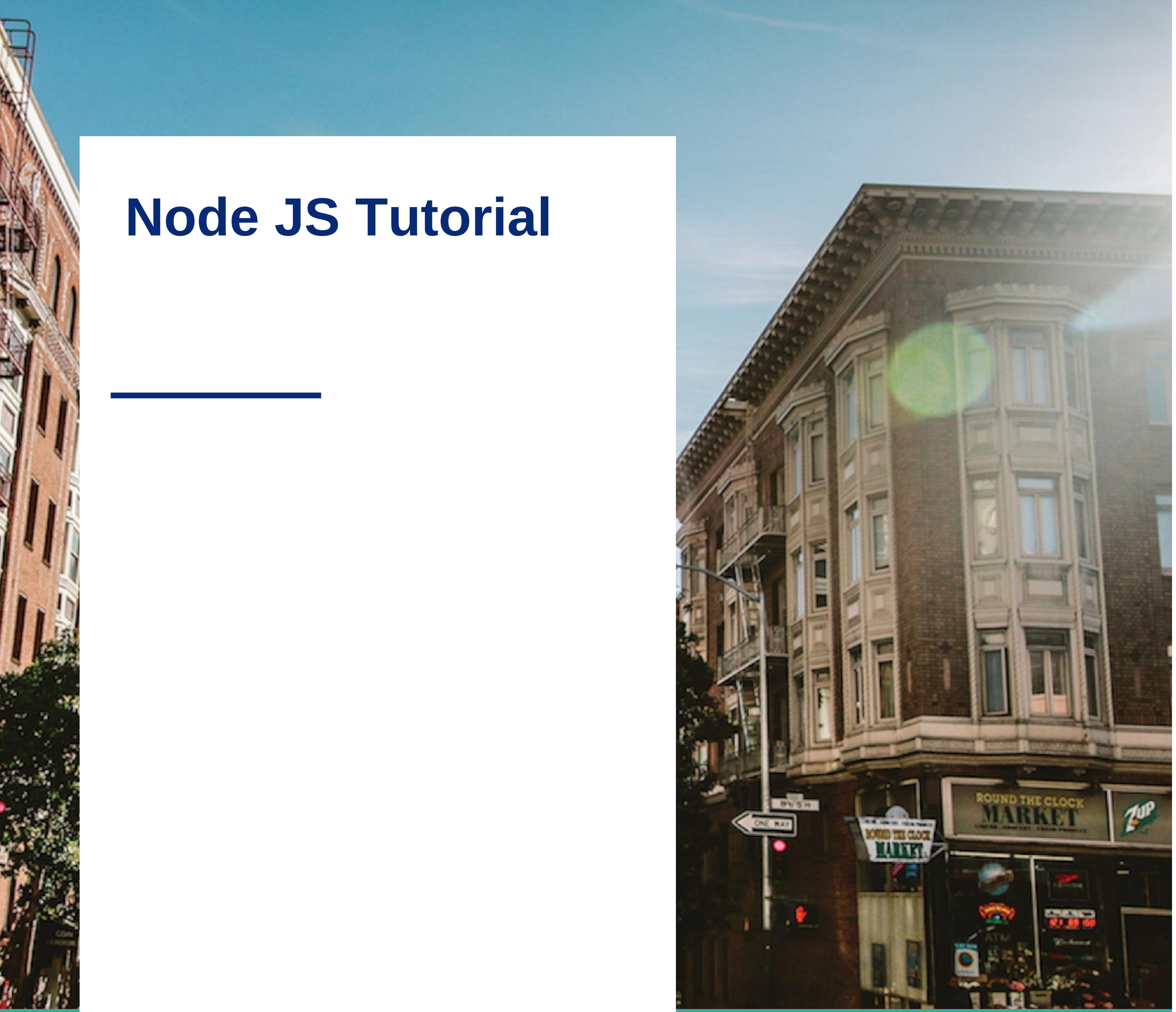

Node JS Tutorial

JANUARY 15

ABES Engineering College

What is Node.js?

Node.js is an open-source, cross-platform runtime environment built on Chrome's V8 JavaScript engine. It allows developers to execute JavaScript code outside of a web browser. It is designed to build scalable, high-performance, and efficient network applications.

Key Features of Node.js

1. **Event-Driven and Non-Blocking I/O:**
    - **Node.js operates on a single-threaded event loop, making it lightweight and efficient.**
    - **Non-blocking I/O operations allow Node.js to handle multiple requests without waiting for any operation to complete.**
2. **Asynchronous Programming:**
    - **Most of the operations in Node.js are asynchronous, meaning they don't block the execution thread. This is achieved through callbacks, promises, or async/await.**
3. **Fast Execution:**
    - **Built on the V8 engine, Node.js can compile JavaScript code into machine code, ensuring high performance.**
4. **Scalability:**
    - **Node.js is well-suited for building scalable applications due to its asynchronous nature and event-driven architecture.**
5. **NPM (Node Package Manager):**
    - **Node.js includes the npm ecosystem, which provides access to thousands of open-source libraries and tools for faster development.**
6. **Cross-Platform:**

- Node.js is compatible with Windows, macOS, and Linux, making it versatile for developers.

Core Modules of Node.js

1. FS (File System): **Provides file I/O operations.**
2. OS: **Provides information about the operating system.**
3. Path: **Offers utilities for working with file and directory paths.**
4. HTTP: **Used for creating servers and handling HTTP requests and responses.**

FS (File System): Provides file I/O operations

The fs (File System) module in Node.js provides a rich set of functionalities to interact with the file system, allowing you to perform operations such as creating, reading, writing, deleting, and modifying files and directories.

---

1. Loading the fs Module

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

---

2. Basic Operations

A. Writing to a File

- **fs.writeFile(file, data, callback): Creates a new file or overwrites an existing file.**

```
fs.writeFile('example.txt', 'Hello, World!', (err) => {
  if (err) throw err;
  console.log('File created and written to!');
});
```

B. Reading a File

- fs.readFile(file, encoding, callback): Reads the content of a file.

```
fs.readFile('example.txt', 'utf-8', (err, data) => {
  if (err) throw err;
  console.log('File content:', data);
});
```

C. Appending to a File

- fs.appendFile(file, data, callback): Appends data to the file.

```
fs.appendFile('example.txt', '\nAppended text.', (err) => {
  if (err) throw err;
  console.log('Data appended to file!');
});
```

D. Deleting a File

- fs.unlink(file, callback): Deletes a file.

```
fs.unlink('example.txt', (err) => {
  if (err) throw err;
  console.log('File deleted!');
});
```

E. Renaming a File

- fs.rename(oldPath, newPath, callback): Renames a file.

```
fs.rename('example.txt', 'renamed_example.txt', (err) => {
  if (err) throw err;
  console.log('File renamed!');
});
```

---

3. Working with Directories

A. Creating a Directory

- fs.mkdir(path, options, callback): Creates a new directory.

```
fs.mkdir('myFolder', { recursive: true }, (err) => {
  if (err) throw err;
  console.log('Directory created!');
});
```

B. Reading a Directory

- fs.readdir(path, callback): Reads the contents of a directory.

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

C. Removing a Directory

- fs.rmdir(path, callback): Removes an empty directory.

```
fs.rmdir('myFolder', (err) => {
  if (err) throw err;
  console.log('Directory removed!');
});
```

---

4. Checking File or Directory Status

A. Checking if a File Exists

- fs.existsSync(path): Checks if a file or directory exists.

```
if (fs.existsSync('example.txt')) {
  console.log('File exists!');
} else {
  console.log('File does not exist!');
}
```

B. Getting File Stats

- fs.stat(path, callback): Retrieves file or directory stats.

```
fs.stat('example.txt', (err, stats) => {
  if (err) throw err;
  console.log('Is file:', stats.isFile());
  console.log('Is directory:', stats.isDirectory());
  console.log('Size:', stats.size, 'bytes');
});
```

Examples of File Handling

OS Module: Provides information about the operating system.

The os module in Node.js provides a set of operating system-related utility methods and properties. It allows you to interact with and retrieve information about the operating system on which your Node.js application is running.

Usage

To use the os module, you must first import it:

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

Common Methods and Properties

1. os.arch()

- Returns the CPU architecture of the system.

  Example: `console.log(os.arch()); // e.g., 'x64'`

2. os.cpus()

- Returns an array of objects containing information about each CPU/core.

Example: `console.log(os.cpus());`

3. os.freemem()

- Returns the amount of free system memory in bytes.

  Example: ``console.log(`Free Memory: ${os.freemem()} bytes`);``

4. os.homedir()

- Returns the path of the current user's home directory.

  Example: `console.log(os.homedir()); // e.g., '/Users/username'`

5. os.hostname()

- Returns the hostname of the operating system.

  Example: `console.log(os.hostname()); // e.g., 'My-PC'`

6. os.platform()

- Returns the operating system platform.

  Example: `console.log(os.platform()); // e.g., 'win32', 'linux', 'darwin'`

Path: Offers utilities for working with file and directory paths.

The path module in Node.js provides utilities for working with file and directory paths. It is part of Node.js' built-in modules, so you don't need to install it. You can use it to handle and transform file paths across platforms.

To use the path module, import it as follows:

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

Common Methods and Properties

1. path.basename(path[, ext])

Returns the last portion of a path (the file name).

If the optional ext parameter is provided, it will be removed from the result.

Example:

```
console.log(path.basename('/user/local/bin/file.txt')); // 'file.txt'

console.log(path.basename('/user/local/bin/file.txt', '.txt')); // 'file'
```

2. path.dirname(path)

- **Returns the directory portion of a path.**

  Example:

```
console.log(path.dirname('/user/local/bin/file.txt')); // '/user/local/bin'
```

3. path.extname(path)

- **Returns the file extension from the path.**

  Example:

```
console.log(path.extname('/user/local/bin/file.txt')); // '.txt'
```

HTTP: Used for creating servers and handling HTTP requests and responses.

The http module in Node.js is a core module that allows you to create and manage HTTP servers and make HTTP requests. It's a fundamental part of Node.js, widely used for building web applications and APIs.

Importing the http Module

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

Creating an HTTP Server

You can create a basic HTTP server using the http.createServer() method.

Create an HTTP server

```
const server = http.createServer((req, res) => {
  // Set the response header and status code
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  // Send response data
  res.end('Hello, World!\n');
});
// Start the server on port 3000
server.listen(3000, () => {
  console.log('Server is running at http://localhost:3000/');
});
```

Handling Requests and Responses

The http.createServer callback function receives two parameters:

1. **req (request): Represents the incoming HTTP request.**
2. **res (response): Represents the outgoing HTTP response.**

Example: Responding to Different Routes

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

const server = http.createServer((req, res) => {

  if (req.url === '/') {

    res.writeHead(200, { 'Content-Type': 'text/plain' });

    res.end('Welcome to the Home Page!');

  } else if (req.url === '/about') {

    res.writeHead(200, { 'Content-Type': 'text/plain' });

    res.end('About Us Page');

  } else {

    res.writeHead(404, { 'Content-Type': 'text/plain' });

    res.end('Page Not Found');

  }

});

server.listen(3000, () => {

  console.log('Server is running at http://localhost:3000/');

});
```

HTTP Methods

You can handle different HTTP methods (e.g., GET, POST) by checking req.method.

Example: Handling GET and POST Requests

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

const server = http.createServer((req, res) => {

  if (req.method === 'GET' && req.url === '/') {

    res.writeHead(200, { 'Content-Type': 'text/plain' });

    res.end('This is a GET request');

  } else if (req.method === 'POST' && req.url === '/submit') {

    let body = '';

    // Collect data chunks

    req.on('data', chunk => {

      body += chunk;

    });

    // Process data when complete

    req.on('end', () => {

      res.writeHead(200, { 'Content-Type': 'text/plain' });

      res.end(`Data received: ${body}`);

    });

  } else {

    res.writeHead(404, { 'Content-Type': 'text/plain' });

    res.end('Not Found');
  }
});

server.listen(3000, () => {
  console.log('Server is running at http://localhost:3000/');
});
```

Sending JSON Data

You can send JSON responses by setting the Content-Type to application/json.

Example: Sending JSON Response

```
const http = require('http');
const server = http.createServer((req, res) => {
  if (req.url === '/api') {
    const data = {
      message: 'Hello, World!',
      status: 'success',
    };
    res.writeHead(200, { 'Content-Type': 'application/json' });
    res.end(JSON.stringify(data));
  } else {
    res.writeHead(404, { 'Content-Type': 'text/plain' });
    res.end('API Not Found');
  }
});
server.listen(3000, () => {
  console.log('Server is running at http://localhost:3000/');
});
```