**Practical - 4**

**URL Parsing and Manipulation:**

**Aim:** Write a program that accepts a URL as user input and uses the URL module to parse it.

Display the protocol, host, path, and query parameters separately.

**Source Code:**

const url = require('url');

const readline = require('readline');

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

function parseURL(inputURL) {

const parsedURL = url.parse(inputURL, true);

console.log('Protocol:', parsedURL.protocol);

console.log('Host:', parsedURL.host);

console.log('Path:', parsedURL.pathname);

console.log('Query Parameters:', parsedURL.query);

}

rl.question('Enter a URL: ', (inputURL) => {

parseURL(inputURL);

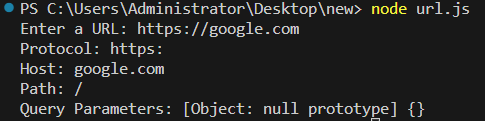
rl.close();

});

**Description:**

* The code starts by importing the necessary modules: URL and readline. The URL module provides URL parsing capabilities, while readline allows the program to read user input from the command line.
* A readline interface (rl) is created using readline.createInterface. This interface will be used to prompt the user for a URL input and read the user's response.
* The parse URL function is defined to handle the URL parsing logic. It takes a URL string as input and uses the url.parse method to parse the URL. The url.parse method splits the URL into its individual components like protocol, host, pathname (path), and query parameters. The true argument in url.parse is used to also parse the query string into an object, making it easier to display the query parameters later.
* The parseURL function then uses console.log to display each of the parsed components of the URL: protocol, host, path, and query parameters.
* The rl.question method is used to prompt the user to enter a URL. When the user provides a URL, the provided input is passed to the parseURL function to extract and display its components.
* After parsing and displaying the URL components, the readline interface is closed using rl.close().

**Output:**

****

**Aim:** Implement a function that takes a base URL and a relative path as input, and uses the

URL module to resolve and display the absolute URL.

**Source Code:**

const url = require('url');

function resolveAbsoluteURL(baseURL, relativePath) {

const absoluteURL = url.resolve(baseURL, relativePath);

console.log('Absolute URL:', absoluteURL);

}

// Example usage:

const baseURL = 'https://www.charusat.ac.in/';

const relativePath = 'path/to/resource';

resolveAbsoluteURL(baseURL, relativePath);

**Description:**

* The url module is imported using require('url'). This module provides utilities for URL parsing and manipulation, including the url.resolve method we use to resolve the absolute URL.
* The resolveAbsoluteURL function is defined, which takes two parameters: baseURL (a string representing the base URL) and relativePath (a string representing the relative path to be appended to the base URL).
* Inside the function, the url.resolve method is used to resolve the absolute URL. It takes the baseURL and relativePath as arguments and returns the concatenated absolute URL formed by combining the two inputs.
* The resolved absolute URL is then displayed using console.log, showing the user the final result.

**Output:**

****

**Query String Operation:**

**Aim:** Write a Node.js program that takes a URL with a query string as input and extracts the

key-value pairs from the query string using the query string module. The program should

display the extracted key-value pairs as output.

**Source Code:**

const url = require('url');

const querystring = require('querystring');

const readline = require('readline');

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

function extractQueryParameters(inputURL) {

const parsedURL = url.parse(inputURL);

const queryParameters = querystring.parse(parsedURL.query);

console.log('Query Parameters:');

console.log(queryParameters);

}

rl.question('Enter a URL with a query string: ', (inputURL) => {

extractQueryParameters(inputURL);

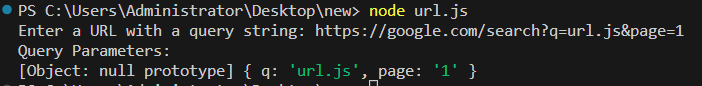
rl.close();

});

**Description:**

* The necessary modules are imported: URL, querystring, and readline. The URL module is used for parsing the URL, the querystring module for parsing the query string, and the readline module for reading user input from the command line.
* A readline interface (rl) is created using readline.createInterface. This interface is used to prompt the user for a URL input and read the user's response.
* The extractQueryParameters function is defined to handle the URL parsing and query string extraction logic. It takes a URL with a query string as input.
* Inside the function, the url.parse method is used to parse the URL and extract its components, including the query string. The querystring.parse method is then applied to the extracted query string, converting it into an object containing key-value pairs.
* The extracted key-value pairs are displayed using console.log.
* The readline interface is used to prompt the user to enter a URL with a query string. When the user provides the input, the extractQueryParameters function is called, and the program extracts and displays the key-value pairs from the query string.
* Finally, the readline interface is closed using rl.close().

**Output:**

****

**Path Operations:**

**Aim:** Create a program that accepts two file paths as input and uses the path module to

determine if they refer to the same file.

**Source Code:**

const fs = require('fs');

const path = require('path');

const readline = require('readline');

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

function areFilesSame(file1Path, file2Path) {

const resolvedPath1 = path.resolve(file1Path);

const resolvedPath2 = path.resolve(file2Path);

fs.stat(resolvedPath1, (err1, stats1) => {

if (err1) {

console.error(`Error accessing file: ${resolvedPath1}`);

return;

}

fs.stat(resolvedPath2, (err2, stats2) => {

if (err2) {

console.error(`Error accessing file: ${resolvedPath2}`);

return;

}

const isSame = stats1.ino === stats2.ino && stats1.dev === stats2.dev;

console.log(`Are the files the same? ${isSame}`);

});

});

}

rl.question('Enter the first file path: ', (file1Path) => {

rl.question('Enter the second file path: ', (file2Path) => {

areFilesSame(file1Path, file2Path);

rl.close();

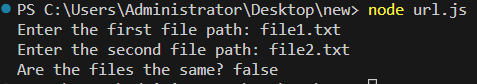
});

});

**Description:**

* The required modules are imported: fs, path, and readline. The fs module is used to access file system-related functionalities, the path module is used for handling file paths, and the readline module is used to read user input from the command line.
* A readline interface (rl) is created using readline.createInterface. This interface will be used to prompt the user for two file paths and read the user's responses.
* The areFilesSame function is defined to handle the file comparison logic. It takes two file paths as input.
* Inside the function, the path.resolve method is used to convert the provided file paths into absolute paths. This step ensures that the paths are in the correct format and can be used for file access.
* The fs.stat method is used to retrieve the file stats for each provided file path. This step checks if the files exist and are accessible.
* If both files are accessible, the areFilesSame function compares the node number (stats.ino) and the device number (stats.dev) of both files. These values uniquely identify a file on the file system. If they match, it indicates that the two paths refer to the same file.
* The program then displays the result using console.log.
* The readline interface is used to prompt the user to enter the two file paths. When the user provides the inputs, the areFilesSame function is called to determine if they refer to the same file.
* Finally, the readline interface is closed using rl.close().

**Output:**

****

**Aim:** Implement a function that accepts a file path as input and uses the path module to

extract the file extension. Display the extracted extension to the user.

**Source Code:**

const path = require('path');

const readline = require('readline');

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

function extractFileExtension(filePath) {

const extension = path.extname(filePath);

console.log('File Extension:', extension);

}

rl.question('Enter a file path: ', (filePath) => {

extractFileExtension(filePath);

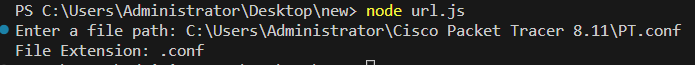
rl.close();

});

**Description:**

* The path module is imported, which is a built-in module in Node.js that provides utilities for handling file paths.
* A readline interface (rl) is created using readline.createInterface. This interface is used to prompt the user for a file path and read the user's response.
* The extractFileExtension function is defined to handle the file extension extraction logic. It takes a file path as input.
* Inside the function, the path.extname method is used to extract the file extension from the provided file path. This method returns the file extension, including the dot (.), if present.
* The extracted file extension is then displayed using console.log.
* The readline interface is used to prompt the user to enter a file path. When the user provides the input, the extractFileExtension function is called to extract and display the file extension.
* Finally, the readline interface is closed using rl.close().

**Output:**

****

**File Paths and Operations:**

**Aim:** Implement a program that accepts a file path as input and uses the path module to

extract the directory name and base name. Display the extracted values separately.

**Source Code:**

const path = require('path');

const readline = require('readline');

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

function extractDirectoryAndBaseNames(filePath) {

const directoryName = path.dirname(filePath);

const baseName = path.basename(filePath);

console.log('Directory Name:', directoryName);

console.log('Base Name:', baseName);

}

rl.question('Enter a file path: ', (filePath) => {

extractDirectoryAndBaseNames(filePath);

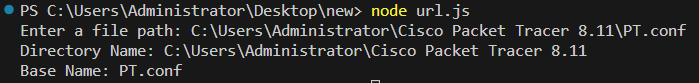
rl.close();

});

**Description:**

* The path module is imported, which is a built-in module in Node.js that provides utilities for handling file paths.
* A readline interface (rl) is created using readline.createInterface. This interface is used to prompt the user for a file path and read the user's response.
* The extractDirectoryAndBaseNames function is defined to handle the directory and base name extraction logic. It takes a file path as input.
* Inside the function, the path.dirname method is used to extract the directory name from the provided file path. This method returns the directory portion of the path.
* The path.basename method is used to extract the base name from the provided file path. This method returns the last portion of the path, typically the filename.
* The extracted directory name and base name are then displayed separately using console.log.
* The readline interface is used to prompt the user to enter a file path. When the user provides the input, the extractDirectoryAndBaseNames function is called to extract and display the directory and base names.
* Finally, the readline interface is closed using rl.close().

**Output:**

****

**Aim:** Write a function that uses the fs module to check if a given file path exists. Display a

success message if the file exists, or an error message if it doesn't.

**Source Code:**

const fs = require('fs');

const readline = require('readline');

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

function checkFileExists(filePath) {

fs.access(filePath, fs.constants.F\_OK, (err) => {

if (err) {

console.error(`Error: File "${filePath}" does not exist.`);

} else {

console.log(`Success: File "${filePath}" exists.`);

}

});

}

rl.question('Enter a file path: ', (filePath) => {

checkFileExists(filePath);

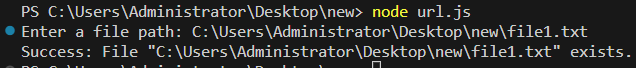
rl.close();

});

**Description:**

* The fs module is imported, which is a built-in module in Node.js that provides file system-related functionalities.
* A readline interface (rl) is created using readline.createInterface. This interface is used to prompt the user for a file path and read the user's response.
* The checkFileExists function is defined to handle the file existence check logic. It takes a file path as input.
* Inside the function, the fs.access method is used to check if the file at the provided file path exists. The fs.constants.F\_OK flag is passed as the second argument to specify that the function should check for the file's existence.
* If an error is encountered during the check (i.e., the file doesn't exist), the function prints an error message using console.error.
* If no error occurs (i.e., the file exists), the function prints a success message using console.log.
* The readline interface is used to prompt the user to enter a file path. When the user provides the input, the checkFileExists function is called to perform the file existence check and display the appropriate message.
* Finally, the readline interface is closed using rl.close().

**Output:**

****

**Learning Outcome:**

CO1: Understand various technologies and trends impacting single-page web applications.

CO4: Demonstrate the use of JavaScript to fulfill the essentials of front-end development to

back-end development