**Git cmds**

Git - - version

Gitadd

Gitcommit

Gitremove

Pwd - conocer ruta

Cd navegar por directorios

Ls - listar directorio

Git init para inicializar un repositorio

Git config

Git config —global user.name para saber el nombre que está configurado

Git config - - List para listar la configuración

Si a continuación añadimos el nombre entre comillas nos modifica la configuración

Git config —global user.mail para saber que correo hay configurado para este git

Man git para acceder al manual (q para salir)

Git help para obtener ayuda específica. Ejemplo git help config

Git status para saber el estado del git

Git add [readme.me](http://readme.me) añade el fichero que estará listo para comiera

Si hago git add . Te añade todo

Git rm - - cache readme.md

Git rm nombre fichero

Git commit -m “commit hecho”

Git push -u origin master para

Touch para crear el fichero independiente

Echo -m “cuarto fichero” >> newFile.txt crea fichero y concatena ese texto

Git checkout -b nombre de la rama para crearla

Git checkout nombre de la rama que te quieres mover

Para unir ramas nos situamos en la rama master y unimos la rama nueva con comando: git merge nombre de la rama. Esto no significa borrar la nueva rama sino unir el contenido de ambas, siempre de la master y traes el contenido de la rama nueva. Nunca al revés.

Git mv nombrefichero.extensión y a continuación el nuevo nombre que le quieras llamar.

También sirve para mover el fichero de directorio. Git mv nombrefichero.extensión a mover a continuación el directorio/nombre fichero.extensión

Gir rm - - cached nombrefichero.extensión para dejar de cachear y “actualizar” ese fichero

/ raíz del directorio

./ directorio dónde estoy

../ un salto arriba ../../ dos saltos y así successivamente

Git log muestra el historial de los comió que hemos hecho

Git log -1 muestra el logro del último logro

Git log - -oneline te pinta sólo el encabezado del commit

Git log - -stat te da más información, te indica que cambios ha habido de forma específica

Git log - -patch da todavía más información del comió

Git stash coge todo lo que no se haya comiteado se guarda en un stash y nos permite trabajar en otras cosas y luego retomar esta tarea como pausada

Git stash list y nos muestra los stash que hay creados

Git stash pop el último estas añadido se puede recuperar y retomamos la tarea

Por el head sabemos en que rama estamos

Cómo hacemos un commit de forma correcta:

Cabecera y shift+enter para añadir la descripción completa del comió

Estructura de un mensaje <https://chris.beams.io/posts/git-commit/>

Crear una carpeta nombre-apellido y hacer la subida al repositorio de Skylab

Git branch para saber que rama estamos

Git checkout nombre-rama cambias de rama

git push origin --delete victor-alemany eliminamos la rama en remoto

Git clone url a clonar si le pongo nombre al final le da ese nombre sino por default le da el nombre del repositorio

Git branch - - all me indica la raise que hay en el repositorio remoto

Git remote add origin url (origin es el nombre que le damos)

Git fetch - - all actualiza la copia local con respecto a la remota pero no el working directory. Es decir no te actualiza la rama

Git pull hace un fetch y luego te hace un merge con tu rama con el contenido nuevo

Librería vs framework

Librería es una agrupación de scripts y un framework es una agrupación de librerías

Git merge origin master antes de hacer el push de mi rama

Después commit otra vez si hay diferencias (por ejemplo commit -m “merge master”)

Y por último ya hacemos el push final

Git push -u origin feature/victor-alemany para hacer el push de la rama al repositorio remoto

Spa single page application

Ajax es usar el objeto xhtml http request take a look w3c try Ajax

Npm repositorio de liberías, código etc por excelencia

Indexar previamente el contenido que vamos a representar en nuestra página web

Lighthouse revisión de un plugin integrado en google donde te hace un análisis de tu código

Challenge/name-topic

Lecture/name-topic

Hot fix/name-topic

User stories how to write correct user stories

As a “power user”

I want see little kitty’s

On my webpage

User story - es una descripción de como una persona interactúa con un producto y utiliza alguna funcionalidad y debe expresar el quien el como y el por qué. Con sus respectivos criterios de aceptación

Epic - es una agrupación de user stories

Theme - es una agrupación de épicas

User stories applied libro de mike Cohn

Snippets vs code

Prettier

marialunarillos.com

<https://www.linkedin.com/sales/ssi>

let singer = {surname:'Turner'};

let pilot = {surname: 'Kamal'};

let temporal = singer.surname;

singer.surname = pilot.surname;

pilot.surname = temporal;

console.log(singer.surname);

console.log(pilot.surname);

//singer.surnam = [pilot.surname, (pilot.surname = singer.surname)][0];

//creamos array y almacenamos 'kamal' en la posición 0

//asignamos 'turner' a pilot.surname

//almacenamos 'turner' en la posición 1

//asignamos a singer surname el valor almacenado en el indice 0 del array, es decir 'kamal'

Restructuring asignment

//que es lo mismo que hacer [singer.surname, pilot.surname ] = [pilot.surname,singer.surname] -- destructuring asignment

Recursividad  concepto cuando un objeto llama a otro objeto

Let president = {president : ‘Putin’};   para luego crear una propiedad que llama a sí misma  president.next = president;  añadimos la propiedad al objeto que así mismo o asignarle esa propiedad al crearlo pero con un valor null y luego posteriormente lo modificamos.

Mutation

Significa cambiar valores de un objeto a través de la referencia de un valor desde otro objeto que apunta al mismo objeto en común a través de la misma propiedad

//singer.surnam = [pilot.surname, (pilot.surname = singer.surname)][0];

//creamos array y almacenamos 'kamal' en la posición 0

//asignamos 'turner' a pilot.surname

//almacenamos 'turner' en la posición 1

//asignamos a singer surname el valor almacenado en el indice 0 del array, es decir 'kamal'

//que es lo mismo que hacer [singer.surname, pilot.surname ] = [pilot.surname,singer.surname] -- destructuring asignment

Prototype

Es cuando hereda una propiedad de un padre con la propiedad porto que se escribe como \_\_proto\_\_

Shadowing   es cuando una propiedad es hererdada porque pregunta primero si pertenece al objeto y si no pertenece, la hereda del prototype. Lo que hace primero es llamara a la propiedad .hasOwnProperty y luego devuelve el valor dependiendo de si tiene o no la propiedad en el objeto.

Por ejemplo

Let human = {teeth:32};  let Gwen = {\_\_proto\_\_: human, teeth: 31};

Hay una propiedad que es hasOwnProperty que nos indica que si tiene esa propiedad definida en el objeto,

Si asignamos una propiedad a un objeto en el cual no existe esta propiedad, la inserta a pesar de que exista un prototype dentro del objeto. Si quisiéramos mutar el valor del prototype deberíamos llamar a la propiedad en si misma \_\_proto\_\_

let sherlock = {

name : ' pepito'};

let watson = {

name : sherlock.name};

let obj = {};

obj.\_\_proto\_\_.smell = 'banana'; —> apunta al prototype de obj que es el objeto generico / padre. Esto es peligroso porque es llamado prototype pollution

console.log(sherlock.smell);

console.log(watson.smell);

Este codigo ejecuta exactamente lo mismo:

// usar prototype es una mala práctica

class Spiderman{

lookOut(){

alert('My spider-sense is tingling');

}

}

let miles = new Spiderman();

miles.lookOut();

let SpidermanPrototype = {

lookOut (){

alert('My spider-sense is tingling');

}

}

let miles = {\_\_proto\_\_: SpidermanPrototype};

miles.lookOut();

Los callbacks y los closures no son más que funciones

* Una función tiene acceso a las variables globales pero las variables de función no pueden ser llamadas desde el scope global

Por definición si no sobreescribimos una variable es una constante. Es decir un let X que luego no vuelve a declararse a let X, se debe cambiar a const por definición.

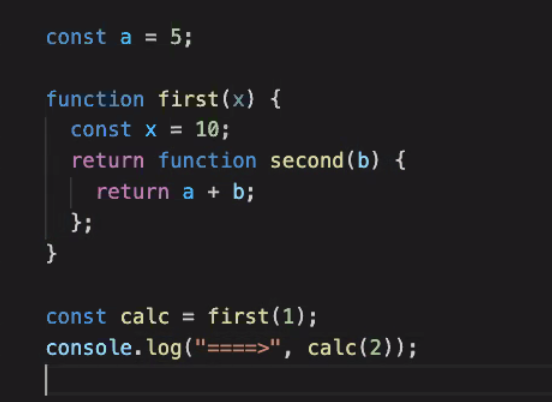
* Un clousure es una función declarada dentro de una función
* Un Callback no es más que una función que recibe por parámetro otra función y que luego es invocada dentro de la función que recibe esta función por parámetro.

Cuando vemos esto myFunction()() significa que

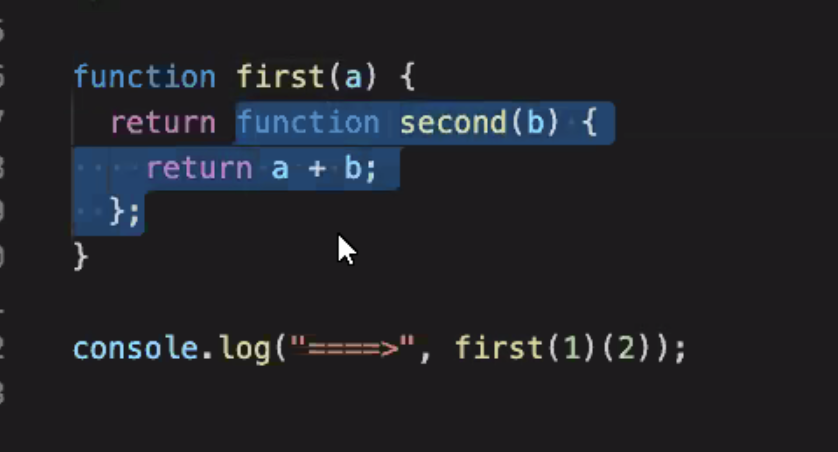
Si le paso valores va a ejecutar la primera función con el primer parámetro y la segunda función con el segundo parámetro. Va buscando en el scope que tiene por encima.

* Si apuntamos a la ruta localhost:9000/ ens obre el sonarqube amb pass admin admin

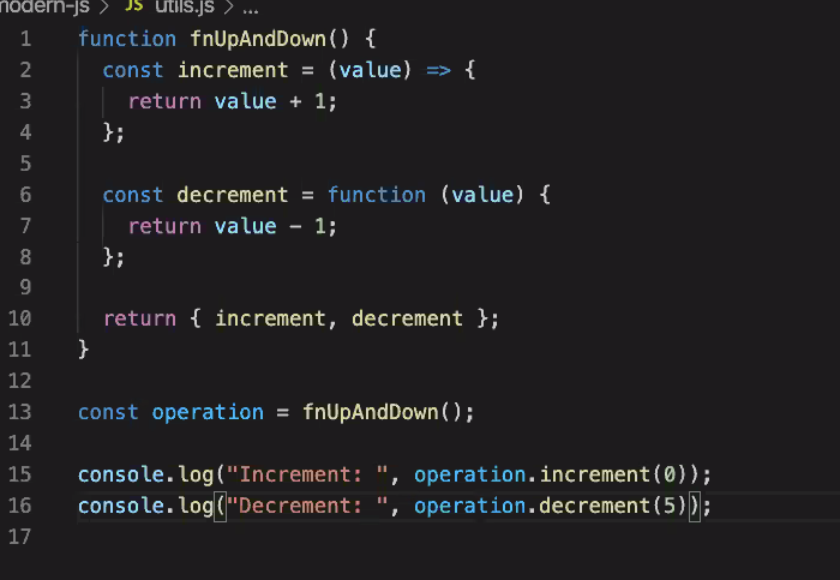
Una variable es un identificador



Que es lo mismo que esto:



Podríamos devolver en una sola función dos valores? A la vez? No pero sí que podemos guardarlos en un objeto y luego acceder a ellos:



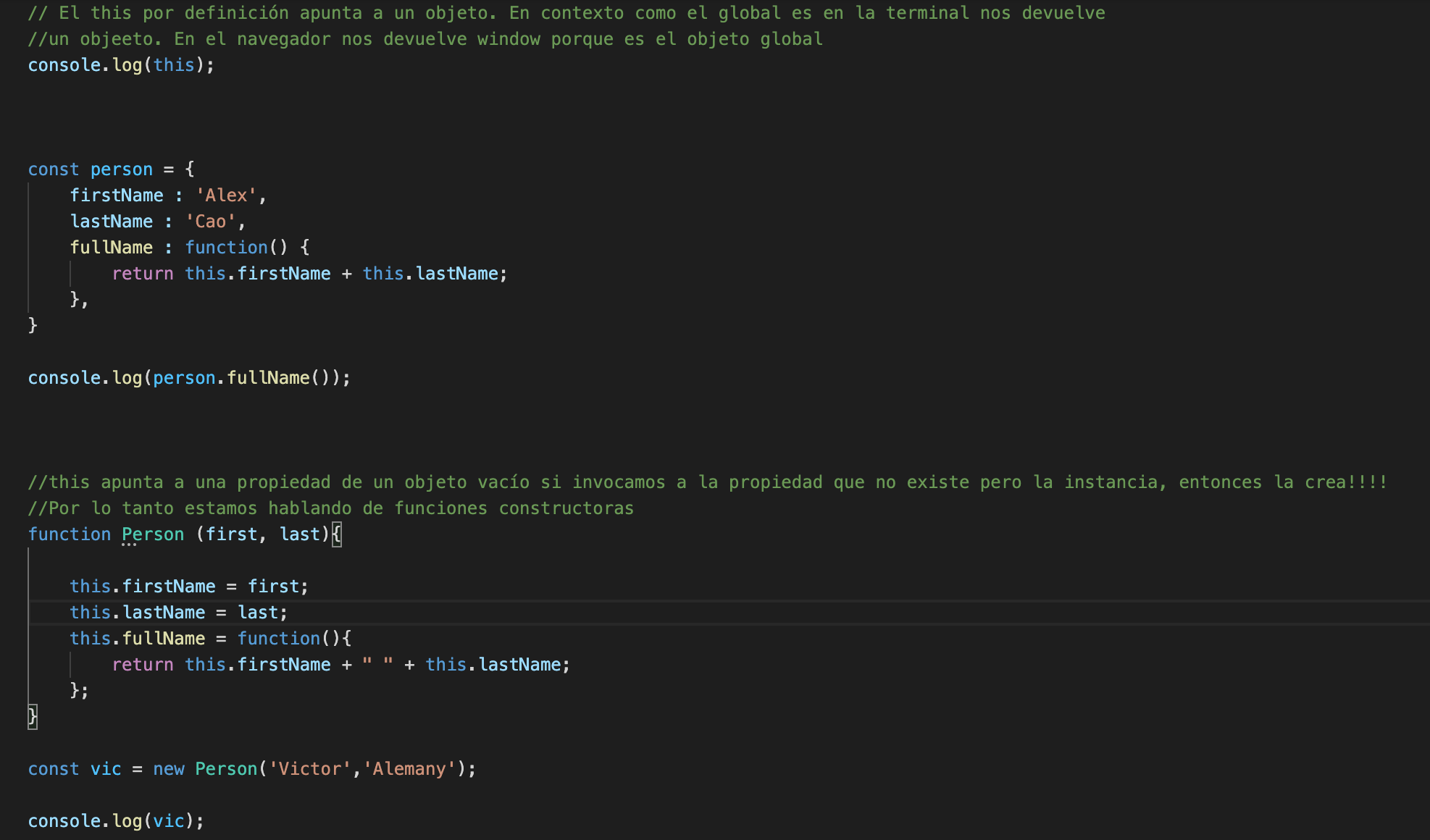
También podríamos hacerlo así:



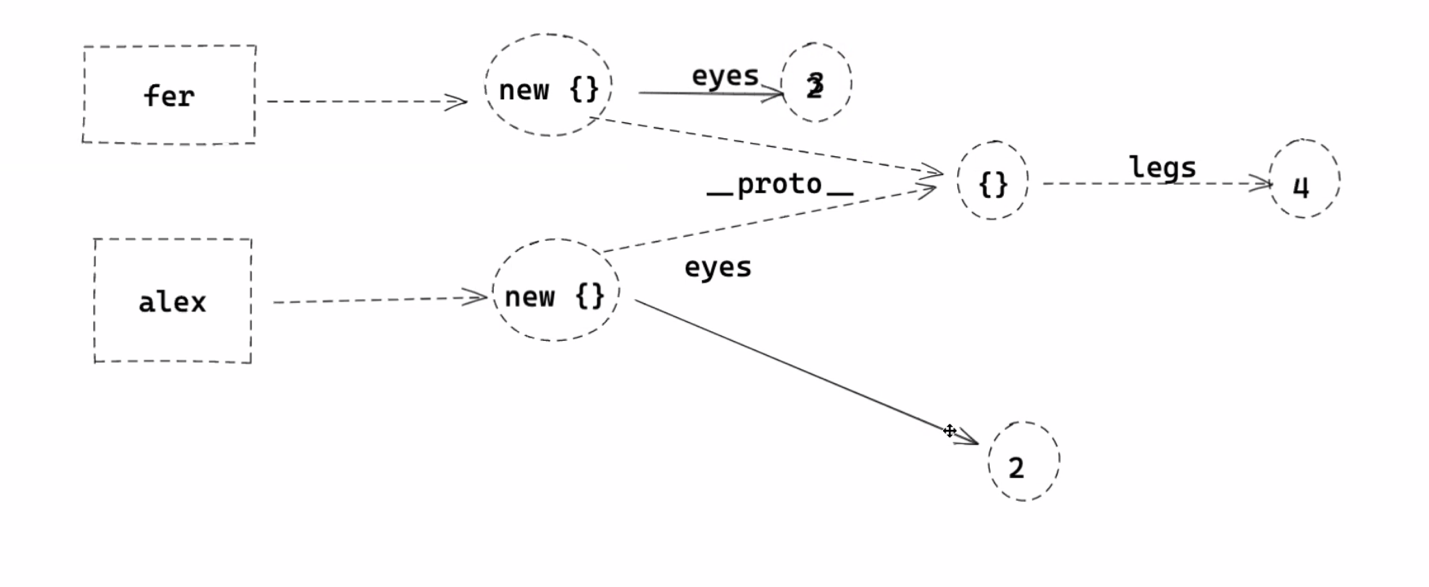
* This en JS es una variable que se comporta diferente a otros lenguajes de programación conocidos. Se comporta distinto en versión estricta o no.

Se comporta también de una forma u otra dependiendo del contexto en el que se ejecuta:

This y constructor



Cuando hacemos new sobre el objeto constructor lo que hace es una inversa de la propiedad es decir construye hacía afuera en el “diagrama”



This en un método el propietario es el objeto al que pertenece

En una función el this pertenece al scope externo o superior

Que es un método vs qué es una función

* Un componente es una clase o una función y que se muestra con html + css

Las tres formas de heredar este comportamiento general es con prototype, funciones y clases

Challenge

Snapshot de todos los componentes del proyecto

Y practicar con el async await de mdn