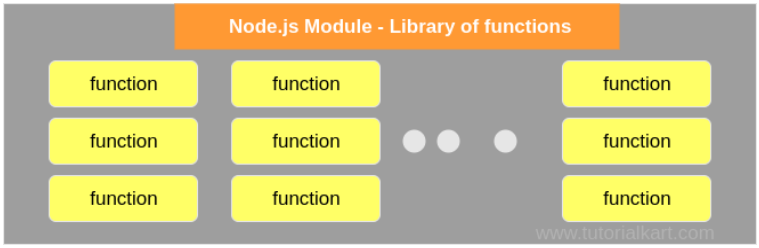
Node.js Modules

In the Node.js module system, each file is treated as a separate module. A Node.js Module is a library of functions that could be used in a Node.js file.



There are three types of modules in Node.js based on their location to access. They are :

1. **Built-in Modules/Core Modules:** These are the modules that come along with Node.js installation.
2. **User defined Modules:** These are the modules written by user or a third party.
3. **Third party Modules:** There are many modules available online which could be used in Node.js. Node Package Manager (NPM) helps to install those modules, extend them if necessary and publish them to repositories like Github for access to distributed machines.

    - Install a Node.js module using NPM
    - Extend a Node.js module
    - Publish a Node.js module to Github using NPM

---

1. Built-in Modules/Core Modules:

The core modules include bare minimum functionalities of Node.js. These core modules are compiled into its binary distribution and load automatically when Node.js process starts. However, you need to import the core module first in order to use it in your application.

The following table lists some of the important core modules in Node.js.

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

Loading Core Modules

In order to use Node.js core or NPM modules, you first need to import it ***using require() function*** as shown below.

```
var module = require('module_name');
```

As per above syntax, specify the module name in the require() function. The require() function will return an object, function, property or any other JavaScript type, depending on what the specified module returns.

The ***following example demonstrates how to use Node.js http module*** to create a web server.

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

var server = http.createServer(function(req, res){

  console.log('http demo')
  res.end('hello world')

});

server.listen(5000);
```

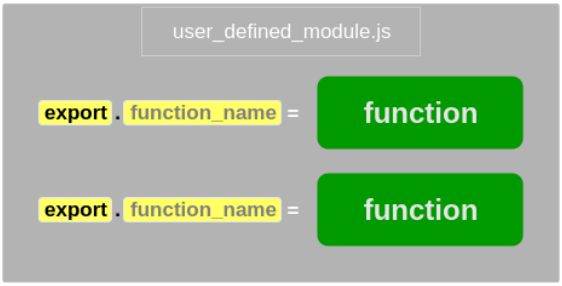
In the above example, require() function returns an object because http module returns its functionality as an object, you can then use its properties and methods using dot notation e.g. http.createServer().

---

2. User defined Modules

- How to Create a Node.js Module

Most of the necessary functions are included in the Built-in Modules. Sometimes it is required that, when you are implementing a Node.js application for a use case, you might want to keep your business logic separately. In such cases you create a Node.js module with all the required functions in it.



Create a Node.js Module

A Node.js Module is a .js file with one or more functions.

Following is the syntax to define a function in Node.js module :

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

exports – It is a keyword which tells Node.js that the function is available outside the module.

Example:

circle.js

```
const {PI}=Math
exports.f1= function (r)
{
   return PI * r ** 2;
}

exports.datetime= function()
{
   return Date();
}

var x1=function()
{
   return 4 * 5
}
exports.s=x1;

exports.multiplyes6 = (a,b) =>
{
   console.log('from arrow test');
```

foo.js

```
var app = require('./circle.js');

console.log(`circle area of 4 is ${app.f1(4)}`);

console.log("date is " +app.datetime());

console.log('Multiplication '+app.s())

console.log('arrow function '
+app.multiplyes6(5,3));
```

return a * b
};
console.log('from circle '+x1());	

Exports an object:

Ex:

log.js

```
var log = {
    info: function (info) {
        console.log('Info: ' + info);
    },
    warning:function (warning) {
        console.log('Warning: ' + warning);
    },
    error:function (error) {
        console.log('Error: ' + error);
    }
};

var str="hello world";

exports.str=str;

exports.sr = log
```

app.js

```
var myLogModule = require('./log');

myLogModule.sr.info('Node.js started');

console.log(myLogModule.str);
```

Output:

H:\node\myexamples>node app.js

Info: Node.js started

hello world

Here, *var* log is an object. Info, warning and error are function. 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.

---

3. **Third party Modules:**

- The 3rd party modules can be downloaded using NPM (Node Package Manager).
- 3rd party modules can be install inside the project folder or globally.
  Load and Use Third Party Module with Example:

3rd party modules can be downloaded using NPM in following way.

```
1. //Syntax
2.
3. npm install -g module_name // Install Globally
4.
5. npm install --save module_name //Install and save in package.json
6.
7.
8.
9. //Install express module
10.
11. npm install --save express
12.
13. npm install --save mongoose
14.
15.
16.
17. //Install multiple module at once
18.
19. npm install --save express mongoos
```

HTTP Module

Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

Node.js as a Web Server

The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.

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
```

The function passed into the http.createServer() method, will be executed when someone tries to access the computer on port 8080. We are using the server.listen function to make our server application listen to client requests on port no 8080. You can specify any available port over here.

Add an HTTP Header

If the response from the HTTP server is supposed to be displayed as HTML, you should 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);
```

The first argument of the res.writeHead() method is the status code, 200 means that all is OK, the second argument is an object containing the response headers.

Status-Code

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

S.N.	Code and Description

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

HTTP status codes are extensible and HTTP applications are not required to understand the meaning of all the registered status codes. Given below is a list of all the status codes.

Read the Query String

The function passed into the http.createServer() has a 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:

demo_http_url.js

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

Save the code above in a file called "demo_http_url.js" and initiate the file:

Initiate demo_http_url.js:

C:\Users\BNJ>node demo_http_url.js

If you have followed the same steps on your computer, you should see two different results when opening these two addresses:

http://localhost:8080/UVPCE

Will produce this result:

/UVPCE

http://localhost:8080/Ganpatuniversity

Will produce this result:
/ Ganpatuniversity

Split the Query String

There are built-in modules to easily split the query string into readable parts, such as the URL module.

Example

Split the query string into readable parts:

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

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  var q = url.parse(req.url, true).query;
  var txt = q.year + " " + q.month;
  res.end(txt);
}).listen(8080);
```

Save the code above in a file called "demo_querystring.js" and initiate the file:

C:\Users\Your Name>node demo_querystring.js

The address:

http://localhost:8080/?year=2020&month=July

Will produce this result:
2020 July

Note:

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

Syntax:

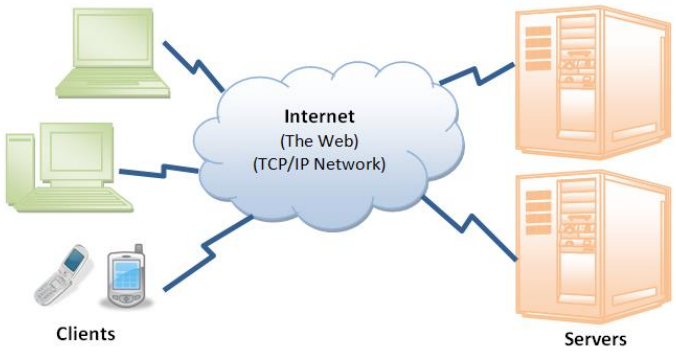
```
url.parse( urlString, parseQueryString, slashesDenoteHost)
```

***Parameters:** This method accepts three parameters a smentioned above and described below:*

- ***urlString:** It holds the URL string which need to parse.*
- ***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.*
- ***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. For example: // https://www.ganpatuniversity.ac.in/exams/cbcs-regulations contains the 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.*

***Return Value:** The **url.parse()** method returns an object with each part of the address*

HTTP (HyperText Transfer Protocol)



As mentioned, whenever you enter a URL in the address box of the browser, the browser translates the URL into a request message according to the specified protocol; and sends the request message to the server.

For example, the browser translated the URL `http://www.nowhere123.com/doc/index.html` into the following request message:

```
GET /docs/index.html HTTP/1.1
Host: www.nowhere123.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
```

When this request message reaches the server, the server can take either one of these actions:

1. The server interprets the request received, maps the request into a *file* under the server's document directory, and returns the file requested to the client.
2. The server interprets the request received, maps the request into a *program* kept in the server, executes the program, and returns the output of the program to the client.
3. The request cannot be satisfied, the server returns an error message.

An example of the HTTP response message is as shown:

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

The browser receives the response message, interprets the message and displays the contents of the message on the browser's window according to the media type of the response (as in the Content-Type response header). Common media type include "`text/plain`", "`text/html`", "`image/gif`", "`image/jpeg`", "`audio/mpeg`", "`video/mpeg`", "`application/msword`", and "`application/pdf`".

In its idling state, an HTTP server does nothing but listening to the IP address(es) and port(s) specified in the configuration for incoming request. When a request arrives, the server analyzes the message header, applies rules specified in the configuration, and takes the appropriate action. The webmaster's main control over the action of web server is via the configuration, which will be dealt with in greater details in the later sections.

File System: To handle file operations like creating, reading, deleting, etc., Node.js provides an inbuilt module called FS (File System). Node.js 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.

To use this File System module, use the require() method:

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

Common use for File System module:

Read Files
Write Files
Append Files
Close Files
Delete Files

What is Synchronous and Asynchronous approach?

Synchronous approach: They are called blocking functions as it waits for each operation to complete, only after that, it executes the next operation, hence blocking the next command from execution i.e. a command will not be executed until & unless the query has finished executing to get all the result from previous commands.

Asynchronous approach: They are called non-blocking functions as 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 i.e. each command will be executed soon after the execution of the previous command. While the previous command runs in the background and loads the result once it is finished processing the data.

Use cases:

If your operations are not doing very heavy lifting like querying huge data from DB then go ahead with Synchronous way otherwise asynchronous way.

In an Asynchronous way, you can show some progress indicator to the user while in the background you can continue with your heavyweight works. This is an ideal scenario for GUI based apps.

Reading File

Use fs.readFile() method to read the physical file asynchronously.

Syntax:

```
fs.readFile (fileName [,options], callback)
```

Parameter Description:

- 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 err and data. This will get called when readFile operation completes.

Example:

```
var fs = require("fs");

// Asynchronous read
fs.readFile('input.txt', function (err, data) {
   if (err) {
      return console.error(err);
   }
   console.log("Asynchronous read: " + data.toString());
});
```

Writing File

Use fs.writeFile() method 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 Description:

- 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 two parameters err and fd. This will get called when write operation completes.

The following example creates a new file called test.txt and writes "Hello World" into it asynchronously.

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

fs.writeFile('test.txt', 'Hello World!', function (err) {
                        if (err)
        console.log(err);
                        else
        console.log('Write operation complete.');
});
```

In the same way, use fs.appendFile() method to append the content to an existing file.

Example: Append File Content

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

fs.appendFile('test.txt', 'Hello World!', function (err) {
      if (err)
        console.log(err);
      else
        console.log('Append operation complete.');
});
```

Delete File

Use fs.unlink() method to delete an existing file.
Syntax :
fs.unlink(path, callback);

The following example deletes an existing file.

Example: Delete file

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

fs.unlink('test.txt', function () {

   console.log('Deleted successfully..');

});
```

Renaming a File

A file can be renamed using fs.rename(). Here, it takes old file name, new file name, and a callback function as arguments.

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

  console.log('Rename complete!');
});
```

For career objective

What Is LeetCode?

It's a website where people–mostly software engineers–practice their coding skills. There are 800+ questions (and growing), each with multiple solutions. Questions are ranked by level of difficulty: easy, medium, and hard.

Similar websites include **HackerRank, Topcoder, InterviewBit**, among others. There's also a popular book, "Cracking the Coding Interview," which some call the Bible for engineers. The Blind community uses a mix of these resources, but based on mentions, LeetCode seems to be the most popular. Our active users cite the following reasons for preferring LeetCode: more questions, better quality, plus a strong user base.