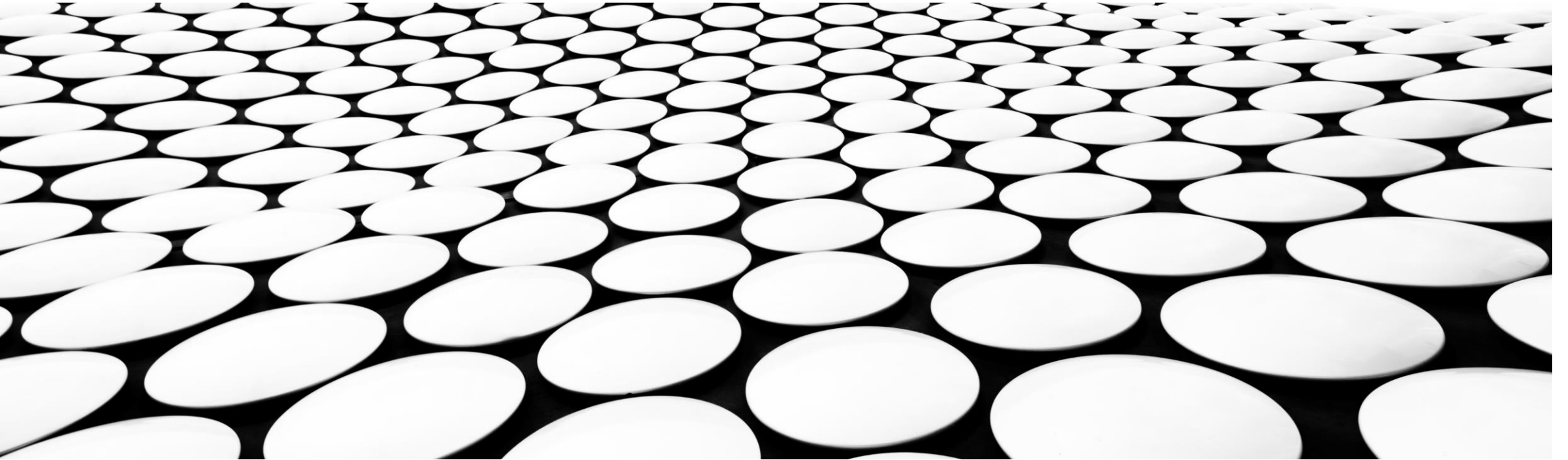


---

# NODE.JS



# WHAT IS NODE.JS ?

- Node.js는 Google 크롬의 JavaScript 엔진(V8 엔진)을 기반으로 구축된 Server-side Platform이다. Node.js는 Ryan Dahl에 의해 2009년에 개발되었다. 공식 문서에서 제공하는 Node.js의 정의는 다음과 같다.
  - Node.js는 빠르고 확장 가능한 네트워크 애플리케이션을 쉽게 구축하기 위해 Chrome의 JavaScript 런타임에 구축된 platform이다. Node.js는 event-driven 기반의 non-blocking I/O 모델을 사용하여 가볍고 효율적이며 분산 장치에서 실행되는 데이터 집약적인 real-time Application에 적합하다.
- Node.js는 Server-Side 및 네트워킹 애플리케이션을 개발하기 위한 Open Source cross-platform runtime 환경이다. Node.js 애플리케이션은 JavaScript로 작성되었으며 OS X, Microsoft Windows 및 Linux의 Node.js runtime 내에서 실행할 수 있다. Node.js는 또한 Node.js를 사용하여 웹 애플리케이션 개발을 크게 단순화하는 다양한 JavaScript 모듈의 풍부한 Library를 제공한다.

Node.js = Runtime Environment + JavaScript Library

# FEATURES OF NODE.JS

- Asynchronous and Event Driven
  - Node.js 라이브러리의 모든 API는 비동기(Asynchronous), 즉 non-blocking 이다. 본질적으로 Node.js 기반 서버는 API가 데이터를 반환할 때까지 절대(never) 기다리지 않는다. Server는 호출 후 다음 API로 이동하며, Node.js의 Events 알림(notification) mechanism은 서버가 이전 API 호출에 대한 응답을 받을 수 있도록 도와준다.
- Very Fast
  - Node.js 라이브러리는 Google Chrome의 V8 JavaScript 엔진을 기반으로 하여 코드 실행이 매우 빠르다.
- Single Threaded but Highly Scalable
  - Node.js는 Event Looping이 있는 Single Thread Model을 사용한다. Event Mechanism은 Server가 non-blocking 방식으로 응답(response)하는 데 도움이 되며 요청(request)을 처리하기 위해 제한된 스레드(limited thread)를 생성하는 기존 서버와 달리 서버를 고도로 확장 가능하게 만든다. Node.js는 Single Thread Program을 사용하며 동일한 프로그램은 Apache HTTP Server와 같은 기존(Traditional) 서버보다 훨씬 많은 수의 요청(request)에 서비스를 제공할 수 있다.
- No Buffering
  - Node.js 애플리케이션은 데이터를 buffering하지 않는다. 이러한 응용 프로그램은 단순히 chunk의 데이터를 출력한다.
- License
  - Node.js is released under the MIT license.

# NODE.JS USES ASYNCHRONOUS PROGRAMMING!

Here is how PHP or ASP handles a file request

- 작업을 컴퓨터의 File System으로 보낸다.
- File System이 열리고 file을 읽는 동안 기다린다.
- Client에게 콘텐츠(content)를 반환(return)한다.
- 다음 요청(request)을 처리(handle)할 준비 한다.

Here is how Node.js handles a file request

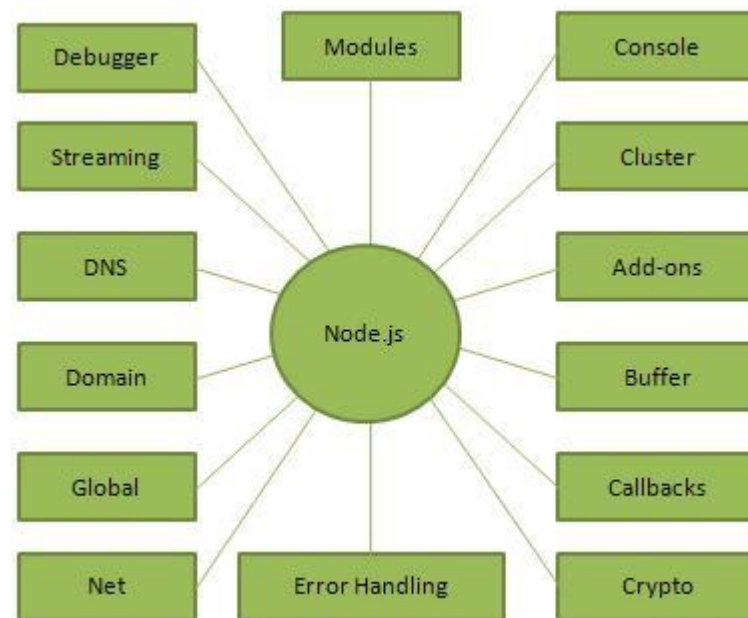
- 작업을 컴퓨터의 File System으로 보낸다.
- 다음 요청(request)을 처리할 준비를 한다.
- File System이 열리고 file을 읽으면 Server는 콘텐츠를(client)에게 반환한다.

Node.js는 대기할 필요가 없고 단순히 다음 요청을 계속한다.

Node.js는 Single-thread, non-blocking, asynchronous programming을 실행하므로 메모리가 매우 효율적이다.

# CONCEPTS

- 다음 Diagram은 다음에서 자세히 논의할 Node.js의 몇 가지 중요한 부분을 보여준다.



# WHERE TO USE NODE.JS?

- 다음은 Node.js가 완벽한 기술 파트너임을 입증하는 영역이다.
  - I/O bound Application.
  - Data Streaming Applications.
  - Data Intensive Real-time Applications(DIRT).
    - 데이터 집약적 실시간 애플리케이션
  - JSON APIs based Applications.
  - Single Page Applications.

## ■ Where Not to Use Node.js?

- CPU 집약적인 애플리케이션에는 Node.js를 사용하지 않는 것이 좋다.

# NODE.JS 설치

- 개발 Project에 따라서 Node의 버전을 여러 개 설치하고, 번갈아 가면서 사용해야 할 경우가 있다.
- 이럴 때는 NVM(Node Version Manager)를 이용해서 한 개발 환경에 여러 버전의 Node를 설치해서, 원하는 Node 버전을 골라서 사용 할 수 있다.
- 기존에 Node가 설치되어 있다면 Node를 제거한다.
- <https://github.com/coreybutler/nvm-windows/releases> 에서 nvm-setup.zip 파일을 다운로드 한다.

# NODE.JS 설치

NVM을 통해 Node를 설치하고, 설치한 Node를 활성화 하기

- `nvm ls`
- <https://nodejs.org/ko/>
  - LTS 버전 확인 후 설치
  - `nvm install lts`
- `nvm use [현재 lts버전]`

```
Windows PowerShell
PS C:\> nvm ls

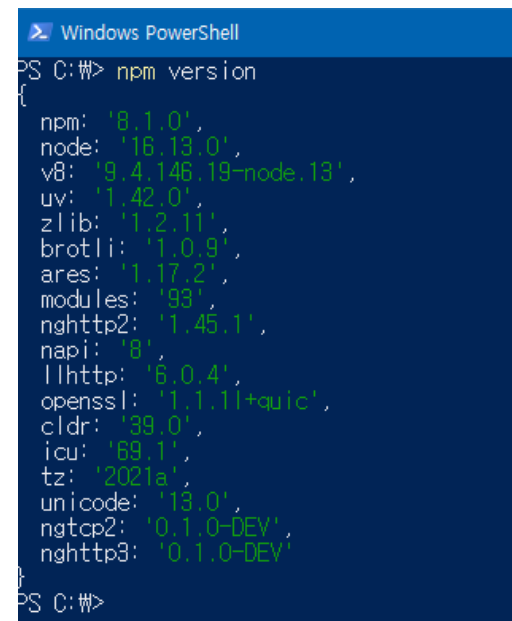
* 16.13.0 (Currently using 64-bit executable)
  14.16.1
PS C:\>
```



# NODE.JS 설치

npm을 통한 nodejs 버전 확인

- npm version



```
Windows PowerShell
PS C:\> npm version
{
  npm: '8.1.0',
  node: '16.13.0',
  v8: '9.4.146.19-node.13',
  uv: '1.42.0',
  zlib: '1.2.11',
  brotli: '1.0.9',
  ares: '1.17.2',
  modules: '93',
  nghttp2: '1.45.1',
  napi: '8',
  llhttp: '6.0.4',
  openssl: '1.1.1f+quic',
  cldr: '39.0',
  icu: '69.1',
  tz: '2021a',
  unicode: '13.0',
  ngtcp2: '0.1.0-DEV',
  nghttp3: '0.1.0-DEV'
}
```

# NODE.JS FIRST APPLICATION

hello world

- 실제 "Hello, World!"를 만들기 전에 Node.js를 사용하는 응용 프로그램에서 Node.js 응용 프로그램의 구성 요소를 살펴보자. Node.js 애플리케이션은 다음 세 가지 중요한 구성 요소로 구성된다.

- Import required modules
  - require 지시 문을 사용하여 Node.js 모듈을 load한다.
- Create server
  - Apache HTTP Server와 유사한 클라이언트의 요청을 수신하는 서버를 생성한다.
- Read request and return response
  - 이전 단계에서 생성된 Server는 Browser 또는 cosole이 될 수 있는 Client가 만든 HTTP request를 읽고 response를 반환한다.

# NODE.JS FIRST APPLICATION

## Creating Node.js Application

- Step 1 - Import Required Module
  - require 지시 문을 사용하여 http 모듈을 load하고 반환된 HTTP 인스턴스를 http 변수에 저장한다.

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

# NODE.JS FIRST APPLICATION

## Creating Node.js Application

### ■ Step 2 - Create Server

- 생성된 http 인스턴스를 사용하고 http.createServer() method를 호출하여 server instance를 만든 다음 server instance 와 연결된 listen method를 사용하여 port 8081에서 binding(연결)한다. Request 및 response parameter가 있는 함수를 전달한다. 항상 "Hello World " 를 반환하도록 샘플 구현을 작성한다.

```
http.createServer(function (request, response) {  
  // Send the HTTP header  
  // HTTP Status: 200 : OK  
  // Content Type: text/plain  
  response.writeHead(200, {'Content-Type': 'text/plain'});  
  
  // Send the response body as "Hello World"  
  response.end('Hello World\n');  
}).listen(8081);  
  
// Console will print the message  
console.log('Server running at http://127.0.0.1:8081/');
```

# NODE.JS

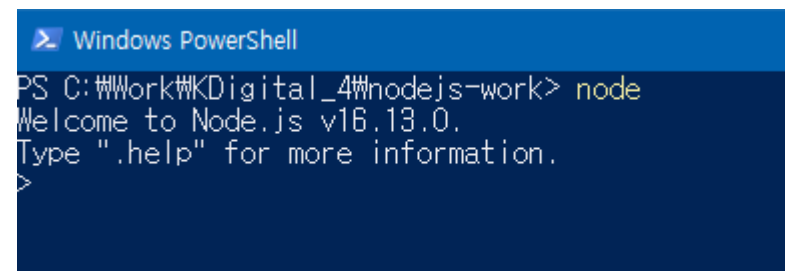
## REPL Terminal

- REPL은 Read, Eval, Print, Loop의 약자로 명령이 입력되고 시스템이 대화형 모드에서 출력으로 응답하는 Windows 콘솔 또는 Unix/Linux shell과 같은 computer 환경을 나타낸다. Node.js 또는 Node는 REPL 환경이 번들로 제공된다
- Node의 REPL 기능은 Node.js 코드를 실험하고 JavaScript 코드를 Debug 할 때 매우 유용하다.
- Read
  - 사용자 입력을 JavaScript Data-Structure로 구문 분석 (Parsing)하고 memory에 저장한다.
- Eval
  - Data Structure를 취하고 평가한다.
- Print
  - 결과를 인쇄한다.
- Loop
  - 사용자가 ctrl-c를 두 번 누를 때까지 위의 명령을 반복한다.

# NODE.JS

## Starting REPL

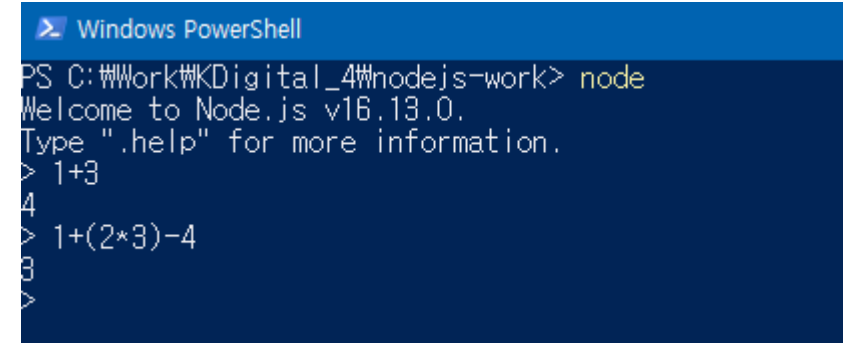
- REPL은 다음과 같이 parameter 없이 shell/console에서 node를 실행한다.
  - node
- REPL 명령 프롬프트 > Node.js 명령을 입력할 수 있는 곳이 표시된다.

A screenshot of a Windows PowerShell terminal window. The title bar is blue with the text 'Windows PowerShell'. The terminal content shows the command 'node' being executed at the prompt 'PS C:\Work\Digital\_4\nodejs-work>'. The output is 'Welcome to Node.js v16.13.0.' followed by 'Type ".help" for more information.' and a new prompt '>'.

# NODE.JS

## Simple Expression

```
$ node
> 1 + 3
4
> 1 + ( 2 * 3 ) - 4
3
>
```

A screenshot of a Windows PowerShell terminal window. The title bar is blue with a white icon and the text "Windows PowerShell". The terminal background is dark blue. The text shown is: "PS C:\Work\Digital\_4\nodejs-work> node", "Welcome to Node.js v16.13.0.", "Type \".help\" for more information.", "> 1+3", "4", "> 1+(2\*3)-4", "3", ">".

```
Windows PowerShell
PS C:\Work\Digital_4\nodejs-work> node
Welcome to Node.js v16.13.0.
Type ".help" for more information.
> 1+3
4
> 1+(2*3)-4
3
>
```

# NODE.JS

## Use Variables

- 기존 스크립트처럼 변수를 사용하여 값을 저장하고 나중에 인쇄할 수 있다. var 키워드를 사용하지 않으면 값이 변수에 저장되고 인쇄된다. 반면에 var 키워드를 사용하면 값이 저장되지만 인쇄되지는 않는다. console.log()를 사용하여 변수를 인쇄할 수 있다.

```
Windows PowerShell
PS C:\Work\Digital_4\nodejs-work> node
Welcome to Node.js v16.13.0.
Type ".help" for more information.
> x = 10
10
> var y = 10
undefined
> x + y
20
> console.log("hello world")
hello world
undefined
>
```

```
$ node
> x = 10
10 > var y = 10
undefined
> x + y
20
> console.log("Hello World")
Hello World
undefined
```

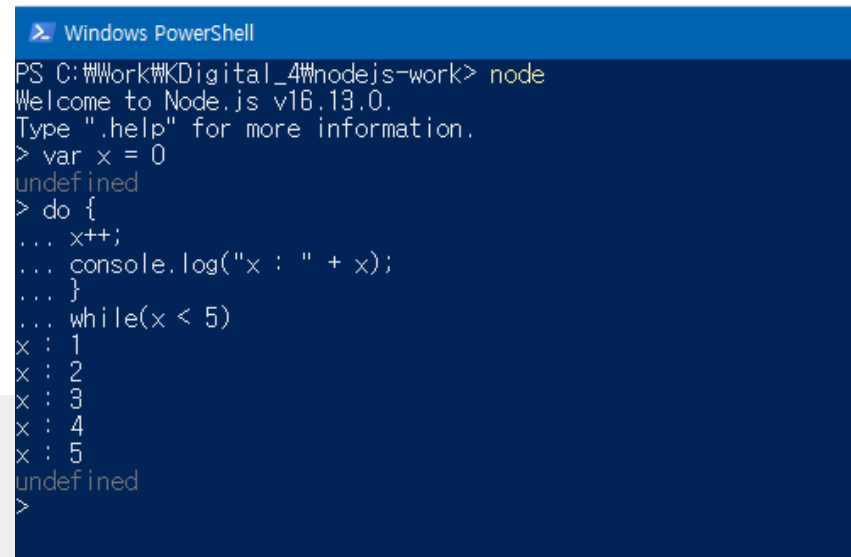


# NODE.JS

## Multiline Expression

- 노드 REPL은 JavaScript와 유사한 Multiline Expression을 지원한다.
- 다음 do-while 루프가 작동하는지 확인해 보자.
- ...는 여는 괄호 뒤에 Enter 키를 누르면 자동으로 옵니다. node는 표현식(expression)의 연속성을 자동으로 확인한다.

```
$ node
> var x = 0
undefined
> do {
  ... x++;
  ... console.log("x: " + x);
  ... }
... while ( x < 5 );
x: 1
x: 2
x: 3
x: 4
x: 5
undefined
>
```

A screenshot of a Windows PowerShell terminal window. The title bar says "Windows PowerShell". The command prompt shows the user running 'node' in a directory 'C:\Work\KDigital\_4\nodejs-work'. The Node.js REPL starts with a welcome message and a prompt. The user enters a multi-line JavaScript code block: 'var x = 0', followed by a 'do' loop containing 'x++' and 'console.log("x: " + x);', and a 'while(x < 5)' condition. The REPL outputs the values of x from 1 to 5, and then 'undefined' after the loop ends, followed by a new prompt '>'.

```
PS C:\Work\KDigital_4\nodejs-work> node
Welcome to Node.js v16.13.0.
Type ".help" for more information.
> var x = 0
undefined
> do {
... x++;
... console.log("x: " + x);
... }
... while(x < 5)
x: 1
x: 2
x: 3
x: 4
x: 5
undefined
>
```

# NODE.JS

## Underscore(\_) Variable

- 밑줄(\_)을 사용하여 마지막 결과를 얻을 수 있다.

```
Windows PowerShell
PS C:\Work\Digital_4\Nodejs-work> node
Welcome to Node.js v16.13.0.
Type ".help" for more information.
> var x = 10
undefined
> var y = 20
undefined
> x + y
30
> var sum = _
undefined
> console.log(sum)
30
undefined
>
```

```
$ node
> var x = 10
undefined
> var y = 20
undefined
> x + y
30
> var sum = _
undefined
> console.log(sum)
30
undefined
>
```

# NODE.JS

## REPL Commands

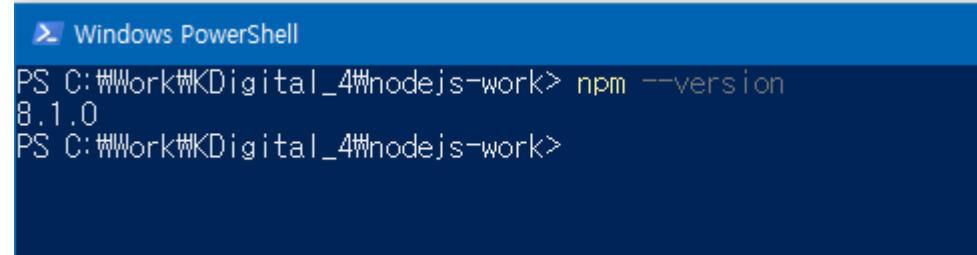
- `ctrl + c` – terminate the current command.
- `ctrl + c` twice – terminate the Node REPL.
- `ctrl + d` – terminate the Node REPL.
- Up/Down Keys – see command history and modify previous commands.
- tab Keys – list of current commands.
- `.help` – list of all commands.
- `.break` – exit from multiline expression.
- `.clear` – exit from multiline expression.
- `.save filename` – save the current Node REPL session to a file.
- `.load filename` – load file content in current Node REPL session.

# NPM(NODE PACKAGE MANAGER)

- 노드 패키지 관리자(NPM)는 두 가지 주요 기능을 제공한다.
  - Search.nodejs.org에서 검색 가능한 node.js package/module용 online repository.
  - Node.js Package를 설치하고 Node.js package의 version 관리 및 dependency(종속성) 관리를 수행하는 command line 유틸리티이다.

# NPM(NODE PACKAGE MANAGER)

- NPM은 v0.6.3 버전 이후에 Node.js 설치 파일과 함께 Bundle(번들)로 제공된다.



```
Windows PowerShell
PS C:\Work\Digital_4\nodejs-work> npm --version
8.1.0
PS C:\Work\Digital_4\nodejs-work>
```

# NPM(NODE PACKAGE MANAGER)

## Installing Modules using NPM

- Node.js module을 설치하는 간단한 구문.
- Now you can use this module in your js file as following

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

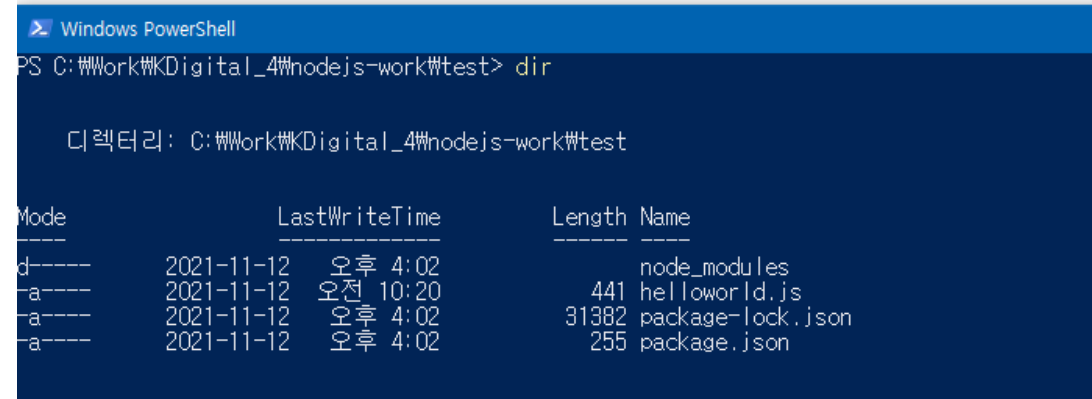
```
$ npm install <Module Name>
```

```
$ npm install express
```

# NPM(NODE PACKAGE MANAGER)

## Global vs Local Installation

- 기본적으로 NPM은 Local mode에서 모든 dependency(종속성)을 설치한다.
- 여기서 Local mode는 Node Application이 있는 folder에 있는 node\_modules directory에 package 설치를 한다.
- Local로 배포된 패키지는 require() 메서드를 통해 액세스할 수 있다. 예를 들어, Express module을 설치할 때 Express 모듈이 설치된 현재 디렉토리에 node\_modules directory를 생성한다.



```
Windows PowerShell
PS C:\Work\Digital_4\nodejs-work\test> dir

디렉터리: C:\Work\Digital_4\nodejs-work\test

Mode                LastWriteTime         Length Name
----                -
d-----         2021-11-12 오후 4:02             node_modules
-a----         2021-11-12 오전 10:20             441 helloworld.js
-a----         2021-11-12 오후 4:02          31382 package-lock.json
-a----         2021-11-12 오후 4:02             255 package.json
```

# NPM(NODE PACKAGE MANAGER)

## Global vs Local Installation

```
$ npm install express -g
```

```
$ npm ls -g
```

- Module은 Global(전역적)로 설치된다.



# NPM(NODE PACKAGE MANAGER)

## Using package.json

- package.json은 모든 Node Application/module의 root directory에 있으며 package의 속성을 정의하는 데 사용된다.
- node\_modules/express/에 있는 express 패키지의 package.json을 열어보자.

```
node_modules > express > {} package.json > {} dependencies
1 {
2   "name": "express",
3   "description": "Fast, unopinionated, minimalist web framework",
4   "version": "4.17.1",
5   "author": "TJ Holowaychuk <tj@vision-media.ca>",
6   "contributors": [
7     "Aaron Heckmann <aaron.heckmann+github@gmail.com>",
8     "Cianan Jessup <ciananj@gmail.com>",
9     "Douglas Christopher Wilson <doug@somethingdoug.com>",
10    "Guillermo Rauch <rauchg@gmail.com>",
11    "Jonathan Ong <me@jongleberry.com>",
12    "Roman Shtylman <shtylman+expressjs@gmail.com>",
13    "Young Jae Sim <hanul@hanul.me>"
14  ],
15   "license": "MIT",
16   "repository": "expressjs/express",
17   "homepage": "http://expressjs.com/",
18   "keywords": [
19     "express",
20     "framework",
21     "sinatra",
22     "web",
23     "rest",
24     "restful",
25     "router",
26     "app",
27     "api"
28  ],
29   "dependencies": {
30     "accepts": "~1.3.2",
31     "array-flatten": "1.1.1",
32     "body-parser": "1.19.0",
33     "content-disposition": "0.5.3",
34     "content-type": "~1.0.4",
35     "cookie": "0.4.0",
36     "cookie-signature": "1.0.6",
37     "debug": "2.6.9",
38     "depd": "~1.1.2",
39     "encodeurl": "~1.0.2",
40     "escape-html": "~1.0.3",
41     "etag": "~1.8.1",
42     "finalhandler": "~1.1.2",
```

# NPM(NODE PACKAGE MANAGER)

## Attributes of Package.json

- name — name of the package
- version — version of the package
- description — description of the package
- homepage — homepage of the package
- author — author of the package
- contributors — name of the contributors to the package
- dependencies — list of dependencies. NPM automatically installs all the dependencies mentioned here in the node\_module folder of the package.
- repository — repository type and URL of the package
- main — entry point of the package
- keywords — keywords

# NPM(NODE PACKAGE MANAGER)

Uninstalling a Module

```
$ npm uninstall express
```

looking at the content

```
$ npm ls
```

# NPM(NODE PACKAGE MANAGER)

## Updating a Module

```
$ npm update express
```

## Search a Module

```
$ npm search express
```

# NPM(NODE PACKAGE MANAGER)

## Create a Module

- 모듈을 생성하려면 package.json을 생성해야 한다.
- Package.json의 기본 골격을 생성할 NPM을 사용하여 package.json을 생성할 수 있다.

```
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sane defaults.

See 'npm help json' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg> --save' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (webmaster)
```

# NODE.JS MODULE

## Built-in Modules(내장 모듈)

- Node.js에는 추가 설치 없이 사용할 수 있는 내장 모듈 세트가 있다.

- assert
  - Assertion test 세트 제공
- Events
  - Event 처리
- fs
  - To handle the file system.
- http
  - Node.js가 HTTP Server 역할을 하도록 한다.
- url
  - URL string parse
- Etc
  - Buffer, cluster, crypto, dgram, `문`, domain, https, os, path, punycode
  - querystring, readline, stream, string\_decoder, timer, tls, util, v8, vm, zlib

# NODE.JS MODULE

## Include Modules

- Module을 포함하려면 module 이름과 함께 require() 함수를 사용한다.
  - var http = require('http');
  - 이제 Application이 HTTP module에 액세스할 수 있으며 Server를 생성할 수 있다.

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

# NODE.JS MODULE

## Create Your Own Modules

- 고유한 Module을 만들고 Application에 쉽게 포함할 수 있다.
- 다음 예제에서는 날짜 및 시간 개체를 반환하는 모듈을 만든다.
- 모듈 파일 외부에서 property와 method를 사용할 수 있도록 하려면 exports keyword를 사용한다.
- Code를 "myfirstmodule.js"라는 파일에 저장한다.

```
exports.myDateTime = function () {  
    return Date();  
};
```



# NODE.JS MODULE

## Include Your Own Module

- 이제 모든 Node.js file에 module을 포함하고 사용할 수 있다.
- ./를 사용하여 module을 찾는다. 이는 module이 Node.js 파일과 동일한 폴더에 있음을 의미한다.

```
var http = require('http');
var dt = require('./myfirstmodule');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write("The date and time are currently: " + dt.myDateTime());
  res.end();
}).listen(8080);
```

# NODE.JS CALLBACK CONCEPT

## What is Callback?

- Callback은 function에 대한 비동기 호출이다. 주어진 작업(task)이 완료되면 callback function이 호출된다. Node는 Callback을 많이 사용한다. Node의 모든 API는 callback을 지원하는 방식으로 작성되었다.
- File을 읽는 함수는 파일 읽기를 시작하고 제어를 실행 환경으로 즉시 반환하여 다음 명령어가 실행될 수 있도록 할 수 있다. 파일 I/O가 완료되면 파일의 내용을 callback function를 parameter(매개변수)로 전달하면서 callback function을 호출한다. 따라서 file I/O를 blocking하거나 기다리지 않는다. 따라서 Node.js는 어떤 함수도 결과를 반환할 때까지 기다리지 않고 많은 수의 요청을 처리할 수 있으므로 확장성이 뛰어나다.

# NODE.JS CALLBACK CONCEPT

## Blocking Code Example

```
var fs = require("fs");  
var data = fs.readFileSync('input.txt');  
  
console.log(data.toString());  
console.log("Program Ended");
```

```
Tutorials Point is giving self learning content  
to teach the world in simple and easy way!!!!  
Program Ended
```

# NODE.JS CALLBACK CONCEPT

## Non-Blocking Code Example

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

fs.readFile('input.txt', function (err, data) {
  if (err) return console.error(err);
  console.log(data.toString());
});

console.log("Program Ended");
```

```
Program Ended
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!
```

# NODE.JS CALLBACK CONCEPT

## Blocking/Non-Blocking Code Example

- 이 두 가지 예는 blocking 및 non-blocking 호출의 개념을 설명한다.
- 첫 번째 예는 프로그램이 파일을 읽을 때까지 blocking한 다음 계속해서 프로그램을 종료하는 것을 보여준다.
- 두 번째 예는 프로그램이 파일 읽기를 기다리지 않고 " Program Ended" 인쇄를 진행함과 동시에 blocking하지 않은 프로그램이 파일 읽기를 계속하는 것을 보여준다.
- 따라서 blocking 프로그램은 매우 많은 순서로 실행된다. 프로그래밍 관점에서는 logic을 구현하는 것이 더 쉽지만 non-blocking 프로그램은 순서대로 실행되지 않는다. 프로그램이 처리할 데이터를 사용해야 하는 경우 동일한 block 내에 보관하여 순차적으로 실행해야 한다.

# NODE.JS ACCESS FILE SYSTEM

## Synchronous vs Asynchronous

- fs module의 모든 method에는 synchronous와 asynchronous가 있다. asynchronous method는 마지막 매개 변수(parameter)를 완료 function callback으로 사용하고 callback function의 첫 번째 매개변수 (parameter)를 오류로 사용한다. synchronous method 대신 asynchronous method 를 사용하는 것이 좋다.
- 전자는 실행 중에 프로그램을 blocking하지 않는 반면 두 번째 method는 blocking하기 때문이다.

# NODE.JS ACCESS FILE SYSTEM

## Synchronous vs Asynchronous

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

// Asynchronous read
fs.readFile('input.txt', function (err, data) {
  if (err) {
    return console.error(err);
  }
  console.log("Asynchronous read: " + data.toString());
});

// Synchronous read
var data = fs.readFileSync('input.txt');
console.log("Synchronous read: " + data.toString());

console.log("Program Ended");
```

```
PS C:\Work\KDigital_4\nodejs-work\test> node .\fstest.js
Synchronous read: Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!
Program Ended
Asynchronous read: Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!
PS C:\Work\KDigital_4\nodejs-work\test> █
```

# NODE.JS ACCESS FILE SYSTEM

## Open a File

### ■ Syntax

- 비동기 모드에서 파일을 여는 방법.
- `fs.open(path, flags[, mode], callback)`

### ■ Parameters

- `path` - 경로를 포함한 파일명을 가지는 문자열이다.
- `Flags` - 플래그는 열려는 파일의 동작을 나타낸다.
- `mode` - 파일 모드(permission 및 고정 비트)를 설정하지만 파일이 생성된 경우에만 설정된다.
- 기본값은 0666이며 읽기 및 쓰기가 가능합니다.
- `Callback` - 두 개의 arguments(err, fd)를 가져오는 callback function이다.



# NODE.JS ACCESS FILE SYSTEM

## Open a File - Flags

Sr.No.	Flag & Description
1	<b>r</b> - Open file for reading. An exception occurs if the file does not exist.
2	<b>r+</b> - Open file for reading and writing. An exception occurs if the file does not exist.
3	<b>rs</b> - Open file for reading in synchronous mode.
4	<b>rs+</b> - Open file for reading and writing, asking the OS to open it synchronously. See notes for 'rs' about using this with caution.
5	<b>W</b> - Open file for writing. The file is created (if it does not exist) or truncated (if it exists).
6	<b>Wx</b> - Like 'w' but fails if the path exists.
7	<b>w+</b> - Open file for reading and writing. The file is created (if it does not exist) or truncated (if it exists).
8	<b>wx+</b> - Like 'w+' but fails if path exists.
9	<b>a</b> - Open file for appending. The file is created if it does not exist.
10	<b>ax</b> - Like 'a' but fails if the path exists.
11	<b>a+</b> - Open file for reading and appending. The file is created if it does not exist.
12	<b>ax+</b> - Like 'a+' but fails if the the path exists.

# NODE.JS ACCESS FILE SYSTEM

Open a File

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

// Asynchronous - Opening File
console.log("Going to open file!");
fs.open('input.txt', 'r+', function(err, fd) {
  if (err) {
    return console.error(err);
  }
  console.log("File opened successfully!");
});
```

```
PS C:\Work\KDigital_4\nodejs-work\test> node .\openfs.js
Going to open file!
File opened successfully!
PS C:\Work\KDigital_4\nodejs-work\test>
```

# NODE.JS ACCESS FILE SYSTEM

## Get File Information

- `fs.stat(path, callback)`
- Parameters
  - `path` - 경로를 포함한 파일명을 가지는 문자열입니다.
  - `callback` - 두 개의 인수(`err`, `stats`)를 가져오는 callback function이다.

# NODE.JS ACCESS FILE SYSTEM

## Get File Information

- fs.Stats class에는 파일 유형을 확인하는 데 사용할 수 있는 몇 가지 유용한 method가 있다.

Sr.No	Method & Description
1	<b>stats.isFile()</b> Returns true if file type of a simple file.
2	<b>stats.isDirectory()</b> Returns true if file type of a directory.
3	<b>stats.isBlockDevice()</b> Returns true if file type of a block device.
4	<b>stats.isCharacterDevice()</b> Returns true if file type of a character device.
5	<b>stats.isSymbolicLink()</b> Returns true if file type of a symbolic link.
6	<b>stats.isFIFO()</b> Returns true if file type of a FIFO.
7	<b>stats.isSocket()</b> Returns true if file type of a socket.

# NODE.JS ACCESS FILE SYSTEM

## Get File Information

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

console.log("Going to get file info!");
fs.stat('input.txt', function (err, stats) {
  if (err) {
    return console.error(err);
  }
  console.log(stats);
  console.log("Got file info successfully!");

  // Check file type
  console.log("isFile ? " + stats.isFile());
  console.log("isDirectory ? " + stats.isDirectory());
});
```

```
PS C:\Work\KDigital_4\nodejs-work\test> node .\stats.js
Going to get file info!
Stats {
  dev: 141247691,
  mode: 33206,
  nlink: 1,
  uid: 0,
  gid: 0,
  rdev: 0,
  blksize: 4096,
  ino: 24206847997161940,
  size: 95,
  blocks: 0,
  atimeMs: 1637047381844.7405,
  mtimeMs: 1637047381773,
  ctimeMs: 1637047381773.7388,
  birthtimeMs: 1636961497656.8174,
  atime: 2021-11-16T07:23:01.845Z,
  mtime: 2021-11-16T07:23:01.773Z,
  ctime: 2021-11-16T07:23:01.774Z,
  birthtime: 2021-11-15T07:31:37.657Z
}
Got file info successfully!
isFile ? true
isDirectory ? false
PS C:\Work\KDigital_4\nodejs-work\test>
```

# NODE.JS ACCESS FILE SYSTEM

## Writing a File

- `fs.writeFile(filename, data[, options], callback)`
- 이 방법은 파일이 이미 있는 경우 파일을 덮어쓴다. 기존 파일에 쓰려면 사용 가능한 다른 방법을 사용해야 한다.
- Parameters
  - `path` - 경로를 포함한 파일명을 가지는 문자열이다.
  - `data` - 파일에 기록될 String(문자열) 또는 Buffer이다.
  - `Options` - 세 번째 매개변수는 {encoding, mode, flag}를 보유할 object(객체)이다. Default값은 Encoding은 utf8, mode는 8진수 값 0666이다. Flag는 'w'이다.
  - `callback` - 쓰기 오류의 경우 오류를 반환하는 단일 parameter `err`을 가져오는 callback function이다.

# NODE.JS ACCESS FILE SYSTEM

## Writing a File

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

console.log("Going to write into existing file");
fs.writeFile('input.txt', 'Simply Easy Learning!', function(err) {
  if (err) {
    return console.error(err);
  }

  console.log("Data written successfully!");
  console.log("Let's read newly written data");

  fs.readFile('input.txt', function (err, data) {
    if (err) {
      return console.error(err);
    }
    console.log("Asynchronous read: " + data.toString());
  });
});
```

```
Going to write into existing file
Data written successfully!
Let's read newly written data
Asynchronous read: Simply Easy Learning!
```

# NODE.JS ACCESS FILE SYSTEM

## Reading a File

- `fs.read(fd, buffer, offset, length, position, callback)`
- file descriptor를 사용하여 파일을 읽는다.
- Parameters
  - `fd` - `fs.open()`에 의해 반환된 file descriptor이다.
  - `buffer` - 데이터가 기록될 buffer이다.
  - `Offset` - 쓰기를 시작하는 buffer의 offset이다.
  - `Length` - 읽을 byte 수를 지정하는 정수이다.
  - `Position` - 이것은 파일에서 읽기 시작 위치를 지정하는 정수이다. `Position`이 null이면 현재 파일 위치에서 데이터를 읽는다.
  - `Callback` - 이것은 세 개의 인수(`err`, `bytesRead`, `buffer`)를 가져오는 callback function이다.



# NODE.JS ACCESS FILE SYSTEM

## Reading a File

```
var fs = require("fs");
var buf = new Buffer(1024);

console.log("Going to open an existing file");
fs.open('input.txt', 'r+', function(err, fd) {
  if (err) {
    return console.error(err);
  }
  console.log("File opened successfully!");
  console.log("Going to read the file");

  fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){
    if (err){
      console.log(err);
    }
    console.log(bytes + " bytes read");

    // Print only read bytes to avoid junk.
    if(bytes > 0){
      console.log(buf.slice(0, bytes).toString());
    }
  });
});
```

```
Going to open an existing file
File opened successfully!
Going to read the file
95 bytes read
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!
```

# NODE.JS ACCESS FILE SYSTEM

## Closing a File

- `fs.close(fd, callback)`
- Parameters
  - `fd` – This is the file descriptor returned by file `fs.open()` method.
  - `callback` – This is the callback function No arguments other than a possible exception are given to the completion callback.

# NODE.JS ACCESS FILE SYSTEM

## Closing a File

```
var fs = require("fs");
var buf = Buffer.alloc(1024);

console.log("Going to open an existing file");
fs.open('input.txt', 'r+', function(err, fd) {
  if (err) {
    return console.error(err);
  }
  console.log("File opened successfully!");
  console.log("Going to read the file");

  fs.read(fd, buf, 0, buf.length, 0, function(err, bytes) {
    if (err) {
      console.log(err);
    }

    // Print only read bytes to avoid junk.
    if (bytes > 0) {
      console.log(buf.slice(0, bytes).toString());
    }

    // Close the opened file.
    fs.close(fd, function(err) {
      if (err) {
        console.log(err);
      }
      console.log("File closed successfully.");
    });
  });
});
```

```
Going to open an existing file
File opened successfully!
Going to read the file
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!
File closed successfully.
```