

## Apunte 09 - Node Js

---

En este apunte vamos a revisar los elementos fundamentales de Nodejs, desde la instalación y prueba en sus diferentes alternativas hasta la ejecución de nuestras primeras aplicaciones Javascript con la conciencia de estar ejecutando en un entorno independiente del navegador y donde la interfaz con el sistema operativo anfitrión pasa a ser directa.

Luego de revisar los elementos principales de node, vamos a mencionar los esquemas de instalación y luego vamos introducir la herramienta **npm** como administrador de proyectos y dependencias de paquetes de node. Esta herramienta nos permitirá desde crear proyectos e instalar las librerías que vayamos a utilizar a ejecutar estos proyectos o llevar a cabo su empaquetado.

Bueno comenzamos retomando un poco lo conversado en el apunte 1.

### Javascript del lado del Servidor - Node Js

#### Node Js como entorno de ejecución y como biblioteca de dependencias

En sus orígenes JavaScript vivía exclusivamente en los browsers, como hemos mencionado era el lenguaje por el cual se daba vida interactiva a las páginas web, los navegadores al recibir el HTML y encontrar en ellos javascript eran capaces de interpretar el código Javascript y ejecutarlo.

Sin embargo, Javascript es hoy, un lenguaje de programación en toda esencia: Se puede usar en muchos contextos y alcanzar con éste, todo lo que se puede alcanzar con cualquier otro lenguaje de programación.

Node.js realmente es sólo otro interprete: te permite ejecutar código JavaScript en sistema operativo anfitrión sin la necesidad de un navegador y por lo tanto es muy útil en la programación del backend de las aplicaciones.

Para ejecutar el código JavaScript que pretendemos correr en el backend, este necesita ser interpretado y, bueno, ejecutado, esto es lo que Node.js realiza, haciendo uso de la Máquina Virtual V8 de Google, al menos en su versión inicial, el mismo entorno de ejecución para JavaScript que Google Chrome utilizaba.

Además, Node.js tiene acceso a una enorme biblioteca de librerías y utilidades, de manera que no tienes que escribir todo de cero, como por ejemplo, algo que ponga un string a la consola. Entonces, Node.js es en realidad dos cosas: un entorno de ejecución y una biblioteca de dependencias.

Para hacer uso de éstas (la biblioteca y el entorno), se necesita instalar Node.js, para no escribir aquí una versión que probablemente quede rápidamente desactualizada preferimos redirigir al sitio oficial de node y su área de descargas [node/downloads](#) y allí en general para la asignatura nos manejaremos siempre con la última versión LTS (Soporte de largo alcance y más estable).

En cualquier caso el primer paso será la instalación y dependiendo del sistema operativo y el esquema de instalación a utilizar se puede optar en este sitio [node/download/package-manager](#). Y también en cualquier

esquema de instalación que utilizemos estaremos incluyendo:

- V8 Engine
- Nodejs API
- NPM manager

Solo será necesario elegir la versión, la plataforma y el administrador de instalación y seguir las instrucciones:



Por ejemplo para windows usando Chocolatey se puede realizar la instalación en base a los siguientes comandos:

```
1 # installs Chocolatey (Windows Package Manager)
2 Set-ExecutionPolicy Bypass -Scope Process -Force;
3 [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -or 3072;
4 iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
5
6 # download and install Node.js
7 choco install nodejs --version="20.12.2"
8
9 # verifies the right Node.js version is in the environment
10 node -v # should print `v20.12.2`
11
12 # verifies the right NPM version is in the environment
13 npm -v # should print `10.5.0`
```

PowerShell

Copy to clipboard

También existe la opción de descargar el instalador para la plataforma utilizada desde [node/download/prebuilt-binaries](https://nodejs.org/en/download/prebuilt-binaries).

Aquí les dejo el link al tutorial de MDN (Mozilla Developer Network) de cómo armar un entorno de desarrollo con Nodejs y NPM [Setting up a Node development environment](https://es.mdn.io/setting-up-a-node-development-environment). Está en castellano 😊

## El Stack de Nodejs

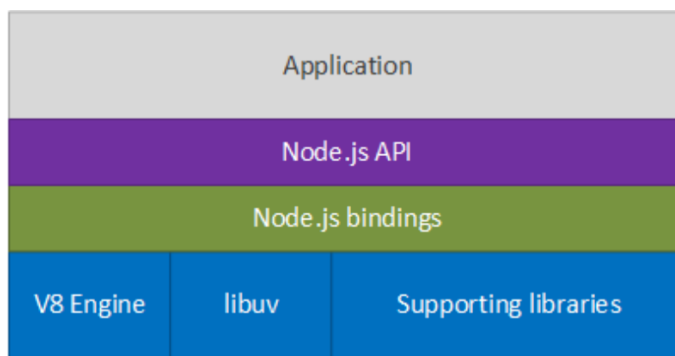
Existen algunas malas interpretaciones comunes acerca de Node.js:

- Node.js **no es** un lenguaje de programación. El lenguaje de programación es Javascript.
- Node.js **no es** un framework para aplicaciones del servidor. Express es un ejemplo de framework para aplicaciones del servidor.

[!IMPORTANT]

*Node.js es en esencia un entorno de ejecución Javascript construido sobre el intérprete V8 de Google Chrome.*

El stack de Node.js incluye varias capas de tecnología que trabajan juntas para permitir la ejecución de código JavaScript en el lado del servidor. Estas son algunas de las capas principales del stack:



1. Sistema Operativo (SO): es el software subyacente en el que se ejecutan todas las aplicaciones. Node.js es compatible con una amplia variedad de sistemas operativos, como Windows, macOS y Linux.
2. V8 Engine: es el motor de JavaScript de código abierto de Google. Node.js utiliza V8 para ejecutar código JavaScript de manera eficiente en el servidor.
3. Node.js Binding: es una capa que proporciona una interfaz para que el código JavaScript interactúe con las API y librerías de C++ y otras lenguajes de programación que son necesarias para acceder a ciertas funcionalidades del sistema operativo o para utilizar alguna biblioteca específica.
4. Node.js API: es la capa principal de Node.js que proporciona un conjunto de módulos y librerías en JavaScript que permiten a los desarrolladores crear aplicaciones web y APIs de manera eficiente. Las características incluyen módulos para manejo de archivos, redes, enrutamiento, manejo de eventos, entre otros.
5. Aplicación de Node.js: es la capa superior del stack de Node.js, que es la aplicación real que se está construyendo. La aplicación de Node.js se construye utilizando la API de Node.js y se ejecuta en el motor V8 de JavaScript.

En resumen, el stack de Node.js es una combinación de tecnologías, que permiten la ejecución de código JavaScript en el lado del servidor y la construcción de aplicaciones web y APIs eficientes. Cada capa del stack se integra con las demás para proporcionar una plataforma completa y robusta para el desarrollo de aplicaciones web.

## Características principales

A continuación se mencionan las características de Node.js que te ayudarán a entenderlo mejor.

- Single Threaded (un solo hilo de ejecución): Solo hay un hilo de usuario para la ejecución del programa Node.js y todas las peticiones de todos los clientes a la vez se responden de forma serializada bajo ese mismo hilo.
- Non blocking I/O (E/S no bloqueante): No espera hasta que la operación de E/S esté completa.
- Asíncrono: Se encarga del código dependiente más tarde una vez que ha terminado.

Dado que Node.js es Single Threaded,

- Los programas de JavaScript no son muy complicados ya que no es necesario bloquear la memoria compartida ni gestionar condiciones de acceso concurrente.
- Las operaciones de E/S se realizan de forma asíncrona, por lo que la ejecución del hilo único no se detiene. Esto se logra a través del Bucle de Eventos.
- Dado que no hay múltiples hilos para admitir la paralelización, las tareas intensivas en CPU no son apropiadas para implementar en Node.js.

## Ventajas de la Arquitectura de Node.js

Las diversas ventajas de la arquitectura de Node.js hacen que la plataforma del lado del servidor tenga mayores capacidades de escalabilidad en comparación con otros lenguajes del lado del servidor.

- Procesa rápida y fácilmente múltiples solicitudes de clientes concurrentes: Utilizando la Cola de Eventos, el servidor Node.js facilita la gestión estructurada de un gran número de solicitudes entrantes.
- No es necesario crear múltiples hilos: Dado que el Bucle de Eventos gestiona todas las solicitudes en orden, no hay necesidad de crear múltiples hilos. Más bien, un solo hilo es suficiente para manejar una solicitud entrante bloqueante.
- Requiere menos memoria y recursos: Debido a la forma en que el servidor Node.js maneja las solicitudes entrantes, por lo general, necesita menos memoria y recursos. Como las solicitudes se manejan una a la vez, este proceso no sobrecarga la memoria.

Gracias a estos beneficios, los servidores creados con Node.js son más rápidos y receptivos en comparación con los servidores desarrollados utilizando otras tecnologías de desarrollo.

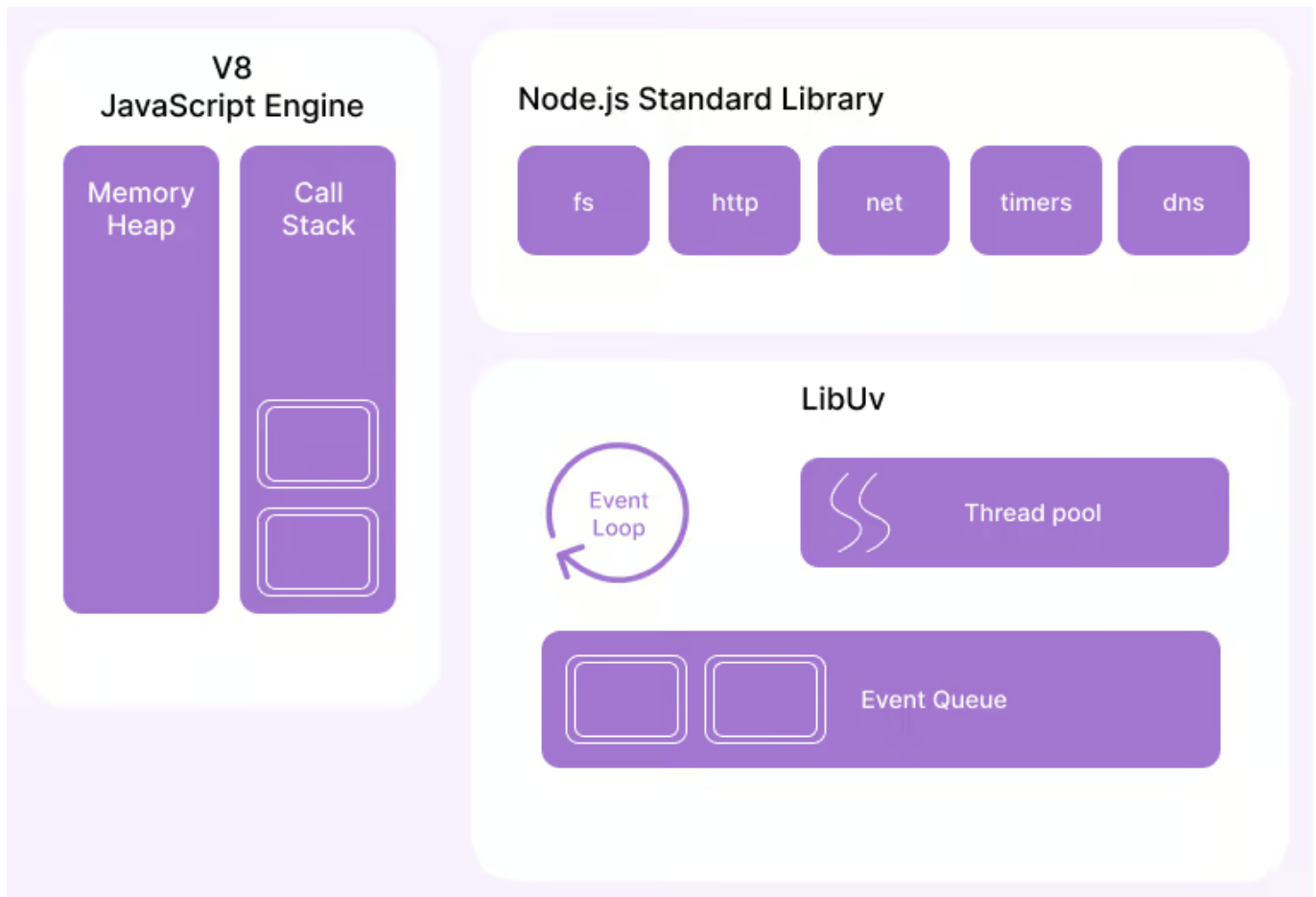
## Entradas y Salidas Asíncronas No Bloqueantes

En el caso del proceso de peticiones que impliquen entrada salida, ya sea accediendo a recursos almacenados o recursos remotos por red, Node.js utiliza un modelo de programación de devolución de llamada asíncrono para manejar operaciones de entrada/salida y liberar un solo hilo.

Cualquier operación externa podría mostrar retrasos vinculados con una función de devolución de llamada que se debe llevar a cabo cuando la operación lenta esté completa. El Grupo de Hilos ejecuta estos eventos en paralelo y el Bucle de Eventos maneja las devoluciones de llamada.

El Bucle de Eventos permite que Node.js realice operaciones de entrada/salida no bloqueantes descargando operaciones al núcleo del sistema siempre que sea posible. Esto es independiente del hecho de que Javascript sea monohilo.

El diagrama mencionado a continuación demuestra cómo interactúan los principales componentes para proporcionar entrada/salida asíncrona:



- La pila de llamadas ejecutaría como se explicó anteriormente, pero si hay bloques con retraso, se bloquearía allí. En cambio, dispararía ese evento usando un callback (función que se ejecuta cuando finalice la acción) asociada y continuará con el código restante.
- La función externa ejecutada se ejecutará en segundo plano (no en el hilo principal único) utilizando el grupo de hilos libuv. Por ejemplo el comando de acceso a un archivo o acceso a una api externa se ejecutará en otro hilo administrado en el caso que fuera necesario.

Cuando se completa, agregará ese evento y devolución de llamada a la Cola de Eventos o Cola de Devolución de Llamada.

- Esta Cola de Eventos contiene todas las funciones de devolución de llamada de las funciones finalizadas que están en espera de ser ejecutadas por el hilo principal. La función de devolución de llamada se ejecuta por el hilo principal y se mueve a la pila de llamadas.
- El Bucle de Eventos entra en juego y transfiere las funciones de devolución de llamada desde la Cola de Eventos a la pila de llamadas para ser ejecutadas por el hilo principal.
- Cuando la pila de llamadas está vacía y la Cola de Eventos tiene funciones pendientes, el Bucle de Eventos mueve el evento y su devolución de llamada desde la Cola de Eventos a la pila de llamadas para que sea ejecutado por el hilo principal.

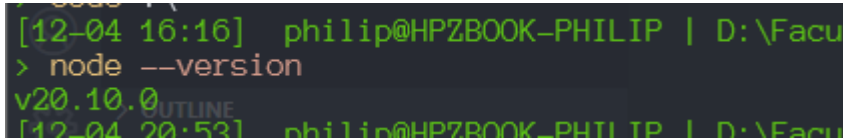
De esta manera, ninguna de las operaciones de E/S lentas bloqueará el hilo principal. En su lugar, se pasan a Node.js para ser ejecutadas en segundo plano.

Por lo tanto, el hilo principal continúa con su programa de manera no bloqueante. Una vez que la operación de E/S está terminada, su devolución de llamada será ejecutada por el hilo principal utilizando el Bucle de Eventos y la Cola de Eventos.

## Bien... Arranquemos con Nodejs

Como ya se explicó en el paso a paso adjunto al apunte 1, la forma más sencilla de testear la instalación de Nodejs en nuestro equipo es simplemente ejecutando node para conocer la versión instalada.

Para esto en una terminal ejecutamos: `node --version` ↵ Enter



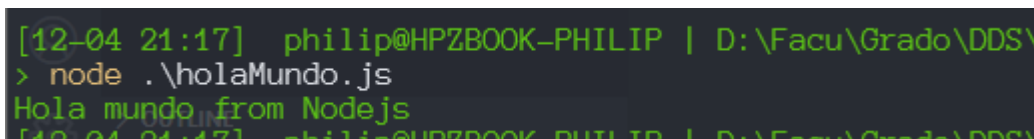
```
[12-04 16:16] philip@HPZBOOK-PHILIP | D:\Facu
> node --version
v20.10.0
[12-04 20:53] philip@HPZBOOK-PHILIP | D:\Facu
```

si devuelve la versión es porque todo está Ok para nuestro primer programa, que en el propio paso a paso se explica como construir escribiendo un archivo con extensión `.js`:

```
// Hola mundo en Javascript para ejecutar con Nodejs
console.log('Hola mundo from Nodejs');

// document.write("Hola Mundo"); // Notar que esta línea provocaría error
//porque no estoy dentro de un html como para
escribir contenido en el documento.
```

Una vez que guardamos el contenido anterior en un archivo con la extensión `js`, por ejemplo: `holaMundo.js`, lo que resta es ejecutar usando node y para eso simplemente tenemos que escribir en la línea de comandos `node holaMundo.js` ↵ Enter.



```
[12-04 21:17] philip@HPZBOOK-PHILIP | D:\Facu\Grado\DDS\
> node .\holaMundo.js
Hola mundo from Nodejs
[12-04 21:17] philip@HPZBOOK-PHILIP | D:\Facu\Grado\DDS\
```

Y veremos que nuestro programa ejecutó mostrando el mensaje en la consola de salida que en el caso de node vemos directamente como resultado en la línea de comandos.

### [!NOTE]

Hemos llegado hasta aquí pudiendo comprobar el funcionamiento de nuestro entorno Nodejs ejecutando además nuestro primer programa. Sin embargo, necesitamos darle más contexto a nuestras aplicaciones, necesitamos poder agregar configuraciones, iniciar las aplicaciones de una forma estándar, agregar a nuestras aplicaciones las librerías necesarias, y podríamos seguir...Para todo esto existe NPM y ahora vamos a continuar por allí.

## ¿Qué es NPM?

Uno de los principales factores del éxito de Node es npm - su popular gestor de paquetes, que permite a los desarrolladores de JavaScript compartir paquetes de forma rápida y fácil.

NPM – o "Node Package Manager" – es el administrador de paquetes predeterminado para el tiempo de ejecución de Javascript Nodejs.

NPM consiste de dos partes principales:

- Herramienta CLI (interfaz de línea de comandos) para la publicación y descarga de paquetes.
- Repositorio en línea que alberga paquetes de JavaScript.

Esencialmente vamos a usar NPM en 3 tareas fundamentales:

- Inicializar el proyecto en un directorio o crear un proyecto en base a una plantilla.
- Gestionar las dependencias como instalar o eliminar dependencias del proyecto.
- Ejecutar el proyecto en base a su configuración.

## Inicializar el proyecto - `npm init`

Cada proyecto en Javascript – ya sea Nodejs o una aplicación de navegador – puede ser enfocado como un paquete npm con su propia información de paquete y su archivo `package.json` para describir y configurar el proyecto. En esencia un proyecto Javascript es un directorio con un archivo `package.json` dentro que lo define como tal.

Evidentemente podemos crear un archivo de texto llamado `package.json` dentro del directorio donde queremos generar nuestro proyecto y ya tendremos un proyecto Javascript, sin embargo `npm` está facultado para llevar a cabo esta tarea por nosotros creando el archivo `package.json` y agregando en él un conjunto de elementos básicos. Esta es la función de `npm init`.

Para ello podemos crear el directorio del proyecto, movernos dentro del directorio creado y ejecutar `npm init` ↵ Enter para iniciar el proceso de creación del proyecto que nos guiará a través de los principales elementos de configuración de la identificación del proyecto que quedarán registrados en el archivo `package.json`

```
[12-04 21:47] philip@HPZBOOK-PHILIP | D:\Facu\Grado\DDS\re
> mkdir primerProyectoNodejs
Directory: D:\Facu\Grado\DDS\repo\catedra\apuntes\sema
Mode                LastWriteTime         Length Name
----                -
d-----          12/4/2024    21:47          primerProyectoNodejs
                Apunte09.md          9+, U
[12-04 21:47] philip@HPZBOOK-PHILIP | D:\Facu\Grado\DDS\re
> cd .\primerProyectoNodejs\
[12-04 21:47] philip@HPZBOOK-PHILIP | D:\Facu\Grado\DDS\re
primerProyectoNodejs:
> npm init
This utility will walk you through creating a package.json
It only covers the most common items, and tries to guess se
See `npm help init` for definitive documentation on these
and exactly what they do.
Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.
Press ^C at any time to quit.
package name: (primerproyectornodejs) prueba
version: (1.0.0)
description: Primer ejemplo de proyecto Javascript
entry point: (index.js)
test command:
git repository:
keywords:
author: philip
license: (ISC)
About to write to D:\Facu\Grado\DDS\repo\catedra\apuntes\se
.json:
{
  "name": "prueba",
  "version": "1.0.0",
  "description": "Primer ejemplo de proyecto Javascript",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "philip",
  "license": "ISC"
}
Is this OK? (yes) _
```

## El archivo `package.json`

Npm solicitará varios datos básicos para configurar el archivo `package.json` los primeros de estos corresponden a la identificación del proyecto, y estos son:



- **name:** es el nombre del proyecto y por lo tanto el nombre del paquete npm que el proyecto pretende construir, este nombre debe seguir una serie de reglas de nomenclatura para ser válido: **Reglas de Nomenclatura:** a continuación se muestra una lista de reglas que el nombre válido de un paquete npm debe cumplir.
  - La longitud del nombre del paquete debe ser mayor que cero.
  - Todos los caracteres en el nombre del paquete deben estar en minúsculas, es decir, no se permiten nombres en mayúsculas o mixtos.
  - El nombre del paquete puede contener guiones.
  - El nombre del paquete no debe contener ningún carácter que no sea seguro para URL (ya que el nombre termina siendo parte de una URL).
  - El nombre del paquete no debe comenzar con . o \_.
  - El nombre del paquete no debe contener espacios.
  - El nombre del paquete no debe contener ninguno de los siguientes caracteres: ~)(!\*
  - El nombre del paquete no puede ser igual al de un módulo principal de node.js/io.js ni a un nombre reservado o prohibido. Por ejemplo, los siguientes nombres son inválidos:
    - http
    - stream
    - node\_modules
    - favicon.ico
  - La longitud del nombre del paquete no puede exceder los 214 caracteres.
- **version:** la versión de tu proyecto. El número de versión debería seguir algún acuerdo pensado con el equipo para que sea representativo. Un acuerdo general para manejar el versionado es utilizar el Versionado semántico.

El comando `npm init -y` se utiliza para inicializar un nuevo proyecto de Node.js. Este comando crea un nuevo archivo `package.json` en el directorio actual con los valores predeterminados.

El archivo `package.json` es un documento que contiene metadatos relevantes para el proyecto. Incluye cosas como el nombre del proyecto, la versión, la descripción, los scripts que se pueden ejecutar, las dependencias del proyecto, y más.

El parámetro `-y` le dice a npm que use los valores predeterminados para todas las preguntas que normalmente haría durante el proceso de inicialización. Esto puede acelerar el proceso si no necesitas personalizar estos valores.

**Acerca de la versión semántica:** para mantener el ecosistema de JavaScript saludable, confiable y seguro, cada vez que realices actualizaciones significativas en un paquete npm que posees, recomendamos publicar una nueva versión del paquete con un número de versión actualizado en el `package.json` que siga la [especificación de versión semántica](#). Seguir la especificación de versión semántica ayuda a otros desarrolladores que dependen de tu código a comprender el alcance de los cambios en una versión determinada y ajustar su propio código si es necesario.

**Nota:** Si introduces un cambio que rompe una dependencia de paquete, recomendamos encarecidamente incrementar el número de versión **mayor**; consulta a continuación para más detalles.

Para ayudar a los desarrolladores que confían en tu código, recomendamos comenzar la versión de tu paquete en `1.0.0` e incrementarla de la siguiente manera:

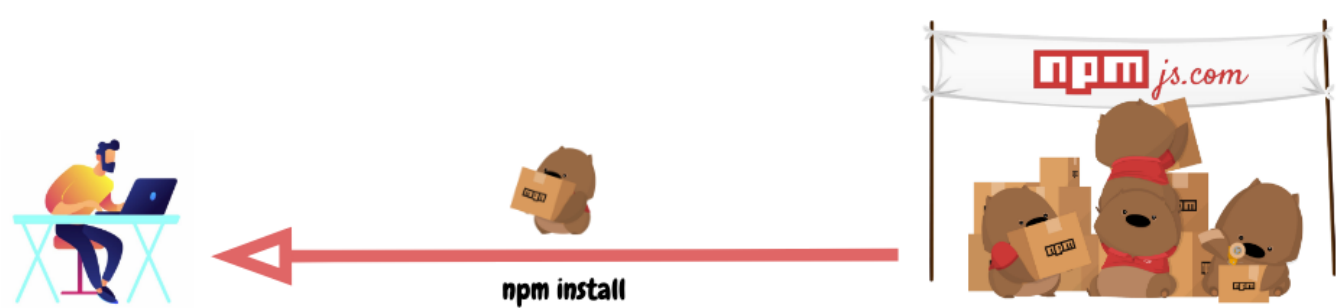
Estado del código	Estadio	Regla	Versión de ejemplo
Primera versión	Nuevo producto	Comienza con 1.0.0	1.0.0
Correcciones de errores compatibles	Lanzamiento de parche	Incrementa el tercer dígito	1.0.1
Nuevas características compatibles	Lanzamiento menor	Incrementa el segundo dígito y restablece el último dígito a cero	1.1.0
Cambios que rompen la compatibilidad	Lanzamiento mayor	Incrementa el primer dígito y restablece los dígitos medio y último a cero	2.0.0

- **description**: Es un valor de documentación donde podemos escribir una descripción breve del proyecto.
- **licence**: Indica la licencia del proyecto.

Finalmente npm nos va a solicitar algunos elementos extra como por ejemplo, quién es el autor, palabras clave o como vamos a ejecutar nuestro proyecto para testearlo o si queremos crear un repositorio git, por ahora podemos simplemente dar `Enter` en cada pregunta para que tome el valor por defecto, en los apartados siguientes explicaremos estos otros elementos.

### Gestionar dependencias - `npm install`

En general nuestros proyectos van a necesitar incluir código ya escrito previamente, la idea es no escribir todo de nuevo sino centrarnos en las funcionalidades necesarias en nuestro proyecto y reutilizar todo lo posible de la librería de Nodejs. El administrador de esta librería es NPM y para ello el comando `npm install {nombre de paquete}` será capaz de instalar dicha dependencia en nuestro proyecto.



El proceso de instalación lleva a cabo al menos dos tareas:

- Descarga la dependencia y la almacena en la carpeta `node_modules` dentro del proyecto.
- Actualiza el archivo `package.json` para registrar en él la propiedad `dependencies` la librería instalada.

Algo importante a tener en cuenta al administrar el registro de dependencias en la propiedad `dependencies` del archivo `package.json`, son los posibles signos que vienen antes de las versiones semánticas (mencionadas anteriormente) del modelo `mayor.minor.patch` de semver, para ello diremos:

- `^`: símbolo que indica que vamos a requerir la última versión menor disponible de dicha librería. Por ejemplo `^1.0.4` podría instalar la versión `1.3.0` si es la última versión de la serie mayor `1`.
- `~`: última versión del parche. De la misma manera que `^` para las versiones menores, la especificación `~1.0.4` podría instalar la versión `1.0.7` si es la última versión menor de la serie menor `1.0`.

De no haber ningún símbolo, npm solo buscará la versión exacta indicada en la configuración de `dependencies`

Esta última configuración es fundamental ya que permite que al compartir el proyecto no necesitemos compartir la carpeta `node_modules` ni subirla al repositorio, puesto que contando con el archivo `package.json` y ejecutando `npm install` (sin nombre de paquete), en el directorio del proyecto, npm instalará todas las dependencias registradas en el archivo `package.json` en nuestro entorno de trabajo, lo que hará mucho más liviana la carga de código a compartir.

#### [!TIP]

Esta es la forma en la que general mente vamos a compartir los ejemplos en las clases de Desarrollo de Software para que sea más rápida la descarga de código.

#### [!TIP]

Para evitar que `node_modules` sea rastreado por Git, debes agregar `node_modules/` a tu archivo `.gitignore`. Aquí está el contenido que debes agregar:

Esto le dice a Git que ignore el directorio `node_modules`, por lo que no se incluirá en tus commits. Esto es útil porque `node_modules` a menudo contiene muchas dependencias que pueden ser fácilmente reinstaladas con `npm install`, por lo que no es necesario incluirlas en tu repositorio.

### `dependencies` vs `devDependencies`

El archivo `package.json` puede registrar dependencias en la propiedad `dependencies` que es donde lo hará por defecto sin hacer nada especial. Sin embargo, en algunos casos podemos requerir dependencias que solo serán necesarias para el proceso de desarrollo pero que no serán necesarias al enviar empaquetar el proyecto para un despliegue en producción.

En este caso podemos agregar la opción `--save-dev` para que la dependencia se configure en la propiedad `devDependencies` y que en un empaquetado para producción no se incluya con el proyecto.

Para hacer este empaquetado deberíamos clonar el repositorio que **no** incluye el directorio `node_modules` y ejecutar `npm install --production` y esta acción no descargará ni prestará atención a las dependencias de desarrollo.

### Ejecutar la aplicación - `npm run`

Como ya vimos la forma de ejecutar una aplicación con Nodejs es simplemente invocar a `node` y luego indicar cuál es el script que inicia la aplicación, por ej. `node index.js` y esto da inicio a la aplicación.

Sin embargo, cuando trabajamos con un proyecto muchas veces queremos configurar diferentes maneras de ejecutar la aplicación, por ejemplo para desarrollo, para ejecutar tests o para una ejecución normal, o incluso podemos querer realizar ejecuciones de otros procesos para nuestro proyecto como podrían ser realizar

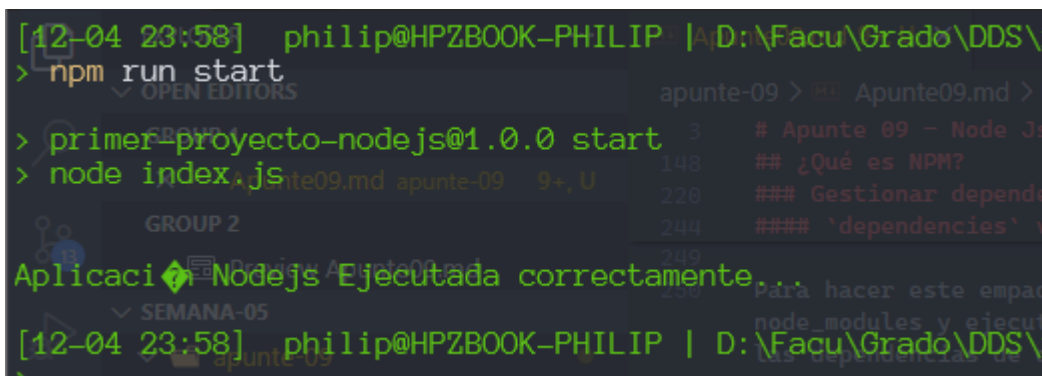
validaciones, reorganizar el código o ejecutar procesos previos al empaquetado. Si estos fueran los casos necesitaríamos conocer cómo realizamos la ejecución para cada caso.

En lugar de eso, lo esperado es que configuremos cada una de estas posibilidades en el archivo `package.json` y para ello podemos utilizar la propiedad `scripts` que nos permite realizar las configuraciones y luego utilizar `npm run {nombre de script}` para lanzar la ejecución sin tener que recordar exactamente cada una de ellas.

En siguiente archivo `package.json` del proyecto que creamos como ejemplo en esta documentación de NPM vamos a agregar a la propiedad `scripts` que agrega npm con una versión por defecto de `"test"` la alternativa `"start"` con el inicio de nuestra aplicación `node index.js`.

```
{
  "name": "primer-proyecto-nodejs",
  "version": "1.0.0",
  "description": "Primer ejemplo de proyecto Javascript",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "philip",
  "license": "ISC"
}
```

Con el archivo `package.json` del proyecto configurado de esta manera podemos ejecutar el proyecto con el comando `npm run start`

A terminal window screenshot showing the execution of a Node.js project. The prompt is [12-04 23:58] philip@HPZBOOK-PHILIP | A:\D:\Facu\Grado\DDS\... The command 'npm run start' is entered, followed by 'primer-proyecto-nodejs@1.0.0 start' and 'node index.js'. The output shows 'Aplicación Nodejs Ejecutada correctamente...' in green. The background shows a file explorer view of the project directory.

Además de `start` `test` existen varias otras configuraciones utilizadas por la comunidad de desarrolladores e incluso podemos agregar las que necesitemos en nuestro equipo si no están incluidas aquí, algunos ejemplos:

- `build`: utilizada generalmente para la ejecución de la construcción del proyecto
- `format`: utilizada para reformatear el código de acuerdo con lo convenido, por ejemplo `"format": "prettier --write **/*.ts"`
- `lint`: utilizada para realizar validaciones de cumplimiento de convenciones de código
- `pack`: utilizada generalmente para la ejecución del empaquetado del proyecto

- `all`: ejecutar todas las anteriores en orden, por ej:

```
"all": "npm run build && npm run format && npm run lint && npm run pack"
```

## Resumen NPM

Bien, nos vamos amigando con la herramienta NPM cuando podemos observar todo lo que es capaz de hacer por nosotros, la verdad que es interesante incluso pensar que aquí solo hemos enfocado nuestro estudio en los elementos necesarios para desarrollar y que aún podríamos seguir investigando elementos como `npm ci` o `npm audit` que brindan alternativas específicas a cada necesidad pero entendemos que con lo expuesto aquí más lo que iremos sumando en los siguientes apuntes podemos atacar con holgura las necesidades la asignatura Desarrollo de Software.

## A trabajar!

Para finalizar el presente apunte queremos mostrar un ejemplo básico donde podamos comprobar los distintos elementos que hemos revisado a lo largo del material. Para ello vamos a generar un proyecto con el objetivo que debería buscar el alumno para resolver el Desafío de programación 1 publicado en el aula virtual de Desarrollo de Software.

En este Desafío se propone construir un programa que genere números aleatorios utilizando una librería específica y luego indique los resultados a los requisitos solicitados. Para ello vamos a ir paso a paso en la construcción, comencemos:

1. El primer paso será construir el proyecto, para ello creamos el directorio que contendrá los archivos de nuestro proyecto, nos movemos dentro del directorio creado y ejecutamos `npm init`

```
[13-04 00:20] philip@HPZBOOK-PHILIP | D:\Facu\Grado\DDS\repo\catedra\apuntes\semana-05\apunte-09:
> mkdir desafio-1
Directory: D:\Facu\Grado\DDS\repo\catedra\apuntes\semana-05\apunte-09
Mode                LastWriteTime         Length Name
----                -
d-----          13/4/2024    00:21           desafio-1
JS index.js
package.json
Apunte09.md
package.json

[13-04 00:21] philip@HPZBOOK-PHILIP | D:\Facu\Grado\DDS\repo\catedra\apuntes\semana-05\apunte-09:
> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.
See `npm help init` for definitive documentation on these fields
and exactly what they do.
Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.
Press ^C at any time to quit.
package name: (apunte-09) desafio-1
version: (1.0.0)
description: Proyecto para dar solución al ejercicio del Desafío 1 de DDS
entry point: (index.js)
test command:
git repository:
keywords:
author: philip
license: (ISC)
About to write to D:\Facu\Grado\DDS\repo\catedra\apuntes\semana-05\apunte-09\package.json:
{
  "name": "desafio-1",
  "version": "1.0.0",
  "description": "Proyecto para dar solución al ejercicio del Desafío 1 de DDS",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "philip",
  "license": "ISC"
}
Is this OK? (yes)
[13-04 00:22] philip@HPZBOOK-PHILIP | D:\Facu\Grado\DDS\repo\catedra\apuntes\semana-05\apunte-09:
>
```

## [!TIP]

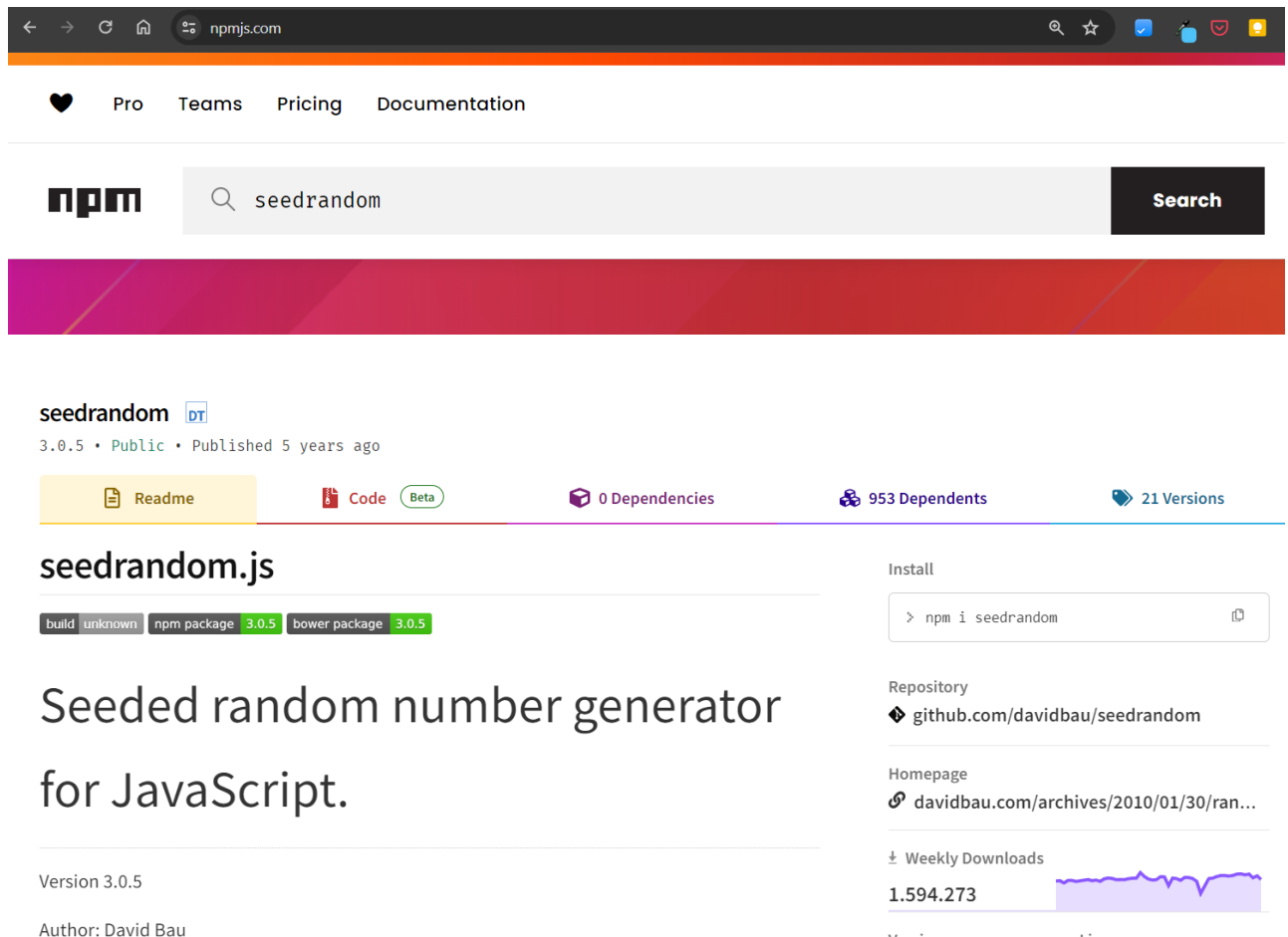
Notar que en las solicitudes no contestadas se saltó con **Enter** lo cual brinda a **npm init** el valor por defecto para la pregunta.

## 2. Ahora vamos a realizar la instalación de las librería requeridas

### 2.1. En primer lugar vamos a instalar la librería que requiere el desafío, esta librería es **seedrandom**.

## [!NOTE]

En los casos donde tenemos que trabajar con una librería que no conocemos, ¿dónde podemos encontrar información?, no hay duda que la respuesta natural: *usar google* va a funcionar, pero en caso de librerías de npm la primero alternativa debería ser la propia **biblioteca de npm** - [npmjs.org](https://www.npmjs.org)



The screenshot shows the npm website interface. At the top, there's a navigation bar with links for Pro, Teams, Pricing, and Documentation. Below this is a search bar with the npm logo and the text 'seedrandom' entered. A 'Search' button is to the right. The main content area features the 'seedrandom' package details. It includes the package name 'seedrandom' with a 'DT' icon, version '3.0.5', and 'Published 5 years ago'. There are buttons for 'Readme', 'Code', and 'Beta'. Statistics show '0 Dependencies', '953 Dependents', and '21 Versions'. The package name 'seedrandom.js' is prominently displayed. Below it, there are tags for 'build', 'unknown', 'npm package 3.0.5', and 'bower package 3.0.5'. The description reads 'Seeded random number generator for JavaScript.' The author is listed as 'David Bau'. On the right side, there's an 'Install' section with the command 'npm i seedrandom'. Below that, the 'Repository' is listed as 'github.com/davidbau/seedrandom' and the 'Homepage' is 'davidbau.com/archives/2010/01/30/ran...'. A 'Weekly Downloads' graph shows a peak of '1.594.273'.

Si observamos sobre la esquina derecha vamos a encontrar el comando de instalación `npm i seedrandom`. Podemos observar que se puede abreviar la palabra `install` con la letra `i`.

Bien allá vamos:

```
[13-04 00:37] philip@HPZBOOK-PHILIP | D:\Facu\Grado\DDS\repo\catedra\apuntes\semana-05\apunte-09\desafio-1:
> npm i seedrandom

added 1 package, and audited 2 packages in 2s

found 0 vulnerabilities
[13-04 00:37] philip@HPZBOOK-PHILIP | D:\Facu\Grado\DDS\repo\catedra\apuntes\semana-05\apunte-09\desafio-1:
>
```

Y veamos cómo quedó la estructura del directorio del proyecto y el archivo `package.json` luego de la instalación:

```

[13-04 00:40] philip@HPZBOOK-PHILIP | D:\Facu\Grado\DDS\repo\catedra\apuntes\semana-05\apunte-09\desafio-1:
> tree
Folder PATH listing for volume Data
Volume serial number is 7CAF-8DFA
D:\Facu\Grado\DDS\repo\catedra\apuntes\semana-05\apunte-09\desafio-1
├── node_modules
│   ├── seedrandom
│   │   ├── nyc_output
│   │   │   ├── processinfo
│   │   │   ├── coverage
│   │   │   │   ├── lcov-report
│   │   │   │   │   ├── seedrandom
│   │   │   │   │   │   └── lib
│   │   └── lib
│   └── lib
└── lib
[13-04 00:40] philip@HPZBOOK-PHILIP | D:\Facu\Grado\DDS\repo\catedra\apuntes\semana-05\apunte-09\desafio-1:
> cat .\package.json
{
  "name": "desafio-1",
  "version": "1.0.0",
  "description": "Proyecto para dar soluci n al ejercicio del Desaf o 1 de DDS",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "philip",
  "license": "ISC",
  "dependencies": {
    "seedrandom": "^3.0.5"
  }
}
[13-04 00:40] philip@HPZBOOK-PHILIP | D:\Facu\Grado\DDS\repo\catedra\apuntes\semana-05\apunte-09\desafio-1:

```

2.2 Ahora vamos a instalar una herramienta de desarrollo que es inmensamente utilizada a nivel global para validar que el c digo Javascript que escribimos est  alineado con las convenciones generales entre otros muchos usos. Para hacer honor al concepto un linter se define como la herramienta para lograr la mejora continua del c digo que escribimos con lo que no solo se enfoca en el cumplimiento de convenciones sino en un sin n mero de controles para lograr c digo de calidad.

ESLint es la dependencia de proyectos Javascript que lleva a cabo esta tarea:

## eslint

9.0.0 • Public • Published 7 days ago

[Readme](#)
[Code](#) Beta
34 Dependencies
23.536 Dependents
354 Versions

npm v9.0.0
downloads 160M/month
CI passing
license scan passing

backers 342
sponsors 133
Follow

## ESLint

[Website](#) | [Configure ESLint](#) | [Rules](#) | [Contribute to ESLint](#) | [Report Bugs](#) | [Code of Conduct](#) | [Twitter](#) | [Discord](#) | [Mastodon](#)

ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code. In many ways, it is similar to JSLint and JSHint with a few exceptions:

- ESLint uses [Espree](#) for JavaScript parsing.
- ESLint uses an AST to evaluate patterns in code.
- ESLint is completely pluggable, every single rule is a plugin and you can add more at runtime.

### Table of Contents

Install

```
> npm i eslint
```

Repository

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

Homepage

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

♥ Fund this package

Weekly Downloads

31.519.569

Esta dependencia en realidad va a realizar controles y nos va a servir mientras estemos desarrollando pero no va a ser necesaria en la aplicaci n definitiva, por esto vamos a instalar la dependencia para desarrollo con:

```
npm i --save-dev eslint
```

El comando `npm i --save-dev eslint` instala eslint como una dependencia de desarrollo en tu proyecto Node.js.



Veamos que hace cada parte del comando:

`npm i`: Es un atajo para `npm install`, que instala un paquete. `--save-dev`: Este flag indica que el paquete debe ser agregado a las dependencias de desarrollo (`devDependencies`) en tu archivo `package.json`. Las dependencias de desarrollo son aquellas que no son necesarias para ejecutar tu aplicación, pero son útiles para el desarrollo, como las herramientas de testing o linting. `eslint`: Este es el paquete que estás instalando. ESLint es una herramienta de linting para JavaScript que ayuda a encontrar y corregir problemas en tu código. Por lo tanto, después de ejecutar este comando, `eslint` estará disponible en tu proyecto para ayudarte a mantener la calidad de tu código.

Veamos como queda el archivo `package.json` después de la última instalación.

```
[13-04 00:51] philip@HPZBOOK-PHILIP | D:\Facu\Grado\DDS\repo\catedra\apuntes\semana-05\apunte-09\desafio-1:
> cat package.json
{
  "name": "desafio-1",
  "version": "1.0.0",
  "description": "Proyecto para dar solución al ejercicio del Desafío 0 1 de DDS",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "philip",
  "license": "ISC",
  "dependencies": {
    "seedrandom": "^3.0.5"
  },
  "devDependencies": {
    "eslint": "^9.0.0"
  }
}
```

No agregamos el tree del proyecto porque el directorio `node_modules` crece bastante al instalar `eslint`, sin embargo nos parece interesante revisar el archivo `package.json` luego de la instalación donde podemos observar la configuración en `devDependencies` separada de la configuración de `seedrandom`.

- Ahora a programar nuestro código: vamos a programar en `index.js` nuestro código y realizar las configuración en `scripts` para ejecutar proyecto a partir de `npm run`. Veamos las novedades en el archivo `package.json`:

```
[13-04 01:05] philip@HPZBOOK-PHILIP | D:\Facu\Grado\DDS\repo\catedra\apuntes\semana-05\apunte-09\desafio-1:
> cat package.json
{
  "name": "desafio-1",
  "version": "1.0.0",
  "description": "Proyecto para dar solución al ejercicio del Desafío 0 1 de DDS",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "philip",
  "license": "ISC",
  "type": "module",
  "dependencies": {
    "seedrandom": "^3.0.5"
  },
  "devDependencies": {
    "eslint": "^9.0.0"
  }
}
```

Podemos observar la alternativa `start` dentro de la propiedad `scripts` y además una configuración que fue mencionada en el Apunte 07 en el apartado de la utilización de módulos que le avisa a Node

que vamos a utilizar `import` para la importación de módulos externos, en este caso para la importación de `seedrandom`.

```
apunte-09 > desafio-1 > JS index.js > ...
1  "use strict";
2
3  import seedrandom from "seedrandom";
4
5  // Establecer la semilla
6  var random = seedrandom(1763519);
7
8  // Generar 1000 números aleatorios en un array.
9  let numerosAleatorios = Array.from({ length: 100 }, () => random.int32());
10
11 // Demostración de forEach para iterar sobre los elementos del array.
12 console.log("Demostración de join: {" + numerosAleatorios.join(", ") + "}");
13
14 // Demostración de filter para buscar los números positivos
15 console.log("Cantidad de positivos", numerosAleatorios.filter(value => value > 0).length);
16 |
```

[!NOTE] Para el ejemplo hemos generado solo 100 números aleatorios y solo hemos contado los positivos.

- Finalmente solo queda ejecutar el programa mediante la utilización de `npm run start` y verificar los resultados:

```
[13-04 01:14] philip@HPZBOOK-PHILIP | D:\Facu\Grado\DDS\repo\catedra\apuntes\semana-05\apunte-09\desafio-1:
> npm run start
apunte-09 > Apunte09.md > # Apunte 09 - Node Js
> /desafio-1 1.0.0 start
> node index.js
Demostración de join: {623759552, 1523424163, -576443027, 1065402314, -2076574705, -2001258260, -772425079, 10
1833851516, 2902383984, -452800445, -329753407, -700205568, -648760359, 390618083, 895409171, -1612528881, 138
3301352, 1388354082, 557503106, -1466185485, -2117130415, -1211660713, 1374044922, -1511211324, -925238450, 88
578540681, 1303705639, -1011345613, -246128175, 1593099511, -881002431, 280339583, -895713289, -692980538, -14
-1856888831, 736534122, -532445789, 1358340112, 1661818758, -1101104410, -2120848327, 799579544, -114091886, -
-1766390775, -1342038811, -85216360, 1674838659, -136619189, 829816439, -443901398, 2011099393, -401966607,
1867448653, 349233884, -163691904, -1675102205, 715646594, 243325580, -784467429, -928040070, 360755199, -16
1873298900, 707315092, 41722652, -2101083846, 101979425, 204691127, -1515891737, 2128261189, -1939670158, 1720
73366626, -1436551903, -1280883346, -2030519550, -1746028645, -1531043981, 1996352238, -174806996, 950560240,
6, -1148489891, -220953191, 1964645754, 51322512, -183338478, -1851854490, 50444974, 489456798, -945665960,
457085920, 796160228}
Cantidad de positivos 42
[13-04 01:14] philip@HPZBOOK-PHILIP | D:\Facu\Grado\DDS\repo\catedra\apuntes\semana-05\apunte-09\desafio-1:
>
```

## Bibliografía

- Mozilla Developer Network - <https://developer.mozilla.org/>
- Nodejs Documentation <https://nodejs.org/docs/latest/api/>
- Artículo - <https://www.esparkinfo.com/blog/node-js-architecture.html>
- Free Code Camp en Español - <https://www.freecodecamp.org/espanol>