## ****Node.js Modules****

The modules in Node.js represents various functionalities that are bundled up into single or multiple JS files. These modules have a unique context, thus, they never interfere nor pollute the scope of other modules.

These modules enable the code reusability and enhance the ease of usage. Node.js basically provides three types of modules:

1. Core Modules
2. Local Modules
3. Third-Party Modules

### ****Core Module****

Since Node.js is a very **lightweight** framework, the core modules bundle the absolute minimum functionalities. These modules generally get loaded when the Node process starts its execution. All you need to do is, import these core modules in order to use them in your code.

Below are a few of the important Core modules.

|  |  |
| --- | --- |
| **Core Module** | **Description** |
| http | Contains classes, methods, and events required to create Node.js HTTP server |
| url | Contains methods for URL resolution and parsing in Node |
| querystring | Contains methods to deal with a query string of Node |
| path | Contains methods to deal with file paths |
| fs | Contains classes, methods, and events to work with file I/O |
| util | Contains utility functions that can be useful for programmers |

You can load your core module, using the below code:

|  |  |
| --- | --- |
| 1 | var module = require('module\_name'); |

Lets now see, what are ‘local modules’.

### ****Local Modules****

The local modules of Node.js are custom modules that are created locally by user/developer in the application. These modules can include various functionalities bundled into distinct files and folders which can be easily distributed in the Node.js community using NPM.

These modules are loaded in a similar way to core modules. Let show you, how to do it using a basic example.

Create your custom/local module.js file

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | var detail = {    name: function (name) {      console.log('Name: ' + name);    },    domain:function (domain) {      console.log('Domain: ' + domain);    }  };    module.exports = detail; |

Include your module file in your main application file.

|  |  |
| --- | --- |
| 1  2  3 | var myLogModule = require('./Local\_module.js');  myLogModule.name('Edureka');  myLogModule.domain('Education'); |

Now you can execute these files using below command:

|  |  |
| --- | --- |
| 1 | node application.js |

Let me now show you what are external modules.

### ****External Modules****

You can use the external or 3rd party modules only by downloading them via NPM. These modules are generally developed by other developers and are free to use. Few of the best external modules are express, react, gulp, mongoose, mocha etc.

Globally Loading the 3rd party modules:

|  |  |
| --- | --- |
| [1](https://www.edureka.co/nodejs-certification-training" \t "_blank) | [npm install --g <module\_name>](https://www.edureka.co/nodejs-certification-training" \t "_blank) |

*[Include your module file in your main application file:](https://www.edureka.co/nodejs-certification-training" \t "_blank)*

|  |  |
| --- | --- |
| [1](https://www.edureka.co/nodejs-certification-training" \t "_blank) | [npm install --save <module\_name>](https://www.edureka.co/nodejs-certification-training" \t "_blank) |

## ****JSON File****

The **package.json file** in Node.js is the heart of the entire application. It is basically the manifest file that contains the metadata of the project. Thus, understanding and working with this file becomes very important for a successful Node project development.

The package.json file generally consists of the metadata of the application, which is further categorized into below two categories:

1. **Identifying metadata properties:**This consists of properties like the project name, current module version, license, author of the project, project description etc.
2. **Writing directly to file:**You can directly write the necessary information into the package.json file and include it, in your project.