What is Node.js?

- Node.js is an open-source server side runtime environment built on Chrome's V8 JavaScript engine. It provides an event driven, non-blocking (asynchronous) I/O and cross-platform runtime environment for building highly scalable server-side application using JavaScript.
- Node.js can be used to build different types of applications such as command line application, web application, real-time chat application, REST API server etc. However, it is mainly used to build network programs like web servers, similar to PHP, Java, or ASP.NET.
- Node.js was written and introduced by Ryan Dahl in 2009
- Node.js official web site: https://nodejs.org
- Node.js on github: https://github.com/nodejs/node
- Node.js community conference http://nodeconf.com

Advantages of Node.js

- Node.js is an open-source framework under MIT license. (MIT license is a free software license originating at the Massachusetts Institute of Technology (MIT).)
- Uses JavaScript to build entire server side application.
- Lightweight framework that includes bare minimum modules. Other modules can be included as per the need of an application.
- Asynchronous by default. So it performs faster than other frameworks.
- Cross-platform framework that runs on Windows, MAC or Linux

Setup Node.js Development Environment

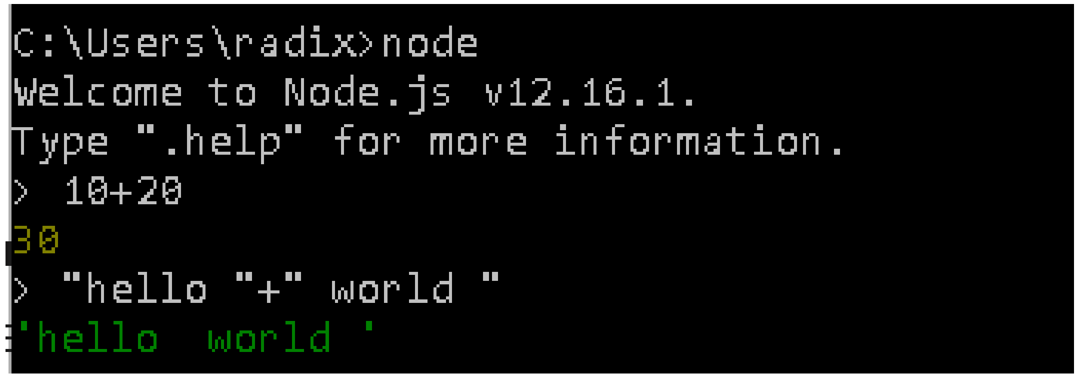
- Node.js development environment can be setup in Windows, Mac, Linux and Solaris. The following tools/SDK are required for developing a Node.js application on any platform.
- Node.js
- Node Package Manager (NPM)
- IDE (Integrated Development Environment) or TextEditor

Node JS Console Command

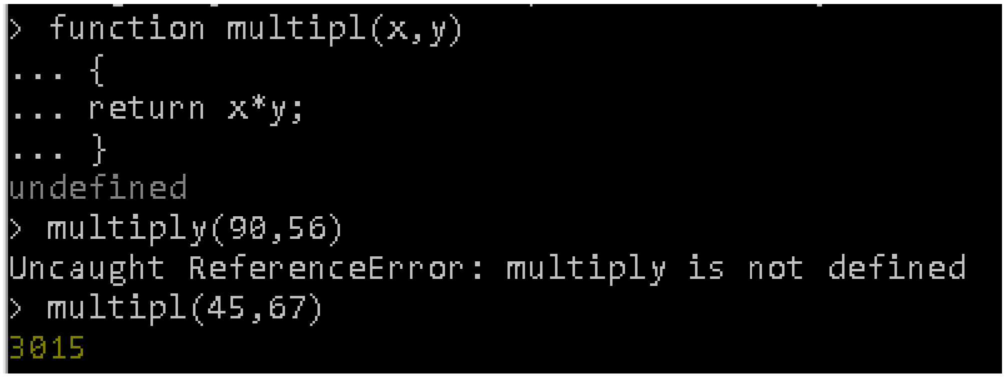
help	**Display help on all the commands**
tab Keys	Display the list of all commands.
Up/Down Keys	See previous commands applied in REPL.
.save filename	Save current Node REPL session to a file.
.load filename	Load the specified file in the current Node REPL session.
ctrl + c	Terminate the current command.
ctrl + c (twice)	Exit from the REPL.
ctrl + d	Exit from the REPL.
.break	Exit from multiline expression.
.clear	Exit from multiline expression.

Node.js Console

- Node.js comes with virtual environment called REPL (aka Node shell). REPL stands for Read-Eval-Print-Loop. It is a quick and easy way to test simple Node.js/JavaScript code.
- To Launch REPL node shell, open command Prompt and type: node



```
C:\Users\radix>node
Welcome to Node.js v12.16.1.
Type ".help" for more information.
> 10+20
30
> "hello "+" world "
'hello  world '
```



```
> function multipl(x,y)
... {
... return x*y;
... }
undefined
> multiply(90,56)
Uncaught ReferenceError: multiply is not defined
> multipl(45,67)
3015
```

Node.js Module

- Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application.
- Each module in Node.js has its own context, so it cannot interfere with other modules or pollute global scope. Also, each module can be placed in a separate .js file under a separate folder.

Node.js Basics

Node.js includes following primitive types:

- String
- Number
- Boolean
- Undefined
- Null
- RegExp

Loose Typing
JavaScript in Node.js supports loose typing like the browser's JavaScript. Use var keyword to declare a variable of any type.

Object Literal

```
var obj=
{
    name:"Roy",
    age:30
}
```

Buffer

Node.js includes an additional data type called Buffer (not available in browser's JavaScript). Buffer is mainly used to store binary data, while reading from a file or receiving packets over the network.

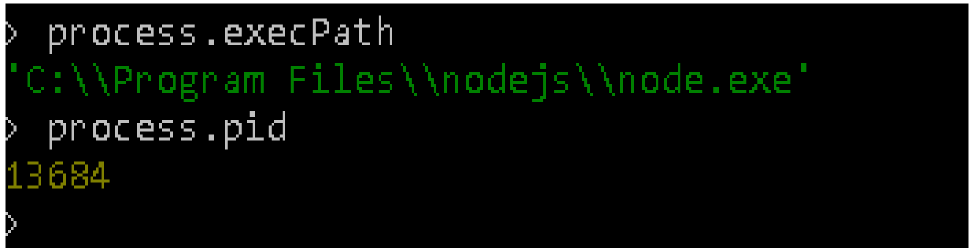
Functions

```
function Display(x) {
    console.log(x);
}

Display(100);
```

Process

Each Node.js script runs in a process. It includes **process** object to get all the information about the current process of Node.js application.



```
> process.execPath
'C:\\Program Files\\nodejs\\node.exe'
> process.pid
13684
>
```

Defaults to local

- Node's JavaScript is different from browser's JavaScript when it comes to global scope. In the browser's JavaScript, variables declared without var keyword become global. In Node.js, everything becomes local by default.

Accessing Global Scope

- In a browser, global scope is the window object. In Node.js, **global** object represents the global scope.
- To add something in global scope, you need to export it using export or module.export. The same way, import modules/object using require() function to access it from the global scope.

- exports.name = object.

Example

```
exports.calc=
{
    add:function(a,b)
    {
        c=a+b;
        console.log(c);
    },
    sub:function(a,b)
    {
        c=a-b;
        console.log(c);
    }
}
```

Now, you can import log object using require() function and use it anywhere in your Node.js project.

Types of Module

- Core module: Modules that come shipped with Node.js, e.g. https, os, fs, net, etc.
- Third-party module: Modules that you install from any package manager. We use these modules to accomplish or simplify any existing task. For example, to simplify our web API development we use express, or to deal with date and time we use moment.
- Local module: These are the modules that we create for our own use. These modules basically consist of core business logic of our code.

Core Module

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.

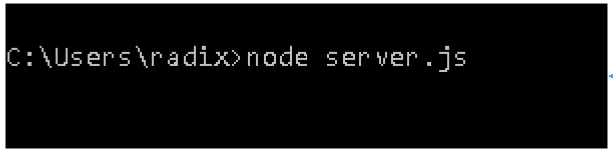
http

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

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

```
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
```



```
C:\Users\radix>node server.js
```

Run code

http://localhost:8080

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:

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

Read the Query String

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

The Built-in URL Module

- The URL module splits up a web address into readable parts.

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

Parse an address with the url.parse() method, and it will return a URL object with each part of the address as properties:

```
var url = require('url');
var adr = 'http://localhost:8080/default.htm?year=2020&month=february';
var q = url.parse(adr, true);

console.log(q.host); //returns 'localhost:8080'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?year=2020&month=february'

var qdata = q.query; //returns an object: { year: 2020, month:
'february' }
console.log(qdata.month); //returns 'february'
```

example

```
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);
```

https://nodejs.org/api/url.html#url_url_host

QueryString

```
var querystring = require('querystring');
var q = querystring.parse('year=2020&month=february');
console.log(q.year);
```

Method	**Description**
escape()	Returns an escaped querystring
parse()	Parses the querystring and returns an object
stringify()	Stringifies an object, and returns a query string
unescape()	Returns an unescaped query string

https://nodejs.org/api/querystring.html#querystring_querystring_parse_str_sep_eq_options

Path

The Path module provides a way of working with directories and file paths.

```
var path = require('path');
var filename = path.basename('/Users/myapp/demo_path.js');
console.log(filename);
```

Path

Method	Description
basename()	Returns the last part of a path
delimiter	Returns the delimiter specified for the platform
dirname()	Returns the directories of a path
extname()	Returns the file extension of a path
format()	Formats a path object into a path string
isAbsolute()	Returns true if a path is an absolute path, otherwise false
join()	Joins the specified paths into one
normalize()	Normalizes the specified path
parse()	Formats a path string into a path object
posix	Returns an object containing POSIX specific properties and methods
relative()	Returns the relative path from one specified path to another specified path
resolve()	Resolves the specified paths into an absolute path
sep	Returns the segment separator specified for the platform
win32	Returns an object containing Windows specific properties and methods

https://nodejs.org/api/path.html

File System

- The Node.js file system module allows you to work with the file system on your computer.
- To include the File System module, use the require() method.

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

Common use for the File System module:

- Read files
- Create files
- Update files
- Delete files
- Rename files

readFile

```
fs.readFile("./db.json",function(err,data)
{
    console.log(JSON.parse(data.toString()).tas
ks);
})
```

fs.open()

```
fs.open( filename, flags, mode, callback )
```

- **filename:** It holds the name of the file to read or the entire path if stored at other location.
- **flag:** The operation in which file has to be opened.
- **mode:** Sets the mode of file i.e. r-read, w-write, r+ -readwrite. It sets to default as readwrite.
- **callback:** It is a callback function that is called after reading a file. It takes two parameters:
    - **err:** If any error occurs.
    - **data:** Contents of the file. It is called after open operation is executed.

Flag

FLAG	DESCRIPTION
r	To open file to read and throws exception if file doesn’t exists.
r+	Open file to read and write. Throws exception if file doesn’t exists.
rs+	Open file in synchronous mode to read and write.
w	Open file for writing. File is created if it doesn’t exists.
wx	It is same as ‘w’ but fails if path exists.
w+	Open file to read and write. File is created if it doesn’t exists.
wx+	It is same as ‘w+’ but fails if path exists.
a	Open file to append. File is created if it doesn’t exists.
ax	It is same as ‘a’ but fails if path exists.
a+	Open file for reading and appending. File is created if it doesn’t exists.
ax+	It is same as ‘a+’ but fails if path exists.

Open Method

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

console.log("Open file!");

// To open file in write and read mode,
// create file if doesn't exists.
fs.open('demo.txt', 'w+', function (err, f) {
   if (err) {
      return console.error(err);
   }
   console.log(f);
   console.log("File opened!!");
});
```

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

fs.appendFile('mynewfile1.txt', 'Hello
content!', function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

The fs.writeFile() method replaces the specified file and content if it exists. If the file does not exist, a new file, containing the specified content, will be created:

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

fs.writeFile('mynewfile3.txt', 'Hello
content!', function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

Delete the File

File System Link

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

fs.unlink('mynewfile2.txt', function (err) {
  if (err) throw err;
  console.log('File deleted!');
});
```

Rename Files

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

fs.rename('mynewfile1.txt', 'myrenamedfile.txt'
, function (err) {
  if (err) throw err;
  console.log('File Renamed!');
});
```

Util Module

- The Util module provides access to some utility functions.

```
var util = require('util');
var txt = 'Congratulate %s on his %dth birthday!';
var result = util.format(txt, minal', 30);

console.log(result);
```

output

```
Congratulate minal on his 30th birthday!
```

Util Method

Util usefull link

Method	Description
debuglog()	Writes debug messages to the error object
deprecate()	Marks the specified function as deprecated
format()	Formats the specified string, using the specified arguments
inherits()	Inherits methods from one function into another
inspect()	Inspects the specified object and returns the object as a string

Node.js Local Module

- Local modules are modules created locally in your Node.js application. These modules include different functionalities of your application in separate files and folders. You can also package it and distribute it via NPM, so that Node.js community can use it.

Writing Simple Module

```
var calc = {
    add: function (a,b) {
        c=a+b;
        console.log('Addition is : ' + c);
    },
    sub:function (a,b) {
        console.log('Substraction is: ' + a-b);
    },
    multi:function (a,b) {
        console.log('Multiplication is : ' + a*b);
    }
};
module.exports = calc
```

Loading Local Module

```
var calc=require("./calc");

console.log( calc.add(12,45));
```

Event Module

- Node.js has a built-in module, called "Events", where you can create-, fire-, and listen for- your own events.
- You can assign event handlers to your own events with the EventEmitter object.
- In the example below we have created a function that will be executed when a "scream" event is fired.
- To fire an event, use the emit() method.

Example

```
eventEmitter.on("Someonerequested",function(data)
{
  fs.readFile('./db.json',function(err,data)
  {
    console.log(JSON.parse(data.toString()).tasks);
  });
  console.log("Request has been made"+data);
})

eventEmitter.emit("Someonerequested","Hello all");
```

```
var fs = require('fs');
var rs = fs.createReadStream('./demofile.txt');
rs.on('open', function () {
  console.log('The file is open');
});
```

Node Package Manager

- Node Package Manager (NPM) is a command line tool that installs, updates or uninstalls Node.js packages in your application. It is also an online repository for open-source Node.js packages. The node community around the world creates useful modules and publishes them as packages in this repository
- Install Package Locally
- npm install <package name>
- Add Dependency into package.json
- npm install <package name> --save

Frameworks for Node.js

- Hapi.js
- Express.js
- Socket.io
- Koa.js
- Meteor.js
- Lad
- https://codebrahma.com/9-best-node-js-frameworks-developers/

Express.js

- Express.js is a web application framework for Node.js. It provides various features that make web application development fast and easy which otherwise takes more time using only Node.js.
- Express.js is based on the Node.js middleware module called ***connect*** which in turn uses **http** module. So, any middleware which is based on connect will also work with Express.js.