***Notas del Curso.***

**Instalación de Node.js**

Si vas a trabajar con Node.js, lo primero que tienes que hacer, es instalarlo en tu máquina. Ya sea con Windows, Linux o Mac, si vas a la web de Node.js (<https://nodejs.org>), la web detectará tu sistema operativo, y te ofrecerá un paquete con el que instalarlo.

Simplemente pulsa en el botón verde de la versión que quieras (mi recomendación es siempre usar las versiones LTS) y completar el proceso de instalación. Dependiendo del sistema operativo, te hará más o menos preguntas, pero con las opciones por defecto se instalará bien.

Como consejo, asegúrate de tener una buena conexión a internet cuando lo instales, para que tarde poco tiempo.

Una vez lo hayas instalado, para comprobar que todo funciona correctamente, abre una terminal (en Windows, CMD o PowerShell valen perfectamente) y escribe:

**node -v**

Ese comando te devolverá la versión de Node.js que se ha instalado.

También nos habrá instalado NPM, el gestor de paquetes. Para asegurarte de que está instalado, puedes ejecutar:

**npm -v**

Y te devolverá la versión de NPM que hay instalada.

Con esto, ya tenemos instalado Node.js y NPM, que es todo lo que necesitamos para empezar con nuestro curso de Fundamentos de Node.js.

# Node: orígenes y filosofía

**¿Qué es?**

* Entorno de ejecución de JS fuera del navegador
* Fuera del navegador
* Su uso en Servidores, IoT, herramientas

**Bases**

* Concurrente  
  – Entradas y salidas asíncronas  
  – Un proceso por cada núcleo del procesador
* Motor V8  
  – JS en código máquina, en vez de interpretarlo.
* Modular  
  – Todo lo que no sea sintaxis, son módulos
* Orientado a eventos  
  – Bucle de eventos que se ejecutan constantemente  
  – Reactivo

Actualmente Node se ha convertido en una de las tecnologías más relevantes en la industria de la programación, debido que nos permite correr código de una manera rápida y escalable de grandes maneras desde un servidor.

Node en si es un entorno de desarrollo que nos permite ejecutar JavaScript fuera del navegador, fue creado en el 2009 y corre con el motor V8 creado con C++, lo que nos ofrece una gran potencia a la hora de desarrollar.

Como características de Node principales, es concurrente, gracias a su procesador que es realmente monohilo, este nos permite realizar procesos de forma asíncrona, haciendo así que nunca se quede bloqueado.

# EventLoop: asíncrona por diseño.

**Event Queue:** Contiene todos los eventos que se generan por nuestro código (Funciones, peticiones, etc.), estos eventos quedan en una cola que van pasando uno a uno al Event Loop.

**Event Loop:** Se encarga de resolver los eventos ultra rápidos que llegan desde el Event Queue. En caso de no poder resolverse rápido, enviá el evento al Thread Pool.

**Thread Pool:** Se encarga de gestionar los eventos de forma asíncrona. Una vez terminado lo devuelve al Event Loop. El Event Loop vera si lo pasa a Event Queue o no.

# 

# Monohilo: implicaciones en diseño y seguridad.

# PROCESO DE NODE

# 1.- Va a abrirse un proceso, ese proceso es un proceso de node

# 2.- Interpreta todo el archivo

# 3.- Convertirlo a código maquina

# 4.- Prepara todo lo que necesita para ejecutarse

# 5.- Se ejecuta

# 6.- Se cierra el proceso, y termina

# DESVENTAJAS MONOHILO

# - Si no se manejan bien los errores y uno truena, ya no continua con los procesos posteriores

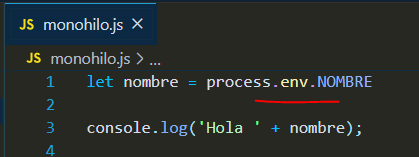
# - Debes estar pendiente de todo el código

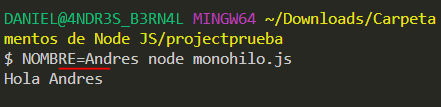
# Importante: Cuando ocurre un error dentro de alguno de los hilos y no se controla apropiadamente (catch); Node detiene todos los hilos de ejecución. Esto puede ser muy peligroso, debido a que es difícil determinar cuál fue el origen del problema y en que punto de ejecución se encontraba cada hilo cuando fue detenido.

# Variables de entorno.

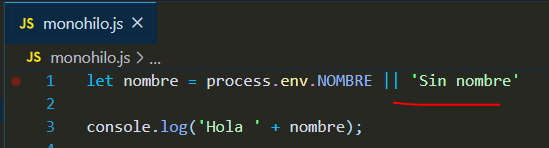
Las variables de entorno son una forma de llamar información de afuera a nuestro software, sirve para definir parámetros sencillos de configuración de los programas de modo que puedan ejecutarse en diferentes ambiente sin necesidad de modificar el código fuente de un script.

El objeto **process** nos da información sobre el procesos que está ejecutando este script.  
La propiedad **env** es la que nos da acceso a las variables de entorno de manera sencilla.

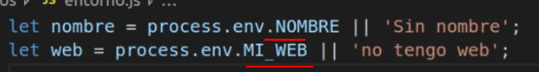




Si queremos dejar un valor por defecto. Quedaría de las siguiente manera:



Detalle importante como buena practica colocar las variables de entorno en MAYUSCULA.



# Herramientas para ser más felices: Nodemon y PM2.

* **Nodemon** (***Desarrollo***): Es un gestor que nos ayuda a detectar los cambios, compilarlo y ejecutarlos.

<https://nodemon.io/>

**Para su instalación:**



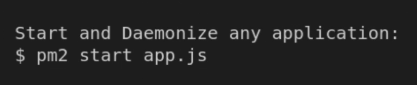
**Para su uso:**



**PM2** (***Producción***): Es parecida a nodemon simplemente más avanzada y mas compleja. No se debe utilizar en desarrollo por que dará más problemas que soluciones. Nos ayudara a ver los datos de nuestra aplicación en producción, como el uso del CPU, memoria, cuantas veces se ha reiniciado.

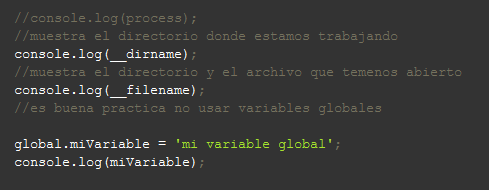
<https://pm2.keymetrics.io/>





# Globals.

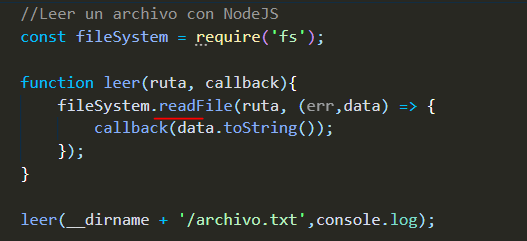
# 

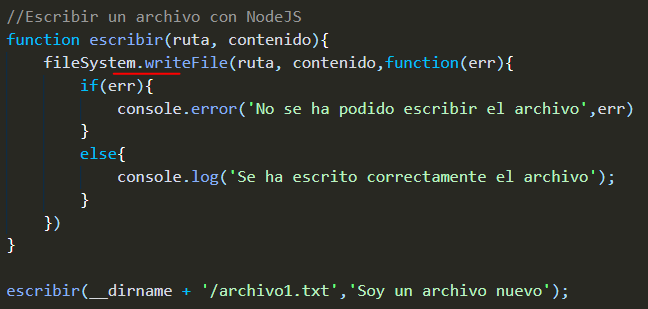


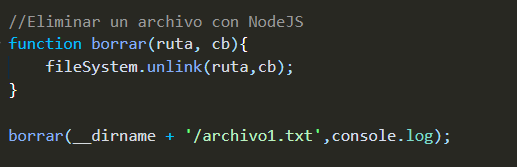
# File System.

En node podemos **crear, editar y eliminar archivos** desde el servidor. La mayoría de estas acciones tienen métodos síncronos y asíncronos, por ejemplo readFile y readFileSync.

“Use readFile siempre que sea posible, readFileSync bloquea el hilo mientras la solicitud es resuelta.”



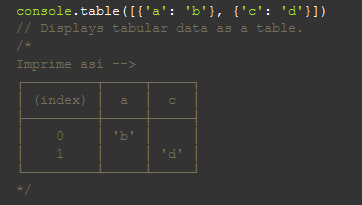


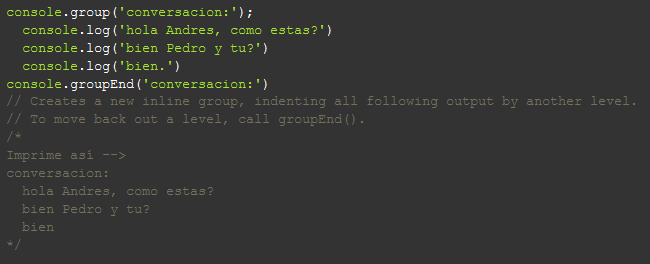


## **Console.**

Con Console podemos imprimir todo tipo de valores por nuestra terminal.

* **console.log**: recibe cualquier tipo y lo muestra en el consola.
* [**console.info**](http://console.info): es equivalente a log pero es usado para informar.
* **console.error**: es equivalente a log pero es usado para errores.
* **console.warn**: es equivalente a log pero es usado para warning.
* **console.table**: muestra una tabla a partir de un objeto.
* **console.count**: inicia un contador autoincremental.
* **console.countReset**: reinicia el contador a 0.
* **console.time**: inicia un cronometro en ms.
* **console.timeEnd**: Finaliza el cronometro.
* **console.group**: permite agrupar errores mediante identación.
* **console.groupEnd**: finaliza la agrupación.
* **console.clear**: Limpia la consola.



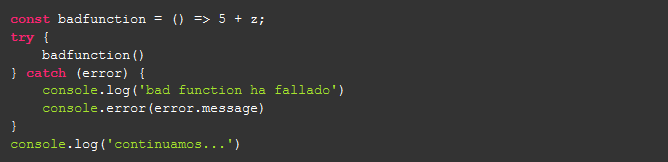


## **Errores (Try/Catch)**

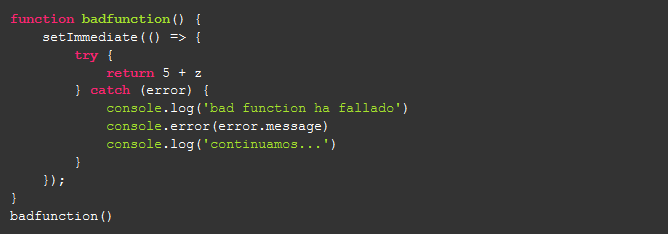
Cuando se genera un error, node propaga el error hacia arriba, hasta que esta es capturado. Si el error no se captura node se detiene.

Siempre que sea posible debemos capturar todos los errores que se puedan generar en nuestros hilos.

Try/Catch nos permite capturar los errores:



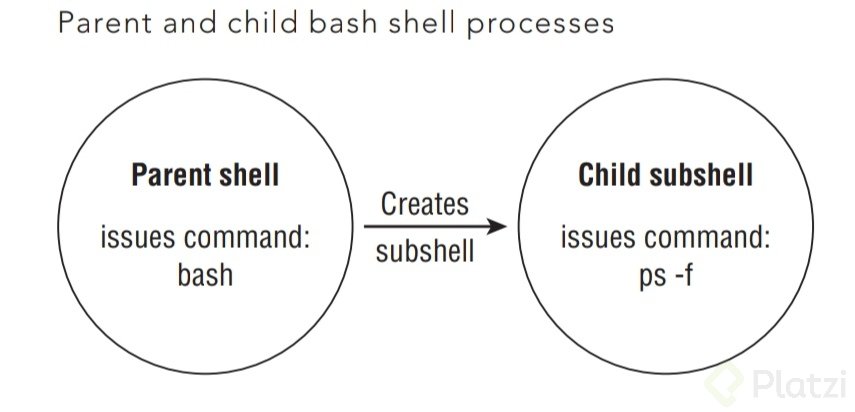
Debemos tener bien presente que hay funciones (en este caso setImmediate) que no se van a ejecutar sobre el hilo principal por lo que tendremos que controlar los errores directamente en la funcion si deseamos manejar errores asíncronos:



## **Child Process.**

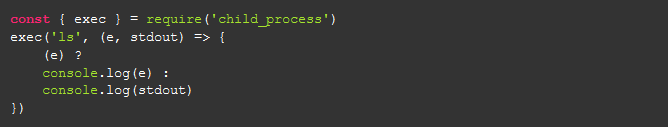
**Funcionamiento de los child process**  
Un proceso es la ejecución de algo (programa, commando, etc) que terminara en un tiempo finito.

Cuando se ejecuta un programa estos corren sobre la terminal y cuando necesitan crear un nuevo proceso crean una nueva terminal(**Parent shell**).

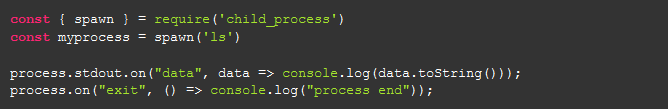


En node podemos crear procesos hijos que ejecuten cualquier acción de nuestro sistema operativo, como si nos encontráramos en la línea de comandos.

Podemos llamar a exec para ejecuciones sencillas:



Podemos llamar a spawn para obtener el proceso: La ventaja de este enfoque es que obtienes mayor control del proceso, y del estado en el que se encuentra.



**HTTP.**

Node nos ofrece el módulo HTTP el cual nos permite principalmente crear un servidor en nuestro computador.  
Uno de los métodos principales de este módulo es createServer, el cual nos permitirá abrir un puerto para crear el servidor.



# OS.

# El módulo de OS, Operative System, nos permite ejecutar acciones de más bajo nivel en nuestro sistema, permitiéndonos conocer una gran variedad de detalles del mismo.

# Como la memoria disponible que tiene, el total de la memoria, la interfaz de red, etc.

# Esto nos será de gran ayuda a la hora de ejecutar o crear proyectos que necesiten información de una máquina para ejecutar una operación.

# 

# Process.

El objecto process es una instancia de EventEmitter; podemos suscribirnos a él para escuchar eventos de node.

* **UncaughtException**: Permite capturar cualquier error que no fue capturado previamente. Esto evita que Node cierre todos los hijos al encontrar un error no manejado.



* **exit**: Se ejecuta cuando node detiene el eventloop y cierra su proceso principal.



# Aquí las opciones de process: <https://nodejs.org/dist/latest-v12.x/docs/api/process.html>

# Gestión de paquetes: NPM.

**npm** (Node Package Manager) es un administrador de paquetes que permiten ejecutar funciones ya realizadas y validadas y de esta manera acelerar y asegurar la calidad de nuestro proceso de desarrollo.

Podemos buscar módulos para casi todo en:

<https://www.npmjs.com>

**// Para instalar un módulo de npm en nuestro proyecto**

$ npm install is-odd

**// Para requerir el modulo**

*const isOdd = require('is-odd');*

*console.log(isOdd(3)); // true*

**// Para revisar que los paquetes no están actualizados a nivel global dentro de nuestro proyecto**

npm outdated -g --depth=0

Va a imprimir algo así:

Package Current Wanted Latest Location

firebase-tools 8.0.1 8.0.2 8.0.2 global

npm 6.13.7 6.14.4 6.14.4 global

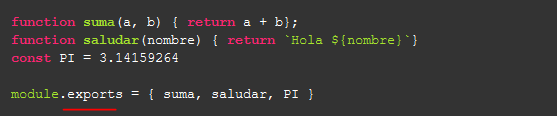
**// Para actualizar todos los paquetes a nivel global dentro del proyecto**

npm update -g

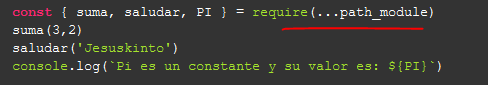
# Construyendo módulos: Require e Import

En Node tenemos una forma de importar módulos la cual es con el método **require**, el cual es la forma por defecto de importar módulos, ya sean nuestros propios módulos como los de otras personas en nuestros proyectos JS, pero suele haber mucha confusión debido al import.

**// Modulo**



**//Importación**



**Import** es la forma de importar módulos en Ecmascript, el cual es un estándar de JavaScript para la web, esta forma de importar en teoría Node no la acepta oficialmente, a no ser que usemos su modo de .mjs “***index.mjs***”.



# Módulos útiles.

# El módulo de cifrado de ****bcrypt**** nos permite construir una plataforma de seguridad utilizando contraseñas encriptadas con Salt.

# <https://www.npmjs.com/package/bcrypt>

# 

# 

# Moment. js es una librería que nos permite solventar problemas con formateo de fechas e implementa un sistema de manejo de fechas mucho más cómodo.

# <https://momentjs.com/>

# 

# 

# Sharp puede convertir imágenes grandes en imágenes JPEG, PNG más pequeñas y compatibles con la web de diferentes dimensiones.

# <https://sharp.pixelplumbing.com/>

# 

# Datos almacenados vs en memoria.

**Memoria RAM**: Es la memoria de corto plazo del computador. Su función principal es recordar la información que tienes en cada una de las aplicaciones abiertas en el computador, mientras este se encuentre encendido. Sin embargo cuando escribimos en memoria es **ultra** **rápido** porque la CPU y la MEMORIA RAM a nivel interno están muy cerca y sus protocolos de conexión son muy rápidos.

**Disco Duro**: El disco duro guarda y protege los datos a largo plazo, lo que significa que quedarán guardados incluso si se apaga el computador. Sin embargo cuando queremos escribir en disco entramos en procesos mucho más lentos, porque el disco esta pensado para almacenar mucha información (1tb, 2tb, etc) en comparación de una Memoria Ram que su uso es exclusivo para la data que se esté manipulando dentro de los procesos corriendo por nuestro procesador y que corren en los thread de los núcleos.

# Buffers.

# **Los buffers son conjuntos de datos en crudo, datos binarios, que podemos tratar en NodeJS para realizar diversos tipos de acciones.**

# **Buffer es la forma en que podemos leer los datos es su forma más sencilla, en este caso Node transmite estos datos Buffer en binario.**

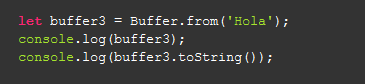
# **Poder convertir datos a Buffer nos ayudara a la velocidad y mejorar el rendimiento de la lectura y escritura de datos, debido que se hace en el lenguaje de más bajo nivel para las maquinas.**

# 

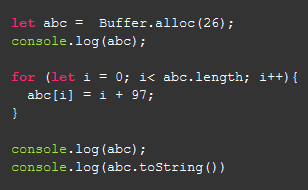
# **Para crear un buffer, con 4 espacios por ejemplo, podemos hacerlo con la siguiente sintaxis.**

# 

**Otras formas de crear un buffer.**



**Guardar el abecedario en un buffer.**



# Streams.

# Los streams son puntos de lectura de Buffers, con estos podemos declarar que acciones ejecutar cuando se recibe un buffer. Existen tres tipos de streams, stream de lectura, de escritura y de lectura y escritura.

# Un gran ejemplo de uso de los streams es el procesamiento de archivos grandes, como imágenes o videos, ya que podemos transformar estos a buffer y a través de los streams cargarlos o guardarlos parte por parte para mejorar el rendimiento de nuestro código.

# 

# Benchmarking (console time y timeEnd).

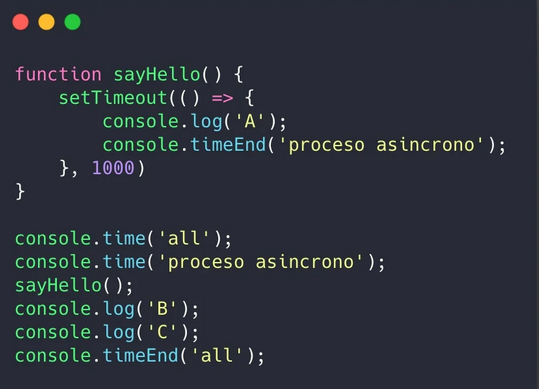
# La función console.time(‘nombre’) inicia un temporizador que se puede usar para rastrear cuánto tiempo dura una operación. El temporizador será identificado por el nombre dado a la consola. Ese mismo nombre se utilizará cuando se llame a console.timeEnd(‘nombre’) para detener el temporizador y obtener el tiempo demorado durante el proceso.

# 

# Su salida sera:



Benchmark = prueba de rendimiento o comparativa en inglés.



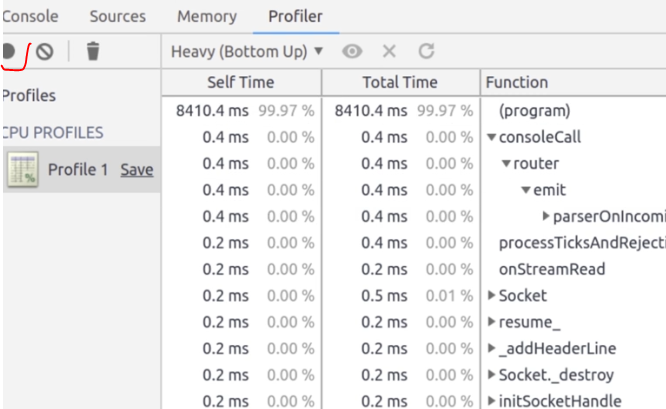
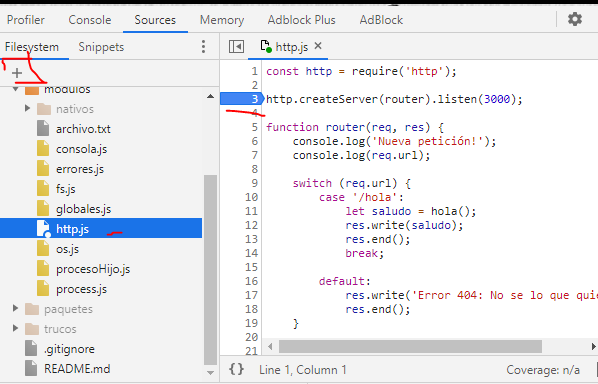
# Debugger.

# La característica de hacer debug de aplicaciones de **Node** en el navegador solo es soportada por Chrome. En **Firefox** no es posible realizarlo debido a que usa un **JavaScript Engine** diferente a **V8**.

# Para debuggear lo que tendremos que hacer es iniciar node con el flag *–inspect.*

# 

# Ingresamos a chrome://inspect y desde allí ya nos debe aparecer lo que estamos corriendo y abrirlo dando click sobre inspect.



# Error First Callbacks.

# *Si el V8 es el corazón de Nodejs, los callbacks son sus venas. 💚*

# Dentro de nuestro código JS siempre vamos a hacer uso de callbacks (funciones que pasamos como parámetros), debemos siempre tener muy presente de que si algo puede fallar , eventualmente fallará, asi que como buena practica debemos controlar los errores dentro de la asincronía. Lanzar una excepcion con Throw dentro de un callback asincrono no va a funcionar, solo nos va a funcionar en un bloque de código síncrono. Es por esto que aplicaremos Error First Callbacks.

# Los Error First Callbacks se utilizan para pasar primero el error y los datos posteriormente. Entonces, puedes verificar el primer argumento, es decir, el objeto de error para ver si algo salió mal y puedes manejarlo. En caso de que no haya ningún error, puedes utilizar los argumentos posteriores y seguir adelante.

# 

# Esto lo que nos permitirá es tener control de errores sobre la asincronía. (Callbacks)

# Scraping.

# Web Scraping es una técnica utilizada mediante programas de software para extraer información de sitios web. Usualmente, estos programas simulan la navegación de un humano en la World Wide Web ya sea utilizando el protocolo HTTP manualmente, o incrustando un navegador en una aplicación.

# 

# Automatización de procesos.

# Gulp es una herramienta que nos va a permitir automatizar tareas o procesos de la siguiente forma. Antes debemos instalarlo previamente con: *npm i gulp gulp-server-livereload*.

# 

# 

# 

# Aplicaciones de escritorio.

# Con NodeJS vamos a poder convertir nuestras aplicaciones Web en aplicaciones de escritorio con ayuda de Electron. Para ello debemos instalar electron:

# *npm i electron*

# Luego importaremos electron y realizaremos las siguientes configuraciones:

# 

# Para finalizar configuramos en el Package.json y ejecutamos:

# 

# 

# Importants Questions and Answers.

# 

# 

# 

# 

# 

# 

# 

# Malas.

# 

