# **Trabajo Práctico: Objetivos – Renzo Di Laudo Co-3**

## **Parte 1: npm y package.json**

### **1. Introducción a npm**

**npm (Node Package Manager)** es el gestor de paquetes de Node.js, utilizado para instalar, compartir y administrar librerías y herramientas en proyectos de JavaScript.

### **2. Creación de un proyecto con npm**

#### **Pasos:**

1. Abre una terminal y crea una nueva carpeta para el proyecto:  
   mkdir mi-proyecto && cd mi-proyecto
2. Inicializa un proyecto de Node.js con:  
   npm init o npm init -y

### **3. Explorando package.json**

El archivo package.json contiene metadatos del proyecto y dependencias. Ejemplo de un archivo package.json:

{

"name": "mi-proyecto",

"version": "1.0.0",

"description": "Un proyecto de prueba",

"main": "index.js",

"scripts": {

"start": "node index.js"

},

"dependencies": {

"yargs": "^17.7.2"

}

}

### **4. Instalación de paquetes**

Para instalar un paquete, usa:

npm install nombre-del-paquete

Por ejemplo, para instalar yargs:

npm install yargs

Para instalar dependencias de desarrollo:

npm install eslint --save-dev

## **Parte 2: Uso de Yargs**

### **1. Introducción a Yargs**

**Yargs** es una librería que facilita el manejo de argumentos en la línea de comandos en Node.js.

### **2. Creación de un script con Yargs**

Crea un archivo index.js y escribe el siguiente código:

const yargs = require('yargs');

const argv = yargs

.command('saludar', 'Muestra un saludo', {

nombre: {

describe: 'Nombre de la persona a saludar',

demandOption: true,

type: 'string'

}

})

.help()

.argv;

if (argv.\_.includes('saludar')) {

console.log(`Hola, ${argv.nombre}!`);

}

### **3. Ejecutando el script**

Ejecuta el siguiente comando en la terminal:

node index.js saludar --nombre="Juan"

Salida esperada:

Hola, Juan!

## **Ejercicios adicionales**

### **Ejercicios teóricos:**

1. **Explica qué es npm y su función en un proyecto de Node.js.**

**npm (Node Package Manager)** es un sistema de gestión de paquetes para **Node.js**. Su principal función es permitir a los desarrolladores gestionar las dependencias (librerías o módulos externos) que necesitan en sus proyectos de Node.js. Con npm se puede instalar, actualizar, desinstalar y gestionar bibliotecas y herramientas de terceros que la aplicación Node.js necesita para funcionar. También permite gestionar los scripts de automatización y las configuraciones de tu proyecto a través del archivo package.json.

1. **Describe la estructura básica del archivo package.json y menciona tres campos importantes**.

El archivo *package.json* es el archivo de configuración principal de un proyecto Node.js y contiene información sobre el proyecto, como su nombre, versión y descripción. También define las dependencias necesarias para el proyecto en los campos *dependencies* y *devDependencies*, especificando qué paquetes deben instalarse para que la aplicación funcione correctamente. También incluye un campo scripts que permite definir comandos personalizados, como npm start o npm test, que facilitan la ejecución de tareas comunes.

Entre los campos importantes tenemos:

1. **dependencies:** Aquí se listan las dependencias necesarias para que la aplicación funcione. Por ejemplo, si usas un framework como **Express**, se incluiría aquí.
2. **devDependencies:** Aquí se listan las dependencias necesarias solo en el entorno de desarrollo, como herramientas de testing o de automatización de tareas, por ejemplo, **Nodemon**.
3. **scripts:** Este campo permite definir comandos personalizados que puedes ejecutar usando npm, como npm run start, npm run test, entre otros.
4. ¿Cuál es la diferencia entre dependencias normales y dependencias de desarrollo en npm?

Las **dependencias normales son** aquellas que son necesarias para que la aplicación funcione correctamente en producción. Se instalan con el comando npm install sin ninguna opción adicional. Mientras que las **dependencias de desarrollo (devDependencies) s**on aquellas que solo se necesitan durante el desarrollo de la aplicación, pero no en producción. Por ejemplo, herramientas de testing, herramientas de compilación o autorefresco.

1. ¿Qué comando utilizamos para actualizar un paquete instalado en nuestro proyecto?

Se utiliza el comando "update” de la siguiente manera:

npm update <nombre-del-paquete>

1. ¿Para que se utiliza el comando npm link?

El comando npm link se utiliza para crear un enlace simbólico entre un paquete globalmente instalado y un proyecto local. Esto es útil cuando estás desarrollando un paquete y quieres probarlo en un proyecto local sin tener que publicarlo o instalarlo como una dependencia externa. Se ejecuta de la siguiente manera:

npm link <nombre-del-paquete>

1. Ver el módulo de inicio de un proyecto

El archivo de inicio de un proyecto suele ser un “index.js”, sin embargo, puede ejecutarse el siguiente comando para ver el archivo que se está utilizando cómo módulo principal:

cat package.json | grep "main"

### **Ejercicios prácticos:**

1. Agrega otro comando a Yargs llamado despedir que muestre "Adiós, [nombre]!".
2. Crea un script con Yargs que permita sumar dos números desde la línea de comandos. (n1 y n2)
3. Implementa un comando que lea un archivo JSON y muestre su contenido en la terminal.
4. Modifica el script para que, si el usuario no proporciona el argumento nombre, se muestre un mensaje de error adecuado.

**Link a github con este y el resto de trabajos prácticos resueltos:** [**https://github.com/Renchu109/LaboratorioIV**](https://github.com/Renchu109/LaboratorioIV)