



Clase 5. Programación Backend

# ***Administradores de Paquetes - NPM***



## ***OBJETIVOS DE LA CLASE***

- Comprender qué es Node.js y su uso en un entorno Backend.
- Comprender la función que cumple npm en Node.js
- Conocer el proceso de instalación de dependencias.

# ***CRONOGRAMA DEL CURSO***

Clase 4



**Manejo de Archivos en  
Javascript**

Clase 5



**Administradores de  
Paquetes - NPM**

Clase 6



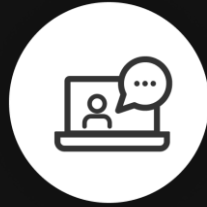
**Servidores Web**

***Repasando...***

- Recordemos que la manipulación de archivo **es solo posible en Node Js** del lado del backend, ya que **en el frontend no es posible** por ser **inseguro**.
- Las **acciones** que podemos hacer sobre un archivo puede ser **lectura, escritura, actualización, borrado** entre otros.
- Estas acciones podemos hacerlas de forma **síncrona** como **asíncrona**. Este último por **callbacks** o **promesas**.

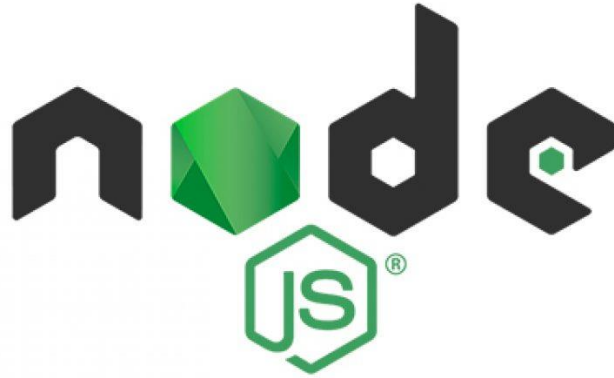
## ***Manejo de Archivos***

Veamos un ejemplo rápido... 



***¿Alguna pregunta hasta  
ahora?***

# *Qué es Node.js*





- Node.js es un entorno de tiempo de ejecución de JavaScript.
- Node.js fue creado por los desarrolladores originales de JavaScript e incluye todo lo que se necesita para **ejecutar un programa escrito en JavaScript por fuera del navegador.**
- Se basa en el motor de tiempo de ejecución JavaScript V8, el mismo que usa Chrome para convertir el Javascript en código máquina.
- Node.js está escrito en C++ y dispone de módulos nativos.

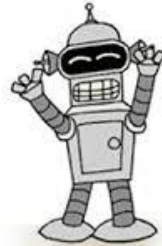


# ***Escribir nuestro primer programa en Node.js***

```
7 var cookieParser = require('cookie-parser');
8 var bodyParser = require('body-parser');
9 var http = require('http');
10 http.
11   Agent
12   var r createClient
13   var u createServer (function) http.createServer(requestListener?: (request:
14     get
15   var a globalAgent
16     request
17   // vi STATUS_CODES
18   app.set('views', path.join(__dirname, 'views')),
19   app.set('view engine', 'jade');
```



Hello world



# ***Requerimientos***

- **Node.js 16.x.x** <https://nodejs.org/es/>
- **Visual Studio code** <https://code.visualstudio.com/>
- **Git** <https://git-scm.com/> (seleccionar la instalación de *git bash* si corresponde, ver próximo punto).
- Una consola (elegir una según corresponda):
  - **PowerShell** (Windows 10, ya viene instalada)
  - **Cmder mini** (Windows < 10) <https://cmder.net/>
  - **Git Bash** (se descarga junto con **Git**)

# ***Node.js: ejecución desde archivo js***

1. Crear una carpeta de proyecto
2. Abrir la carpeta con el vscode
3. Crear un archivo llamado **main.js** dentro de esa carpeta
4. Escribir código en **main.js** y guardarlo.
5. Abrir una terminal (la que hayan elegido), ya sea en forma externa o desde dentro del vscode.
6. Ejecutar el programa desde la terminal corriendo la instrucción: **node main.js**



# ***Proyecto en Node***

Vamos a practicar lo aprendido hasta ahora

# 1- NÚMEROS ALEATORIOS

Desafío  
generico



A- Crear un proyecto en node.js que genere 10000 números aleatorios en el rango de 1 a 20.

B- Crear un objeto cuyas claves sean los números salidos y el valor asociado a cada clave será la cantidad de veces que salió dicho número.  
Representar por consola los resultados.

*Tiempo: 5 minutos*

## 2- ARRAY DE OBJETOS

Desafío  
generico



Desarrollar un proyecto en node.js que declare un array de objetos de este tipo:

```
const productos = [  
  { id:1, nombre:'Escuadra', precio:323.45 },  
  { id:2, nombre:'Calculadora', precio:234.56 },  
  { id:3, nombre:'Globo Terráqueo', precio:45.67 },  
  { id:4, nombre:'Paleta Pintura', precio:456.78 },  
  { id:5, nombre:'Reloj', precio:67.89 },  
  { id:6, nombre:'Agenda', precio:78.90 }  
]
```

## 2- ARRAY DE OBJETOS

Desafío  
generico



Y obtenga la siguiente información de dicho array

- A) Los nombres de los productos en un string separados por comas.
- B) El precio total
- C) El precio promedio
- D) El producto con menor precio
- E) El producto con mayor precio
- F) Con los datos de los puntos 1 al 5 crear un objeto y representarlo por consola

**Aclaración:** todos los valores monetarios serán expresados con 2 decimales

*Tiempo: 10 minutos*

**CODER HOUSE**



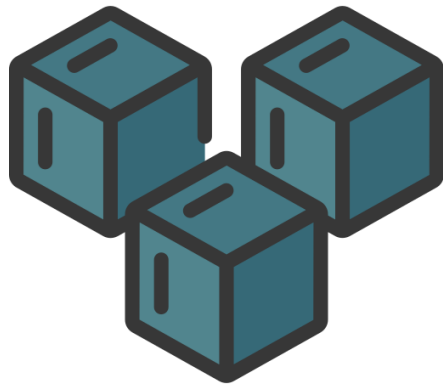
***¿Alguna pregunta hasta  
ahora?***



# ***Módulos en Node.js***

# ***¿Qué son?***

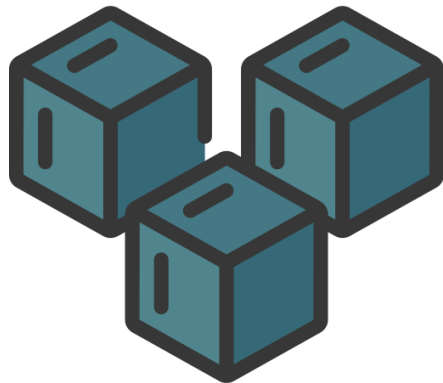
- Un módulo no es nada más que una unidad de código organizado en archivos o directorios, la cual puede ser exportada con facilidad para poder reutilizarse en otras partes de la aplicación.



# ***Tipos de Módulos en Node.js***

# ***Built-in modules***

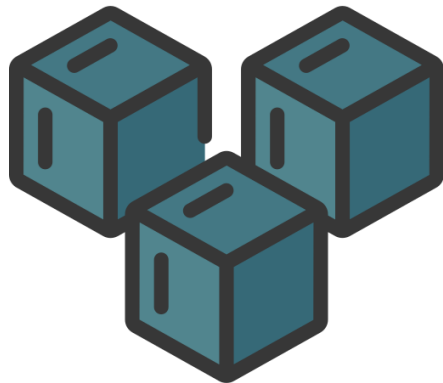
- Son los **módulos nativos** de la **API** de Node.js. No hace falta que se instalen, ya que vienen incluidos **por defecto** con Node.js. Algunos ejemplos son los módulos **fs** o **http**. Estos paquetes solo son actualizados si cambias la versión de Node.js.



***Veamos un ejemplo...*** 

# ***Local modules***

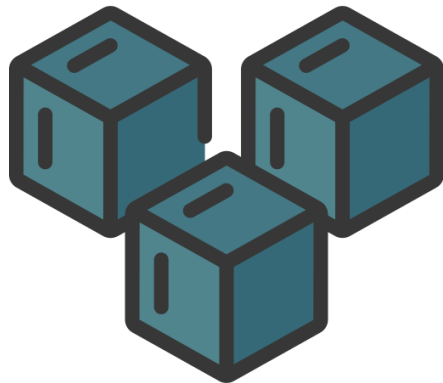
- Son los módulos escritos por los desarrolladores y forman en su conjunto gran parte de la aplicación. Como ya has leído, se estructuran así con la finalidad de poder ser un código reutilizable.



***Veamos un ejemplo...*** 

# ***External modules***

- Son, en esencia, los paquetes de terceros distribuidos a través de npm (aunque pueden provenir de otros repositorios). Estos paquetes se instalan como dependencias y, aunque aportan funcionalidad a la aplicación, no deben incluirse en el repositorio ya que no son parte de la misma.





***Veamos un ejemplo...*** 



***¿Alguna pregunta hasta  
ahora?***



***BREAK***

**¡5/10 MINUTOS Y VOLVEMOS!**

# ***Administradores de Paquetes (Package Managers)***



# Concepto



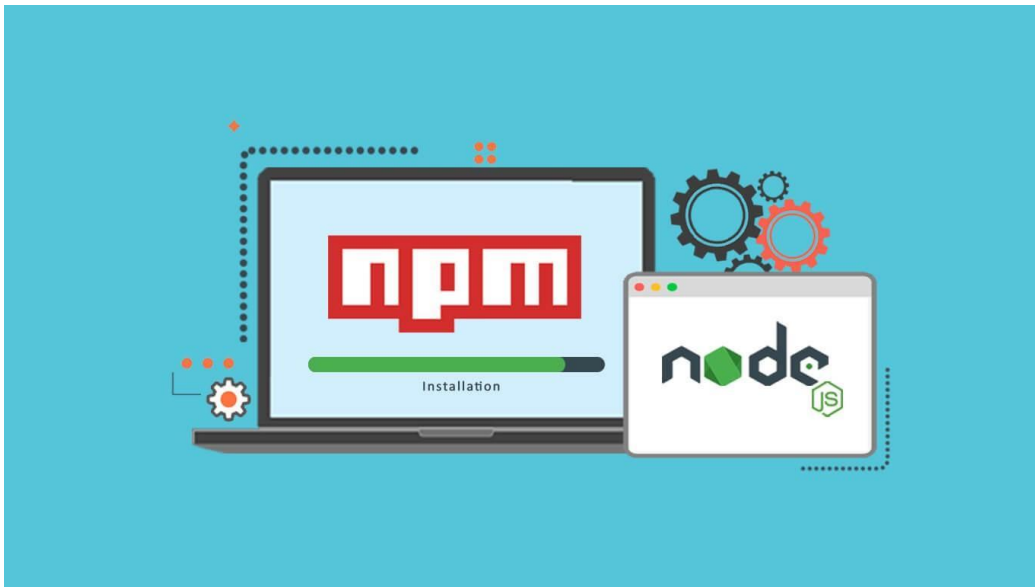
- Los **Package Managers** (o Administradores de paquetes) sirven para no tener que descargar, instalar y mantener las dependencias de un proyecto a mano.
- Estas aplicaciones facilitan la **descarga e instalación de las librerías** que utiliza el proyecto.
- Se requiere que conozcamos el nombre exacto de la **librería** (y versión deseada si es necesario) y contar con conexión a Internet.
- Mediante un comando se descargará de un **repositorio centralizado** la versión correspondiente de la dependencia especificada y se agregará al proyecto.

***NPM***

# ¿Que es NPM?



- NodeJS cuenta con su propio Administrador de Paquetes: **NPM** (*NodeJS Package Manager*).



# ***Instalando dependencias con NPM***



```
//Global:  
npm install -g nombre-de-la-librería  
//Local:  
npm install nombre-de-la-librería
```

- Las dependencias pueden instalarse en **forma global o local**.
- Si instalamos una dependencia en **forma global**, todos nuestros **programas** desarrollados en NodeJS contarán con esa **librería**, y con la versión que haya sido instalada.
- En cambio, si instalamos en **forma local**, podremos **elegir** exactamente qué **librería** y con qué **versión** contará **cada proyecto** que desarrollemos.





***¿Alguna pregunta hasta  
ahora?***

***El archivo "package.json"***

# Package.json

```
{
  "type": "module",
  "name": "mi-proyecto",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.16.4",
    "joi": "~14.3.1",
    "sleep": "*6.0.0"
  },
  "devDependencies": {
    "jest": "latest"
  }
}
```

- Es un archivo de configuración en formato **JSON** que es parte de un proyecto Node.js.
- Podemos crearlo mediante la instrucción: **npm init**
- Podemos especificar en este archivo la lista de dependencias, que son las librerías que usa el proyecto para funcionar o para realizar distintos tipos de testing.

# ***Manejo automatizado de dependencias***

- ❑ Siempre que hayamos especificado nuestras dependencias en el archivo de configuración (package.json) podremos actualizar y mantener de forma fácil y segura las dependencias del proyecto con el comando ***npm install***
- ❑ Además podemos hacer que npm agregue como dependencia al package.json un módulo que estamos instalando. Si lo queremos como **dependencia del proyecto**, al comando *'install'* le agregamos el nombre del módulo: ***npm install <algún-módulo>***
- ❑ Si sólo es una **dependencia del entorno de desarrollo**, le agregamos ***--save-dev*** ó ***-D*** : ***npm install --save-dev <algún-módulo-dev>*** ó ***npm install -D <algún-módulo-dev>***

***Versionado***

# Versionado

*Las librerías de NPM siguen un estándar de versionado de tres números, separados entre sí por un punto:*



- **Major Release:** El primer número corresponde a actualizaciones grandes/significativas que incluyen muchas nuevas características, o que cambian de manera significativa el funcionamiento de las existentes.
- **Minor Release:** El segundo número corresponde a actualizaciones pequeñas que agregan pocas cosas nuevas o actualizan algún detalle del funcionamiento de la librería.
- **Patches:** El tercer número corresponde a arreglos o parches que corrigen defectos en las funcionalidades de la librería.

# ***Manejo Avanzado del Versionado***

Cada una de las versiones de las dependencias está precedida por un símbolo (  $\sim$   $\wedge$   $*$  ) que indica la forma en la que deseamos que se actualice un módulo cada vez que ejecutemos npm install

# ~ (solo patches)

Si escribimos en nuestro package.json: ~0.13.0

- Cuando salga la versión 0.13.1 se actualizará en nuestro proyecto, ya que es un Patch.
- Cuando salga la versión 0.14.0 no se actualizará, ya que es una Minor Release.
- Cuando salga la versión 1.1.0 no se actualizará, ya que es una Major Release.



# ^ (patches y actualizaciones menores)

Si escribimos en nuestro package.json: ^0.13.0

- Cuando salga la versión 0.13.1 se actualizará en nuestro proyecto, ya que es un Patch.
- Cuando salga la versión 0.14.0 se actualizará, ya que es una Minor Release.
- Cuando salga la versión 1.1.0 no se actualizará, ya que es una Major Release

# \* (todas las actualizaciones)

Si escribimos en nuestro package.json: \*0.13.0

- Cuando salga la versión 0.13.1 se actualizará en nuestro proyecto, ya que es un Patch
- Cuando salga la versión 0.14.0 se actualizará, ya que es una Minor Release
- Cuando salga la versión 1.1.0 se actualizará, ya que es una Major Release



***¿Alguna pregunta hasta  
ahora?***



# ***Calculadora de edad***

Vamos a practicar lo aprendido hasta ahora

***Tiempo: 5/10 minutos***

***CODER HOUSE***



Realizar un proyecto en node.js que permita calcular **cuántos años y días totales transcurrieron desde la fecha de tu nacimiento**. Para ello utiliza la dependencia **dayjs** instalándose en forma local desde *npm*. Imprimir los resultados por consola. Hacer las modificaciones necesarias para que sólo se actualicen los **patches** para la librería recién instalada.

### Un ejemplo de salida:

*Hoy es 11/01/2021*

*Nací el 29/11/1968*

*Desde mi nacimiento han pasado 52 años.*

*Desde mi nacimiento han pasado 19036 días.*

### Ayuda:

Utilizar los métodos `diff` y `format` de la librería *dayjs*.

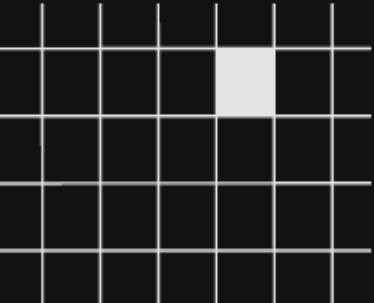
***¿PREGUNTAS?***





# ***¡MUCHAS GRACIAS!***

Resumen de lo visto en clase hoy:

- Node.js
  - NPM
  - Package.json
  - Nodemon
- 



***OPINA Y VALORA ESTA CLASE***



***#DEMOCRATIZANDO LA EDUCACIÓN***