Taller de Ejercicios JS

**Programación Avanzada**

*Profe. Jossy Tello*

*Investiga los diferentes módulos que vienen integrados en node.js.*

* *investiga las propiedades y funciones de cada módulo.*

1. *módulo console, módulo process, módulo http, módulo OS,*

*módulo Timers, módulo fs.*

1. *realizar un ejemplo práctico de cada módulo.*
2. *Consultar que es MPN y como crear un paquete.*
   * + 1. *Módulo console:*

*El módulo console proporciona una forma de interactuar con la consola en Node.js. Se utiliza para imprimir mensajes en la consola, lo que es útil para depuración y seguimiento del flujo de ejecución.*

*Propiedades y funciones importantes:*

* 1. *console.log(message[, ...args]): Imprime un mensaje en la consola, aceptando varios argumentos que se concatenan para formar el mensaje.*

Copiar

console.log('Hello,', 'Node.js!');

* 1. *console.error([data][, ...args]): Similar a console.log(), pero se utiliza para imprimir mensajes de error.*

console.error('This is an error message.');

* 1. *console.warn([data][, ...args]): Imprime un mensaje de advertencia en la consola.*

console.warn('This is a warning.')

* 1. *console.clear(): Limpia la consola.*

console.clear();

1. *Módulo process:*

*El módulo process proporciona información y control sobre el proceso en ejecución. Contiene información sobre los argumentos de la línea de comandos, entorno, flujo de entrada/salida y más.*

*Propiedades y funciones importantes:*

* 1. *process.argv: Un array que contiene los argumentos de la línea de comandos proporcionados al ejecutar el script.*

console.log(process.argv);

* 1. *process.env: Un objeto que contiene las variables de entorno del sistema.*

console.log(process.env.NODE\_ENV);

* 1. *process.cwd(): Devuelve el directorio de trabajo actual del proceso.*

console.log('Current working directory:', process.cwd());

* 1. *process.exit([code]): Finaliza el proceso. El código de salida por defecto es 0 (éxito), pero puedes proporcionar un código personalizado.*

if (someCondition) { process.exit(1); // Termina con código de error }

else { process.exit(0); // Termina con éxito }

* 1. *process.on(eventName, callback): Establece un manejador de eventos para el proceso.*

process.on('exit', (code) => {   
console.log(`Process exited with code ${code}`); });

1. *Módulo http:*

*El módulo http proporciona funcionalidades para crear servidores HTTP y realizar solicitudes a través de HTTP en Node.js.*

*Propiedades y funciones importantes:*

* 1. *http.createServer([options][, requestListener]): Crea un servidor HTTP y devuelve una instancia del servidor.*

const http = require('http');   
const server = http.createServer((req, res) => {  
 res.writeHead(200, { 'Content-Type': 'text/plain' });   
res.end('Hello, world!\n'); });   
  
server.listen(3000, () => {  
 console.log('Server is running on port 3000'); });

* 1. *http.request(options[, callback]): Realiza una solicitud HTTP a un servidor.*

const http = require('http');  
const options = {  
hostname: 'www.example.com',  
port: 80,  
path: '/',  
method: 'GET'  
};  
  
const req = http.request(options, (res) => {  
console.log(`Response status code: ${res.statusCode}`);  
});  
  
req.end();

1. *Módulo os:*

*El módulo os proporciona información sobre el sistema operativo y el entorno del sistema.*

*Propiedades y funciones importantes:*

* 1. *os.cpus(): Devuelve información sobre los núcleos de la CPU del sistema.*

const os = require('os');  
const cpus = os.cpus();  
console.log(cpus);

* 1. *os.totalmem(): Devuelve la cantidad total de memoria del sistema en bytes.*

const os = require('os');  
const totalMemory = os.totalmem();  
console.log(`Total memory: ${totalMemory} bytes`);

* 1. *os.hostname(): Devuelve el nombre de host del sistema.*

const os = require('os');  
const hostname = os.hostname();  
console.log(`Hostname: ${hostname}`);

* 1. *os.platform(): Devuelve la plataforma del sistema operativo.*

const os = require('os');  
const platform = os.platform();  
console.log(`Platform: ${platform}`);

* 1. *os.arch(): Devuelve la arquitectura de la CPU del sistema.*

const os = require('os');  
const architecture = os.arch();  
console.log(`Architecture: ${architecture}`);

1. *Módulo timers:* *El módulo timers proporciona funcionalidades para programar tareas para que se ejecuten en el futuro.*

*Funciones importantes:*

* 1. *setTimeout(callback, delay[, ...args]): Ejecuta una función después de un cierto retraso en milisegundos.*

const delayedFunction = () => {  
console.log('Delayed function executed');  
};  
  
setTimeout(delayedFunction, 2000); // Ejecutará delayedFunction después de 2 segundos

* 1. *setInterval(callback, delay[, ...args]): Ejecuta una función repetidamente con un intervalo específico en milisegundos*

const intervalFunction = () => {  
console.log('Interval function executed');  
};  
  
setInterval(intervalFunction, 1000); // Ejecutará intervalFunction cada segundo

* 1. *clearTimeout(timeoutObject): Cancela una ejecución programada con setTimeout.*

const timeoutFunction = () => {  
console.log('This should not be executed');  
};  
  
const timeout = setTimeout(timeoutFunction, 1000);  
clearTimeout(timeout); // Cancela la ejecución programada

* 1. *clearInterval(intervalObject): Cancela la repetición programada con setInterval.*

const intervalFunction = () => {  
console.log('This should not be repeatedly executed');  
};  
  
const interval = setInterval(intervalFunction, 1000);  
clearInterval(interval); // Detiene la repetición programada

1. *Módulo fs:*

*El módulo fs proporciona funcionalidades para trabajar con el sistema de archivos en Node.js.*

*Funciones importantes:*

* 1. *fs.readFile(path[, options], callback): Lee un archivo asincrónicamente.*

const fs = require('fs');  
const content = 'Hello, Node.js!';  
fs.writeFile('file.txt', content, (err) => {  
if (err) throw err;

console.log('File has been written.');

});

* 1. *fs.readdir(path[, options], callback): Lee el contenido de un directorio asincrónicamente.*

const fs = require('fs');  
const content = 'Hello, Node.js!';  
fs.writeFile('file.txt', content, (err) => {  
if (err) throw err;  
console.log('File has been written.');  
});

* 1. *fs.readdir(path[, options], callback): Lee el contenido de un directorio asincrónicamente.*

const fs = require('fs');  
fs.readdir('.', (err, files) => {  
if (err) throw err;  
console.log('Files in current directory:', files);   
});

* 1. *fs.stat(path[, options], callback): Obtiene información sobre un archivo o directorio.*

const fs = require('fs');  
fs.stat('file.txt', (err, stats) => {  
if (err) throw err;  
console.log('Is file:', stats.isFile());  
console.log('Is directory:', stats.isDirectory());  
});

1. *npm (Node Package Manager) es el gestor de paquetes oficial de Node.js y se utiliza para instalar, gestionar y compartir paquetes y módulos de código reutilizables. Los paquetes son conjuntos de código JavaScript que pueden contener librerías, frameworks, herramientas y cualquier recurso necesario para un proyecto.*

*Crear un paquete npm implica empaquetar tu código, recursos y metadata en un formato específico que pueda ser compartido y utilizado por otros desarrolladores en sus proyectos.*

*cómo puedes crear y publicar npm:*

*Estructura del proyecto: Organiza tu código y recursos en una estructura de directorios coherente. Asegúrate de incluir un archivo package.json, que contiene la información de metadatos y configuración del paquete.*

* 1. *Crear package.json: Puedes crear este archivo manualmente o usando el comando npm init. Aquí es donde proporcionas detalles sobre el paquete, como nombre, versión, descripción, autor, dependencias, scripts y más.*
  2. *Código y recursos: Asegúrate de que tu código esté en una estructura coherente y que todos los recursos necesarios estén incluidos.*
  3. *Pruebas: Si es posible, incluye pruebas unitarias y/o de integración para asegurarte de que tu paquete funciona como se espera.*
  4. *Registrar una cuenta npm: Si aún no tienes una cuenta en el registro de npm, regístrate en .*
  5. *Iniciar sesión en npm: Abre tu terminal y ejecuta npm login para iniciar sesión en tu cuenta de npm.*
  6. *Publicar el paquete: Ejecuta npm publish en el directorio raíz de tu paquete. Esto enviará tu paquete al registro de npm para que otros lo puedan instalar.*
  7. *Actualizar el paquete: Si necesitas hacer cambios en tu paquete, puedes actualizar el código y la versión en el package.json, luego ejecuta npm publish nuevamente para actualizar la versión.*
  8. *Usar el paquete: Otros desarrolladores pueden instalar tu paquete en sus proyectos usando npm install <nombre-del-paquete>. Luego, pueden importar y usar las funciones o componentes que has proporcionado.+*

*Publicar paquetes en el registro de npm significa que tu código será público y compartido con la comunidad. Asegúrate de proporcionar una documentación clara y útil para que otros puedan entender y utilizar tu paquete.*