# Good Parts

## Participantes :

* Javi : [javidevictoria@gmail.com](mailto:javidevictoria@gmail.com)
* Alice : [adacostac@gmail.com](mailto:adacostac@gmail.com)
* Renata: [rmonteal14@gmail.com](mailto:rmonteal14@gmail.com)
* Pau ( PROFE) : [pfraces@gmail.com](mailto:pfraces@gmail.com)
* Salva: [puglisisalva@gmail.com](mailto:puglisisalva@gmail.com)
* Jose: [jfontenla.glez@gmail.com](mailto:jfontenla.glez@gmail.com)
* David : [loscabezonesdavid@gmail.com](mailto:loscabezonesdavid@gmail.com)

## Enlaces del curso:

* Repositorio: <https://github.com/pfraces-wip/curso-javascript>
* Foro: <http://cursojavascript.foroactivo.com/>
* Chat: <https://cursojavascriptsede.slack.com>
* Drive : <https://drive.google.com/drive/folders/1T3kmj2KNmG0FHyDhbSieRyT7zWF8uBzb?usp=sharing_eip&ts=5bb7910c>

## Explicaciones futuras

* TDD(TEST-DRIVER-DEVELOPER)
* UNIT-TEST
* MARKDOWN
* NODEJS
* SEMVER(version semantica)
* JSON
* Package-lock.json
* Git
* BDD
* *\*Cascada\**
* *\*npx\**
* *\*regex\**

## INTRODUCCIÓN

MDS (MARKDOWN)

**Package.json** te dice las dependencias del proyecto.

**¿Qué es el package.json?** Lo podemos considerar como un manifiesto de nuestro proyecto

Npm es una herramienta de Node para administrar los paquetes. Se suele instalar junto con Node y tiene dos roles fundamentales:

* **Manejar la publicación de un proyecto al regístro público de npm** (para que otros puedan descargarlo y utilizarlo como dependencia en sus propios proyectos).
* **Administrar las dependencias del proyecto**

Dentro de este archivo se definen y maneja características como:

* Nombre del proyecto
* Versión
* Dependencias
* Autores
* Licencia
* Y mas…

(**yarn ->** administración de paquetes para Node creado por FACEBOOK)

Con el Package.json podemos asegurar que quienes tengan una copia del mismo, podrán acceder a las mismas propiedades y sincronizar entre múltiples partes de cada vez que decidan hacer un cambio. *Por esto lo podemos considerar el manifiesto de nuestro proyecto*

**npm init : Para crear un package.json automáticamente.**

(**yarn init :** administración de paquetes Node FACEBOOK)

Se crea con npm init –y -> nos crea el package.json

El archivo package-lock.json se borra

**Instalación de dependencias :**

Poner el comando “*Npm install*” continuado de estas dos librerías js

Mocha

Expect.js

**Npm install [librerias] –save o npm install [librerias] –save-dev :** te guardan las dependencias en package.json

Node\_modules estarán todas las librerías de las depndencias, esta carpeta ~~es bueno ignorarla~~ [GIT IGNORE EN LA CARPETA DEL PROYECTO PARA INDICARSELO A GIT]

Git add –all

Git remove HEAD

Git commit –m “ ”

Git log

**[*notas*]Quitar los proxis**  en carpeta del usuario

npm config set proxy http://10.110.8.42:8080

npm config set https-proxy http://10.110.8.42:8080

npm config rm proxy

npm config rm https-proxy

npm config edit

git config --global http.proxy http://10.110.8.42:8080

git config --global --unset http.proxy

## lenguajes tipados y no tipados

javascript lenguaje no tipado

Principio del formulario

Tipos simples

* numeros
* strings
* boolean ( ocupan 1bit 0 ó 1)
* Null
* Undefined

Nota : node en cualquier simbolo del sistema te abre un interprete de node

“var” declarar la variable. Por defecto sino se define por defecto javascrip te lo inicializa a undefined

Definimos la variable

Variable = 3 // aquí tendremos un tipo de numero

JAVASCRIPT ES UN LENGUAJE NO TIPADO DINÁMICO

*REPL interprete de node*

Operadores Lógicos son EXPRESIONES (porque te devuelven un resultado) Todas los valores de tipos son expresiones porque tiene valor.(EXPRESIONES CIUDADANOS DE 1º ORDEN)

SENTENCIA no devuelve valor

EJEMPLO

EXPRESION -> x=2

SENTENCIA -> var x

Var x = 2 (sentencia y expresión)

If (EXPRESION BOOLEANA)

{

}

Else

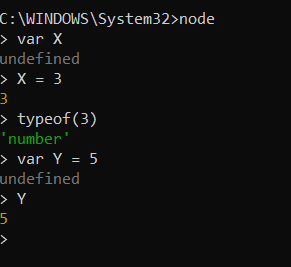
{

}

Operadores

* Unarios(unario)
* Binaries
* Ternarios

**Duda a PAU**



**Codigos de powershell**

**UNIT TEST**

Mocha

Expect.js

Los ficheros tiene que tener al final del nombre **.spec.js** el **spec es de specification.**

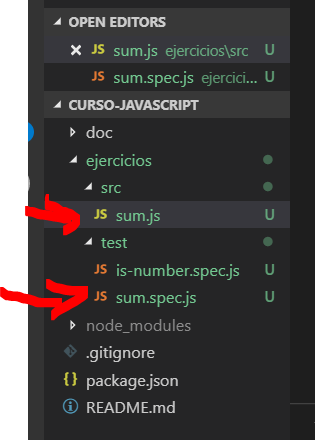
Describe es con mocha

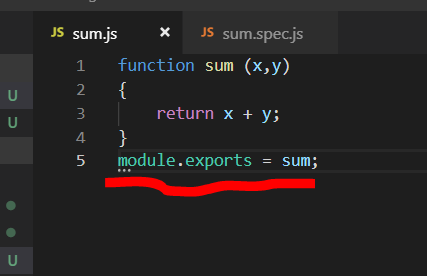
Expect(“lo que voy a testear”,”el resultado que tienes que hacer” )

LAS DOS CARPETAS AL MISMO NIVEL

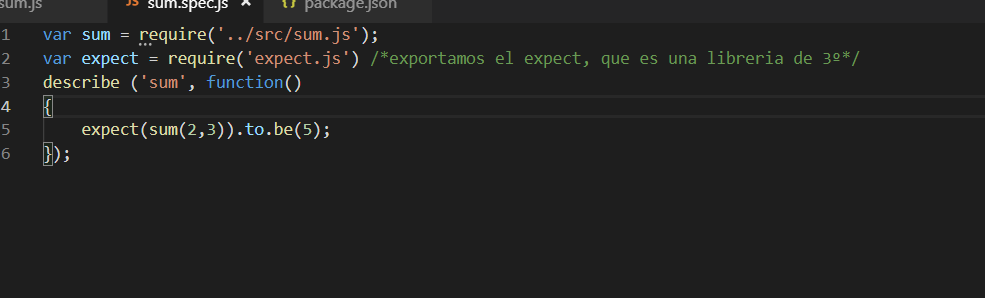
SRC ->

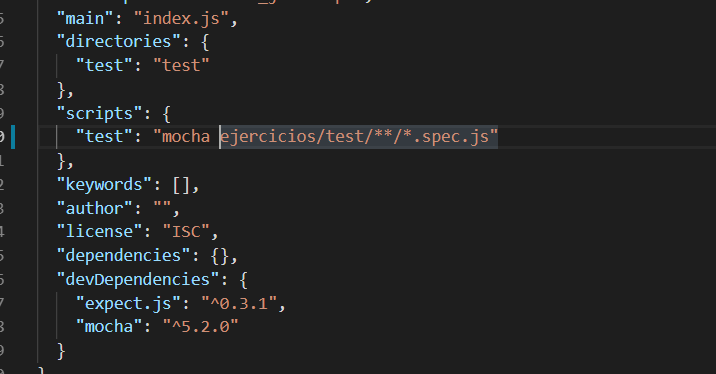
TEST->MISMA ESTRUCTURA DE CARPETAS Y FICHEROS QUE SRC





\*MIRAR EL MODULE.EXPORTS\*





Indicamos que tenemos test para ejecutar. Para ejecutarlos se usa el comando npm run <scripts> **(se pone entre <> cuando es una variable….realmente iría sin <>)**

**HOISTING // function steatments**

### Clase 3 :

Variables (bindings) <https://eloquentjavascript.net/02_program_structure.html>

ECMAscript 3

\*NOTA\* LET Y/O CONST => SERÁ NUESTRO VAR

Paso de variables a una función por valor y referencia ( ver las variables simples y las compuestas ….escalares??)**Consultar también para C#**

Paso por valor por defecto para tipos simples

Paso por refecerncia por defecto para los tipos compuestos

**Expresion Statments**

**Statements group**

Toda función que devuelve un boolean es un predicado .LOS PREDICADOS NUNCA LANZAN UN THROW. (BUSCAR SI ESTO ES ASÍ).

CODE SMELL -> ESTUDIAR SUS REFACTORS

## 3.- Objects

## 4.- Functions

## 5.- Inheritance

## 6.- Arrays

## 7.- Regular Expressions

## Apendice A.- Awful Parts

## Apendice B.- Bad Parts