



**UTN.BA**  
UNIVERSIDAD TECNOLÓGICA NACIONAL  
FACULTAD REGIONAL BUENOS AIRES

**Centro de  
e-Learning**

**CURSO:**

# **Desarrollo en Node JS**

**Centro de e-Learning SCEU UTN - BA.**

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

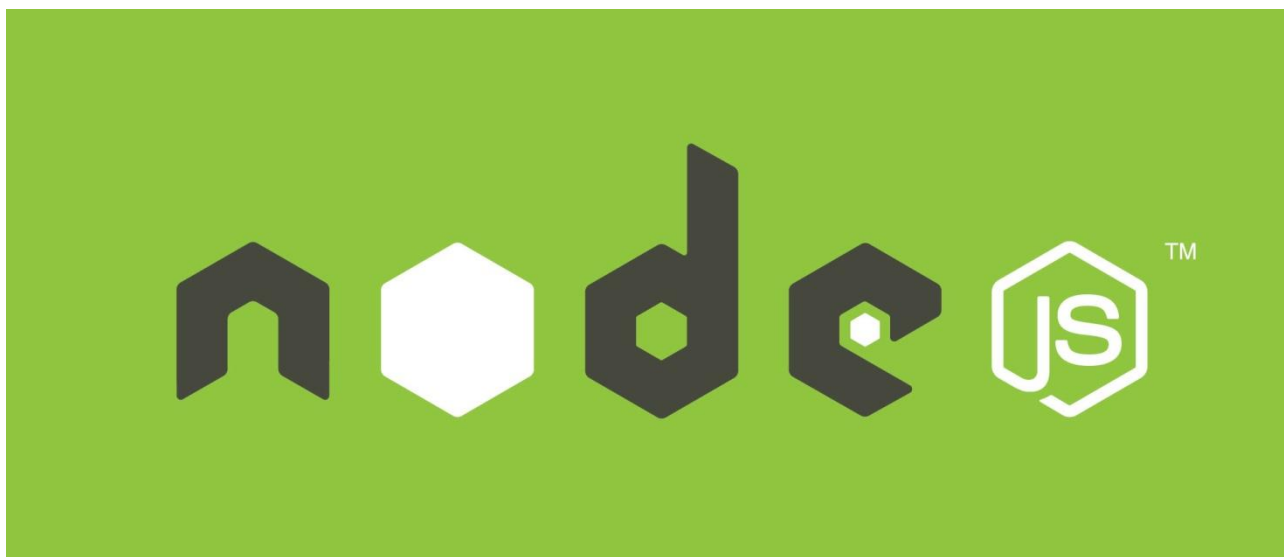
**[www.sceu.frba.utn.edu.ar/e-learning](http://www.sceu.frba.utn.edu.ar/e-learning)**

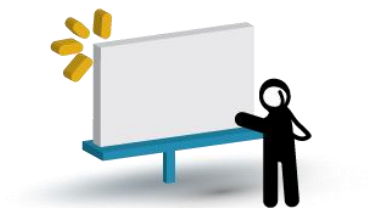


## Unidad 1:

### Introducción a NodeJS

---





## Presentación:

En esta primera Unidad del curso nos introducimos en Node JS, vemos cuándo surge y debido a qué necesidades, aprendemos qué es, quiénes lo usan y por qué.

También hacemos mención de otras tecnologías asociadas y frameworks basados en Node.

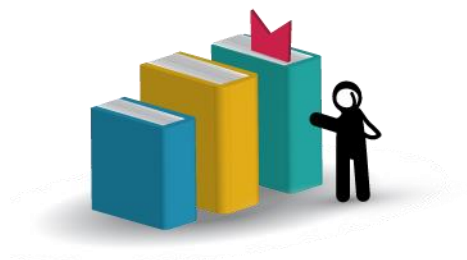
Repasamos algunos conceptos como la programación asíncrona y la programación orientada a eventos y damos los primeros pasos en la instalación y utilización de los primeros comandos repasando sus características fundamentales.



## Objetivo:

Al terminar la Unidad los participantes:

- Sabrán cómo crear módulos exportando nuestras funciones.
- Sabrán cómo incluir módulos con require.
- Conocerán algunos de los métodos importantes de algunos módulos incluidos en Node JS y sabrán utilizarlos.
- Conocerán el uso de eventos.



## Bloques temáticos:

### 1. NodeJS

¿Qué es NodeJS?

¿Quién usa NodeJS?

Más tecnologías y frameworks basados en NodeJS

### 2. Instalación de NodeJS

Instalación de NodeJS en Windows, Linux y Mac

Probando los primeros comandos NodeJS

### 3. Características fundamentales de Node.JS

Un Javascript "sin restricciones"

Programación Asíncrona

Problema del código piramidal

Programación orientada a eventos (POE)

### 4. Módulos en NodeJS

Crear nuestros propios módulos

Incluir módulos con "require"

Modulo para administrar el sistema de archivos

Modulo http

### 5. Eventos en NodeJS

Módulo de eventos, cómo definir un evento en NodeJS



## Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC\*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

- El MEC es el modelo de E-learning colaborativo de nuestro Centro.



### Tomen nota:

---

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.



## 1 - NodeJS

---

NodeJS, surge en 2009 como respuesta a algunas necesidades encontradas a la hora de desarrollar sitios web, específicamente el caso de la concurrencia y la velocidad.

NodeJS es un plataforma súper-rápida, especialmente diseñada para realizar operaciones de entrada / salida (Input / Output o simplemente I/O en inglés) en redes informáticas por medio de distintos protocolos, apegada a la filosofía UNIX. Es además uno de los actores que ha provocado, junto con HTML5, que Javascript gane gran relevancia en los últimos tiempos, pues ha conseguido llevar al lenguaje a nuevas fronteras como es el trabajo del lado del servidor.

Presenta grandes ventajas en la implementación de aplicaciones que deben responder en tiempo real como por ejemplo los juegos multijugador.

Poco sentido tiene utilizar Node.js en sitios web mayormente estáticos. Hay muchas aplicaciones actuales donde el contenido de la página varía constantemente: Twitter, Facebook, chats, juegos multijugador etc. donde Node.js puede hacer más eficiente la aplicación que se ejecuta en el servidor.

### ¿Qué es NodeJS?

NodeJS, es básicamente un framework para implementar operaciones de entrada y salida, como decíamos anteriormente. Está basado en eventos, streams y construido encima del motor de Javascript V8, que es con el que funciona el Javascript de Google Chrome.

La mejor manera de aproximarse a Node es a través de la definición que aparece en su página web:

“Node.js es una plataforma construida encima del entorno de ejecución javascript de Chrome para fácilmente construir rápidas, escalables aplicaciones de red.

Node.js usa un modelo de E/S no bloqueante dirigido por eventos que lo hace ligero y eficiente, perfecto para aplicaciones data-intensive en tiempo real”

Si queremos entender esta plataforma, lo primero que debemos de hacer es desprendernos de varias ideas que los desarrolladores de Javascript hemos ido adquiriendo a lo largo de los años que llevamos usando ese lenguaje. Para empezar, NodeJS se programa del lado del servidor, lo que indica que los procesos para el desarrollo de software en "Node" se realizan de una manera muy diferente que los de Javascript del lado del cliente.



Entre alguno de los conceptos que cambian al estar Node.JS del lado del servidor, está el asunto del "Cross Browser", que indica la necesidad en el lado del cliente de hacer código que se interprete bien en todos los navegadores. Cuando trabajamos con Node solamente necesitamos preocuparnos de que el código que escribas se ejecute correctamente en tu servidor.

Otras de las cosas que deberías tener en cuenta cuando trabajas con NodeJS, son la programación asíncrona y la programación orientada a eventos, con la particularidad que los eventos en esta plataforma son orientados a cosas que suceden del lado del servidor y no del lado del cliente como los que conocemos anteriormente en Javascript "común".

Además, NodeJS implementa los protocolos de comunicaciones en redes más habituales, de los usados en Internet, como puede ser el HTTP, DNS, TLS, SSL, etc.

Otro aspecto sobre el que está basada nodeJS son los "streams", que son flujos de datos que están entrando en un proceso. Un Stream en Node es un objeto que encapsula un flujo binario de datos y provee mecanismos para la escritura y/o lectura en él. Por tanto, pueden ser readable, writable o ambas cosas.



## ¿Quién usa NodeJS?

Existen varios ejemplos de sitios y empresas que ya están usando Node en sitios en producción. Uno de ellos es LinkedIn, la plataforma de contacto entre profesionales a modo de red social. Al pasar a NodeJS, LinkedIn ha reducido sensiblemente el número de servidores que tenían en funcionamiento para dar servicio a sus usuarios, específicamente de 30 servidores a 3.

Lo que sí queda claro es que NodeJS tiene un footprint de memoria menor. Es decir, los procesos de NodeJS ocupan niveles de memoria sensiblemente menores que los de otros lenguajes, por lo que los requisitos de servidor para atender al mismo número de usuarios son menores. Por aproximar algo, podríamos llegar a tener 1.000 usuarios conectados a la vez y el proceso de NodeJS ocuparía solamente 5 MB de memoria. Al final, todo esto se traduce en que empresas grandes pueden tener un ahorro importante en costos de infraestructura.

Otros ejemplos, además de LinkedIn son eBay, Microsoft, empresas dedicadas a hosting como Nodester o Nodejitsu, redes sociales como Geekli.st, y muchos más. Podemos obtener más referencias acerca de casos de uso y empresas que implementan NodeJS en el enlace [nodeknockout.com](http://nodeknockout.com) que es un concurso mundial de aplicaciones en Node.

Además de la gran actividad en Internet, el concurso mundial de aplicaciones Node Knockout, la comunidad de Node tiene una cita anualmente con el ciclo de conferencias NodeConf (<http://www.nodeconf.com>), donde se presentan todos los avances de la plataforma, o con la JSConf (<http://jsconf.com>), un evento para desarrolladores de JavaScript donde la plataforma es protagonista de muchas de las conferencias que se realizan.



## Más tecnologías y frameworks basados en NodeJS

No todo termina con NodeJS, en la actualidad existen diversos proyectos interesantes que basan su funcionamiento en Node y que nos dan una idea de la madurez que está adquiriendo esta plataforma. Es el caso de proyectos como:



**Meteor JS**: Un framework Open Source para crear aplicaciones web rápidamente, basado en programación con "Javascript puro" que se ejecuta sobre el motor de Node.JS.



**Grunt**: Un conjunto de herramientas que te ayudan como desarrollador web Javascript. Minifica archivos, los verifica, los organiza, etc. Todo basado en línea de comandos.



**Yeoman**: Otra herramienta, esta vez basada en Grunt, que todavía ofrece más utilidades que ayudan a simplificar diversas tareas en la creación de proyectos, basados en muchas otras librerías y frameworks habituales como Bootstrap, BackboneJS...



## Express

Infraestructura web rápida,  
minimalista y flexible para  
**Node.js**

**Express**: Un framework Open Source desarrollado por la comunidad de Node, compuesto por un conjunto de módulos, que nos facilita y nos ordena el desarrollo de sitios web.

Estos son algunos ejemplos destacados, entre muchos otros que hay en Internet. Son programas basados en Node que nos facilitan labores de desarrollo de aplicaciones web.



## 2 - Instalación de NodeJS

---

Si no tienes instalado todavía NodeJS el proceso es bastante fácil. Por supuesto, todo comienza por dirigirse a la página de inicio de NodeJS:

[nodejs.org](https://nodejs.org)

Allí encontrarás el botón para instalarlo "Install" que pulsas y simplemente sigues las instrucciones.

Los procesos de instalación son sencillos y ahora los describimos. Aunque son ligeramente distintos de instalar dependiendo del sistema operativo, una vez que lo tienes instalado, el modo de trabajo con NodeJS es independiente de la plataforma y teóricamente no existe una preferencia dada por uno u otro sistema, Windows, Linux, Mac, etc. Sin embargo, dependiendo de tu sistema operativo sí puede haber unos módulos diferentes que otros, esto es, unos pueden funcionar en Linux y no así en otros sistemas, y viceversa.

Los módulos que no hemos visto en detalle todavía, en la práctica se usan constantemente durante el desarrollo en NodeJS. Son como paquetes de software o componentes desarrollados y que puedes incluir en tu programa para tener soporte a distintas necesidades, como, por ejemplo, comunicaciones mediante un protocolo. A la hora de hacer tu programa puedes incluir esos módulos para acceder a funcionalidades adicionales de Node.

## Instalación de NodeJS en Windows, Linux y Mac

Si estás en Windows, al pulsar sobre Install te descargará el instalador para este sistema, un archivo con extensión "msi" que como ya sabes, te mostrará el típico asistente de instalación de software.

Una vez descargado, ejecutas el instalador y ¡ya lo tienes!

A partir de ahora, para ejecutar "Node" tienes que irte a la línea de comandos de Windows e introducir el comando "node".

Nota: Ya debes saberlo, pero a la línea de comandos de Windows se accede desde el menú de inicio, buscas "cmd" y encuentras el cmd.exe, que abre la línea de comandos. En algunos sistemas Windows anteriores a 7 accedes también desde el menú de inicio, utilizas la opción "Ejecutar" del menú de inicio y escribes "cmd".

Entonces entrarás en la línea de comandos del propio NodeJS donde puedes ya escribir tus comandos Node, que luego veremos.

## Instalar NodeJS en Linux

Puedes instalarlo de muchas maneras. Lo más interesante sería bajártelo de los repositorios de tu distribución para que se actualice automáticamente cuando suban nuevas versiones. En este caso en el centro de software de Ubuntu se ofrece una versión menor que la que ofrecen en el sitio oficial de NodeJS.

Para solucionar esta situación podemos usar otros repositorios y para ello te ofrecen en la siguiente referencia instrucciones para instalar NodeJS desde diferentes gestores de paquetes, para las distribuciones más habituales.

[github.com](https://github.com)

Siguiendo esa referencia, en Ubuntu hay que ejecutar los siguientes comandos para instalar la última versión:



```
sudo apt-get install python-software-properties
sudo add-apt-repository ppa:chris-lea/node.js
sudo apt-get update
sudo apt-get install nodejs npm
```

También si lo deseas, desde la página de descargas de Node accederás a los binarios de Linux o al código fuente para compilarlo.

## Instalar NodeJS en Mac

La instalación en Mac es muy sencilla si cuentas con el gestor de paquetes "homebrew". Es tan fácil como lanzar el comando:

```
brew install nodejs
```

Durante la instalación se te pedirá incluir en tu sistema un paquete de utilidades por línea de comandos de xcode, si es que no lo tienes ya instalado en tu OS X. Si se produce un error durante la instalación prueba a hacer un update de homebrew.

```
brew update
```

Además está la opción de compilar el código fuente, pero quizás mejor dejarla para los que les guste mucho trabajar a bajo nivel.

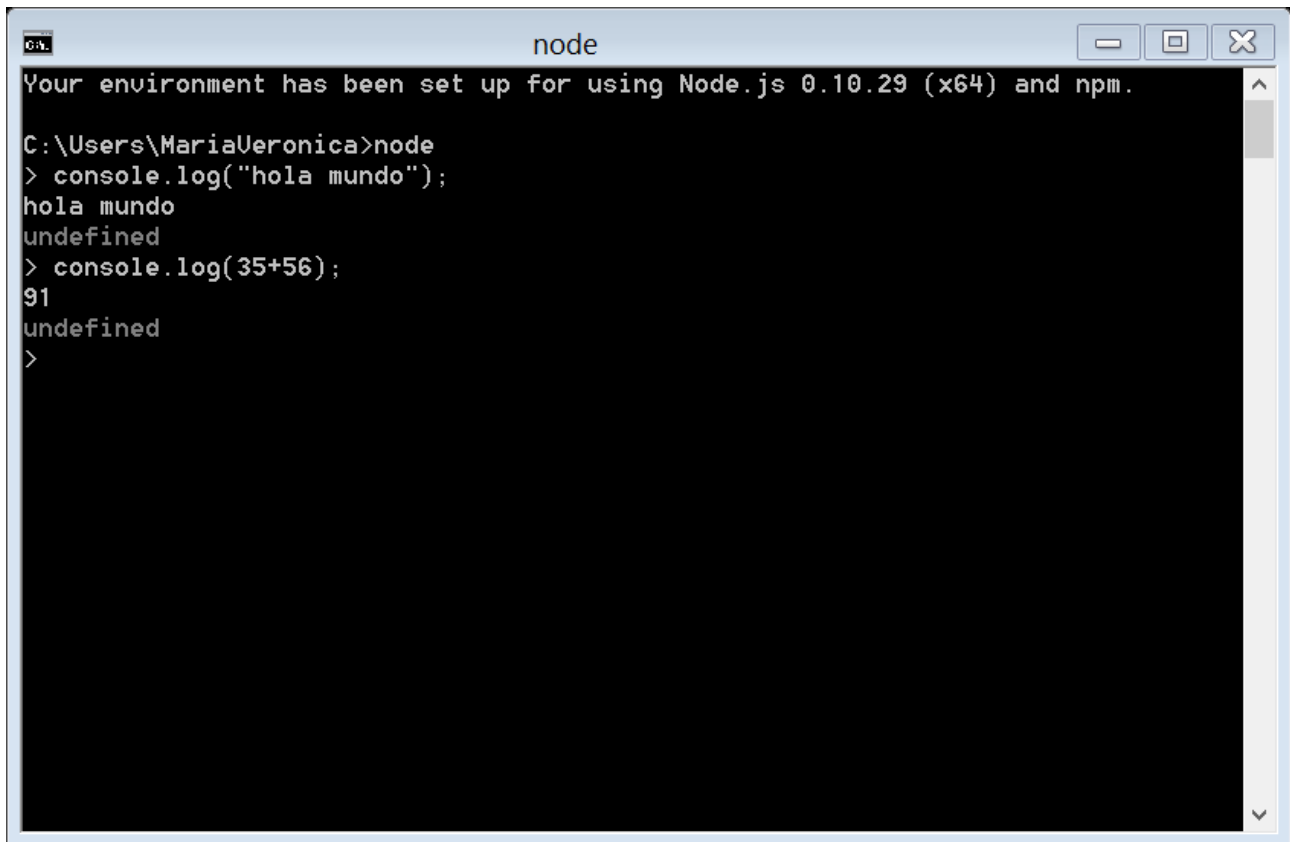
## Probando los primeros comandos NodeJS

En NodeJS la consola de Node puedes escribir instrucciones Javascript. Si lo deseas, puedes mandar mensajes a la consola con `console.log()` por ejemplo:

```
$ node
console.log("hola mundo");
```

Te mostrará el mensaje "hola mundo" en la consola.

Podemos ver en la siguiente imagen un par de mensajes a la consola de Node que podemos utilizar para comenzar nuestro camino.



```
node
Your environment has been set up for using Node.js 0.10.29 (x64) and npm.

C:\Users\MariaVeronica>node
> console.log("hola mundo");
hola mundo
undefined
> console.log(35+56);
91
undefined
>
```

Definimos una variable array llamada lenguajes y luego imprimimos su contenido con `console.log(lenguajes);`





```
node
Your environment has been set up for using Node.js 0.10.29 (x64) and npm.

C:\Users\MariaVeronica>node
> var lenguajes = ['javascript', 'php', 'java']
undefined
> console.log(lenguajes);
[ 'javascript', 'php', 'java' ]
undefined
>
```

Como vemos definimos un vector llamado lenguajes y lo inicializamos con 3 string. Luego mediante el objeto console llamamos al método log para que muestre el contenido de la variable de tipo vector llamada lenguajes.

Nota: observarás que te salen unos mensajes "undefined" en la consola y es porque las instrucciones que estamos ejecutando como "console.log()" no devuelven ningún valor.

Desde la línea de comandos de node.js podemos ejecutar instrucciones de Javascript pero si lo que necesitamos es implementar un problema más complejo debemos codificar con un editor de texto un programa y almacenarlo con extensión js para luego ejecutarlo llamando al programa node.

Para eso creamos un archivo ejercicio1.js con las siguientes instrucciones:

```
console.log('Mi primer programa con Node.js');
console.log('Fin');
```



Lo guardamos en una carpeta y nuevamente desde la consola nos posicionamos en la carpeta en donde hemos guardado el archivo y tipeamos `node ejercicio1.js` y veremos la salida producida por el programa

```
C:\Users\MariaVeronica\Documents\UTN-Desarrollo con NodeJS\Unidad 1\ejemplos>node ejercicio1.js
Mi primer programa con Node.js
Fin
C:\Users\MariaVeronica\Documents\UTN-Desarrollo con NodeJS\Unidad 1\ejemplos>
```

**Nota:** Para salir de la línea de comandos de Node.JS pulsas CTRL + D o dos veces CTRL + c.

En el sitio Node.js Hispano tienen diversos artículos sobre NodeJS, incluidos varios sobre la [instalación de NodeJS en diversas plataformas](#).



## 3 - Características fundamentales de Node.JS

---

Podemos enumerar como características destacadas de NodeJS las siguientes:

- JavaScript sin restricciones,
- Programación asíncrona y
- Programación orientada a eventos.

Todos estos puntos los veremos a continuación.

### Un Javascript "sin restricciones"

Con NodeJS tenemos un "Javascript sin restricciones", ya que todo se ejecuta en el servidor y no tenemos que preocuparnos si nuestro código será compatible o no con distintos clientes. Todo lo que escribas en Node JS y te funcione en tu servidor, estarás seguro que funcionará bien, sea cual sea el sistema que se conecte, porque toda la ejecución de código del servidor se queda aislada en el servidor.

**Nota:** Este detalle de fiabilidad y compatibilidad, o lo que el autor ha llamado como Javascript "sin restricciones" (por estar ejecutado en un ambiente seguro, del lado del servidor) no deja de ser una ventaja de todos los lenguajes del lado del servidor, como PHP, ASP.NET, JSP, etc.

Pero no sólo eso, el Javascript original tiene algunas estructuras de control que realmente no se utilizan en el día a día, pero que realmente existen y están disponibles en NodeJS. En algunas ocasiones resulta especialmente útil alguna de las mejoras de Javascript en temas como la herencia.

Otro ejemplo es que en Javascript haces:

```
for(var key in obj){ }
```

Mientras que en las nuevas versiones de Javascript podrías hacer esto otro:

```
Object.keys(obj).forEach()
```



## Programación Asíncrona

Éste es un concepto que ahora toma especial importancia, dado que NodeJS fue pensado desde el primer momento para potenciar los beneficios de la programación asíncrona.

Imaginemos que un programa tiene un fragmento de código que tarda cinco segundos en resolverse. En la mayoría de los lenguajes de programación precedentes, durante todo ese tiempo el hilo de ejecución se encuentra parado, esperando a que pasen esos cinco segundos, o los que sean, antes de continuar con las siguientes instrucciones. En la programación asíncrona eres capaz de crear diferentes hilos, con diferentes procesos que llevarán un tiempo en ejecutarse, de modo que se hagan todos a la vez. Además, podrás especificar código (*callbacks*) que se ejecute al final de cada uno de esos procesos largos.

La filosofía detrás de Node.JS es hacer programas que no bloqueen la línea de ejecución de código con respecto a entradas y salidas, de modo que los ciclos de procesamiento se queden disponibles cuando se está esperando a que se completen tales acciones. Esto se implementa a partir de "callbacks" que son funciones que se indican con cada operación I/O y que se ejecutan cuando las entradas o salidas se han completado.

**Nota:** Estos callbacks son más o menos como los que podemos conocer de otros sistemas Javascript como jQuery. Funciones que se ponen en ejecución cuando terminan de ejecutarse otras.

Por ejemplo miremos este código:

```
console.log("hola");  
fs.readFile("x.txt", function(error, archivo){  
  console.log("archivo");  
})  
console.log("ya!");
```

Realmente Javascript es síncrono y ejecuta las líneas de código una detrás de otra, pero por la forma de ejecutarse el código hace posible la programación asíncrona. La segunda instrucción (que hace la lectura del archivo) tarda un rato en ejecutarse y en ella indicamos además una función con un `console.log("archivo")`, esa es la función callback que se ejecutará solamente cuando termine la lectura del archivo. A continuación y antes que se llegue a terminar la lectura del archivo ejecuta la instrucción con el último `console.log()`.



Como resultado, primero veremos el mensaje "hola" en la consola, luego el mensaje "ya!" y por último, cuando el fichero terminó su lectura, veremos el mensaje "archivo".

## Problema del código piramidal

El uso intensivo de callbacks en la programación asíncrona produce el poco deseable efecto de código piramidal. Al utilizarse los callbacks, se meten unas funciones dentro de otras y se va entrando en niveles de profundidad que hacen un código menos sencillo de entender visualmente y de mantener.

La solución es hacer un esfuerzo adicional por estructurar nuestro código, que básicamente se trata de modularizar el código de cada una de las funciones, para escribirlas aparte y al indicar la función callback, en vez de escribir el código, escribimos el nombre de la función que se ha definido aparte. Podrías incluso definir las funciones en archivos aparte y requiriéndolas con `require("nombre_archivo")` en el código de tu aplicación.

Al conseguir niveles de indentación menos profundos, estamos ordenando el código, con lo que será más sencillo de entender y también más fácil de encontrar posibles errores. Además, a la larga conseguirás que sea más escalable y puedas extenderlo en el futuro o mantenerlo por cualquier cuestión.

Algunos consejos a la hora de escribir código para que éste sea de mayor calidad:

- Escribe código modularizado (un archivo con más de 500 líneas de código puede que esté mal planteado)
- No abuses, no repitas las mismas cosas, mejor reúsa.
- Usa librerías que ayuden al control (como `async` que te ayuda a ordenar ese montón de callbacks)
- Usa promesas (`promise` en Node) y futuros (`future` en Node)
- Conoce el lenguaje

## Programación orientada a eventos (POE)

Conocemos la programación orientada a eventos porque la hemos utilizado en Javascript para escribir aplicaciones del lado del cliente. Estamos acostumbrados al sistema, que en NodeJS es algo distinto, aunque sigue el mismo concepto.

En Javascript del lado del cliente tenemos objetos como "window" o "document" pero en NodeJS no existen, pues estamos en el lado del servidor.

Eventos que podremos captar en el servidor serán diferentes, como "uncaughtError", que se produce cuando se encuentra un error por el cual un proceso ya no pueda continuar. El evento "data" es cuando vienen datos por un *stream*. El evento "request" sobre un servidor también se puede detectar y ejecutar cosas cuando se produzca ese evento.

Volvemos a insistir, NodeJS sería un equivalente a PHP, JSP, ASP.NET y entonces todo lo que sean eventos de Node, serán cosas que ocurran en el lado del servidor, en diferencia con los eventos de Javascript común que son del lado del cliente y ocurren en el navegador.

**Nota:** La comparación de NodeJS con lenguajes del lado del servidor viene bien para entender cómo pueden ser sus eventos distintos que los de Javascript del lado del cliente. Sin embargo debemos saber que NodeJS es más bien un *framework* basado en Javascript del lado del servidor, por lo que su equivalente más próximo podría ser algo como Django o Ruby on Rails. Aunque habría que matizar que esa comparación puede ser un poco atrevida, debido que NodeJS estaría en un nivel mucho más bajo (más cercano a la máquina) y aunque lo podrías usar sin más añadidos para el desarrollo de sitios web, se combina con otros frameworks como ExpressJS para dar mayores facilidades para la creación de sitios aplicaciones web.



## 4 – Módulos en NodeJS

---

En NodeJS el código se organiza por medio de módulos. Son como los paquetes o librerías de otros lenguajes como Java. Por su parte, NPM es el nombre del gestor de paquetes (package manager) que usamos en Node JS.

Si conoces los gestores de paquetes de Linux podrás hacerte una buena idea de lo que es npm, si no, pues simplemente entiéndelo como una forma de administrar módulos que deseas tener instalados, distribuir los paquetes y agregar dependencias a tus programas.

Nota: Por ejemplo, si estás acostumbrado a Ubuntu, el gestor de paquetes que se utiliza es el "apt-get". Por su parte, Fedora o Red Hat usan Yum. En Mac, por poner otro ejemplo, existe un gestor de paquetes llamado Homebrew.

El gestor de paquetes npm, no obstante, es un poquito distinto a otros gestores de paquetes que podemos conocer, porque los instala localmente en los proyectos. Es decir, al descargarse un módulo, se agrega a un proyecto local, que es el que lo tendrá disponible para incluir. Aunque cabe decir que también existe la posibilidad de instalar los paquetes de manera global en nuestro sistema.

### Crear nuestros propios Módulos

La primera gran diferencia en el JavaScript que se utiliza en Node.js es el concepto de módulo. Sabemos que cuando nuestra aplicación comienza a crecer un único archivo es imposible de manejar todas las funcionalidades, lo mismo ocurre con librerías desarrolladas por otros programadores.

Un módulo contiene funciones, objetos, variables donde indicamos cuales serán exportados para ser utilizados por otros programas.

Vamos a crear un programa muy sencillo que nos permita sumar, restar y dividir números y mostrarlos por la consola. Las funcionalidades las dispondremos en un módulo de archivo y veremos como la consumimos en nuestro programa principal.

Primero creemos nuestro módulo llamado matematica.js con el siguiente código:



```
var PI=3.14;

function sumar(x1,x2)
{
    return x1+x2;
}

function restar(x1,x2)
{
    return x1-x2;
}

function dividir(x1,x2)
{
    if (x2==0)
    {
        mostrarErrorDivision();
    }
    else
    {
        return x1/x2;
    }
}

function mostrarErrorDivision() {
    console.log('No se puede dividir por cero');
}

exports.sumar=sumar;
exports.restar=restar;
exports.dividir=dividir;
exports.PI=PI;
```

En este archivo podemos definir variables, funciones, objetos etc. y los que necesitamos que sean accedidos desde otro archivo los exportamos agregándolos al objeto exports:

```
exports.sumar=sumar;
exports.restar=restar;
exports.dividir=dividir;
exports.PI=PI;
```

Aquello que no necesitemos llamarlo desde otra archivo como en este ejemplo pasa con la función mostrarErrorDivision simplemente no la agregamos al objeto exports.

Codifiquemos ahora nuestra aplicación principal que la llamaremos ejercicio2.js y también la guardamos en la misma carpeta donde tenemos el archivo matematica.js:





```
var mat=require('./matematica');  
  
console.log('La suma de 2+2='+mat.sumar(2,2));  
console.log('La resta de 4-1='+mat.restar(4,1));  
console.log('La división de 6/3='+mat.dividir(6,3));  
console.log('El valor de PI='+mat.PI);
```

El JavaScript que se ejecuta en un navegador web actualmente no puede hacer referencia a otro archivo, en Node.js esto se hace llamando a la función require e indicando el path donde se encuentra en este caso el archivo matematica.js (no es necesario indicar la extensión):

```
var mat=require('./matematica');
```

Luego la variable mat tiene acceso a todas las variables, funciones y objetos exportados.

Llamamos luego a las funciones y accedemos a las variables mediante la variable mat:

```
console.log('La suma de 2+2='+mat.sumar(2,2));  
console.log('La resta de 4-1='+mat.restar(4,1));  
console.log('La división de 6/3='+mat.dividir(6,3));  
console.log('El valor de PI='+mat.PI);
```

En nuestra consola al ejecutar nuestro programa ejercicio2.js tenemos como resultado:



```
Node.js command prompt
Your environment has been set up for using Node.js 0.10.29 (x64) and npm.

C:\Users\MariaVeronica>cd documents\UTN-Desarrollo con NodeJS\Unidad 1\ejemplos\ejercicio2

C:\Users\MariaVeronica\Documents\UTN-Desarrollo con NodeJS\Unidad 1\ejemplos\ejercicio2>node ejercicio2.js
La suma de 2+2=4
La resta de 4-1=3
La división de 6/3=2
El valor de PI=3.14

C:\Users\MariaVeronica\Documents\UTN-Desarrollo con NodeJS\Unidad 1\ejemplos\ejercicio2>
```

Tengamos en cuenta que solo podemos acceder del módulo a aquellos elementos exportados, si por ejemplo tratamos de acceder a la función `mostrarErrorDivision` del módulo `matematica.js`:

```
mat.mostrarErrorDivision();
```

Tendremos como resultado un error en tiempo de ejecución ("undefined is not a function"):

Esta pequeña introducción de módulos es para entender en los próximos temas cada vez que consumamos módulos que ya vienen con Node.js y de otras librerías que descarguemos de internet.

Veremos más adelante que los módulos pueden ser una carpeta que contiene un conjunto de archivos y de subcarpetas (esto es muy útil si la funcionalidad del módulo es muy compleja y por lo tanto no debería estar en un único archivo).



La forma de consumir módulos con la función `require` es idéntica ya sea que el módulo sea un único archivo o una carpeta.

## Incluir módulos con `require`

Hemos visto que para utilizar la funcionalidad de un módulo en Node.js debemos llamar a la función `require` pasando entre comillas el nombre del módulo.

La plataforma Node.js incorpora una serie de módulos de uso muy común en los programas que los incorpora el núcleo de Node.js y que se encuentran codificados en C++ para mayor eficiencia.

Para utilizar los módulos del núcleo también utilizamos la función `require` indicando el nombre del módulo.

Algunos de los módulos del núcleo de Node.js son: `os`, `fs`, `http`, `url`, `net`, `path`, `process`, `dns`, etc.

En nuestro `ejercicio3.js` vamos a emplear el módulo `"os"` que provee información básica del sistema donde se ejecuta la plataforma del Node.js. Para hacer uso de un módulo del núcleo de Node.js solo hacemos referencia a su nombre:

```
var os=require('os');

console.log('Sistema operativo:'+os.platform());
console.log('Versión del sistema operativo:'+os.release());
console.log('Memoria total:'+os.totalmem()+' bytes');
console.log('Memoria libre:'+os.freemem()+' bytes');
```

Recordemos que cuando incorporamos un módulo que codificamos nosotros indicamos el path:  
`var mat=require('./matematica');`

La variable `os` almacena una referencia al módulo `'os'` y mediante esta referencia podemos acceder a las variables, funciones, objetos etc. que el módulo exporta (debemos tener la documentación del módulo para poder utilizar su funcionalidad)

Como dijimos el módulo `'os'` tiene una serie de métodos que nos informan el ambiente donde se está ejecutando Node.js como puede ser el sistema operativo donde está instalado, la versión del sistema operativo, la cantidad de memoria ram disponible, memoria libre etc.

En el sitio [nodejs.org](https://nodejs.org) podemos consultar la documentación de cada uno de los módulos del núcleo: <https://nodejs.org/api/index.html>



El resultado de ejecutar este programa en la consola (recordemos que este script se ejecuta en forma local, más adelante veremos las herramientas que nos provee Node.js para desarrollar programas que se ejecutan en un servidor de internet y los podamos consumir desde un navegador por ejemplo):

```
C:\Users\MariaVeronica\Documents\UTN-Desarrollo con NodeJS\Unidad 1\ejemplos>node ejercicio3.js
Sistema operativo:win32
Versión del sistema operativo:6.2.9200
Memoria total:8463343616 bytes
Memoria libre:5231190016 bytes

C:\Users\MariaVeronica\Documents\UTN-Desarrollo con NodeJS\Unidad 1\ejemplos>
```

En el archivo ejercicio4.js confeccionamos un programa que requiera el módulo 'os' para recuperar el espacio libre de memoria. Mostramos inicialmente el espacio libre mediante el método `freemem()`. Luego de crear un vector y mediante el método `push` almacenamos 1000000 de enteros. Mostramos luego de la creación y carga del vector la cantidad de espacio libre.

Todos los métodos del módulo `os` son los siguientes:

- [os.EOL](#)
- [os.arch\(\)](#)
- [os.cpus\(\)](#)
- [os.endianness\(\)](#)
- [os.freemem\(\)](#)
- [os.homedir\(\)](#)
- [os.hostname\(\)](#)
- [os.loadavg\(\)](#)
- [os.networkInterfaces\(\)](#)
- [os.platform\(\)](#)
- [os.release\(\)](#)
- [os.tmpdir\(\)](#)
- [os.totalmem\(\)](#)
- [os.type\(\)](#)
- [os.uptime\(\)](#)

Y los pueden consultar en <https://nodejs.org/api/os.html>



Confeccione un programa que, usando los métodos del módulo, muestre por consola: el nombre del directorio temporal, el espacio total de memoria y el espacio libre de memoria.

**Comparta el programa con sus compañeros en el foro.**

## Módulo para administrar el sistema de archivos

Ya sabemos cómo crear un módulo mínimo, como consumirlo y también como consumir módulos que vienen por defecto en Node.js.

En el ejemplo anterior vimos el módulo `os` ahora veremos un ejemplo sencillo con el uso del módulo `fs`, que viene implementado en Node.js por defecto y nos permite acceder al sistema de archivos para poder leer sus contenidos y crear otros archivos o carpetas.

Anteriormente vimos un ejemplo a través del cual explicábamos el concepto de asincronismo. El módulo de administración de archivos "`fs`" implementa la programación asíncronica para procesar su creación, lectura, modificación, borrado etc.

El módulo '`fs`' tiene muchas funciones como crear, leer, borrar y renombrar archivos, crear directorios, borrar directorios, retornar información de archivos etc. para consultar estas funciones podemos visitar **el api de Node.js**: <https://nodejs.org/api/fs.html>

En el ejercicio5.js vemos el uso de una de las funciones de este módulo para crear un archivo y vemos también en el ejemplo lo que explicábamos al comienzo de la unidad con respecto al asincronismo.

```
var fs=require('fs');

fs.writeFile('./archivo1.txt','línea 1\nLínea 2',function(error){
    if (error)
        console.log(error);
    else
        console.log('El archivo fue creado');
});

console.log('última línea del programa');
```

La programación asíncronica podemos ver qué sucede al mostrar el mensaje 'última línea del programa' antes de informarnos que el archivo fue creado. Es decir que cuando llamamos a la función `writeFile` el programa no se detiene en esta línea hasta que el archivo se crea sino que continúa con las siguientes instrucciones.

En este programita en particular no tiene grandes ventajas utilizar la programación asíncrona ya que luego de llamar a la función `writeFile` solo procedemos a mostrar un mensaje por la consola, pero en otras circunstancias podríamos estar ejecutando más actividades que no dependieran de la creación de dicho archivo (por ejemplo ordenando un vector en memoria).

Llamamos a la función `writeFile` a través de la variable `fs`. Esta función tiene tres parámetros:



- El primer parámetro es el nombre del archivo de texto a crear. Indicamos el path donde debe crearse (en este caso se crea en la misma carpeta donde se encuentra nuestro programa ejercicio5.js)
- El segundo parámetro es el string a grabar en el archivo de texto (mediante los caracteres \n generamos el salto de línea en el archivo de texto)
- Finalmente el tercer parámetro es una función anónima que será llamada desde el interior de la función writeFile cuando haya terminado de crear y grabar el string en el archivo de texto. La función recibe como parámetro un objeto literal con el error en caso de no poderse crear el archivo, en el caso de haber podido crear el archivo retorna en la variable error el valor null.
- El if se verifica verdadero si en la variable error viene un objeto sino con el valor null almacenado en el parámetro error se ejecuta el bloque del else donde mostramos el mensaje de que el archivo fue creado.

Ahora creamos un archivo ejercicio6.js para leer el contenido del archivo que creamos con el ejemplo anterior.

```
var fs=require('fs');

fs.readFile('./archivo1.txt',function(error,datos){
  if (error) {
    console.log(error);
  }
  else {
    console.log(datos.toString());
  }
});

console.log('última línea del programa');
```

Para mostrar el contenido del Buffer en formato texto llamamos al método toString(). Si no hacemos esto en pantalla mostrará los valores numéricos de los caracteres.

El empleo de funciones anónimas en JavaScript es muy común pero podemos volver a codificar el problema anterior pasando el nombre de una función:



```
var fs=require('fs');

function leer(error,datos){
    if (error) {
        console.log(error);
    }
    else {
        console.log(datos.toString());
    }
}

fs.readFile('./archivo1.txt',leer);

console.log('última línea del programa');
```

No es obligatorio que la implementación de la función esté definida antes de llamar a `readFile`, podría estar implementada al final, incluso podría estar implementada en otro módulo y requerirla. El hecho de definir funciones y llamarlas en lugar de utilizar funciones anónimas Esto ayuda a evitar el código piramidal del que hablábamos al principio de la unidad.

## Módulo http

Un módulo fundamental para implementar aplicaciones en un servidor web es el 'http'.

'http' es un módulo del core de Node.js e implementado en C para una mayor eficiencia en las conexiones web.

Vamos a repasar el funcionamiento de este protocolo tan importante en Internet. HTTP (HyperText Transfer Protocol) o (Protocolo de transferencia de hipertexto) permite la transferencia de datos entre un servidor web y normalmente un navegador.

Cuando accedemos a un sitio web desde un navegador escribimos entre otras cosas:

`http://host[:puerto][/ruta y archivo][?parámetros]`

- `http` (indica el protocolo que utilizamos para conectarnos con un servidor web)
- `host` (es el nombre del dominio por ej. `google.com.ar`)
- `puerto` (es un número que generalmente no lo disponemos ya que por defecto el protocolo `http` utiliza el nro 80, salvo que nuestro servidor escuche peticiones en otro puerto que ya en este caso si debemos indicarlo)
- `[/ruta y archivo]` (indica donde se encuentra el archivo en el servidor)





- ?parámetros (datos que se pueden enviar desde el cliente para una mayor identificación del recurso que solicitamos)

Desde el navegador parte el pedido y el servidor tiene por objetivo responder a esa petición.

Cuando trabajamos con la plataforma Node.js debemos codificar nosotros el servidor web (muy distinto a como se trabaja con PHP, Asp.Net, JSP etc. donde disponemos en forma paralela un servidor web como puede ser el Apache o IIS)

Para la implementación de un servidor web es necesario utilizar el módulo 'http', también existen otros módulos que nos facilitan el desarrollo de sitios web complejos (de todos modos siempre utilizando y extendiendo las funcionalidades del módulo 'http').

Creamos el archivo ejercicio7.js con el siguiente código

```
var http=require('http');

var servidor=http.createServer(function(pedido, respuesta) {
    respuesta.writeHead(200, {'Content-Type': 'text/html'});
    respuesta.write('<!doctype html><html><head></head>'+
        '<body><h1>Sitio en desarrollo</h1></body></html>');
    respuesta.end();
});

servidor.listen(8888);

console.log('Servidor web iniciado');
```

Analizamos un poco el código:

Primero requerimos el módulo 'http' y guardamos una referencia en la variable http:

```
var http=require('http');
```

El módulo 'http' tiene una función llamada `createServer` que tiene por objetivo crear un servidor que implementa el protocolo HTTP.

A la función `createServer` debemos enviarle una función anónima (o podemos implementar una función definida como vimos anteriormente) con dos parámetros que los hemos llamado `pedido` y `respuesta`.

Los objetos `pedido` y `respuesta` los crea la misma función `createServer` y los pasa cuando se dispara el pedido de una página u otro recurso al servidor.



Igual que cuando vimos archivos de texto estamos utilizando programación asincrónica, la función `createServer` se ejecuta en forma asíncrona lo que significa que no se detiene, sino que sigue con la ejecución de la siguiente función:

```
servidor.listen(8888);
```

La función `listen` (escucha) que también es asíncrona se queda esperando a recibir peticiones.

Antes que solicitemos una página desde el navegador podemos ver en la consola el mensaje de: 'Servidor web iniciado'.

El programa como podemos ver desde la consola no ha finalizado sino que esta ejecutándose un ciclo infinito en la función `listen` esperando peticiones de recursos.

Dijimos que cuando hay una solicitud de recursos al servidor se dispara la función anónima llegando dos objetos como parámetro:

```
var servidor=http.createServer(function(pedido, respuesta) {  
    respuesta.writeHead(200, {'Content-Type': 'text/html'});  
    respuesta.write('<!doctype html><html><head></head>'+  
        '<body><h1>Sitio en desarrollo</h1></body></html>');  
    respuesta.end();  
});
```

El primer parámetro contiene entre otros datos el nombre del archivo que solicitamos, información del navegador que hace la petición etc. En este programa al parámetro `pedido` no lo utilizamos.

El parámetro `respuesta` es el que tenemos que llamar a los métodos:

- `writeHead`: es para indicar la cabecera de la petición HTTP (en esta caso indicamos con el código 200 que la petición fue Ok y con el segundo parámetro inicializamos la propiedad `Content-Type` indicando que retornaremos una corriente de datos de tipo HTML)
- `write`: mediante la función `write` indicamos todos los datos propiamente dicho del recurso a devolver al cliente (en este caso indicamos en la cabecera de la petición que se trataba de HTML)
- `end`: finalmente llamando a la función `end` finalizamos la corriente de datos del recurso (podemos llamar varias veces a la función `write` previo a llamar por única vez a `end`)

Otra cosa importante de notar de nuestro servidor web elemental es que no importa que archivo pidamos a nuestro servidor web siempre nos devolverá el código HTML que indicamos en la

función anónima que le pasamos a `createServer` (en este ejemplo solicito el archivo `pagina1.html`, lo mismo sucedería si pido otras páginas: `pagina2.html`, `pagina3.php` etc.).

Si vamos a la consola y ejecutamos `ejercicio7.js` veremos la leyenda `Servidor Web iniciado`, el servidor se queda esperando peticiones en el puerto 8888. Si abrimos el navegador y tipeamos `localhost:8888` veremos la página `Sitio en Desarrollo`.

Si detenemos el servidor que creamos desde la consola (Presionamos las teclas `Ctrl C` que aborta el programa) y solicitamos nuevamente datos al servidor podremos ver que ahora el servidor no responde.

Es importante entonces tener en cuenta que el programa `Node.js` este ejecutándose para poder pedirle recursos. Otra cosa muy importante es que cada vez que hagamos cambios en el código fuente de nuestra aplicación en JavaScript debemos detener y volver a lanzar el programa para que tenga en cuenta las modificaciones.



## 6 - Eventos en NodeJS

---

Los eventos del lado del servidor, no tienen nada que ver con los eventos Javascript que conocemos y utilizamos en las aplicaciones web del lado del cliente. Aquí los eventos se producen en el servidor y pueden ser de diversos tipos dependiendo de las librerías o clases que estemos trabajando.

Por ejemplo pensemos en un servidor HTTP, donde tendríamos el evento de recibir una solicitud. O en un stream de datos tendríamos un evento cuando se recibe un dato como una parte del flujo.

Los eventos se encuentran en un módulo independiente que tenemos que requerir en nuestros programas creados con Node JS con la sentencia "require" que ya hemos visto.

```
var eventos = require('events');
```

Dentro de esta librería o módulo hay una serie de utilidades para trabajar con eventos.

Veamos primero el emisor de eventos, que encuentras en la propiedad EventEmitter.

```
var EmisorEventos = eventos.EventEmitter;
```

En Node existe un bucle de eventos, de modo que cuando declaras un evento, el sistema se queda escuchando en el momento que se produce, para ejecutar entonces una función. Esa función se conoce como "callback" o como "manejador de eventos" y contiene el código que quieras que se ejecute en el momento que se produzca el evento al que la hemos asociado.

Primero tendremos que "instanciar" un objeto de la clase EventEmitter, que hemos guardado en la variable EmisorEventos.

```
var ee = new EmisorEventos();
```

Luego tendremos que usar el método on() para definir las funciones manejadoras de eventos, o su equivalente addEventListener(). Para emitir un evento mediante código Javascript usamos el método emit().

Como ven, se encuentran muchas similitudes a la hora de escribir eventos en otras librerías Javascript como jQuery. El método on() es exactamente igual, al menos su sintaxis. El método emit() sería un equivalente a trigger() de jQuery.



Por ejemplo, voy a emitir un evento llamado "datos", con este código.

```
ee.emit('datos', Date.now());
```

Ahora voy a hacer una función manejadora de eventos que se asocie al evento definido en "datos".

```
ee.on('datos', function (fecha) {  
    console.log(fecha);  
});
```

Si deseamos aprovechar algunas de las características más interesantes de aplicaciones NodeJS quizás nos venga bien usar setInterval() y así podremos estar emitiendo datos cada cierto tiempo:

```
setInterval(function() {  
    ee.emit('datos', Date.now());  
}, 500);
```

El código completo sería el siguiente:

```
var eventos = require('events');  
  
var EmisorEventos = eventos.EventEmitter;  
var ee = new EmisorEventos();  
ee.on('datos', function (fecha) {  
    console.log(fecha);  
});  
setInterval(function() {  
    ee.emit('datos', Date.now());  
}, 500);
```

Esto lo podemos guardar como "ejercicio8.js"

Para ponerlo en ejecución nos vamos en línea de comandos hasta la carpeta donde hayamos colocado el archivo "ejercicio8.js" y escribimos:

```
node ejercicio8.js
```

Como resultado, veremos que empiezan a aparecer líneas en la ventana del terminal del sistema operativo, con un número, que es el "timestamp" de la fecha de cada instante inicial. Para salir del programa pulsamos como siempre CTRL + c.



## **Lo que vimos:**

---

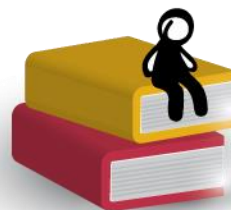
En esta Unidad nos adentramos en las características de NodeJS.



## **Lo que viene:**

---

En la próxima unidad continuaremos viendo en mayor detalle el modulo http y cómo recibir datos de un formulario o de una url por GET, el gestor de paquetes npm y algunos otros paquetes de utilidad.



## Bibliografía

---

- Alex Young y Marc Harter (2014). Node.js in practice. Manning.
- Barsarat Ali Syed (2014). Beggining Node.js. Apress.
- Ethan Brown (2014). Web Development with Node & Express. O'Reilly.
- Gimson, Matthew. (2015). Practical Guide for Begginers. Kindle Edition.
- Sitios web de referencia y consulta:
  - <https://nodejs.org/api/index.html>
  - <https://www.npmjs.com/>
  - <http://www.desarrolloweb.com/manuales/manual-nodejs.html>
  - <http://www.javascriptya.com.ar/nodejsya/>