# Javascript

Codigo inline y external link

No ocupa mucha memoria y tiene simple sintax

Puede ser usado en diferentes plataformas y es orientado a objetos

## Dynamic typing

No se tiene que definir la variable Javascript la asigna automatico.

var num = 5; // integer

var name = 'hardif'; // string

console.log(typeof name); // this will print string

## Type coerción

Es cuando uno imprime y el consle log lo convierte en string

var firstName = 'Jhon';

var age = 28;

//Type Coercion

console.log(firstName + ' ' + age); // Jhon 28

## Escaping

Cuando se usa back slash para meter caracteres especiales

console.log('John\'s team Win. ' + 'average team john: ' + averageJohnT + ' average team Mike: ' + averageMikeT);

## Printing

Se pueden imprimir caracteres separados por coma

var firstName = 'Jhon';

var age = 28;

console.log(firstName, age); // Jhon 28

## Ternary operator

Es un if corto por ejemplo

age >= 18 ? console.log('mayor') : console.log('menor');

switch (key) {

case value:

break;

default:

break;

}

# Function Staments and Expressions y declarations

## Diferencias entre statements y expressions

Las expresiones siempre producen un valor no importa lo largo que sean mientras que el codigo termine en un valor o resultado.

// Function Declaration

function name(params) {}

//Function expression

var whatDoYouDo = function (job,firstName) {

switch (job) {

case 'teacher':

return firstName + ' Teaches kids how to code';

case 'driver':

return firstName + ' drives';

case 'designer':

return firstName + ' designs';

default:

return firstName + ' does something else';

}

}

console.log(whatDoYouDo('teacher', 'John'));

2+3 es una expresión

Cada vez que esperemos un valor es un siempre tenemos que escribir una expresión

## FL Statement

Statements hacen acciones pero no producen un resultado inmediato

Lo que esta dentro de los paréntesis del if(4>3) es una expresión pero el if como tal es un statement porque no devuelve ningún valor y se ve undefined en la consola.

if (5 > 4) {

// el if como tal es un statement pero adentro hay una expresion

}

# Objetos y propiedades

Objeto literal ejemplo

Diferencia entre arrays y objects

Los arrays importa mucho el orden en cambio en los objetos no

var john = {

}

## This

Keyword this dentro de un objeto quiere decir el mismo objeto.propiedad

## LOOPS

For and while

Continue and break statements

## Continue

Se brinca uno y continua con el loop

Se puede poner que cuando sea diferente de string se brinque esa iteración

function loop() {

for (var i = 0; i < 10; i++) {

if (i == 5) {

continue; // breaks when i is greater than 5 and ends the loop

}

console.log(i); // Won't print 5

}

}

## Break

Quiera las iteraciones y se sale como decir que quiebre apenas se encuentre algo que no sea string

function loop() {

for (var i = 0; i < 10; i++) {

console.log(i); // it will print from 0 to 6

if (i > 5) {

console.log(i); // it will print 6

break; // breaks when i is greater than 5 and ends the loop

}

}

}

## Return

Tambien termina las funciones o los procesos

function testReturn() {

if (age) {

return age;

}

return age;

}

Funciones anónimas son funciones sin nombre

function (params) {}

## Contexto de ejecución y ejecución stack también creación y fases de ejecución mas hoisting

Es una caja que encierra todo nuestro codigo

Global context es cuando todo el código que no esta dentro de funciones es ejecutado entonces es solo para variables y funciones que no están adentro de funciones todo lo que ejecutamos en el global context se atacha a el window por ejemplo declarar

Es lo primero que se ejecuta por default código que no esta dentro de funciones statments y es asociado con global object en el buscador es el Windows object todo lo que declaramos es automática mente añadido al window

Var lastName = ‘hardif’

Lastname === Window.lastName also window.prop

DOM = Document object model

Despues de esto la primer function que se invoca o se llama comienza hacer nuevos contextos de ejecucion sobre el global context y se hace el action context

Hosting functions declarattions and variables son leidas y puestas en el contexto global las variables como undfined y las funciones como funciones. Antes que que fase de ejecución comience y las variables van a ser puestas hasta la ejecución

Creation phase y ejecución phase

Hay grados de ejecución las variables están en el global execution context object y las funciones y lo que esta adentro esta dentro de variable object of the execution global content de la función foo mientras que las variables están en el variable object of the global execution context object

Document.function() y funciona

Document.prop = undefined

Solo carga las funciones declarations y no las function expressions

caculateAge(); // will work because of hoisting

console.log(ageJohn); // Undefined

function caculateAge(birthyear){

console.log(2018 - birthyear);

}

var ageJohn = 28;

Las funciones declarations son puestas en el varable object content apuntando a la función por eso se pueden usar mas rápido mientras las variables son puestas en el mismo contexto pero undefined

Donde podemos accesar ciertas variables o funciones

Cada función crea un scope que es un espacio donde las variables que ellas defines son accesibles

## Lexical scoping

Quiere decir que las funciones dentro de otras funciones pueden accesar el contenido de la función padre

## Scope chain

Es cuando vamos de abajo para arriba por ejemplo aquí

var a = 'Hello!';

first();

function first() {

var b = 'Hi!';

second();

function second() {

var c = 'Hey!';

console.log(a + b + c);

}

}

# Objetos en JS

Dicen que todo en JS son objetos es casi el 100% correcto

Hay dos tipos grandes tipos de valores primitivos y objetos

## Primitivos

Numbers Strings Booleans Undefined NULL después de eso todo son objetos

## Object constructor

En JS las clases de llaman constructores o prototipos y se pueden crear las instancias que uno desee.

var Person = function (name, yearOfBirth, job) {

this.name = name;

this.yearOfBirth = yearOfBirth;

this.job = job;

}

Todos los objetos en js tienen prototype y eso hace posible la herencia

var Person = function (name, yearOfBirth, job) {

this.name = name;

this.yearOfBirth = yearOfBirth;

this.job = job;

}

Person.prototype.calculateAge = function(){

console.log(2019 - this.yearOfBirth);

};

## PROTORYPE CHAIN

Es cuando un objeto busca propiedades o métodos de su padre pasando por el padre y la clase superior objeto y terminando en null

## Function constructor

La forma mas usada de hacer prototipos

Los métodos tienen que ser metidos a la función constructor por prototype es la mejor forma asi no todos los cargan pero lo pueden usar cuando se usa el prototype los métodos no están adjuntos a las instancias pero lo pueden usar

## Diferencia entre Object.create y function constructor

Object.create construye un objeto directamente del que le pasamos como parámetro pero el otro viene del prototipo de los objetos

var personProto = {

calculateAge: function () {

console.log(2016 - this.yearOfBirth);

}

}

var john = Object.create(personProto);

john.name = 'John';

john.yearOfBirth = 1990;

var jane = Object.create(personProto, {

name: {value: 'jane'},

yearOfBirth: {value: 1969}

})

Nos deja agregar cosas mas complicadas que las funciones constructor son las dos formas mas comunes de hacer objetos

# Primitivos vs objetos

## Primitivos

Numbers, strings, bolleans, null y undefined

var a = 23

var b = a;

a = 46;

console.log(a,b); //a = 46 y b = 23

Porque no hay referencia simplemente se hace una copia de la variable

## Objects

var obj1 = {

name: 'Jhon',

age: 26

}

var obj2 = obj1;

obj1.age = 28;

console.log(obj1.age); // age = 28

console.log(obj2.age); // age = 28

Past by value

Pasa normal en js con todos los tipos primitivos porque crea una nueva copia en memoria en cambio.

Past by reference

Con las funciones cuando uno le pasa un objeto copia al objeto que esta adentro porque es pasado por referencia lo que se le haga al objeto dentro de la funcion se va a ver reflejado en el objeto afuera de la función.

# Functions

## Callback Functions

Son funciones que reciben una function com argumento

var years = [1990, 1965, 1937, 2005, 1998];

function arrayCalc(arr, fn) {

var arrRes = [];

for(var i = 0; i < arr.length; i++){

arrRes.push(fn(arr[i]));

}

return arrRes;

}

function caculateAge(el) {

return 2019 - el;

}

function isFullAge(el) {

return el >= 18;

}

var ages = arrayCalc(years, caculateAge);

var fullAges = arrayCalc(ages,isFinite);

## Functions que devuelven functions

function interviewQuestion(job) {

if (job === 'designer') {

return function (name) {

console.log(name + ', can you please explain what UX design is?');

}

}else if (job === 'teacher') {

return function (name) {

console.log('What subject do you teach, ' + name + '?');

}

} else {

return function (name) {

console.log('Hello ' + name + ' what do you do?');

}

}

}

var treacherQuestion = interviewQuestion('teacher');

treacherQuestion('hardif');

interviewQuestion('teacher')('Mark')

las funciones son instancias de los objetos

para hacer funciones anónimas y asi ocultar propiedades o informacion podemos usar las IIFEs que son funciones invocadas inmediatamente y también se le puede pasar variables y no se pueden ver las variables dentro de la IIFE sirve para data privacy.

## IIFE Immediately Invoked Function Expressions

(function (){

var score = Math.random() \* 10;

console.log(score);

})();

(function (goodLuck){

var score = Math.random() \* 10;

console.log(score);

})(5);

Para crear un buen proyecto es necesario usar modules eso nos ayuda a encapsular data y mantener las cosas separadas y organizadas y cada modulo mantiene su contenido privado pero también se puede tener modulos públicos

Se crean con una IIFE y se devuelve un objeto con las funciones que queremos usar afuera del IIFE

Sepramiento de consentimiento quiere decir que cada parte tiene que hacer lo suyo independientemente

Y Pero si lo ponemos asi nos va a dar un error porque no tiene nombre y va a pensar que es una declaración

function () {

}

## Closures

Son funciones que devuelven una función y pueden usar las variables y parámetros de la función padre gracias al scope chain

function retirament(retiramentAge) {

var a = ' years left';

return function calcRetirament(yearOfBirth) {

var age = 2018 - yearOfBirth;

console.log((retiramentAge - age) + a);

}

}

var retiramentUS = retirament(66);

console.log(retiramentUS(1990));

function interviewQuestion(job) {

return function test(name) {

if (job === 'designer') {

console.log(name + ', can you please explain what UX design is?');

}else if (job === 'teacher') {

console.log('What subject do you teach, ' + name + '?');

} else {

console.log('Hello ' + name + ' what do you do?');

}

}

}

var designerQuestion = interviewQuestion('designer');

console.log(designerQuestion('hardif'));

## Bind, Call and Apply methods

Barrowing functions sin paréntesis

Con call se puede usar el metodo barrowing con call se pasan por coma y apply se pasa un array pero aquí no va a funcionar porque la función pide dos parámetros no un array

var john = {

name: 'John',

age: 26,

job: 'teacher',

presentation: function(style, timeOfDay) {

if (style === 'formal') {

console.log('Good ' + timeOfDay + ', Ladies and gentlemen! I\'m ' + this.name + ', I\'m a ' + this.job + ' and I\'m ' + this.age + ' years old.');

} else if (style === 'friendly') {

console.log('Hey! What\'s up? I\'m ' + this.name + ', I\'m a ' + this.job + ' and I\'m ' + this.age + ' years old. Have a nice ' + timeOfDay + '.');

}

}

};

var emily = {

name: 'Emily',

age: 35,

job: 'designer'

};

john.presentation('formal', 'morning');

john.presentation.call(emily, 'friendly', 'afternoon');

## Call

Se usa para method barrowing de esta forma

john.presentation.call(emily, 'friendly', 'afternoon');

## Apply

Los parametros se ponen como si fuera un array la función o el método debería de aceptar un array

john.presentation.apply(emily,['friendly', 'afternoon']);

## Bind

Return a function. Preselecciona una parte de la function o un parametron dejandonos agregar el otro parametros después. Preset algunos argmentos y de hecho se llama carrying.

Cambia el this entonces agarra el this del objeto que le pasemos como referencia

Bind nos deja preseleccionar parámetros entonces nos deja agregar parámetros y guardarlos como función y después usar el otro parámetro y nos deja crear una función nueva con un parámetro introducido

var johnFriendly = john.presentation.bind(john, 'friendly');

johnFriendly('morning');

var years = [1990, 1965, 1937, 2005, 1998];

function arrayCalc(arr, fn) {

var arrRes = [];

for(var i = 0; i < arr.length; i++){

arrRes.push(fn(arr[i]));

}

return arrRes;

}

function caculateAge(age, el) {

return age - el;

}

function isFullAge(limit, el) {

return el >= limit;

}

var ages = arrayCalc(years, caculateAge.bind(this,2019));

var japanFullAge = arrayCalc(ages,isFullAge.bind(this,20));

console.log(ages);

console.log(japanFullAge);

## Event delegation

Event bubbling cuando un elemento es fire o trigger cuando se apreta un botón que esta en cierto lugar comienza a subir hasta llegar al DOM y eso hace un target y es guardado en como una propiedad

El target es el elemento al que le damos click o el evento en ese momento

Nos permite adjuntar un evento que escucha en el nodo padre que va a disparar para todos los hijos es usado para listas, botones y nos ahorra mucho código

Usamos event delegation cunado hay un elemento con muchos hijos en lugar de hacer un handler para cada uno mejor se hace un evento delegation

Cuando queremos agregar un evento a un elemento cuando la pagina aun no esta cargada

El target puede usar una función parentNode y para eliminar se usa el child.

var ctrlDeleteItem = function (e) {

var itemID, splitID, type, ID;

itemID = e.target.parentNode.parentNode.parentNode.parentNode.id;

if (itemID) {

splitID = itemID.split('-');

type = splitID[0];

ID = splitID[1];

console.log(itemID);

}

# Modules

Se hacen para separar el código y las funcionalidades estructurando asi las cosas por partes.

La mayoría de las veces es bueno separar los controladores y que no se comuniquen entre asi asi después cuando queremos hacer el App mas grande esto no interfiere por ejemplo tener un controlador para el Budget y uno para la interfaz UIController y BudgetController

Esto se llama Separation of concerns porque la UI esta separada de la data y no se comunica con ella. Entonces creamos un AppCOntroller para que este maneje los dos otros controladores.

Agregar las clases o los ID a una objeto asi después es mas fácil de cambiarlos. Todos lo que tenga que ver con el UI va en el UI controler.

var UIController = (function () {

var DOMstrings = {

inputType: '.add\_\_type',

inputDescription: '.add\_\_description',

inputValue: '.add\_\_value',

inputBtn: '.add\_\_btn'

}

return{

getInput: function () {

return{

type: document.querySelector(DOMstrings.inputType).value,

description: document.querySelector(DOMstrings.inputDescription).value,

value: document.querySelector(DOMstrings.inputValue).value

}

},

getDOMstrings: function () {

return DOMstrings;

}

}

})();

Javascript ES6

Const y let

Const crea constantes y no se pueden cambiar después se declara y se inicializa de un solo sino da error

También let y const son block scope if, for, foreach todo son bloques para let y const

Otra diferencia con var y let es que el hoisting no funciona con let no lo carga antes lo carga como not defined es por algo que se llama temporal death zone que dice que se pueden usar hasta que estén declaradas y definidas

Otra diferencia es con las varibles iguales si usamos i como varible y después lo usamos en un for con var se cambia la variable pero con let no

let i= 26;

for (let i = 0; i < 5; i++) {

console.log(i) // 0, 1, 2, 3, 4

}

console.log(i); //26

con var el segundo log seria 5

template string

backticks

se mete todo el código dentro de las comillas que van para atrás ` ` y las variables nada mas se les pone ${nombre variable} tambuen se pueden llamar funciones y calacular cosas dentro de las ${function(parameter}}

Destructors

Permite hacer inicializaciones mas rápido por ejemplo tanto en arrays como en objeto y podemos usar las variables con el nombre que les pasamos

const [name,age] = ['hardif' , 27];

console.log(name);

console.log(age);

const {firstName,lastName} = obj;

console.log(firstName);

console.log(lastName);

sino queremos el mismo nombre solo le tenemos que hacer lo siguiente

const {firstName = a,lastName = b} = obj;

console.log(a);

console.log(b);

callback hell es arreglado con promises que estamos por verlo

es un evento que mantiene el rastro sobre si un evento ha pasado o no determina lo que pasa después de que el evento ha pasado implementa el concepto de el valor futuro que estamos esperando

las promises tienen diferentes estados el primero es

Pendin

Settled/Resolve si sale bien fulfilled o Rejected si sale mal

Se inicializa con un new y recibe una función

const getIDs = new Promise((resolve,reject) => {

});

En este caso no vamos a usar el reject pero se usa cuando estamos solicitando información del server

ASYNC / AWAY

Para consumir promises se usa async function (){}

APIS and Ajax so called same origin policy in JS nos previene de hacer request de Ajax

Cross origin resource sharing

# RULES

No pueden comenzar por números o símbolos solo si es underscore \_ o dólar signo $

No se pueden usar palabras reservadas como nombre de variables

Truly and falsy

Falsy variables = undefined, null, 0, '',NaN

# Buenas practicas

Siempre ponerle nombre significativo a las variables

Usar camel case

Primero inicializar variables y después declararla es una practica común

DONT REPEAT YOUTRSELF (DRY)

Crear IIFES para proteger el código

Crear un objeto con las variables del DOM para que sea mas fácil de cambiarlas a futuro

Crear modulos para proteger nuestro código

Usar event delegation para no crear un monton de listeners

Crear una funtion de iniciación

Es mejor crear una estructura para tener variables de las mismas asi no andan un monton de variables regadas por todo lado y las podemos traspasar con mas facilidad

Hay muchas versiones de javascript después del 2015 pero se sigue usando ES5 los navegadores usan transpiling and polyfilling.