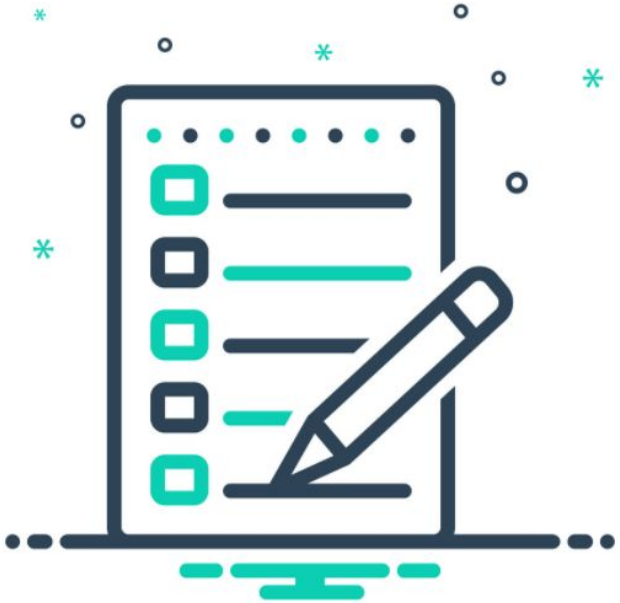
# NODEJS INTRO

by Kulinskyi Vitalii
updated by Revutska Natalya

softserve

# AGENDA

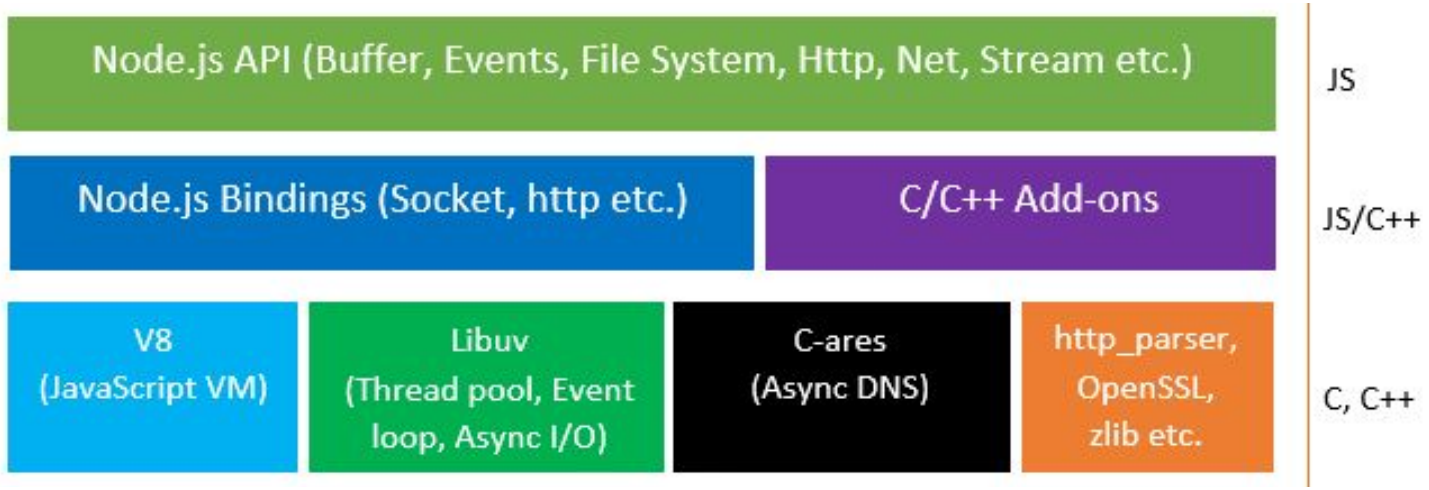
- What is NodeJS?
- Blocking and Non-blocking I/O
- REPL Console
- NPM & Module System
- Global Objects
- Filesystem module
- HTTP Module



softserve

# What is NodeJS ?

- **NodeJS** is an open source , cross platform runtime environment for server side and networking applications
- It is written in **C/C++ & JavaScript** and can run on Linux , Mac , Windows
- It provides an event driven architecture with **non blocking I/O** that is optimal for scalability.
- It uses **Google JavaScript V8** Engine to Execute Code




softserve

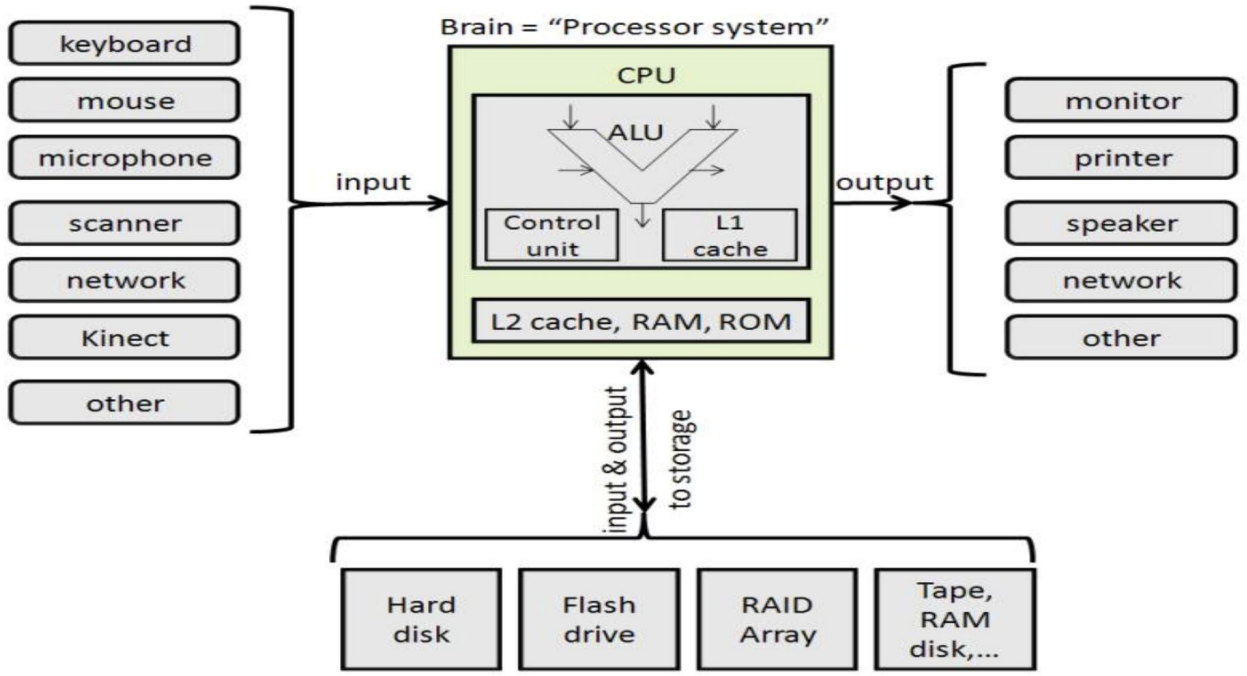
# JAVASCRIPT V8 ENGINE

**V8** is an open-source JavaScript engine developed by The Chromium Project for Google Chrome and Chromium web browsers.

V8 compiles **JavaScript** to **native machine** code before executing it

V8 can run standalone, or can be embedded into other application

# Input/Output (I/O)



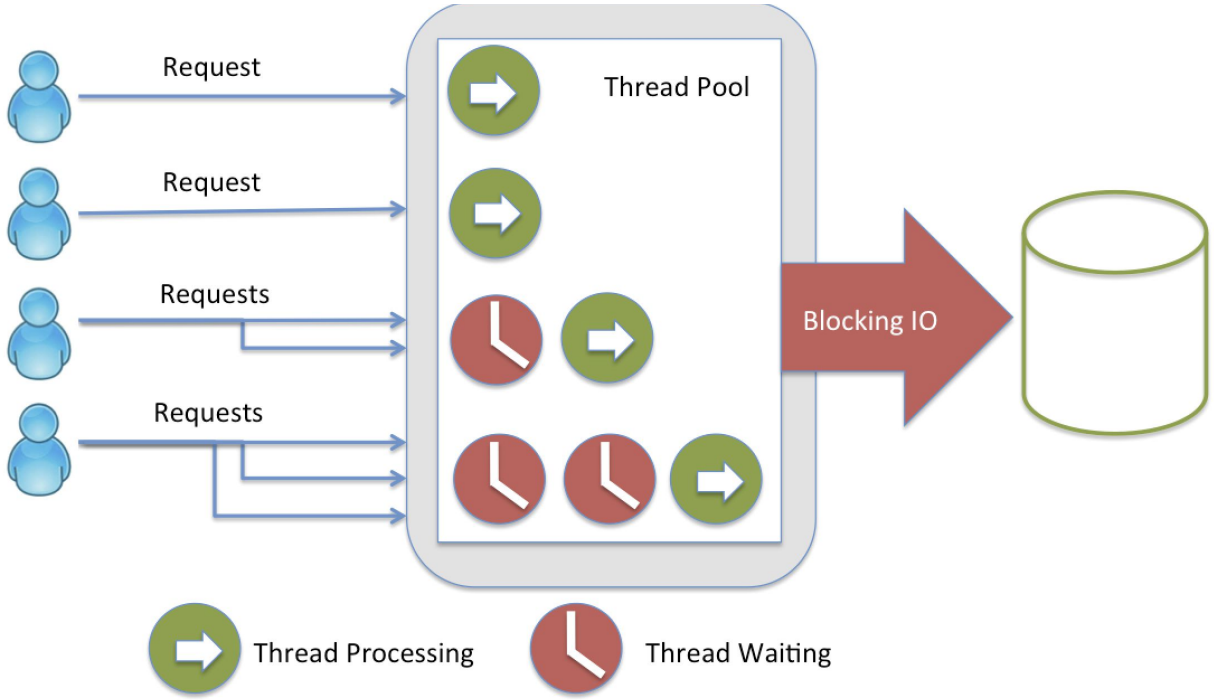

**Input/Output (I/O)** is the communication between an information processing system, such as a computer, and the outside world, possibly a human or another information processing system.

**Inputs** are the signals or data **received by** the system

**Outputs** are the signals or data **sent from** it.

softserve

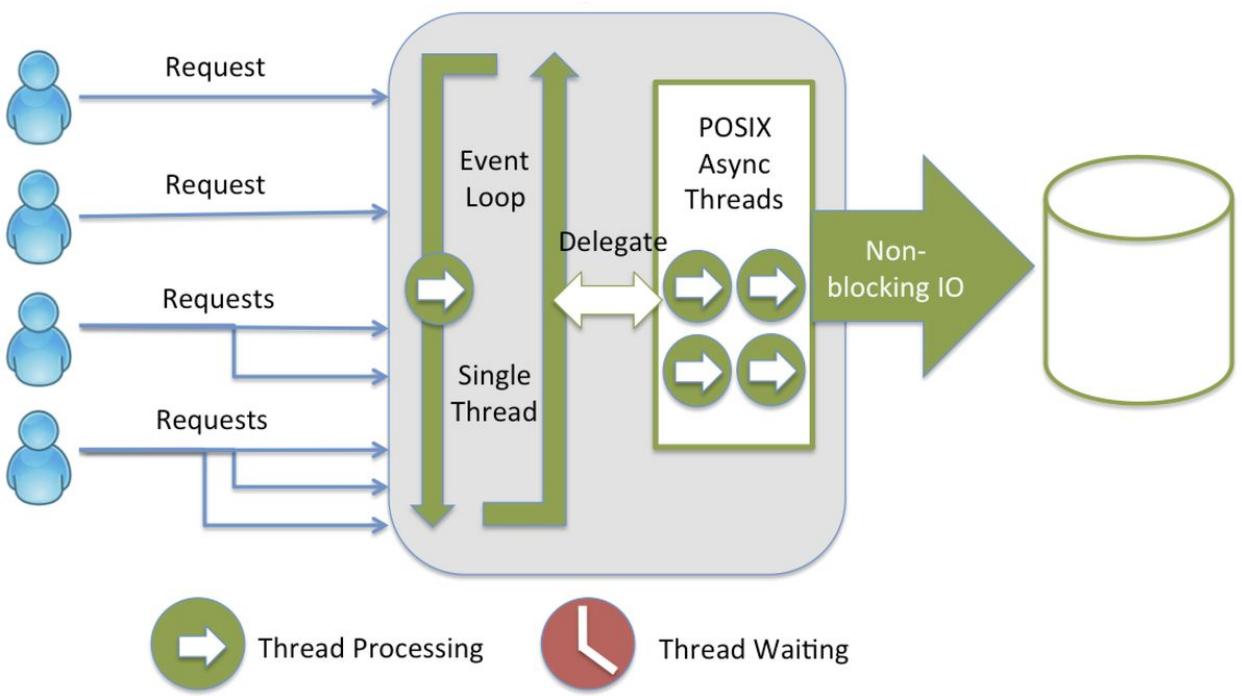
# BLOKING I/O




**Blocking** is when the execution of additional JavaScript in the Node.js process must wait until a non-JavaScript operation completes.

softserve

# NON-BLOCKING I/O

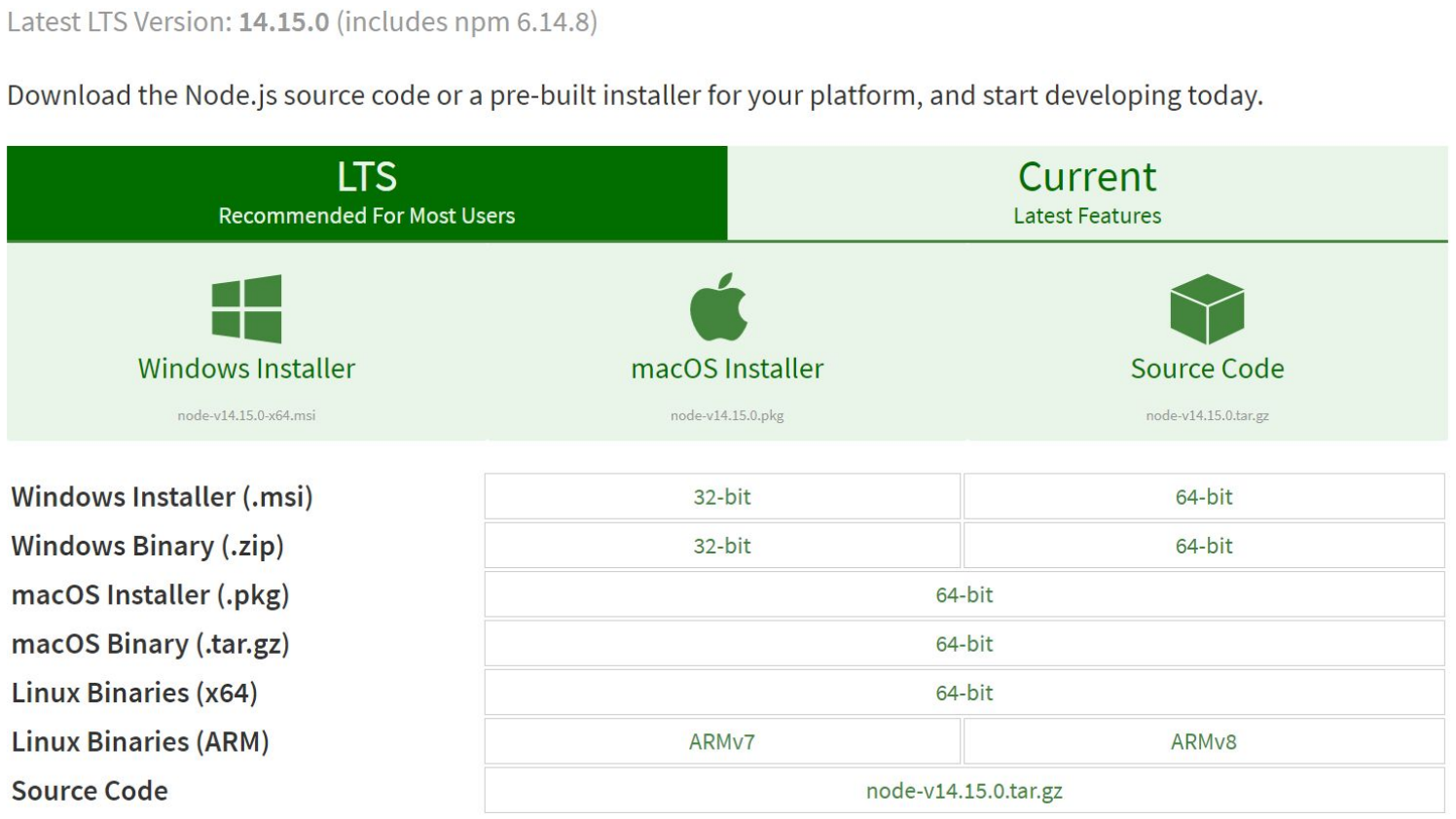



In **Non-blocking I/O** once the request is made we continue on to the next line of code before waiting for the time consuming request to finish.

softserve

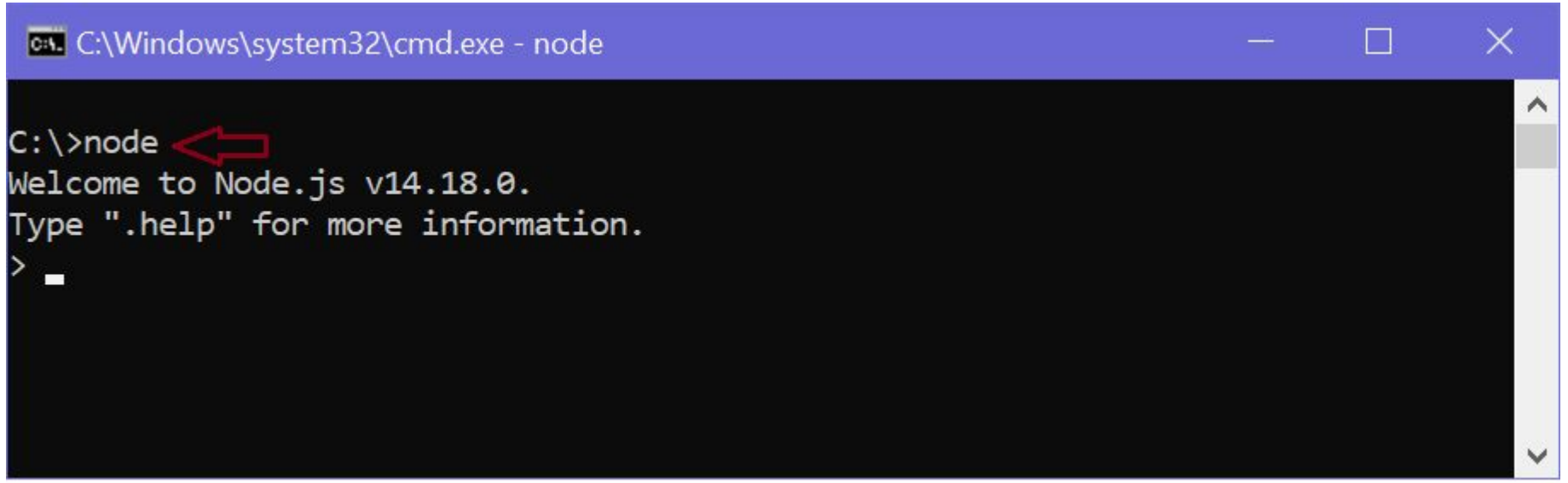
# NODEJS SETUP

https://nodejs.org/en/download/




softserve

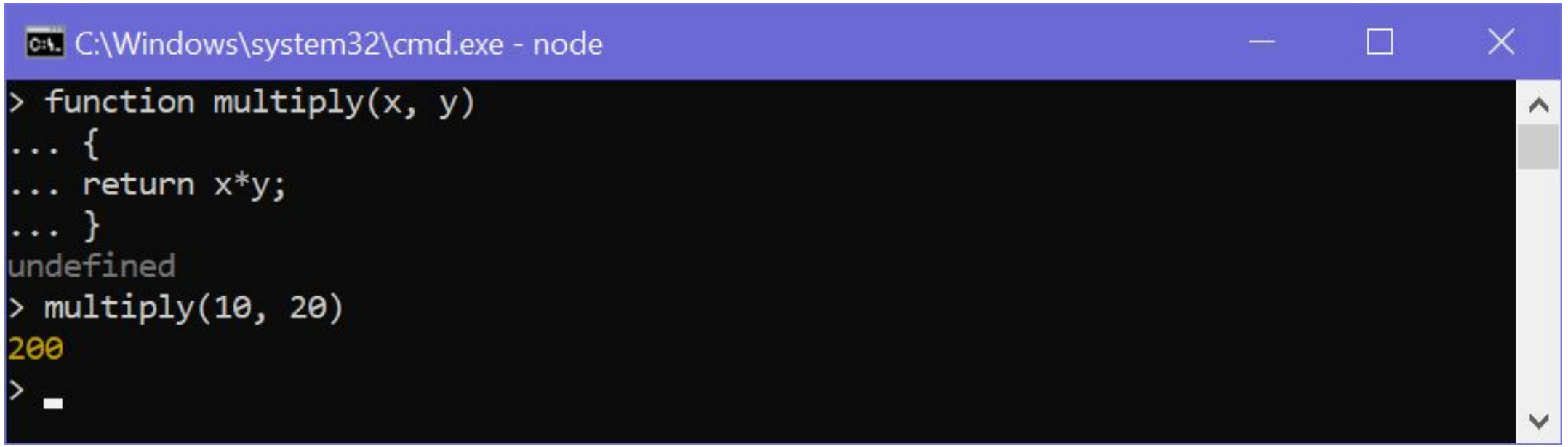
# REPL CONSOLE





softserve

# REPL CONSOLE




softserve

# REPL CONSOLE

mynodejs-app.js

```
console.log("Hello World");
```


rve

| REPL Command | Description |
| --- | --- |
| **.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. |

# REPL COMMANDS

softserve

# 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.

https://www.npmjs.com/

# INSTALLING A NPM MODULE

All the modules installed using NPM are installed under **node_modules** folder.

```
npm install <name>
npm install <name>@<tag>
npm install <name>@<version>
npm install <name> [--save|--save-dev]
```

softserve

# PACKAGE.JSON

The **package.json** file is kind of a manifest for your project. You can add a it to your package to make it easy for others to manage and install. Packages published to the registry must contain a package.json file.


A package.json file:

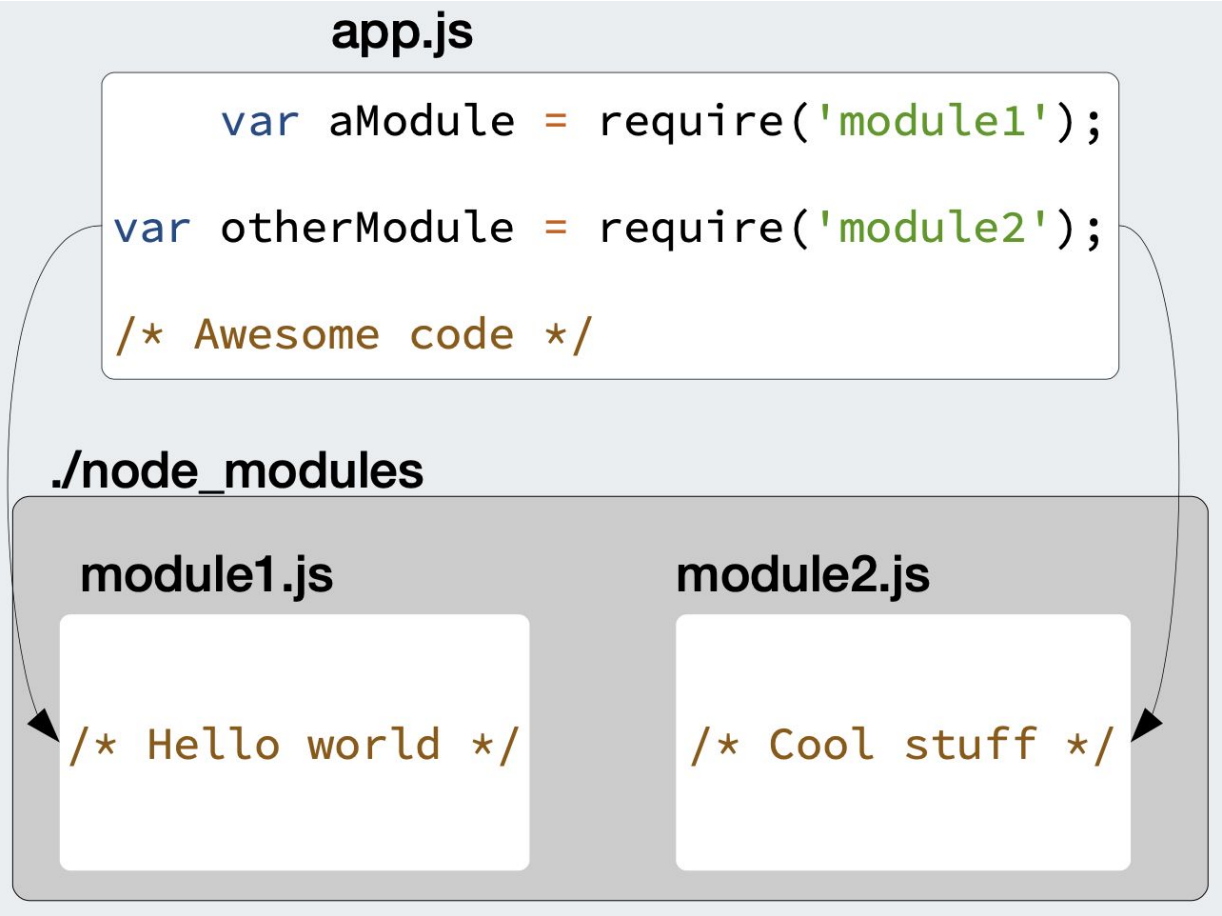
- lists the packages your project depends on
- specifies versions of a package that your project can use
- makes your build reproducible, and therefore easier to share with other developers

softserve

# MODULES IN NODEJS

- **NodeJS Module** system lets developers create a manageable structure of their application
- Each **module** is just a **JavaScript** file with code (functions, variables, objects, etc.)
- In Node, modules are referenced either by file path or by name
  - Referenced by name modules are either core modules (preinstalled with Node) or third-party modules installed using NPM
- Each module exposes public API that the developer can use after the module is imported into the current script

# LOADING AND EXPORTING MODULES




To **load/import** a module, you have to use the **require** function

The **require** function returns an object that represents the JavaScript API exposed by the module

Depending on the module, that object can be any JavaScript value — a function, an object with some properties that can be functions, an array, or any other type of JavaScript object

softserve

# LOADING AND EXPORTING MODULES

To **export** an object/function/variable from a module, use the **module.exports** object

```
// greetings.js
var hello = function() {
    console.log('Hello %s!', name || '');
}
var bye = function() {
    console.log('Bye %s!', name || '');
}
module.exports = {
    hello: hello,
    bye: bye
};
```

```
// hola.js
module.exports = function(name) {
    console.log('Hola %s!', name || '');
};
```

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

greetings.hello();
// Hello!
hola('John doe');
// Hola Jane doe!

// If we only need to call once

require('./greetings').bye('Jane');
// Goodbye Jane!
```

softserve

# GLOBAL OBJECTS

These objects are available in all modules:

- **process** - In computing, a process is an instance of a computer program that is being executed.
- **console** - For printing to stdout and stderr.
- **require** - used to load or import local files, JSON and modules..
- **__filename** – returns the absolute path of the file being executed. It is not available in the Node.js REPL..
- **__dirname** – returns the path of the directory the script is executing in. It is not available in the Node.js REPL.
- **module.exports** is used for defining what a module exports and makes available through require().
- **setTimeout(cb, ms)** – Run callback **cb** after at least **ms** milliseconds. The timeout must be in the range of 1-2,147,483,647 cannot span more than 24.8 days.
- **clearTimeout(t)** – Stop a timer that was previously created with **setTimeout**().
- **setInterval(cb, ms)** – Run callback **cb** repeatedly every **ms** milliseconds
- **clearInterval(t)** - Stop a timer that was previously created with setInterval().

# FILE SYSTEM MODULE

The **fs** module provides a lot of very useful functionality to access and interact with the file system.

```
const fs = require('fs')
```

softserve

# FILE SYSTEM MODULE

For example let's examine the fs.rename() method:

The asynchronous API is used with a callback:

```
const fs = require('fs')

fs.rename('before.json', 'after.json', err => {
  if (err) {
    return console.error(err)
  }

  //done
})
```

A synchronous API can be used like this, with a try/catch block to handle errors:

```
const fs = require('fs')

try {
  fs.renameSync('before.json', 'after.json')
  //done
} catch (err) {
  console.error(err)
}
```

softserve

# HTTP MODULE

It is easy to create an HTTP server in Node.js. A Node server is typically created using the **createServer** method of the **http module** and run this with

```
// include http module in the file
const http = require('http');

// create a server
http.createServer(function (req, res) {
    // http header
    // 200 - is the OK message
    // to respond with html content, 'Content-Type' should be 'text/html'
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('Node.js says hello!'); //write a response to the client
    res.end(); //end the response
}).listen(3000); //the server object listens on port 3000
```

softserve


TO
THE
FUTURE
softserve