11.2 Getting started with Node.js

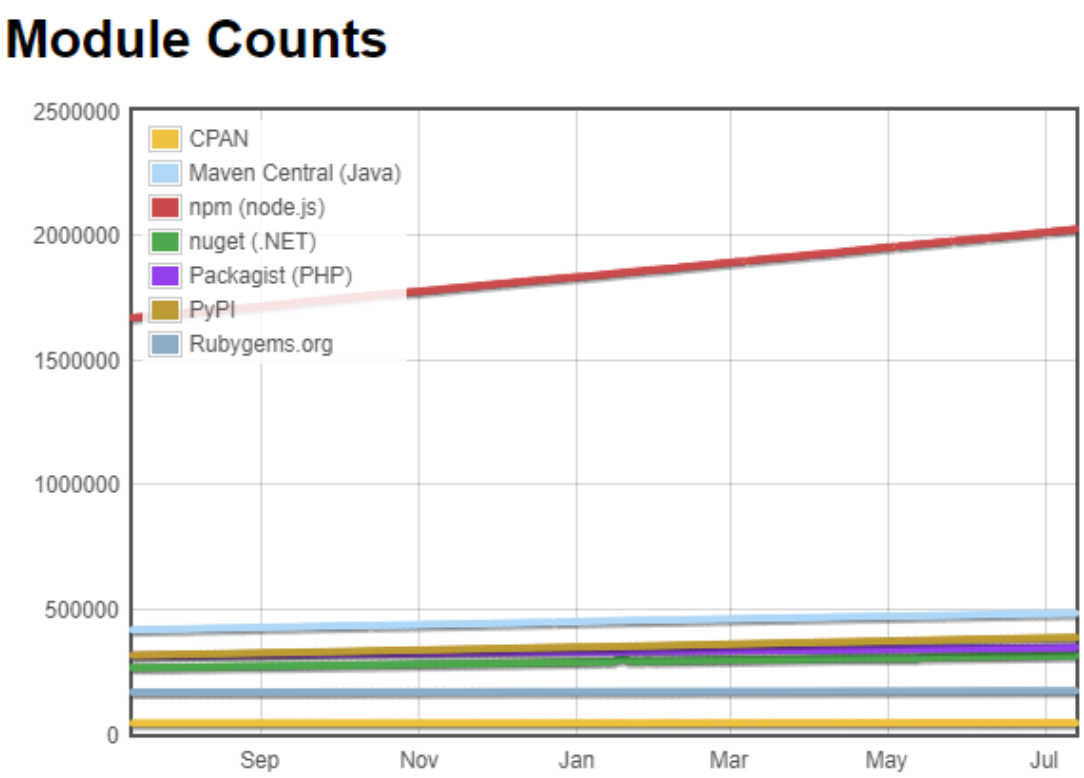
Introduction to Node.js

Node.js is a JavaScript runtime environment that is primarily used to run server-side web applications. Node.js has many benefits:

- The event-driven, non-blocking I/O architecture of Node.js allows Node.js to handle high loads.
- Node.js allows developers to write JavaScript on the server and client, simplifying some development tasks.
- Node.js provides a simple mechanism to create and distribute modules. A Node.js ***module*** is a JavaScript file that provides some useful functionality.
- Node.js works seamlessly with MongoDB, a document database that stores JSON and uses JavaScript as a query language. Web development is greatly simplified when JSON is used between the client and server, and between the server and database.

Companies using Node.js include Netflix, Walmart, Ebay, and LinkedIn. Adoption by these companies helped validate the Node.js approach and spur development of more Node.js modules.

Figure 11.2.1: Number of publicly accessible modules for Node.js (npm) in 2022 far exceeds other languages.



Source: ModuleCounts.com

Feedback?

PARTICIPATION ACTIVITY 11.2.1: Introduction to Node.js.

1) Node.js programs are written in JavaScript.
   - (•) True
   - ( ) False

Correct

Node.js uses Google's V8 JavaScript engine to run JavaScript programs. Some Node.js programs are written in other languages, like TypeScript, but must be converted into JavaScript to execute.

2) Node.js programs run in the web browser.
   - ( ) True
   - (•) False

Correct

Node.js programs run on the web server.

3) Node.js has over one million modules.
   - (•) True
   - ( ) False

Correct

More modules are available for Node.js than any other web development language.

Feedback?

Installing and running

Developers may install Node.js using installers from the Node.js website for Windows, macOS, and other operating systems.

After installing Node.js, a developer can start the Node.js interactive shell and execute JavaScript statements. The figure below shows a command line prompt from which the user started the Node.js interactive shell by entering "node". The ".exit" command exits the interactive shell.

Figure 11.2.2: Node.js interactive shell.

```
$ node
Welcome to Node.js
> console.log("Hello,
Node.js!")
Hello, Node.js!
undefined
> x = 2
2
> .exit
```

Feedback?

A developer may write a Node.js program in a text editor and execute the program using the "node myprogram.js" command.

Figure 11.2.3: Simple Node.js program.

```
// hello.js
for (let i = 0; i < 5; i++) {
   console.log("Hello,
Node.js!");
}
```

```
$ node hello.js
Hello, Node.js!
Hello, Node.js!
Hello, Node.js!
Hello, Node.js!
Hello, Node.js!
```

Feedback?

Online services

Online IDEs like Replit and StackBlitz allow developers to run Node.js programs in the cloud instead of installing Node.js on the developer's machine.

PARTICIPATION ACTIVITY 11.2.2: Running Node.js.

1) The Node.js command-line program only runs on Windows.
   - True
   - False (selected)

Correct

Node.js runs on many different operating systems. The "node" command-line program runs after Node.js is installed on the computer.

2) The command `node test.js` starts the Node.js interactive shell.
   - True
   - False (selected)

Correct

The command `node test.js` runs the test.js program. No program should be specified to start the interactive shell.

3) A Node.js program can display output to the console using the `console.log()` method call.
   - True (selected)
   - False

Correct

The figure above shows a Node.js program that outputs "Hello, Node.js!" using `console.log()`.

Feedback?

Creating a simple web server

The ***http module*** allows a Node.js application to create a simple web server. The http module's ***createServer()*** method creates a web server that can receive HTTP requests and send HTTP responses. The ***listen()*** method starts the server listening for HTTP requests on a particular port. The server continues to run until the developer enters Ctrl+C to kill the Node.js application.

The program below shows the http module being imported with `require()`. The ***require()*** function imports a module for use in a Node.js program.

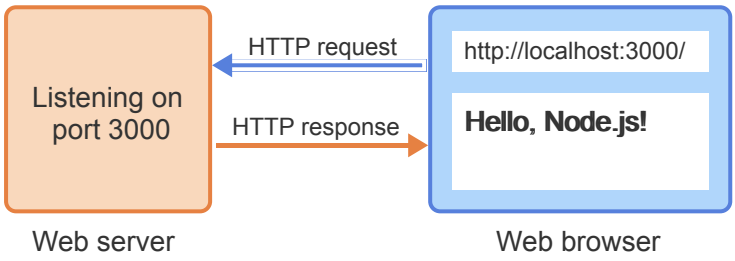
PARTICIPATION ACTIVITY 11.2.3: A simple Node.js web server.

1 2 3 4 5 6 7 8 9 2x speed

$ node server.js

```
// server.js
const http = require("http");

http.createServer(function(request, response) {
   response.writeHead(200, {"Content-Type": "text/html"});
   response.write("<h1>Hello, Node.js!</h1>");
   response.end();
}).listen(3000);
```



end() sends the HTTP response to the web browser, which renders the HTML.

Captions

1. server.js is executed from the command line.
2. The require("http") command imports the "http" module.
3. createServer() creates a web server object that calls the provided callback function when an HTTP request is received.
4. listen() starts the web server listening on port 3000 for HTTP requests.
5. The user enters a URL to access the web server running on the same machine on port 3000.
6. The HTTP request is routed to the web server, causing the request callback function to execute.
7. writeHead() creates an HTTP response with a 200 status code and text/html content type.
8. write() sends the HTML to the HTTP response object.
9. end() sends the HTTP response to the web browser, which renders the HTML.

Feedback?

PARTICIPATION ACTIVITY 11.2.4: Node.js web server.

1) What method causes the web server to

Correct

begin listening for HTTP requests?

- require()
- createServer()
- listen()

listen() starts the web server, after which the web server can respond to incoming requests.

2) What URL accesses a web server running locally and listening on port 8080?

- http://localhost/
- http://localhost:8080/
- http://8080/

Correct

localhost indicates the web server is running on the same machine as the web browser. ":8080" indicates the port to which HTTP requests are routed.

3) What method sets the status code for the HTTP response?

- response.writeHead()
- response.write()
- response.end()

Correct

writeHead() sets the status code and HTTP headers on the HTTP response object.

4) What keyboard command kills the web server program?

- Ctrl+C
- Ctrl+Z
- Ctrl+X

Correct

Ctrl+C kills a running Node.js program when entered in the console window executing the program. Developers frequently kill their program, revise code, and re-run the program.

Feedback?

Projects and npm

Node.js programs are typically organized into projects. A ***Node.js project*** is a collection of JavaScript files, packages, configuration files, and other miscellaneous files that are stored in a directory.

Figure 11.2.4: Example Node.js project with a single JavaScript file.

```
myproject
└──
server.js
```

Feedback?

A ***package*** is a directory containing one or more modules and a package.json file. A ***package.json*** file contains JSON that lists the package's name, version, license, dependencies, and other package metadata.

The ***Node Package Manager (npm)*** is the package manager for Node.js that allows developers to download, install, and update packaged modules. npm is installed with Node.js and is executed from the command line.

Figure 11.2.5: Display npm's version.

```
$ npm -v
8.7.0
```

Feedback?

npm can install packages in one of two modes:

- Local mode: Packages are installed in a `node_modules` directory in the parent working directory. Ex: `npm install mypackage`
- Global mode: Packages are installed in a `{prefix}/node_modules` directory, where `{prefix}` is a location set in npm's configuration. The `--global` flag (or `-g`) directs npm to install in global mode. Ex: `npm install mypackage --global`

Local mode is ideal for installing project dependencies. A ***dependency*** is a package that a Node.js project must be able to access to run. Global mode is typically for installing command-line tools.

Figure 11.2.6: Get npm's prefix directory where global packages are installed.

```
$ npm config get prefix
/usr/local
```

The prefix directory will be different for Windows users.

Feedback?

The figure below shows a developer installing the nodemon package globally. ***Nodemon*** is a utility that saves developers time by restarting a Node.js application whenever the files in a project are modified.

Figure 11.2.7: Installing and running nodemon.

```
$ npm install nodemon --global
...
+ nodemon@1.19.1
added 147 packages from 90 contributors in
41.265s

$ nodemon myproject/server.js
[nodemon] 1.9.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node myproject\server.js`
```

Feedback?

Underscore is a library of helpful functions that extends some built-in JavaScript objects. The figure below shows a developer changing to the `myproject` directory that stores a Node.js project, installing underscore as a local package, and producing a list of the project's local packages. The underscore module is stored in `myproject/node_modules/underscore`.

Figure 11.2.8: Installing "underscore" as a local package.

```
$ cd myproject
$ npm install underscore
...
+ underscore@1.9.1
added 1 package from 1 contributor and audited 1 package in
1.983s

$ npm list
/myproject
└── underscore@1.9.1
```

Feedback?

A module is imported and assigned to a variable with `require()`. *Good practice is to assign imported modules to variables that are named similar to the module name. Ex: Variable* ***`http`*** *for the "http" module.* However, the underscore module is usually assigned to the variable `_`, as shown in the figure below.

Figure 11.2.9: Using the underscore package to get random dice rolls.

```
const http = require("http");
const _ = require("underscore");

http.createServer(function(request, response) {
    response.writeHead(200, {"Content-Type": "text/html"});
    response.write("<!DOCTYPE html>\n<html>\n");
    response.write("<title>Dice Roll</title>\n");
    response.write("<body>\n");

    for (let i = 0; i < 5; i++) {
        // Use underscore to get a random number between 1 and 6
        let randNum = _.random(1, 6);

        response.write("<p>" + randNum + "</p>\n");
    }
    response.write("</body>\n</html>");
    response.end();
}).listen(3000);
```

```
<!DOCTYPE html>
<html>
<title>Dice Roll</title>
<body>
<p>3</p>
<p>5</p>
<p>5</p>
<p>6</p>
<p>1</p>
</body>
</html>
```

Feedback?

Table 11.2.1: Summary of npm commands.

Command	Description	Example
`config`	Manage npm configuration files	`npm config list` `npm config get prefix`
`install`	Install package locally or globally (-g)	`npm install nodemon -g`
`list`	List all installed local or global (-g) packages	`npm list`

Command	Description	Example
`update`	Update a local or global (-g) package	`npm update lodash`
`uninstall`	Uninstall a local or global (-g) package	`npm uninstall lodash`

Feedback?

PARTICIPATION ACTIVITY 11.2.5: Using npm.

1) Where does the npm command below install the grunt package?

```
$ npm install grunt -g
```

- The project's `node_modules` directory
- `{prefix}/node_modules` directory
- `{prefix}` directory

Correct

All modules installed with -g or --global are installed globally in the same directory.

2) Which command displays all the installed global npm packages?

- npm install --global
- npm list
- npm list --global

Correct

The list command with --global flag lists all globally installed packages.

3) Which command updates the local mkdirp package?

- npm install mkdirp
- npm update mkdirp
- npm update mkdirp -g

Correct

The update command replaces the existing module with the most up-to-date version.

4) Which command uninstalls the local mkdirp package?

- npm update mkdirp
- npm uninstall mkdirp -g
- npm uninstall mkdirp

Correct

The uninstall command removes the local package from the project's node_modules directory.

Feedback?

The package.json and package-lock.json files

Node.js projects use package.json to list information about the project, including the project's name, version, license, and package dependencies. Developers can manually create package.json or use the `npm init` command, which prompts the user to enter various fields and generates package.json automatically.

Figure 11.2.10: Example package.json file.

```
{
  "name": "my-web-server",
  "version": "1.0.0",
  "description": "A simple web server",
  "main": "server.js",
  "dependencies": {
    "underscore": "^1.9.1"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit
1",
    "start": "node server.js"
  },
  "author": "",
  "license": "ISC"
}
```

Feedback?

When a project's package.json file is present, all the project's dependencies can be installed with a single command: `npm install`.

A ***package-lock.json*** file is created or modified when project dependencies are added or removed. The file ensures that the same dependency versions are always used when the project is installed on different machines. Ex: A project's package.json file may indicate a

dependency version ^1.9.1. The caret character (^) means that npm should install the highest version of the library that exists, as long as the major version number, the number following the caret, is the same. So if version 1.9.2 is available, 1.9.2 is installed. But version 2.0.0 is not installed since ^1 requires the major version number to be 1. If the package-lock.json indicates that version 1.9.1 should be used, npm will install version 1.9.1 instead of any newer versions.

Semantic versioning

npm uses semantic versioning to ensure the correct package version is installed. ***Semantic versioning*** *is a popular software versioning scheme that uses a sequence of three digits: major.minor.patch. Ex: 1.2.3.*

- *The major number indicates a major version of the package, which adds new functionality and possibly alters how previous functions now work.*
- *The minor number indicates a minor change to the package, which usually entails bug fixes and minor changes to how the package's functions work.*
- *The patch number indicates a bug fix to a minor version.*

Figure 11.2.11: Files composing Node.js project.

```
myproject
├── node_modules
│   └── underscore
├── package.json
├── package-
lock.json
└── server.js
```

Feedback?

PARTICIPATION ACTIVITY 11.2.6: Node.js project's package.json file.

1) A package.json file may list the project developers, homepage, and bugs.

- (●) True
- ( ) False

Correct

Much of the project's information may be included in a project's package.json.

2) Packages installed from npm occasionally have package.json files.

- ( ) True
- (●) False

Correct

npm requires packages to have a package.json file.

3) The "scripts" value in the example package.json above allows the web server to be started with the command:

```
npm start
```

- (●) True
- ( ) False

Correct

The `npm start` command executes the instruction: `node server.js`.

4) The following command installs the mkdirp module and adds mkdirp to the "dependencies" block of package.json:

```
npm install
mkdirp
```

- (●) True
- ( ) False

Correct

The "dependencies" block names the modules and version numbers required to run the project.

5) A project's package.json and package-lock.json files may list different dependency versions.

- (●) True
- ( ) False

Correct

The package.json may list the dependency version as ^3.0.1, but package-lock.json may list 3.2.1 if 3.2.1 was the most recent version available when the package was installed.

Feedback?

Exploring further:

- Node.js website
- npm documentation
- package.json documentation
- package-lock.json documentation
- Understanding module.exports and exports in Node.js

How was this section? Provide section feedback