Certified Tech Developer
The Ultimate Degree

Programación Imperativa

Cheat Sheet - JavaScript

Está *cheat sheet* nos va a ayudar cada vez que necesitemos recordar las herramientas que podemos utilizar en JavaScript. Se actualizará junto con el contenido de Playground.

VARIABLES		
Sintaxis	Uso	Ejemplo
`var`	SE RECOMIENDA NO UTILIZAR Le indica a JavaScript que vamos a declarar una variable de tipo **var** a la cual le podemos asignar un valor.	`var nombre = 'Hackerman';`
`let`	Solo será accesible en el bloque de código en el que fue declarada, no puede volver declararse	`let contador = 0;`
`const`	Al igual que let solo es accesible en el bloque de código en el que fue declarada. Además, no podemos cambiar su valor.	`const url = 'http://digitalhouse.com.ar';`

DigitalHouse >
Coding School

TIPOS DE DATOS

Tipos	Explicación	Ejemplo
numéricos (number)	Pueden ser enteros o con decimales.	`let age = 27;`
cadenas de caracteres (string)	Cadenas de textos. Se escriben entre comillas dobles o simples.	`let saludar = 'Hello';`
lógicos o booleanos	Su valor puede ser true o false (verdadero o falso).	`let hayAsado = true;` `let hayMates = false;`
NaN (Not a number)	No es un número. (no puede ser parseado como número)	`let malaDivision = '27' / 2;`
NULL (Nulo)	Los asignamos nosotros para indicar un valor vacío o desconocido.	`let aprobado = null;`
UNDEFINED (Valor sin definir)	Las variables tienen un valor indefinido hasta que les asignamos un valor.	`let saludo;` `saludo = 'Hola';`
`//objeto literal` `let object = {` `  clave:valor` `}`	Son colecciones de datos, que contienen propiedades agrupados en pares de {clave :valor}, asignados a una variable	`let persona = {` ` nombre: 'Martin',` ` edad: 27,` ` profesion: Desarollador` `}`
`//array` `array = ['dato1','dato2',...]`	Nos permite agrupar varios tipos de datos en una sola variable, no tiene claves, tiene índices numéricos que empiezan en el nro 0	`let frutas = ['Pera', 'Kiwi', 'Banana'];` `let edades = [10, 23, 37];`

OPERADORES DE ASIGNACIÓN

Sintaxis	Uso	Ejemplo
`let variable = valor;`	nos permiten asignar un valor a una variable determinada	`let edad = 27;`

OPERADORES ARITMÉTICOS

Sintaxis	Uso	Ejemplo
`+`	suma	`10 + 5`
`-`	resta	`10 - 15`
`*`	multiplicación	`10 * 15`
`/`	división	`10 / 5`
`++`	incremento en uno	`15++ //16`
`--`	decremento	`15-- // 14`
`%`	módulo, nos devuelve el resto de una división	`15 % 5 // 0`

OPERADORES DE COMPARACIÓN SIMPLE

Sintaxis	Uso	Ejemplo
`==`	simple nos permite preguntar si un valor es = a otro, nos devuelve un dato booleano	`5 == 5 //true` `'5' == 5 //true` `3 == 5 //false`
`!=`	Desigualdad simple nos permite comparar si un valor es opuesto a otro	`'5' != 5 //false` `5 != 5 //false` `25 != 5 //true`

OPERADORES DE COMPARACIÓN ESTRICTA

Sintaxis	Uso	Ejemplo
`===`	estricta que la comparación simple, solo que además del valor compara el tipo de dato	`5 === 5 //true` `'5' === 5 //false`
`!==`	Desigualdad estricta nos permite comparar si un valor es opuesto a otro, también el tipo de dato	`'5' !== 5 //true` `5 !== 5 //false`

OPERADORES DE COMPARACIÓN

Sintaxis	Uso	Ejemplo
`>`	pregunta si un número es mayor que otro	`5 > 4 // true`
`>=`	pregunta si un número es mayor o igual que otro	`45 >= 4 // true` `4 >= 4 // true`
`<`	pregunta si un número es menor que otro	`4 < 9 //true` `9 < 9 //false`
`<=`	pregunta si un número es menor o igual que otro	`9 <= 9 //true` `10 <= 9 //false`

OPERADORES LÓGICOS

Sintaxis	Uso	Ejemplo
`sentencia1 && sentencia2`	AND, todos los valores deben evaluar como true para que el resultado sea true, cada sentencia debe poder leerse por separado indistintamente, devuelve un dato booleano	`let dia = 'lunes'` `let mates = true` `dia == 'lunes' && mates == true` `//true`
`sentencia1 || sentencia2`	OR, al menos un valor debe evaluar como true para que el resultado sea true, a diferencia del AND, devuelve un dato booleano	`let dia = 'martes'` `let mates = true` `dia == 'lunes' || mates == true` `//true`
`!`	NOT, niega la condición, si era true, es false y viceversa	`let mates = true;` `console.log(!mates);`

		`//false`

OPERADORES DE CONCATENACIÓN

Sintaxis	Uso	Ejemplo
`+`	Sirve para unir dos o más cadenas de texto en una sola, Devuelve otra cadena de texto, si mezclamos otros tipos de datos, por ejemplo un number, el resultado será un string	`let nombre = 'Martin';` `let apellido = 'Cejas';` `nombre + '  ' +apellido` `//'Martin Cejas'` `1 + '1' // '11'`

FUNCIONES DECLARADAS

Sintaxis	Uso	Ejemplo
`function nombre() {...}`	Son aquellas que se declaran usando la **estructura básica**. Pueden recibir un **nombre**, escrito a continuación de la palabra reservada **function**, a través del cual podremos invocar. Se cargan antes de que cualquier código sea ejecutado	`function saludar() {` `console.log('Hola, soy una funcion declarada');` `}` `saludar();` `// 'Hola, soy una funcion declarada'`

FUNCIONES EXPRESADAS

Sintaxis	Uso	Ejemplo
`let variable = function() {...}`	Son aquellas que se asignan como valor a una variable a través del cual podremos invocar. Se carga únicamente cuando el intérprete alcanza la	`let saludar = function() {` `console.log('Hola, soy una funcion expresada');`

	línea de código donde se encuentra la función	`}` `saludar();` `// 'Hola, soy una funcion expresada'`
`let saludar = function(param1,param2,...paramN)` `{...}` `saludar(arg1, arg2,..., argN);` `function(param1 = 'Valor por defecto')` `{...}`	Puede recibir **parámetros**, los cuales deben ir dentro de los paréntesis, lo que importa es respetar el orden de los parámetros al momento de invocar la función, Javascript asigna valores en el orden que estos lleguen. Los parámetros pueden estar definidos con valores por defecto. Por último, llamamos *parámetros* a las variables que escribimos cuando definimos la función, y *argumentos* a los valores que enviamos cuando invocamos la función.	`let saludar = function(nombre, apellido) {` `console.log(nombre + ' ' + apellido);` `}` `saludar('Martin','Cejas');` `// 'Martin Cejas'`

SCOPE o AMBITO LOCAL

Sintaxis	Uso	Ejemplo
`function nombre() {` `let variable = 'hola';` `//scope local solo //visible dentro de la //función nombre()` `}`	Se da cuando existen variables declaradas exclusivamente *dentro de una función*. **Fuera de esta las variables son inexistentes.** Las variables con scope local tienen predominancia sobre las con scope global	`function saludo() {` `//var local` `let saludo = 'Holis';` `return saludo;` `}` `console.log(saludo());` `// 'Holis'` `console.log(saludo);` `// Undefined Variable`

SCOPE GLOBAL

Sintaxis	Uso	Ejemplo
`let variable = 'hola';` `//visible fuera de la //función` `function nombre() {` `//scope local` `//visible en la función` `...` `}`	Cuando las variables se declaran fuera de cualquier función y así tienen un alcance global, visible en cualquier lugar de código, incluso dentro de la función.	`//var local` `let saludo = 'Holis';` `function saludo() {` `return saludo;` `}` `console.log(saludo());` `// 'Holis'` `console.log(saludo);` `// 'Holis'`

CONDICIONAL IF/ELSE IF

Sintaxis	Uso	Ejemplo
`if (condicion){` `//codigo que ejecuta si //la condicion es //verdadero` `}else{` `//código que ejecuta si //es false` `}`	SIMPLE: El condicional nos permite evaluar condiciones y realizar diferentes acciones según el resultado de esas evaluaciones Flujo: Si esta condición es verdadera, hacer esto. Si es falsa, hacer esto otro.	`let numero = 4;` `if (numero == 4){` `return 'Si es 4';` `}else{` `return 'No es 4';` `}`
`if (condicion){` `...` `}else if (condicion) {` `//código que ejecuta si`	Al igual que el anterior, else if es un adicional, para evaluar otra condición en caso de que la 1ra sea falsa, se puede agregar todos los bloques else	`let numero = 4;` `if (numero > 4){` `return 'Si es mayor';` `}else if (numero == 4){`

`//la 2da condicion es //verdadera` `}else{` `//código que ejecuta si //ambas condiciones` `//son falsas` `}`	if que queramos, solo uno será verdadero, de lo contrario entrará en acción el bloque else, si es que este existe. SOLO UNA CONDICIÓN PUEDE SER VERDADERA. Flujo: Si esta condición es verdadera, hacer esto. Si es falsa, ver si esto es verdadero. Si es verdadero, hacer esto. Si es falso, hacer esto otro.	`return 'Es es 4';` `}else {` `return 'es menor';` `}`

CONDICIONAL IF TERNARIO

Sintaxis	Uso	Ejemplo
`condicion ? expresion para el true : expresion para el false;`	No lleva llaves **{}** del bloque de código a evaluar, ni la palabra reservada **if** y else, se escribe de forma horizontal, en la misma línea. Resultado: se puede asignar a una variable.	`let clase = 1;` `let res = clase == 1 ? 'Presentarse' : 'Iniciar';` `console.log(res);` `//'Presentarse'`

CONDICIONAL SWITCH

Sintaxis	Uso	Ejemplo
`switch (expresion){` `case caso1:` `console.log('caso1');` `break;` `case caso2:` `console.log('caso2');` `break;` `default:` `console.log('default');` `break;` `}`	Pregunta por algo, si es verdadero, ejecuta un bloque de código, similar a los condicionales que vimos, solo que usamos una expresión para evaluar si se cumple en algún caso. El default, es opcional, se ejecutara su bloque de código en caso de que no encuentra ninguna coincidencia Se pueden agrupar casos, listándolos uno encima de otro, para luego declarar/ejecutar un mismo bloque de código para cualquier caso de ese grupo.	`let clase = 2;` `switch (clase){` `case 1:` `console.log('Intro');` `break;` `case 2:` `console.log('Bases');` `break;` `case 3:` `console.log('Funciones');` `break;` `default:` `console.log('Examen!');`

		break; }

ARRAYS

Sintaxis	Uso	Ejemplo
let array = [elemento1, elemento2,...,elementoN];	**array**, similar a definir una variable solo que para indicar que es un array utilizaremos corchetes **[ ]** para indicar el inicio y fin del mismo, una coma **,** para separar los elementos. Podemos almacenar la cantidad de elementos que queramos, sin importar el t**ipo de dato** Cada elemento ocupa una posición numerada en el array, siempre comienza en 0. Para acceder a un elemento en particular debemos especificar el nombre del array seguido del **índice** (array[i])x.	let array = ['Martin',27, true]; console.log(array[0]); //'Martin'
array.length; //long del array	Longitud de un array puede resultar muy útil, para esto utilizamos la palabra length	let array = ['Martin',27, true]; console.log(array.length); //3

MÉTODOS DE ARRAYS

Sintaxis	Uso	Ejemplo
let array = [elemento1, elemento2]; array.push(elemento);	.**push()** nos permite agregar al final de array uno o más elementos, modifican al mismo la longitud como la cantidad de índices, los elemento a insertar tenemos que pasar como	let nombres = ['Martin','Eze', 'Lean']; nombres.push('Lopi','Esteban'); console.log(nombres); //['Martin', 'Eze', 'Lean', 'Lopi', 'Esteban']

	parámetros separados por coma.	
`array.pop();`	.**pop()** nos permite sacar el último elemento del array, no recibe ningún parámetro, modifican la longitud como la cantidad de índices. Retorna el elemento extraído del array original, podemos guardalo en una variable	`let nombres = [ 'Martin', 'Eze', 'Lean', 'Lopi', 'Esteban'];` `let ultimo = nombres.pop();` `console.log(ultimo);` `//Esteban`
`array.shift();`	.**shift()** extrae del array al elemento ubicado en el índice 0. Retorna el elemento extraído del array original, podemos guardar él mismo en una variable	`let numeros = [3,4,5];` `let primero = numeros.shift();` `console.log(ultimo);` `//3`
`array.unshift();`	.**unshift()** nos permite agregar al inicio de array uno o más elementos, modifican al mismo la longitud como la cantidad de índices. los elemento a insertar tenemos que pasar como parámetros separados por coma.	`let numeros = [3,4,5];` `números.unshift(1,2);` `console.log(numeros );` `//[1,2,3,4,5]`
`array.indexOf(elemento);`	**.indexOf()**, se ejecuta sobre un array definido y recibe como parámetro el elemento que deseamos **buscar,** en caso de encontrar el valor retorna el índice del elemento, caso contrario devuelve -1. Él orden de búsqueda es desde él índice 0 hacia la derecha del array	`let numeros = [1,2,3,2];` `console.log(numeros.indexOf(2));` `// 1` `console.log(numeros.indexOf(4));` `// -1`
`array.lastIndexOf(elemento);`	**.indexOf()**, idem a indexOf() El orden de búsqueda es desde el final del array hacia él índice 0	`let numeros = [1,2,3,2];` `console.log(numeros.indexOf(2));` `// 3`
`array.join();`	**.join(),** nos permite unificar todos los elementos de una	`let numeros = [1,2,3,4];` `console.log(numeros.join());`

	array en un string separados por comas , recibe como parámetro cualquier carácter como elemento delimitador, en caso de que no queramos la , como separador	`// 1,2,3,4`

CICLOS (FOR)

Sintaxis	Uso	Ejemplo
`for(let i = 0; i < n; i++){` `//código que queremos iterar` `}`	**for**, consta de tres partes, variable inicializadora, condición a evaluar y el modificador de la variable inicializadora. **for (let i = 0**; i< n; i++**)** definimos en que nro empezaremos a iterar, **for** (let i = 0**; i< n**; i++**)** la condición a evaluar, si es true continua iterando, si es false, detiene el ciclo. **for** (let i = 0; i< n; **i++)** modificador (incremento o decremento), luego de ejecutar la 1ra iteración modificamos la variable inicializadora, puede ser en 1 o con alguna cuenta	`for(let i = 0; i < 5; i++){` `console.log("contando: "+ i);` `}`

ARROW FUNCTIONS

Sintaxis	Uso	Ejemplo
`let nombreFuncion = (parámetros) => //cuerpo de la función;`	Se las nombra asignándolas a una variable (función expresada), si no, son anónimas. Se la llama con el nombre de la variable. **sin parámetros:** () **un parámetro:** sin paréntesis **dos parámetros:** (a, b) **=>:** reemplaza a la palabra	`let sumar = (a, b) => a + b;`

	function **omitir {} y return:** si hay una línea de código y es la que hay que retornar.	

CALLBACKS

Sintaxis	Uso	Ejemplo
`funcion(callback) => {//cuerpo de la función que ejecuta la función pasada como callback cuando lo especifiquemos` `};`	Función que se pasa como parámetro de otra. Es ejecutada cuando es necesario por la función que la recibe. **callback anónimo:** se escribe la función anónima como argumento **callback definido:** se pasa el nombre de la función como argumento, *sin paréntesis.* **dos parámetros:** (a, b) **=>:** reemplaza a la palabra function **omitir {} y return:** si hay una línea de código y es la que hay que retornar.	`//declaro una función` `funcion nombreCompleto(nombre, apellido) {` `return nombre + " " + apellido;` `};` `//declaro una función que recibe otra por parámetro` `function saludar (nombre, apellido, callback) {` `return "Hola "+ callback(nombre, apellido);` `};` `//invoco la función y paso como argumento cualquier otra función` `saludar ("Armando", "Barreda", nombreCompleto);`

CLOSURES

Sintaxis	Uso	Ejemplo
`funcion contenedora () => {` `function interna () {` `//cuerpo de la función` `}` `//cuerpo de la función` `};`	Función anidada, vive dentro de otra. Existe solo cuando y mientras se ejecuta la función contenedora. La función close tiene acceso a todos los **parámetros** y **variables** de la función contenedora. **llamado local:** las closure se llaman/invocan dentro de la función contenedora.	`//declaro una función` `funcion saludoPersonalizado(nombre) {` `let saludoGenerico = "Hola ";` `//closure/función local` `function saludar() {` `return saludoGenerico + nombre;`

		} //llamado local return saludar(); };

OBJETOS LITERALES

Sintaxis	Uso	Ejemplo
let objeto = { clave: valor, clave2: valor2, clave3: function (this.clave2) { //cuerpo de la función }, } //función constructora: function Nombre (propiedad1, propiedad2) { this.propiedad1 = propiedad1, this.propiedad2 = propiedad2, } //creación de una instancia/objeto a partir de la función: let variable = new constructora(valorprop1, valorprop2);	Estructura de datos que puede contener propiedades y métodos. Van separados por comas. Es buena práctica poner coma después del último elemento. **propiedad:** clave: valor, **método:** función asignada como valor de una clave. Se invocan> **objeto.metodo()**. *Recordar que los paréntesis hacen que se ejecute.* **dot notation:** objeto.propiedad, así accedemos a una propiedad específica de un elemento. **this.propiedad** llama a una propiedad estando dentro del objeto. ***this*** refiere al objeto. **FUNCIONES CONSTRUCTURAS** El **nombre** de la función constructora con primera letra mayúscula, por convención. Los **parámetros** son obligatorios si no se define lo contrario. **this.propiedad** define la propiedad del objeto que se está creando, su **valor** viene por parámetro.	//función constructora function Auto (marca, modelo) { this.marca = marca, this.modelo = modelo, }; //instanciar let miAuto = new Auto("Hyundai". "i10");

	INSTANCIAR En una nueva variable, con el método **new**, se llama a la constructora por el **nombre** y se agregan entre paréntesis los **valores** correspondientes a cada propiedad del objeto literal, **en orden**.	

DESTRUCTURING

Sintaxis	Uso	Ejemplo
`//ARRAYS` `let [nombre1, nombre2, nombre3] = array;` `//saltar un valor` `let [nombre1, , nombre2] = array;`	**Desestructurar** un array es almacenar todos o algunos de sus elementos en una nueva variable o muchas (una para cada uno). ***No modifica*** *el array de origen,* ***copia*** *los valores en variables***.** **+Extraer varios datos:** Los nombres de las variables a crear van entre corchetes, luego de declarar el tipo de variable. El valor se les asignará siguiendo el orden del array. *Si quiero saltar un dato, dejo un espacio vacío entre comas en la posición correspondiente a ese dato.* **+Extraer un solo dato:** *también hay que poner los corchetes* con el nombre de la variable adentro. Se iguala a la variable que contiene el array, y tendrá el valor del índice 0 del array.	`let stock= [“tomate”, “manzana”, “hummus”, “pan”]` `//destructuring` `let [verduras, frutas, aderezos, pan] = stock` `/*En cada variable quedó almacenado un valor del array, siguiendo el del mismo:*/` `console.log(verduras) // “tomate”` `console.log(frutas) // “frutas”` `console.log(aderezos) // “aderezos”` `console.log(pan) // “pan”`
`// OBJETOS LITERALES`	**Desestructurar** un objeto es almacenar todos o algunos de sus elementos o propiedades en una nueva variable o muchas (una para cada uno). ***No modifica*** *el array de origen,* ***copia*** *los valores en variables***.**	

SPREAD OPERATOR		
Sintaxis	Uso	Ejemplo