del string actual con una cadena determinada que le pasamos eventualmente en el segundo parámetro, sino rellena con espacios en blanco.
`String.raw()`	Es una función de plantilla de literales. Se utiliza para obtener un string a partir de una plantilla de string.
`repeat(cantidad)`	Retorna una nueva cadena que es una copia del string actual repetida la cantidad de veces que le pasamos como parámetro.

`replace(string actual,string nuevo)`	Retorna una nueva cadena que es una copia del string actual pero con la primera ocurrencia reemplazada por la cadena nueva.
`replace(expresion regular,string nuevo)`	Retorna una nueva cadena que es una copia del string actual pero con todas las ocurrencias de la expresión regular reemplazadas por la cadena nueva.
`replaceAll(string actual,string nuevo)`	Retorna una nueva cadena que es una copia del string actual pero con todas las ocurrencias reemplazadas por la cadena nueva.
`replaceAll(expresion regular,string nuevo)`	Retorna una nueva cadena que es una copia del string actual pero con todas las ocurrencias de la expresión regular reemplazadas por la cadena nueva.
`search(expresion regular)`	Retorna la posición de la primera ocurrencia de la expresión regular en el string.
`slice(inicio, [fin])`	Retorna un trozo de la cadena actual. El primer parámetro es el índice de inicio y el segundo es opcional y es el índice de fin.
`split(separador,[limite])`	Retorna un array con todas las partes de la cadena que se separan por el separador. El segundo parámetro es opcional y es el número máximo de elementos que se retornarán.
`startsWith(string, [posicion])`	Retorna true si el string actual comienza con el string que le pasamos como parámetro. El segundo parámetro es opcional y es la posición de inicio de la búsqueda.
`substring(indice inicial, [indice final])`	Retorna un trozo de la cadena actual. El primer parámetro es el índice de inicio y el segundo es opcional y es el índice de fin.
`toLocaleLowerCase([locale])`	Retorna una copia de la cadena actual en minúsculas. El segundo parámetro es

opcional y es el nombre del lenguaje local.

`toLocaleUpperCase([locale])` Retorna una copia de la cadena actual en mayúsculas. El segundo parámetro es opcional y es el nombre del lenguaje local.

`toLowerCase()` Retorna una copia de la cadena actual en minúsculas.

`toString()` Retorna una copia de la cadena actual.

`toUpperCase()` Retorna una copia de la cadena actual en mayúsculas.

`trim()` Retorna una copia de la cadena actual sin los espacios en blanco al principio y al final.

`trimEnd()` Retorna una copia de la cadena actual sin los espacios en blanco al final.

`trimStart()` Retorna una copia de la cadena actual sin los espacios en blanco al principio.

`valueOf()` Retorna una copia de la cadena actual.

Clase: Math

Jugar

`Math.E` Propiedad que representa la base de los logaritmos naturales, e, aproximadamente 2.71828.

`Math.LN10` Propiedad que representa el logaritmo natural en base 10

`Math.LN2` Propiedad que representa el logaritmo natural en base 2

`Math.LOG10E` Propiedad que representa el logaritmo base 10 de e

`Math.LOG2E` Propiedad que representa la base 2 del logaritmo natural de E

`Math.PI` Propiedad que representa la relacion entre la longitud de la circunferencia de un circulo y su diametro, la cual es aproximadamente 3.14159

`Math.SQRT1_2` Propiedad que representa la raiz cuadrada de 1/2

`Math.SQRT2` Propiedad que representa la raiz cuadrada de 2

`Math.abs(valor)` Retorna el valor absoluto del parámetro.

`Math.acos(valor)` Retorna el arco coseno del parámetro. valor debe tomar un número entre 1 y -1.

`Math.acosh(valor)` Retorna el arco coseno hiperbólico del parámetro.

`Math.asin(valor)` Retorna El arco seno en radianes de un número

`Math.asinh(valor)` Retorna El arco seno hiperbólico de un número

`Math.atan(valor)` Retorna el arcotangentem, en radianes, de un numero

`Math.atan2(y, x)` Retorna la arcotangente del cociente de los parámetros

`Math.atanh(valor)` Nos retorna el arco tangente hiperbólico.

`Math.cbrt(valor)` Nos retorna la raiz cúbica del parámetro

`Math.ceil(valor)` Retorna el número entero mayor o igual más próximo.

`Math.clz32(valor)` Retorna el número de bits con ceros a la izquierda del número, en la representación binaria de 32 bits.

`Math.cos(valor)` Retorna el coseno del ángulo, que debe pasarse en radianes.

`Math.cosh(valor)` Retorna el coseno hiperbólico del ángulo, que debe pasarse en radianes.

`Math.exp(x)` Retorna 'e' elevado a la 'x'. Donde x es el parámetro y 'e' es el número de Euler.

`Math.expm1(x)` Retorna 'e' elevado a la 'x' menos 1. Donde x es el parámetro y 'e' es el número de Euler.

`Math.floor(valor)` Retorna el máximo entero menor o igual a un número.

`Math.fround(valor)` Retorna el valor flotante más cercano con precisión de 32 bits.

`Math.hypot(valor1, valor2, ..., valorN)` Retorna la raíz cuadrada de la suma de los cuadrados de los valores pasados a la función.

`Math.imul(valor1, valor2)` Retorna la multiplicación de enteros de 32 bits con un algoritmo similar al lenguaje C.

`Math.log(valor)` Retorna el logaritmo natural en base E de un número.

`Math.log10(valor)` Retorna el logaritmo en base 10 de un número.

`Math.log1p(valor)` Retorna el logaritmo natural (en base e) de 1 más el parámetro.

`Math.log2(valor)` Retorna el logaritmo en base 2.

`Math.max(valor1, valor2, ...)` Retorna el máximo valor.

`Math.min(valor1, valor2, ...)` Retorna el mínimo valor.

`Math.pow(base, exponente)` Retorna la base elevada al exponente.

`Math.random()` Retorna un punto flotante, un número pseudo-aleatorio dentro del rango [0, 1). Esto es, desde el 0 (Incluido) hasta el 1 pero sin incluirlo (excluido).

`Math.round(valor)` Retorna el valor de un número redondeado al entero más cercano.

`Math.sign(valor)` Retorna el signo de un número, indicando si el número es positivo (un 1), negativo (un -1) o cero (un 0).

`Math.sin(valor)` Retorna seno de un número.

`Math.sinh(valor)` Retorna seno hiperbólico de un número.

`Math.sqrt(valor)` Retorna la raíz cuadrada de un número.

`Math.tan(valor)` Retorna la tangente de un número.

`Math.tanh(valor)` Retorna la tangente hiperbólica de un número.

`Math.trunc(valor)` Retorna la parte entera del número.

Clase: Date

Jugar

`new Date()` Crea un objeto de la clase Date con la fecha y hora local.

`new Date(milisegundos)` El parámetro representa el número de milisegundos desde las 00:00:00 UTC del 1 de enero de 1970

`new Date(año,mes,dia,[hora,minuto,segundo,milisegundo])` Creamos una fecha indicando cada una de sus componentes como parámetros del constructor de la clase Date.

`getDate()` Retorna el día del mes para la fecha especificada de acuerdo con la hora local (un valor comprendido entre 1 y 31)

`getDay()` Retorna el día de la semana de la fecha especificada en función de la fecha local, siendo 0 (Domingo) el primer día.

`getFullYear()` Retorna el año de la fecha de acuerdo a la hora local.

`getHours()` Retorna la hora de acuerdo a la fecha local (un valor entre 0 y 23).

`getMilliseconds()` Retorna la cantidad de milisegundos de acuerdo a la fecha local (un valor entre 0 y 999).

`getMinutes()` Retorna la cantidad de minutos de acuerdo a la fecha local (un valor entre 0 y 59).

`getMonth()` Retorna el número de mes (un valor entre 0 y 11, 0 representa a Enero).

`getSeconds()` Retorna la cantidad de segundos de acuerdo a la hora local (un valor entre 0 y 59).

`getTime()` El valor devuelto es un número de milisegundos transcurridos desde el 1 de enero de 1970 00:00:00 UTC

`getTimezoneOffset()` Retorna la diferencia en minutos entre entre la hora UTC y la hora local.

`getUTCDate()` Retorna el día del mes de acuerdo a la hora universal (un valor entre 1 y 31)

`getUTCDay()` Retorna el día de la semana de la fecha especificada en función de la hora universal, siendo 0 (Domingo) el primer día.

`getUTCFullYear()` Retorna el año de la fecha de acuerdo a la hora universal.

`getUTCHours()` Retorna la hora de acuerdo a la fecha universal (un valor entre 0 y 23).

`getUTCMilliseconds()` Retorna la cantidad de milisegundos de acuerdo a la fecha universal (un valor entre 0 y 999).

`getUTCMinutes()` Retorna la cantidad de minutos de acuerdo a la fecha universal (un valor entre 0 y 59).

`getUTCMonth()` Retorna el número de mes en horario universal (un valor entre 0 y 11, 0 representa a Enero).

`getUTCSeconds()` Retorna la cantidad de segundos de acuerdo a la hora universal (un valor entre 0 y 59).

`Date.now()` El método estático 'now' retorna la cantidad de milisegundos transcurridos desde el 1 de enero de 1970 00:00:00 UTC

`Date.parse(cadenaFecha)` Transforma una cadena con la representación de una fecha y hora, y devuelve el número de milisegundos desde las 00:00:00 del 1 de enero de 1970, en hora local.

`setDate(día)` Cambia el día del mes basado en la hora local.

`setFullYear(año)` Cambia el año basado en la hora local.

`setHours(hora)` Cambia la hora basado en la hora local.

`setMilliseconds(milisegundos)` Cambia los milisegundos basado en la hora local.

`setMinutes(minutos)` Cambia los minutos basado en la hora local.

`setMonth(mes)` Cambia el número de mes (un valor entre 0 y 11, 0 representa a Enero).

`setSeconds(segundos)` Fija la cantidad de segundos de acuerdo a la hora local (un valor entre 0 y 59).

`setTime(milisegundos)` Fijamos la cantidad de milisegundos desde 00:00:00 del 1 de enero de 1970.

`setUTCDate(día)` Cambia el día del mes basado en la hora universal.

`setUTCFullYear(año)` Fija el año de la fecha de acuerdo a la hora universal.

`setUTCHours(hora)` Fija la hora de acuerdo a la fecha universal (un valor entre 0 y 23).

`setUTCMilliseconds(milisegundos)` Cambia los milisegundos basado en la hora universal.

`setUTCMinutes(minutos)` Cambia los minutos basado en la hora universal.

`setUTCMonth(mes)` Cambia el número de mes (un valor entre 0 y 11, 0 representa a Enero) en la hora universal.

`setUTCSeconds(segundos)` Fija la cantidad de segundos de acuerdo a la hora universal (un valor entre 0 y 59).

`toDateString()` Retornar solo la parte de la fecha en un formato legible en Inglés.

`toISOString()` Retornar una cadena en el formato ISO 8601, que siempre mide 24 o 27 caracteres de largo: (YYYY-MM-DDTHH:mm:ss.sssZ o ±YYYYYY-MM-DDTHH:mm:ss.sssZ)

`toJSON()` Retornar una cadena que representa la fecha y hora del objeto Date.

`toLocaleDateString([region], [opciones])` Retornar una cadena con una representación de acuerdo al lenguaje indicado en el primer parámetro.

`toLocaleString([region], [opciones])` Retornar una cadena con una representación de acuerdo al lenguaje indicado en el primer parámetro.

`toLocaleTimeString([region], [opciones])` Retornar una cadena con una representación de la parte del tiempo según el lenguaje indicado en el primer parámetro.

`toString()` Retornar una cadena que representa la fecha almacenada en el objeto Date.

`toTimeString()` Retornar una cadena con una representación de la parte del tiempo.

`toUTCString()` Retornar una cadena con una representación de la fecha y hora en el horario universal.

`Date.UTC(año, [mes, dia, hora, minuto, segundo, milisegundos])` El método estático UTC tiene los mismos parámetros que el constructor de la clase Date, pero indicamos una fecha y hora según el horario universal.

`valueOf()` Retorna el número de milisegundos entre el 1 de enero de 1970 a las 00:00:00 UTC y la fecha indicada.

Clase: Map

Jugar

`size` La propiedad size almacena la cantidad de elementos del mapa.

`new Map()` El constructor crea un objeto de la clas Map o diccionario. Utiliza una clave para acceder a un valor.

`new Map(iterable)` Este segundo constructor podemos pasarle como parámetro un objeto iterable para inicializar los elementos del mapa.

`clear()` Elimina todos los elementos del mapa.

`delete(clave)` El método 'delete' retorna true si se eliminó la entrada en el Map y false en caso que le hayamos pasado una clave que no exista.

`entries()` El método entries() retorna un objeto iterable que contiene un arreglo [clave, valor] de cada elemento del mapa en el orden de inserción.

`forEach(función)` Se le pasa una función que se ejecuta una vez por cada elemento del mapa en el orden de inserción.

`get(clave)` Recuperar un valor para una determinada clave del 'Map', en el caso que no existe retorna undefined.

`has(clave)` Retorna true si la clave existe en el Map, en caso contrario retorna false.

`keys()` El método keys() retorna un iterable que contiene todas las claves del Map en el orden de inserción.

`set(clave, valor)` Agrega o modifica un elemento del mapa indicando como primer parámetro la clave y como segundo parámetro el valor a almacenar para dicha clave.

`values()` Retorna un iterador con los valores almacenados para cada clave.

Clase: Set

Jugar

`size` La propiedad size almacena la cantidad de elementos del conjunto.

`new Set()` Crea un conjunto vacío.

`new Set(iterable)` Este segundo constructor podemos pasarle como parámetro un objeto iterable para inicializar los elementos del conjunto.

`add(elemento)` El método add añade un nuevo elemento al final del conjunto. Si el elemento ya existe en el conjunto el mismo no se agrega. El método add retorna la referencia del conjunto.

`clear()` Elimina todos los elementos del conjunto.

`delete(elemento)` El método 'delete' retorna true si se eliminó el elemento del conjunto y false en caso que le hayamos pasado un elemento inexistente.

`entries()` El método entries() retorna un objeto iterable que contiene un arreglo [elemento, elemento] de cada elemento del conjunto en el orden de inserción.

`forEach(función)` Se le pasa una función que se ejecuta una vez por cada elemento del conjunto en el orden de inserción.

`has(elemennto)` Retorna true si el elemento existe en el conjunto, en caso contrario retorna false.

`values()` Retorna un iterador con los elementos almacenados en el conjunto.

Clase: Number

Jugar

`Number.isFinite(valor)` Retorna true si el parámetro almacena un valor finito.

`Number.isInteger(valor)` Retorna true si el parámetro pasado almacena un entero.

`Number.isNaN(value)` Retorna true si el parámetro pasado almacena el valor NaN.

`Number.parseFloat(string)` El método estático parseFloat recibe como parámetro una cadena y regresa un número de punto flotante. Tiene el mismo objetivo que la función global 'parseFloat(string)'.

`Number.parseInt(string,[base])` El método estático parseInt recibe como parámetro una cadena y regresa un número entero. Tiene el mismo objetivo que la función global 'parseInt(string, [base])'. Opcionalmente podemos pasar la base, un valor entre 2 y 36.

`toExponential([cant. dígitos fraccionarios])` Retorna un string que representa el objeto Number con notación exponencial.

`toFixed([cant. dígitos])` Retorna un string con el valor numérico usando notación de punto fijo.

`toLocaleString([region], [opciones])` Retornar una cadena con una representación de acuerdo al lenguaje indicado en el primer parámetro.

`toPrecision([precisión])` El método toPrecision devuelve un string que representa un objeto Number según la precisión especificada.

`toString([base])` Retorna una cadena que representa el objeto Number especificado, la base puede ser un valore entre 2 y 36.

`valueOf()` Retorna el valor numérico primitivo del objeto Number.

`Number.MAX_VALUE` La propiedad MAX_VALUE representa el número máximo que puede almacenar un objeto Number.

`Number.MIN_VALUE` La propiedad MIN_VALUE representa el número mínimo que puede almacenar un objeto Number.

Tema: funciones

`Declaración` Nos permiten definir un bloque de instrucciones para poderlo invocar posteriormente la cantidad de veces que necesitemos.

`Parámetros` Nos permite pasar datos a la función para que sean procesados en su interior.

`Valor de retorno` Una función devuelve un dato a la línea del código donde se la llamó. Se lo indica con la palabra clave return. Retorna 'undefined' si no lo indicamos.

Parámetros por defecto Nos permite asignar un valor por defecto a un parámetro para los casos en que en la llamada a la misma no se le envíe.

Parámetros rest Podemos pasar una lista indefinida de valores y que los reciba un vector. Debemos anteceder 3 puntos al nombre del parámetro.

Expresión de función Una expresión de función por lo general es anónima y se le asigna a una variable. Son convenientes cuando se pasan como parámetro a otra función. Luego de definida la variable que se le asigna la función se puede llamar a la función, a diferencia de las funciones declaradas.

función flecha Otra sintaxis de JavaScript para definir una función, una forma es asignar dicha función a una variable y luego la llamamos en forma similar a otras funciones, podemos también crear una función flecha anónima y pasar la misma como parámetro. Una diferencia importante es que no se tiene 'this', si se accede a 'this' se toma el contexto del exterior.

Closures Una closure permite acceder al ámbito de una función exterior desde una función interior.

Tema: objetos

Creación Podemos crear una instancia de la clase Object o más común es directamente abrir y cerrar llaves y en su interior definir sus propiedades.

Definición de propiedades Podemos definir propiedades al objeto en el momento que se lo crea o posteriormente. El nombre de una propiedad puede tener espacios en blanco pero estamos obligados a utilizar la sintaxis de corchetes.

Definición de métodos Además de las propiedades, un objeto puede definir métodos que pueden acceder a las propiedades.

'this' en un método Para acceder a las propiedades de un objeto debemos anteceder la palabra clave this al nombre de la propiedad.

`'this' en una función` El contexto de this en una función independiente se resuelve en tiempo de ejecución y depende del objeto que hace su llamada.

`Propiedades a partir de variables` Podemos definir propiedades de un objeto indicando el nombre de una variable independiente.

Tema: clase

`Función constructora` Proporcionan los medios para crear tantos objetos como se necesiten.

`class` A partir de ECMAScript 2015 JavaScript dispone de una sintaxis más sencilla para la declaración de clases, sus atributos y métodos.

`extends` La palabra clave 'extends' se utiliza para declarar que una clase hereda de otra.

`set y get` Los métodos get y set establecen o recuperar un valor de una propiedad. Permiten mejorar el encapsulamiento de nuestra clase.

`static` Un método estático pertenece a la clase y no a una instancia u objeto de una clase. Para llamar luego a un método estático lo hacemos directamente antecediendo el nombre de la clase. No pueden acceder a los atributos mediante this.

Tema: Object

`Object.keys(objeto)` Retorna un array con las propiedades del objeto que le pasamos como parámetro.

`Object.values(objeto)` Retorna un array con los valores asociados a cada propiedad del objeto.

`Object.entries(objeto)` Retorna un array con componentes de tipo array de dos elementos con la propiedad y valor de dicha propiedad.

`Object.fromEntries(objeto)` El método fromEntries crea un nuevo objeto a partir de un Array, Map, Set etc. Deben ser

estructuras iterables que contengan una clave/valor.

`Object.assign(objeto-destino,objeto-fuente1, ...)` Copia todas las propiedades enumerables de uno o más objetos fuente a un objeto destino. Devuelve el objeto destino.

`Object.create(objeto)` El método create(objeto) crea un objeto nuevo, utilizando un objeto existente como el prototipo del nuevo objeto creado.

`Object.freeze(objeto)` El método freeze nos permite congelar un objeto, esto quiere decir que no podemos modificar sus propiedades ni sus métodos. No permite agregar nuevas propiedades ni nuevos métodos, ni borrarlos.

`Object.isFrozen(objeto)` El método isFrozen(objeto) devuelve true si el objeto está congelado, false en caso contrario.

`Object.seal(objeto)` El método seal sella un objeto evitando que se puedan agregar nuevas propiedades o borrar existentes, pero se pueden modificar los valores de las ya existentes.

`Object.isSealed(objeto)` El método isSealed devuelve true si el objeto está sellado, false en caso contrario.

`Object.preventExtensions(objeto)` El método preventExtensions evita agregar nuevas propiedades pero si podemos borrar las que ya existen. Nos permite modificar los datos de las propiedades existentes.

`Object.isExtensible(objeto)` El método isExtensible retorna true o false de acuerdo a si el objeto puede ser extendido.

`Object.setPrototypeOf(objeto,prototipo)` Establece el prototipo (modifica la propiedad interna [Prototype]) de un objeto especificado a otro objeto.

`Object.getPrototypeOf(objeto)` El método getPrototypeOf() devuelve el prototipo (es decir, el valor de la

	propiedad interna [Prototype]) del objeto especificado.
`isPrototypeOf(objeto)`	El método isPrototypeOf devuelve true si el objeto especificado es un prototipo del objeto actual.
`Object.is(valor1,valor2)`	El método 'is' determina si dos valores son iguales. Coincide en casi todo con el operador de comparación === salvo cuatro casos que se muestran en el ejemplo.
`valueOf()`	El método valueOf() retorna el valor primitivo del objeto especificado. Podemos sobreescribir el método para nuestros problemas particulares.
`toString()`	Retorna una cadena que representa el objeto. Se se llama automáticamente cuando el objeto se representa como un valor de texto o cuando un objeto se referencia de tal manera que se espera una cadena.

Tema: Function

`constructor(parametro1, parametro2 ..., cadena de la función)`	Crea un objeto de la clase Function. Permite crear funciones en forma dinámica y que sea una forma más segura que la función global eval. Solo se ejecutan en el ámbito global.

`call(this, parámetro1, parámetro2 ...)` Llama a un método de un objeto y le pasa el this de otro objeto. El this de la función se refiere al objeto que se está llamando.

`apply(this, arreglo)` Llama a un método de un objeto y le pasa el this de otro objeto, la única diferencia con call es que los parámetros se envían mediante un arreglo.

`bind(this, parámetro1, parámetro2 ...)` El método bind() crea una nueva función, que cuando es llamada, asigna a su operador this el valor pasado en el primer parámetro. Además, todos los parámetros pasados a partir del segundo parámetro se pasan a la nueva función.

`name` La propiedad name es una cadena que representa el nombre de la función. Si se trata de una función anónima almacena el valor 'anonymous'.

`length` La propiedad length es un número que representa el número de parámetros de la función.

`toString()` El método toString() retorna una cadena que representa la función propiamente dicha.