**Node JS Basics:**

**Data Types**

Like any other programming language, Node.js has various datatypes, which are further categorized into Primitive and Non-Primitive datatypes.

***Primitive Data Types are:***

1. String
2. Number
3. Boolean
4. Null
5. Undefined

***Non-Primitive Data Types are:***

1. Object
2. Date
3. Array

**Variables**

Variable are entities that hold values which may vary during the course of a program. To create a variable in Node.js, you need to make use of a reserved keyword var. You do not have to assign a data type, as the compiler will automatically pick it.

***Syntax:***

|  |  |
| --- | --- |
| 1 | var varName = value; |

**Operators**

Node.js supports the below operators:

|  |  |
| --- | --- |
| **Operator Type** | **Operators** |
| ***Arithmetic*** | +, -, /, \*,  %, ++, — |
| ***Assignment*** | =, +=, -=, \*=, /=, %= |
| ***Conditional*** | =? |
| ***Comparison*** | ==, ===, !=, !==, >, >=, <, <=, |
| ***Logical*** | &&, ||, ! |
| ***Bitwise*** | &, |, ^, ~, <<, >>, >>> |

**Functions**

Functions in Node.js is a block of code that has a name and is written to achieve a particular task. You need to use the keyword function to create it. A function is generally a two-step process. First is defining the function and the second is invoking it. Below is the syntax of creating and invoking a function:

***Example:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | //Defining a function  function display\_Name(firstName, lastName) {    alert("Hello " + firstName + " " + lastName);  }    //Invoking the function  display\_Name("Park", "Jimin"); |

**Objects**

An object is a non-primitive data type that can hold multiple values in terms of properties and methods. Node.js objects are standalone entities as there is no concept of class. You can create an object in two ways:

1. Using Object literal
2. Using Object constructor

***Example:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | // object with properties and method  var employee = {    //properties    firstName: "Minho",    lastName: "Choi",    age: 35,    salary:50000,    //method    getFullName: function () {      return this.firstName + ' ' + this.lastName    }  }; |

**File System**

To access the physical file system, Node.js makes use of the **fs** module which basically takes care of all asynchronous and synchronous file I/O operations. This module is imported using the below command:

|  |  |
| --- | --- |
| 1 | var fs = require('fs'); |

Some of the general use for the File System modules are:

* **Read files**
  1. fs.readFile()

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | var http = require('http');  var fs = require('fs');  http.createServer(function (req, res) {    fs.readFile('script.txt', function(err, data) {      res.writeHead(200, {'Content-Type': 'text/html'});      res.write(data);      res.end();    });  }).listen(8080); |

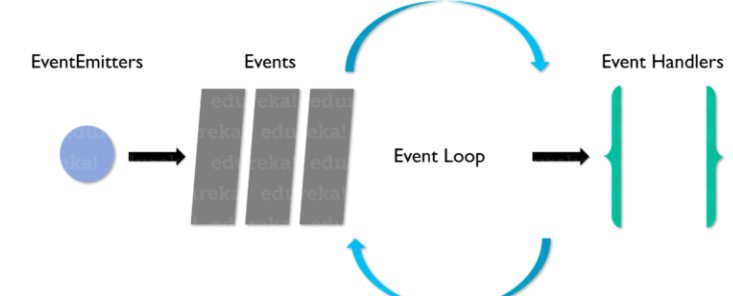
* **Create files**
  1. appendFile()
  2. open()
  3. writeFile()
* **Update files**
  1. fs.appendFile()
  2. fs.writeFile()
* **Delete files**
  1. fs.unlink()
* **Rename files**
  1. fs.rename()

## ****Events****

## Node.js applications are single threaded and event-driven. Node.js supports concurrency as it is event-driven, and thus makes use of concepts like events and callbacks. The async function calls help Node.js in maintaining concurrency throughout the application.

Basically, in a Node.js application, there is a main loop which waits and listens for events, and once any event is completed, it immediately initiates a callback function.

Below diagram represents how the events are driven in Node.js.



One thing that you must note here is that, even though events look similar to callback functions but the difference lies in their functionalities. When an asynchronous function returns its results callbacks are invoked on the other hand event handling completely works on the observer pattern. And in Node.js, methods which listen to the events are called the observers. The moment, an event is triggered, its listener function automatically starts executing. Event modules and EventEmitter class provide multiple in-built events which are used to bind events with event listeners.

**Binding Event to an Event Listener**

|  |  |
| --- | --- |
| 1  2  3  4 | // Import events module  var my\_Events = require('events');  // Create an eventEmitter object  var my\_EveEmitter = new my\_Events.EventEmitter(); |

**Binding Event Handler to an Event**

|  |  |
| --- | --- |
| 1  2 | // Binding event and event handler  my\_EveEmitter.on('eventName', eventHandler); |

**Firing an Event**

|  |  |
| --- | --- |
| 1  2 | // Fire an event  my\_EveEmitter.emit('eventName'); |

Now let’s try to implement the things that I have discussed in this Node.js Event section. The below code shows a simple representation of Event execution in Node.js.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | var emitter = require('events').EventEmitter;  function iterateProcessor(num) {    var emt = new emitter();    setTimeout(function () {          for (var i = 1; i <= num; i++) {            emt.emit('BeforeProcess', i);            console.log('Processing Iteration:' + i);            emt.emit('AfterProcess', i);          }        }        , 5000)    return emt;  }  var it = iterateProcessor(5);    it.on('BeforeProcess', function (info) {    console.log('Starting the process for ' + info);  });    it.on('AfterProcess', function (info) {    console.log('Finishing processing for ' + info); |

The most important module of Node.js called the HTTP Module.

## ****HTTP Module****

Generally, Node.js is used for developing server-based applications. But using the module, you can easily create web servers that can respond to the client requests. Thus it is also referred to Web Module and provides modules like HTTP and request that facilitate Node.js in processing the server requests.

You can easily include this module in your Node.js application just by writing the below code:

|  |  |
| --- | --- |
| 1 | var http = require('http'); |

How to develop a Web Server in Node.js.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | //calling http library  var http = require('http');  var url = require('url');    //creating server  var server = http.createServer(function (req, res) {    //setting content header    res.writeHead(200, ('Content-Type', 'text/html'));    var q = url.parse(req.url, true).query;    var txt = q.year + " " + q.month;    //send string to response    res.end(txt);  });    //assigning 8082 as server listening port  server.listen(8082); |

In the next section we will be talking about Express.js which is heavily used for server-side web application development.

## ****Express.js****

Express.js is a framework built on top of Node.js that facilitates the management of the flow of data between server and routes in the server-side applications.  It is a lightweight and flexible framework that provides a wide range of features required for the web as well as mobile application development.

Express.js is developed on the middleware module of Node.js called **connect**. The connect module further makes use of **http** module to communicate with Node.js. Thus, if you are working with any of the connect based middleware modules, then you can easily integrate with Express.js.

 Not, only this, few of the major advantages of Express.js are:

* Makes web application development faster
* Helps in building mobile and web application of single-page, multi-page, and hybrid types
* Express provides two templating engines namely, Jade and EJS
* Express follows the Model-View-Controller (MVC) architecture
* Makes integration with databases such as MongoDB, Redis, MySQL
* Defines an error handling middleware
* Simplifies configuration and customization easy for the application.

With all these features, Express takes responsibility of backend part in the MEAN stack. Mean Stack is the open-source JavaScript software stack that is used for building dynamic websites and web applications. Here, **MEAN**stands for **M**ongoDB, **E**xpress.js, **A**ngularJS, and **N**ode.js.

Lets now see a simple example to understand, how Express.js works with Node.js to ease our work. But before you start working with Express.js, you need to install it in your system.

To install Express.js globally you can use the below command:

|  |  |
| --- | --- |
| 1 | npm install -g express |

Or, if you want to install it locally into your project folder, you need to execute the below command:

|  |  |
| --- | --- |
| 1 | npm install express --save |

Since we are done with all the preparations, let’s now jump directly into practical implementation.