

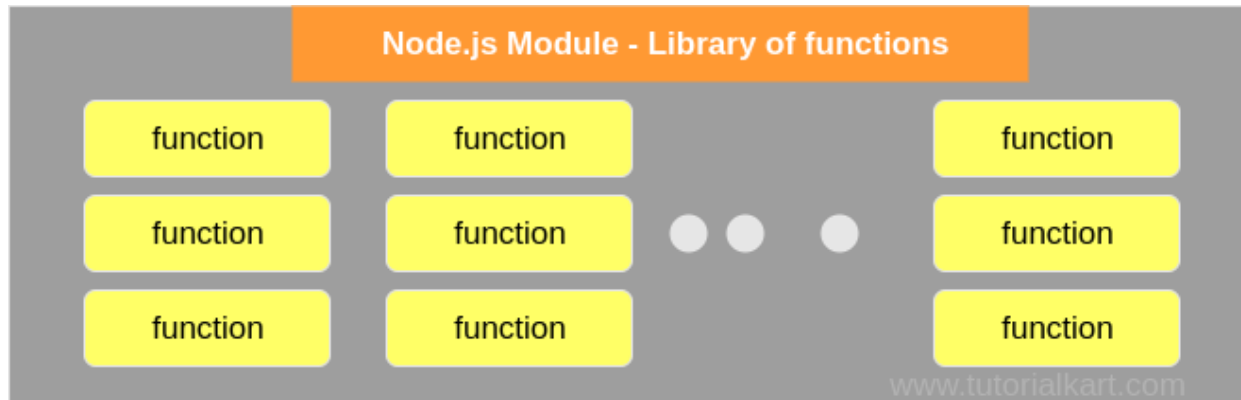


# THE SERVER SIDE JAVASCRIPT

Prof. Rachana V. Modi

# NODE.JS MODULES

- Modules are encapsulated code blocks that communicate with an external application.
- It can be a single file or a collection of multiple files/folders.
- It is reusable.



# TYPES OF MODULES

- There are three types of modules:
  - 1) Core Modules - come along with Node.js installation
  - 2) Local Modules - written by user
  - 3) Third-party Modules - available online



# NODE.JS CORE MODULES

- Built-in modules of node.js that are part of node.js and come with the Node.js installation process are known as **core modules**.
- To load/include this module in program, use the **require** function.  

```
let module = require('module_name')
```
- Return type of require() function depends on what the particular module returns.
- **Example:** http, fs, path, util and url modules are core modules.



# NODE.JS LOCAL MODULES

- Local modules are created by user locally in Node.js application.
- It should be included in program in the same way as core module.
- Example:** Sum module performs addition of two numbers
- Syntax:** To create and export local module

## Sum.js – Name of Module

```
exports.add=function(n,m){  
  return n+m;  
};
```

```
exports.<function_name>  
= function (argument_1, argument_2,  
.. argument_N)  
{ /** function body */ };
```

- Exports keyword is used to make properties and methods available outside the file.



# NODE.JS LOCAL MODULES

- In order to include the add function in our index.js file we use the require function

- **Syntax:** To include local module  
**index.js file**

```
let sum = require('./sum')
```

```
console.log("Sum of 10 and 20 is ", sum.add(10, 20))
```

**Note:** The ***module.exports*** is a special object which is included in every JS file in the Node.js application by default.

Use **module.exports** or **exports** to expose a function, object or variable as a module in Node.js.



# NODE.JS THIRD PARTY MODULE

- Modules are available online and they are installed using the Node Package Manager (NPM) are called third party modules.
- **Examples:** express, mongoose, angular, react etc.



# NODE PACKAGE MANAGER (NPM)

- **Node Package Manager (NPM)** is a library and registry for JavaScript software packages. NPM has command-line tools to help install the different packages and manage their dependencies.
- Official website: <https://www.npmjs.com>
- In September 2022 over 2.1 million packages were reported being listed in the npm registry, making it the biggest single language code repository.
- NPM helps:
  - Install a Node.js module
  - Extend a Node.js module
  - Publish a Node.js module to repositories like Github





# NODE PACKAGE MANAGER (NPM)

- `npm install -g module_name` // Install Globally
- `npm install --save module_name` //Install and save in package.json

## Example:

- `npm install --save express` //Install express module
- `npm install --save express mongoos` //Install multiple modules at once



# CORE MODULES

- The core modules include minimum functionalities of Node.js.
- It is compiled into its binary distribution and load automatically when Node.js process starts.
- To load/include this module in program, use the **require** function.  
`let module = require('module_name')`



# CORE MODULES

Core Module	Description
<b>http</b>	http module includes classes, methods and events to create Node.js http server.
<b>url</b>	url module includes methods for URL resolution and parsing.
<b>querystring</b>	querystring module includes methods to deal with query string.
<b>path</b>	path module includes methods to deal with file paths.
<b>fs</b>	fs module includes classes, methods, and events to work with file I/O.
<b>util</b>	util module includes utility functions useful for programmers.



# CORE MODULES: HTTP

- The HTTP module can create HTTP server that listens to server ports and gives a response back to the client.
- HTTP module allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

- **Example:**

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

```
//create a server object:
```

```
http.createServer(function (req, res) {
```

```
res.write('Hello World!'); //write a response to the client
```

```
res.end(); //end the response
```

```
}).listen(8080); //the server object listens on port 8080
```



# CORE MODULES: HTTP

## Add an HTTP Header

- If the response from the HTTP server is supposed to be displayed as HTML then include an HTTP header with the correct content type.

- **Example:**

```
var http = require('http');  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.write('Hello World!');  
  res.end();  
}).listen(8080);
```



# CORE MODULES: HTTP

## Status-Code:

- It is a 3-digit integer where the first digit of the Status-Code defines the class of response and last two digits do not have any categorization role.
- There are 5 values for the first digit:

Code and Description	
1xx: Informational	It means the request has been received and the process is continuing.
2xx: Success	It means the action was successfully received, understood and accepted.
3xx: Redirection	It means further action must be taken in order to complete the request.
4xx: Client Error	It means the request contains incorrect syntax or cannot be fulfilled.
5xx: Server Error	It means the server failed to fulfill an apparently valid request.

# CORE MODULES: HTTP

## Read the Query String:

- The function passed into the `http.createServer()` has `req` argument that represents the request from the client as an object (`http.IncomingMessage` object).
- This object has a property called `"url"` which holds the part of the url that comes after the domain name.

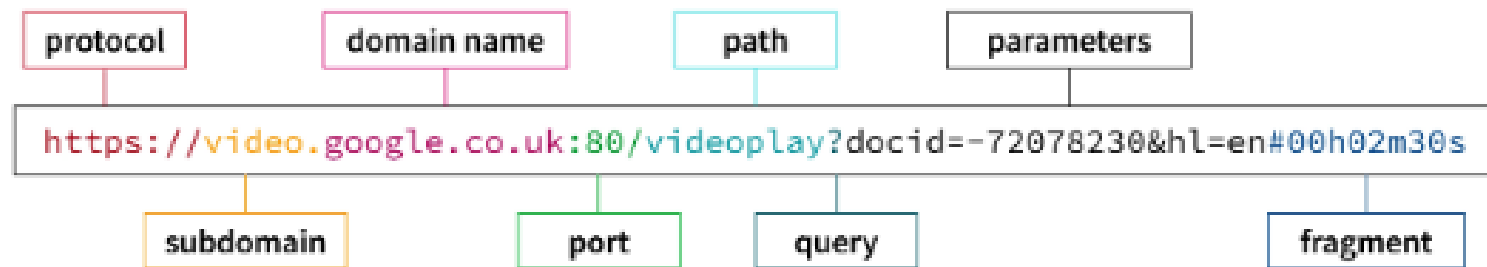
- **Example:**

```
var http = require('http');  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.write(req.url);  
  res.end();  
}).listen(8080);
```



# CORE MODULES: URL

- URL module provides utilities for URL resolution and parsing.





# CORE MODULES: URL

The **url.parse()** method takes a URL string, parses it and return URL object with each part of the address as properties of URL object.

**Syntax:** `url.parse( urlString, parseQueryString, slashesDenoteHost)`

**Return Value:** The `url.parse()` method returns an object with each part of the address like protocol, hostname, pathname, query, href



# CORE MODULES: URL

**Parameters:** This method accepts three parameters:

- 1) **urlString:** It holds the URL string which need to parse.
- 2) **parseQueryString:** It is a boolean value. If it set to true then the query property will be set to an object returned by the querystring module's parse() method. If it set to false then the query property on the returned URL object will be an unparsed, undecoded string. Its **default value is false.**
- 3) **slashesDenoteHost:** It is a boolean value. If it set to true then the first token after the literal string // and preceding the next / will be interpreted as the host.

eg <https://www.ganpatuniversity.ac.in/exams/cbcs-regulations>

result {host: 'www.ganpatuniversity.ac.in, pathname: '/exams/cbcs-regulations'} rather than {pathname: '/www.ganpatuniversity.ac.in/exams/cbcs-regulations'}.

Its **default value is false.**



# CORE MODULES: URL

Property	Description
.href	Provides us the complete url string
.host	Gives us host name and port number
.hostname	Hostname in the url
.path	Gives us path name of the url
.pathname	Provides host name , port and pathname
.port	Gives us port number specified in url
.auth	Authorization part of url
.protocol	Protocol used for the request
.search	Returns query string attached with url



# CORE MODULES: URL

## Example:

```
var http = require('http');  
var url = require('url');
```

```
http.createServer(function (req, res) {  
    var components = url.parse(req.url, true);  
    console.log(components);  
}).listen(4200);
```



# CORE MODULES: Fs

- FS (File System) in-built module to handle file operations like creating, reading, writing, deleting, renaming file.
- It gives the functionality of file I/O by providing wrappers around the standard POSIX(IEEE STANDARD) functions.
- All file system operations can have synchronous and asynchronous forms depending upon user requirements.



# CORE MODULES: Fs

**Synchronous approach:** called **blocking functions**

- It waits for each operation to complete, only after that, it executes the next operation, hence blocking the next command from execution

**Asynchronous approach:** called **non-blocking functions**

- It never waits for each operation to complete, rather it executes all operations in the first go itself. The result of each operation will be handled once the result is available




# CORE MODULES: Fs - READING FILE

- fs.readFile() method is used to read the file asynchronously.
- **Syntax:** fs.readFile (fileName ,[options], callback)
- **Parameter:**
  - filename: Full path and name of the file as a string.
  - options: The options parameter can be an object or string which can include encoding and flag. The default encoding is utf8 and default flag is "r". This field is optional.
  - callback: A function with two parameters error and data. This will get called when readFile operation completes.
- Example



# CORE MODULES: Fs - WRITING FILE

- fs.writeFile() method is used to write data to a file. If file already exists then it overwrites the existing content otherwise it creates a new file and writes data into it.
  - **Syntax:** fs.writeFile(filename, data[, options], callback)
  - **Parameter:**
    - filename: Full path and name of the file as a string.
    - data: The content to be written in a file.
    - options: The options parameter can be an object or string which can include encoding, mode and flag. The default encoding is utf8 and default flag is "r".
    - callback: A function with parameter error. This will get called when write operation completes.
  - Example
- 



# CORE MODULES: Fs - APPENDING FILE

- fs.appendFile() method is used to asynchronously append the given data to a file. A new file is created if it does not exist.
- **Syntax:** fs.appendFile( path, data[, options], callback )
- **Parameters:**
  - path: It is a String, Buffer, URL or number that denotes the source filename or file descriptor that will be appended to.
  - data: It is a String or Buffer that denotes the data that has to be appended.
  - options: It is an string or an object that can be used to specify optional parameters that will affect the output. It has three optional parameters: encoding, mode, flag
  - callback: It is a function that would be called when the method is executed.

# CORE MODULES: Fs - DELETING FILE

- fs.unlink() method to delete an existing file.

- **Syntax:** fs.unlink(path, callback);

- **Example: Delete file**

```
var fs = require('fs');  
fs.unlink('test.txt', function () {  
  console.log('File deleted successfully..');  
});
```



# CORE MODULES: Fs - RENAMING FILE

- A file can be renamed using fs.rename() method.
- Parameters of fs.rename() are old file name, new file name and a callback function.

- **Example:**

```
fs.rename('hello1.txt', 'hello2.txt', function(err)
{
    if (err)
        console.log(err);
    else
        console.log('File renamed');
}
```



# CORE MODULES: PATH

- Node.js provides path module which allows to interact with file paths easily.
- The path module has many useful properties and methods to access and manipulate paths in the file system.

## Properties:

- path.sep - represents the platform-specific path separator.  
It returns \ on Windows and / on Linux and macOS.
- path.delimiter - represents the path delimiter.  
It returns ; on Windows and : on Linux and macOS.

# CORE MODULES: PATH

## Methods:

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

`path.dirname(path)`

`path.extname(path)`

`path.format(pathObj)`

`path.isAbsolute(path)`

`path.join(...path)`

`path.normalize(path)`

`path.parse(path)`

`path.relative(from, to)`

`path.resolve(...path)`



# CORE MODULES: UTIL

- The node.js "util" module provides some functions to print formatted strings as well as some 'utility' functions are helpful for debugging purposes.
- Use *require('util')* to access these functions.



# CORE MODULES: UTIL

## Methods:

[util.format\(format, \[...\]\)](#)

[util.debug\(string\)](#)

[util.error\(\[...\]\)](#)

[util.puts\(\[...\]\)](#)

[util.print\(\[...\]\)](#)

[util.log\(string\)](#)

[util.inspect\(object, \[options\]\)](#)

[Customizing util.inspect colors](#)

[util.isArray\(object\)](#)

[util.isRegExp\(object\)](#)

[util.isDate\(object\)](#)

[util.isError\(object\)](#)

[util.inherits\(constructor, superConstructor\)](#)



# CORE MODULES: OS

- OS module in Node.js establishes interaction between the operating system and the application.
- The OS module consists of certain properties and methods which help in performing system-related activities.
- The OS module provides several information about the Operating System to Node.js like hostname, free (unused) memory, total memory, operating system name, user information, etc.





# CORE MODULES: OS

## Properties:

- **os.EOL** - end of the line marker
- **os.constants** - constants for process signals, error codes

## Methods:

- **os.type()** - name of the operating system
- **os.arch()** - architecture of the operating system
- **os.platform()** – operating system platform name
- **os.cpuinfo()** - information of the CPU like model, speed



# CORE MODULES: OS

## Methods:

- **os.userInfo()** - information about the current user
- **os.uptime()** - system uptime in seconds
- **os.hostname()** - hostname of the operating system
- **os.totalmem()** - total memory in bytes
- **os.freemem()** - unused memory in the hard disk of the operating system in bytes
- **os.networkInterfaces()** - information about the network interfaces like IP address, MAC address, netmask, etc..



Any query??

