

keywords=typeof, new, true, false, catch, function, return, null,
catch, switch, var, if, in, while, do, else, case, break,
keywordstyle=,

ndkeywords=class, export, boolean, throw, implements,
import, this,

ndkeywordstyle=,

identifierstyle=,

sensitive=false,

comment=[l]//,

morecomment=[s]//,

commentstyle=,

stringstyle=,

morestring=[b] ',

morestring=[b] "

language=JavaScript,

backgroundcolor=,

extendedchars=true,

basicstyle=

Contenido

Introducción a NodeJS

Instalación y Primeros Pasos

Módulos en NodeJS

Manejo de Archivos

Manejo de Eventos

Servidor Web Básico

Asincronía en NodeJS






Depuración

Mejores Prácticas

¿Qué es NodeJS?

- ▶ Entorno de ejecución de JavaScript del lado del servidor
- ▶ Creado por Ryan Dahl en 2009
- ▶ Permite ejecutar JavaScript fuera del navegador
- ▶ Utiliza el motor V8 de Google Chrome
- ▶ Código abierto y multiplataforma

Ventajas de NodeJS

- ▶  Velocidad y eficiencia
- ▶  Ecosistema rico (NPM)
- ▶  Versatilidad
- ▶  Misma lengua en frontend y backend
- ▶  Gran comunidad y soporte

Instalación de NodeJS

1. Visitar <https://nodejs.org>
2. Descargar versión LTS
3. Seguir instrucciones del instalador
4. Verificar instalación:

```
node —version
```

```
npm —version
```

Creación y Ejecución de un Script Básico

1. Crear archivo `hola_mundo.js`:

```
console.log('Hola , mundo!');
```

2. Ejecutar en terminal:

```
node hola_mundo.js
```

Módulos Incorporados

- ▶ NodeJS incluye módulos como `fs`, `http`, `path`
- ▶ Uso de `require()`:

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

Creación de Módulos Personalizados

```
// miModulo.js  
function saludar(nombre) {  
    return 'Hola , ${nombre}!';  
}
```

```
module.exports = {  
    saludar: saludar  
};
```

```
// uso.js  
const miModulo = require('./miModulo');  
console.log(miModulo.saludar("Juan"));
```


Lectura de Archivos

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

fs.readFile('archivo.txt', 'utf8', (err, data) => {
  if (err) {
    console.error('Error:', err);
    return;
  }
  console.log('Contenido:', data);
});
```

Escritura de Archivos

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

const contenido = 'Nuevo contenido';

fs.writeFile('nuevo.txt', contenido, (err) => {
  if (err) {
    console.error('Error:', err);
    return;
  }
  console.log('Archivo guardado');
});
```

Patrón Observer en NodeJS

```
const EventEmitter = require('events');

class MiEmissor extends EventEmitter {}

const miEmissor = new MiEmissor();

miEmissor.on('evento', () => {
    console.log('Evento ocurrido!');
});

miEmissor.emit('evento');
```

Creación de un Servidor HTTP

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

const servidor = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('  Hola  , mundo!');
});

servidor.listen(3000, () => {
  console.log('Servidor en http://localhost:3000/');
});
```

Callbacks

```
function operacionAsincrona(callback) {  
    setTimeout(() => {  
        callback('Operaci n completada');  
    }, 1000);  
}  
  
operacionAsincrona((resultado) => {  
    console.log(resultado);  
});
```

Promesas

```
function operacionAsincrona() {  
    return new Promise((resolve, reject) => {  
        setTimeout(() => {  
            resolve('Operaci n completada');  
        }, 1000);  
    });  
}
```

```
operacionAsincrona()  
    .then(resultado => console.log(resultado))  
    .catch(error => console.error(error));
```

Async/Await

```
async function ejecutarOperacion() {  
    try {  
        const resultado = await operacionAsincrona(  
            console.log(resultado);  
        ) catch (error) {  
            console.error(error);  
        }  
    }  
}  
  
ejecutarOperacion();
```

Técnicas de Depuración

- ▶ Uso de `console.log()`
- ▶ Depurador incorporado de NodeJS
 - ▶ `node --inspect miScript.js`
 - ▶ Abrir Chrome: `chrome://inspect`
- ▶ Depurador de IDE (ej. Visual Studio Code)

Buenas Prácticas en NodeJS

- ▶ Manejo adecuado de errores
- ▶ Usar estándar de codificación (ESLint)
- ▶ Evitar callback hell
- ▶ Modularizar el código
- ▶ Usar variables de entorno
- ▶ Implementar logging
- ▶ Realizar pruebas unitarias y de integración

Conclusión

- ▶ NodeJS: potente plataforma para JavaScript del lado del servidor
- ▶ Modelo asíncrono eficiente
- ▶ Amplio ecosistema de módulos
- ▶ La práctica es clave para dominar NodeJS
- ▶ Explora y experimenta con diferentes módulos y técnicas