

---

## 2장. 노드 간단하게 살펴보기

# 강의 주제 및 목차

## 강의 주제

노드를 어떻게 사용하는지 먼저 간단하게 알아봐요.

## 목 차



1

첫 번째 노드 프로젝트 만들기

2

콘솔에 로그 뿌리기

3

프로세스 객체 간단하게 살펴보기

4

노드에서 모듈 사용하기

5

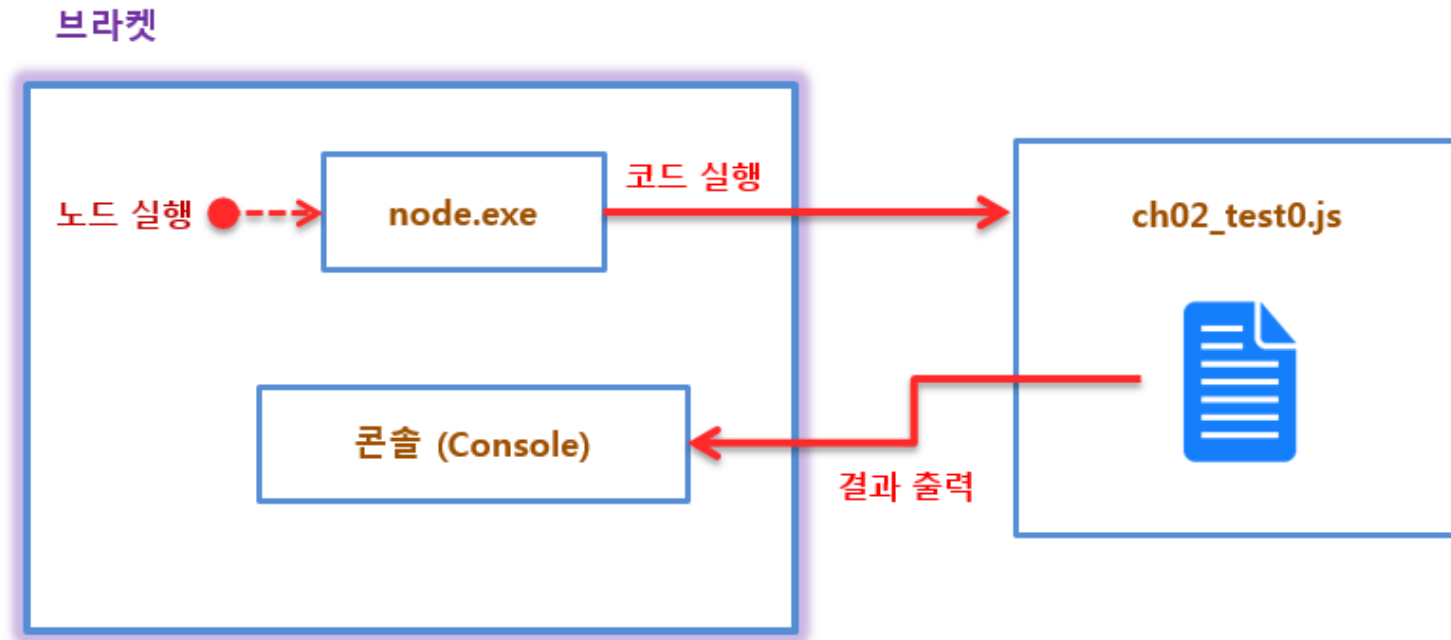
간단한 내장 모듈 사용하기

1.

첫 번째 노드 프로젝트 만들기

# 노드 프로그램의 실행 과정

- node 명령어를 이용해 실행하면 지정한 자바스크립트 파일의 내용을 읽어 들인 후 실행
- 실행 결과는 콘솔에 출력됨



- 파일>폴더 열기 메뉴를 누르고 nodejs 폴더 아래의 NodeExample 폴더 지정  
(기존에 지정되어 있으면 그대로 사용)

---

2.

콘솔에 로그 뿌리기

# 명령프롬프트에서 자바스크립트 파일 실행하기

---

- cmd 명령어로 명령프롬프트 실행하면 사용자 폴더가 기본 위치로 지정됨
- NodeExample 폴더로 이동 후 node 명령어를 이용해 실행 (% 기호는 입력 안함)

```
% cd nodejs\NodeExample
```

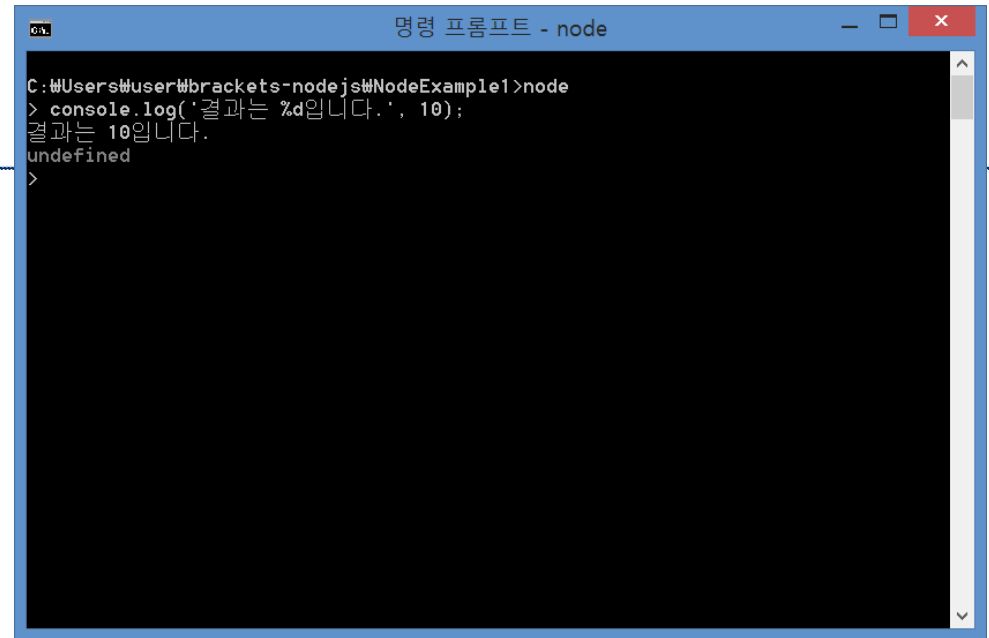
```
% node ch02_test1.js
```

# 명령프롬프트에서 직접 코드 입력하여 실행하기

- 명령프롬프트에 직접 코드를 입력할 수 있음
- node 명령어를 실행하면 코드를 입력할 수 있는 상태로 바뀜

```
% node
```

```
% console.log('결과는 %d입니다.', 10);
```



```
C:\Users\user\brackets-nodejs\NodeExample1>node
> console.log('결과는 %d입니다.', 10);
결과는 10입니다.
undefined
>
```

# console 객체

- console은 어디서든 사용할 수 있는 전역 객체로 로그를 출력할 수 있도록 함

| 전역 객체 이름 | 설명                      |
|----------|-------------------------|
| console  | 콘솔 창에 결과를 보여주는 객체       |
| process  | 프로세스의 실행에 대한 정보를 다루는 객체 |
| exports  | 모듈을 다루는 객체              |

```
% console.log('숫자 보여주기 : %d', 10);  
% console.log('문자열 보여주기 : %s', '안녕!');  
% console.log('JSON 객체 보여주기 : %j', {name: '소녀시대'});
```



# console 객체의 주요 메소드

- Console객체에는 dir, time, timeEnd 등의 메소드가 있음

| 메소드 이름      | 설명                           |
|-------------|------------------------------|
| dir(object) | 자바스크립트 객체의 속성들을 출력합니다.       |
| time(id)    | 실행 시간을 측정하기 위한 시작 시간을 기록합니다. |
| timeEnd(id) | 실행 시간을 측정하기 위한 끝 시간을 기록합니다.  |

# 코드 실행한 시간 체크하기

- 왼쪽 [NodeExample] 프로젝트 선택 후 오른쪽 마우스 버튼 눌러 [파일 만들기] 메뉴 선택
- 파일 이름으로 ch02\_test1.js 입력
- 새로 만들어진 파일 안에 아래 내용 코드 입력
- 실행

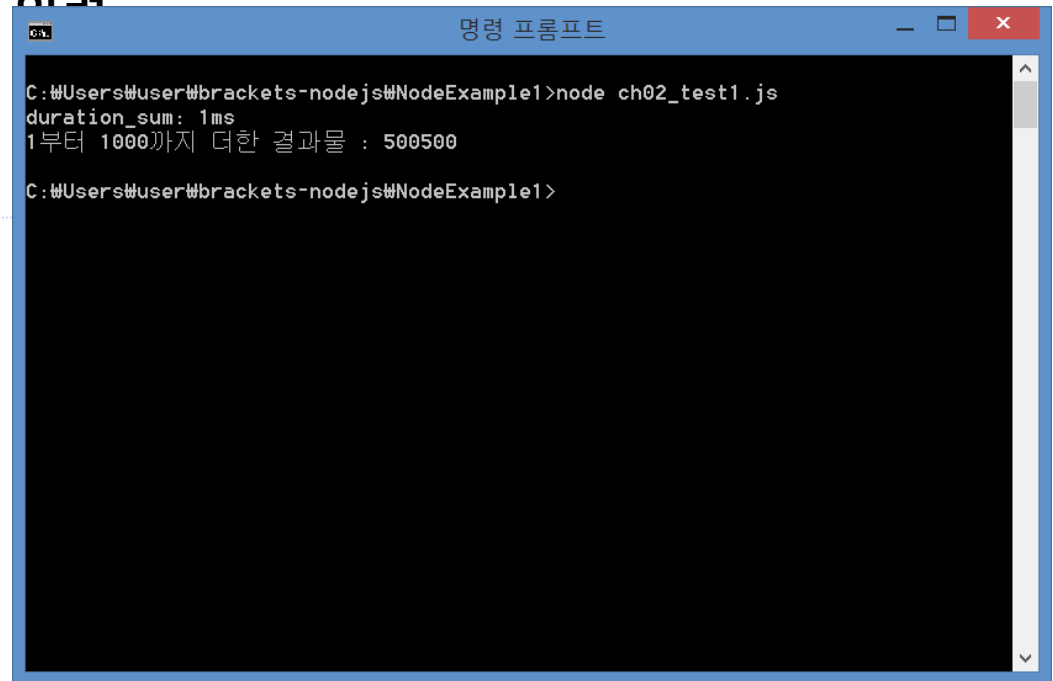
```
var result = 0;
```

```
console.time('duration_sum');
```

```
for (var i = 1; i <= 1000; i++) {  
    result += i;  
}
```

```
console.timeEnd('duration_sum');
```

```
console.log('1부터 1000까지 더한 결과물 : %d', result);
```



A screenshot of a Windows command prompt window titled '명령 프롬프트'. The window shows the execution of a Node.js script. The command entered is 'C:\Users\User\brackets-nodejs\NodeExample1>node ch02\_test1.js'. The output shows 'duration\_sum: 1ms' and '1부터 1000까지 더한 결과물 : 500500'. The prompt then returns to 'C:\Users\User\brackets-nodejs\NodeExample1>'.

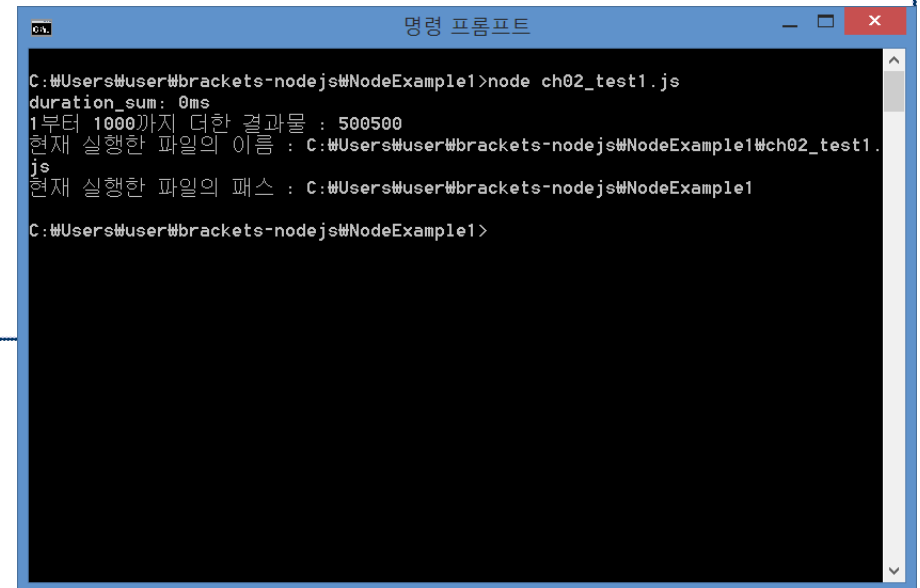
# 실행한 파일 이름과 객체 정보 출력

- `__filename`, `__dirname` 전역 변수 사용하여 파일 이름 출력
- `console.dir` 함수 이용하여 객체 정보 출력
- 자바스크립트의 객체는 `{ }` 를 이용해서 만들고 형태는 `{ 속성이름 : 속성값, 속성이름 :`

## ... 중략

```
console.log('현재 실행한 파일의 이름 : %s', __filename);  
console.log('현재 실행한 파일의 패스 : %s', __dirname);
```

```
var Person = {name:"소녀시대", age:20};  
console.dir(Person);
```



```
명령 프롬프트  
C:\Users\User\brackets-nodejs\NodeExample1>node ch02_test1.js  
duration_sum: 0ms  
1부터 1000까지 더한 결과물 : 500500  
현재 실행한 파일의 이름 : C:\Users\User\brackets-nodejs\NodeExample1\ch02_test1.js  
현재 실행한 파일의 패스 : C:\Users\User\brackets-nodejs\NodeExample1  
C:\Users\User\brackets-nodejs\NodeExample1>
```

## ch02\_test1.js console 객체 사용하기 - 실행 소요 시간 측정하기

```
var result = 0;

console.time('duration_sum');

for (var i = 1; i <= 1000; i++) {
    result += i;
}

console.timeEnd('duration_sum');
console.log('1부터 1000까지 더한 결과물 : %d', result);

console.log('현재 실행한 파일의 이름 : %s', __filename);
console.log('현재 실행한 파일의 패스 : %s', __dirname);

// dir() 메소드 사용하기
var Person = {name:"소녀시대", age:20};
console.dir(Person);
```

---

3.

프로세스 객체 살펴보기

# 프로세스 객체

- process 객체는 프로세스 정보를 다루는 객체

| 속성/메소드 이름 | 설명                             |
|-----------|--------------------------------|
| argv      | 프로세스를 실행할 때 전달되는 파라미터(매개변수) 정보 |
| env       | 환경 변수 정보                       |
| exit( )   | 프로세스를 끝내는 메소드                  |

```
console.log('argv 속성의 파라미터 수 : ' + process.argv.length);  
console.dir(process.argv);
```

```
process.argv.forEach(function(item, index) {  
    console.log(index + ' : ', item);  
});
```

## ch02\_test2.j process 객체 살펴보기

```
console.log('argv 속성의 파라미터 수 : ' + process.argv.length);
console.dir(process.argv);

if (process.argv.length > 2) {
    console.log('세번째 파라미터의 값 : %s', process.argv[2]);
}

process.argv.forEach(function(item, index) {
    console.log(index + ' : ', item);
});
```

## ch02\_test3.j env 객체 사용하여 환경변수의 값 알아내기

---

```
console.dir(process.env);
```

```
console.log('OS 환경변수의 값 : ' + process.env[OS]);
```



---

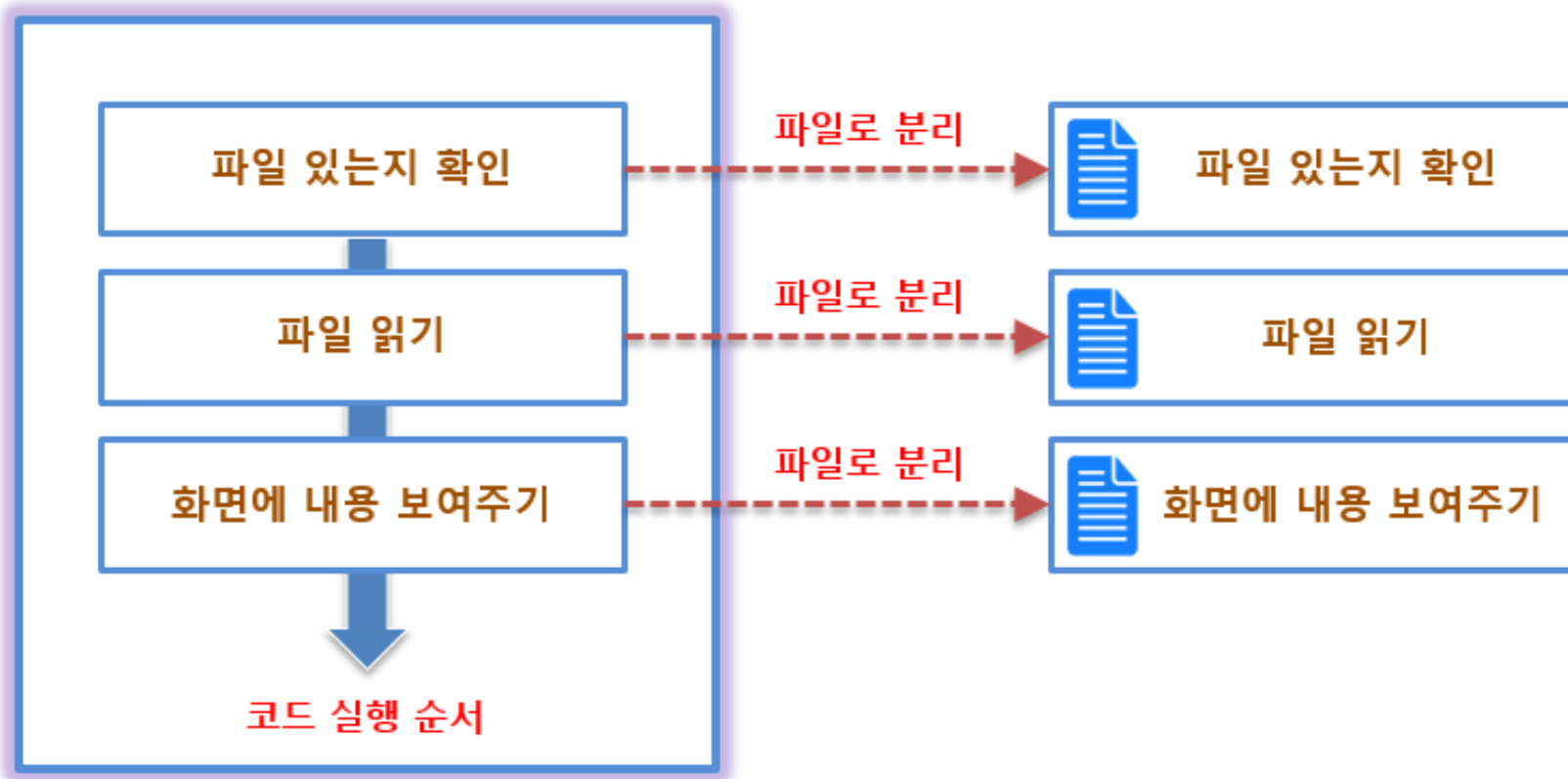
4.

노드에서 모듈 사용하기

# 기능을 각각의 파일로 분리하기

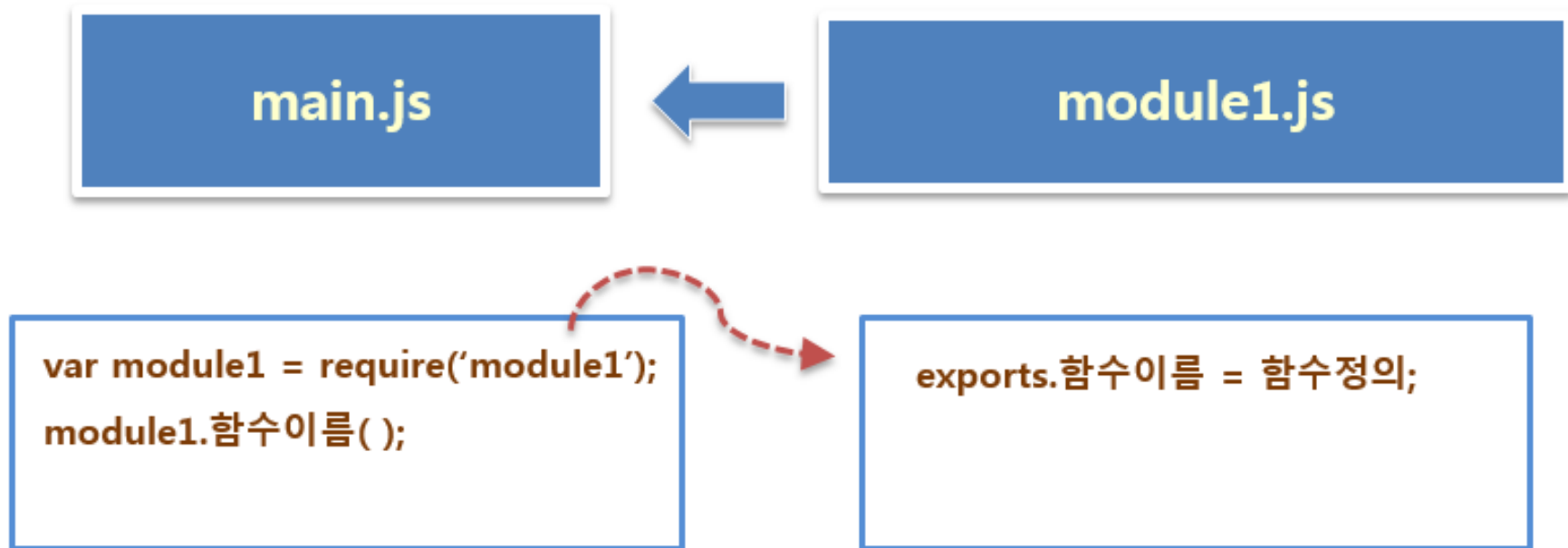
- 파일을 읽을 때 필요한 코드를 기능별로 각각의 파일에 나누어 넣을 수 있음

메인 파일



# 모듈

- 별도의 파일로 분리된 독립 기능을 모듈이라고 함
- 모듈을 만들어 놓으면 다른 파일에서 모듈을 불러와 사용할 수 있음
- CommonJs 표준 스펙을 따르며 exports 전역 객체를 사용함



# exports와 module.exports의 사용

- 모듈 파일 안에서는 exports를 사용할 수도 있고 module.exports를 사용할 수도 있음
- 객체를 직접 할당하려면 module.exports를 사용함

module1.js

```
exports.add = function(a, b) {  
  return a + b;  
};  
  
exports.multiply = function(a, b) {  
  return a * b;  
};
```

VS.

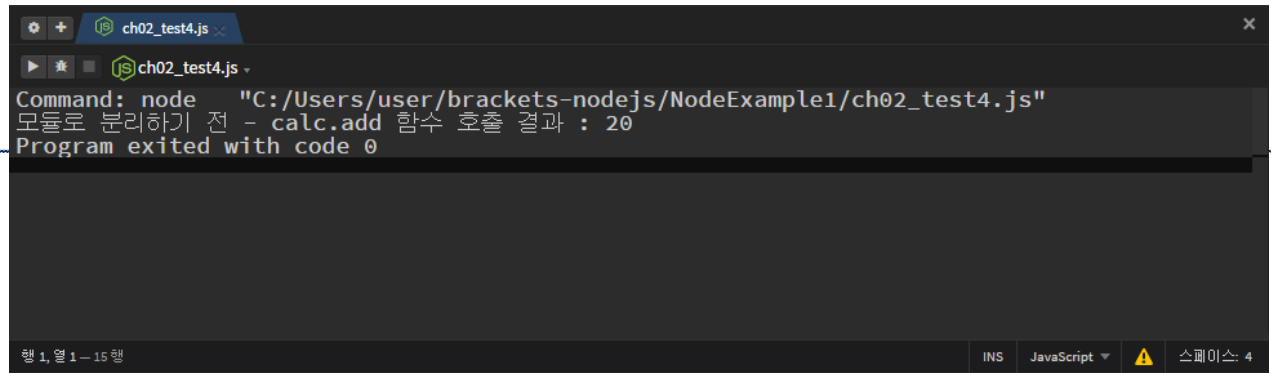
module2.js

```
var calc = {};  
  
calc.add = function(a, b) {  
  return a + b;  
};  
  
calc.multiply = function(a, b) {  
  return a * b;  
};  
  
module.exports = calc;
```

# 더하기 함수를 모듈로 분리하기

- 모듈 분리 전의 더하기 함수 사용 코드

```
var calc = {};  
calc.add = function(a, b) {  
    return a + b;  
}  
console.log('모듈로 분리하기 전 - calc.add 함수 호출 결과 : %d',  
    calc.add(10, 10));
```



The screenshot shows a terminal window with the following content:

```
ch02_test4.js  
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch02_test4.js"  
모듈로 분리하기 전 - calc.add 함수 호출 결과 : 20  
Program exited with code 0
```

The terminal window has a dark background and a light-colored text. The command prompt shows the file path and the command used to run the script. The output shows the result of the function call and the exit code.

# 모듈 파일

---

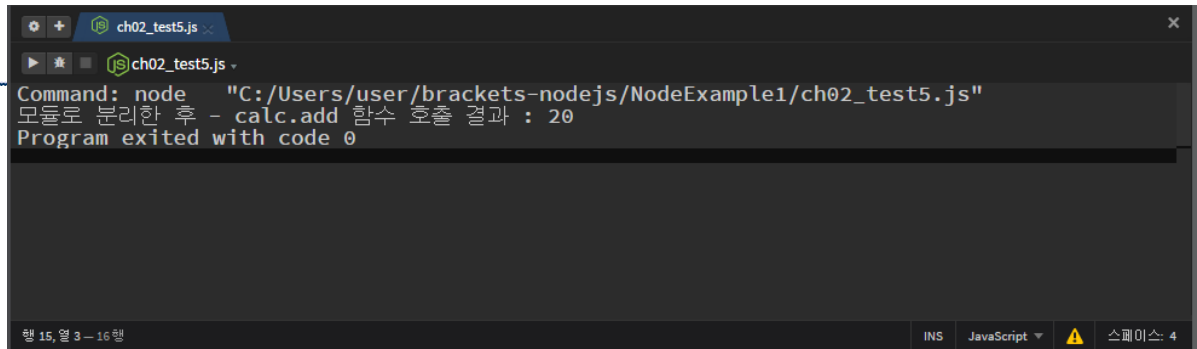
- 모듈 파일 생성, calc.js
- 함수를 exports 객체의 add 속성으로 추가

```
exports.add = function(a, b) {  
    return a + b;  
}
```

# 메인 파일

- 모듈 파일을 불러들인 후 add 함수 호출
- require 함수를 이용해 모듈 파일을 불러들임
- 불러들인 결과 객체는 exports 객체로 간주할 수 있음
- 파일이 아닌 폴더를 지정하면 그 폴더 안에 들어있는 index.js 파일을 불러들임

```
var calc = require('./calc');  
console.log('모듈로 분리한 후 - calc.add 함수 호출 결과 : %d',  
            calc.add(10, 10));
```



The screenshot shows a terminal window with the title 'ch02\_test5.js'. The command executed is 'node "C:/Users/user/brackets-nodejs/NodeExample1/ch02\_test5.js"'. The output is '모듈로 분리한 후 - calc.add 함수 호출 결과 : 20' and 'Program exited with code 0'. The status bar at the bottom indicates '행 15, 열 3 - 16 행', 'INS', 'JavaScript', and '스페이스: 4'.

```
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch02_test5.js"  
모듈로 분리한 후 - calc.add 함수 호출 결과 : 20  
Program exited with code 0
```

```
var calc = {};  
calc.add = function(a, b) {  
    return a + b;  
}  
  
console.log('모듈로 분리하기 전 - calc.add 함수 호출 결과 : %d', calc.add(10,  
10));
```



```
exports.add = function(a, b) {  
  return a + b;  
}
```

```
var calc = require('./calc');  
console.log('모듈로 분리한 후 - calc.add 함수 호출 결과 : %d', calc.add(10,  
10));
```

- 함수를 할당한 경우

# 모듈 파일에서 module.exports 사용

---

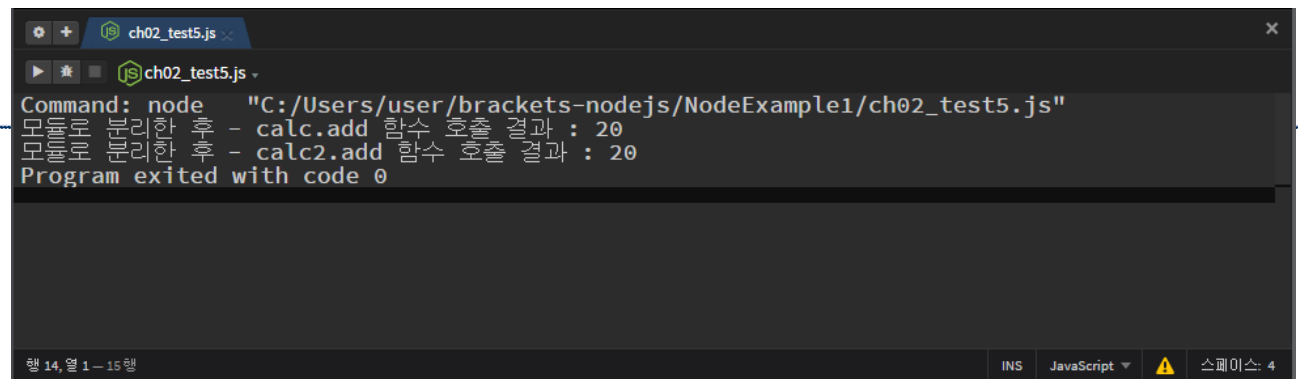
- 모듈 파일 생성, calc2.js
- calc 객체를 만들고 그 객체 그대로 module.exports에 할당

```
var calc = {};  
calc.add = function(a, b) {  
  return a + b;  
}  
module.exports = calc;
```

# 메인 파일

- 모듈 파일을 불러들인 후 add 함수 호출
- require 함수를 이용해 모듈 파일을 불러들임
- 불러들인 결과 객체는 module.exports 객체로 간주할 수 있음

```
var calc2 = require('./calc2');  
console.log('모듈로 분리한 후 - calc2.add 함수 호출 결과 : %d',  
            calc2.add(10, 10));
```



The screenshot shows a terminal window with the following content:

```
ch02_test5.js  
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch02_test5.js"  
모듈로 분리한 후 - calc.add 함수 호출 결과 : 20  
모듈로 분리한 후 - calc2.add 함수 호출 결과 : 20  
Program exited with code 0
```

At the bottom of the terminal, it shows "행 14, 열 1 - 15 행" and "INS JavaScript 스페이스: 4".

```
var calc = {};  
  
calc.add = function(a, b) {  
    return a + b;  
};  
  
module.exports = calc;
```

```
var calc = require('./calc');  
console.log('모듈로 분리한 후 - calc.add 함수 호출 결과 : %d', calc.add(10,  
10));  
  
var calc2 = require('./calc2');  
console.log('모듈로 분리한 후 - calc2.add 함수 호출 결과 : %d', calc2.add(10,  
10));
```

- calc 함수를 할당한 경우
- calc2는 객체를 할당한 경우

# 외장 모듈의 사용

---

- nconf 모듈을 사용하면 시스템 환경 변수에 접근할 수 있음
- 외장 모듈을 사용할 때는 상대 패스를 사용하지 않음

```
var nconf = require('nconf');  
nconf.env();  
console.log('OS 환경 변수의 값 : %s',  
            nconf.get('OS'));
```

## ch02\_test6.j 외부 모듈인 nconf를 이용해 환경변수 확인하기

```
var nconf = require('nconf');  
  
nconf.env();  
  
console.log('OS 환경변수의 값 : %s', nconf.get('OS'));
```

- >npm install nconf --save

- >npm init  
name: nodeexample

.....

package.json 생성

- >npm install

.....

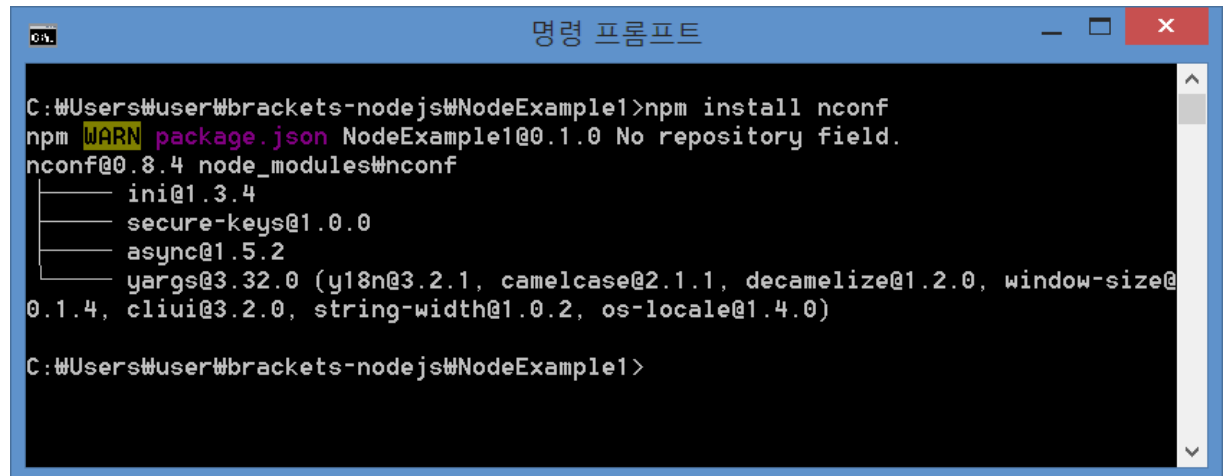
node\_modules 생성



# 외장 모듈의 설치

- 외장 모듈을 사용하려면 npm을 이용해 패키지를 설치해야 함
- 패키지 설치 후 앞에서 만든 자바스크립트 파일을 실행하면 OS 환경변수가 출력됨

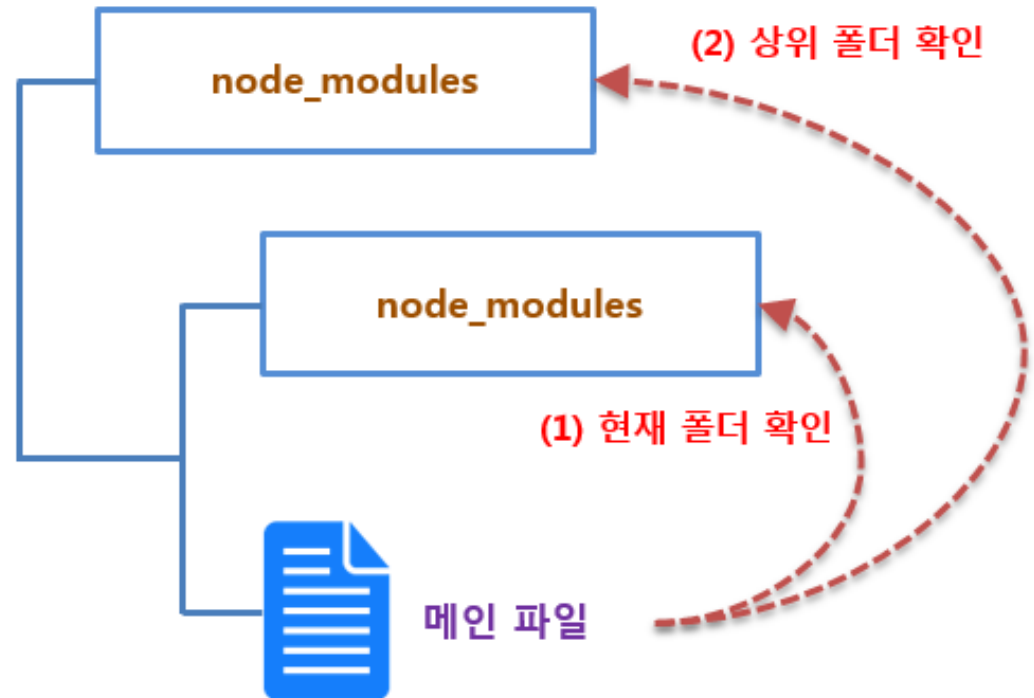
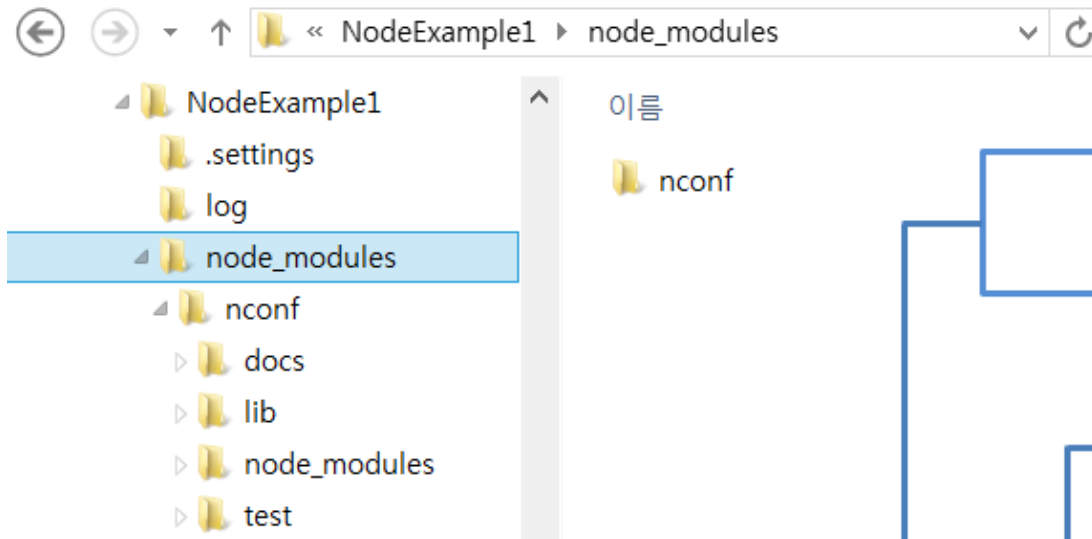
% npm install nconf



```
C:\Users\User\brackets-nodejs\NodeExample1>npm install nconf
npm WARN package.json NodeExample1@0.1.0 No repository field.
nconf@0.8.4 node_modules\nconf
├── ini@1.3.4
├── secure-keys@1.0.0
├── async@1.5.2
└── yargs@3.32.0 (y18n@3.2.1, camelcase@2.1.1, decamelize@1.2.0, window-size@0.1.4, cliui@3.2.0, string-width@1.0.2, os-locale@1.4.0)
C:\Users\User\brackets-nodejs\NodeExample1>
```

# 설치된 외장 모듈의 사용

- node\_modules 폴더 안에 패키지가 설치됨
- 프로젝트별로 사용 가능하나 프로젝트 상위 폴더에 있어도 사용 가능



# package.json 파일

- npm으로 설치한 패키지 정보를 확인할 수 있음

```
{
  "name": "NodeExample",
  "version": "0.1.0",
  "description": "NodeExample1",
  "main": "hello-world-server.js",
  "scripts": {
    "test": "echo ₩Error: no test specified! Configure in package.json₩ && exit 1"
  },
  "repository": "",
  "keywords": [
    "node.js",
    "eclipse",
    "nodeclipse"
  ],
  "author": "",
  "license": "MIT",
  "readmeFilename": "README.md"
}
```

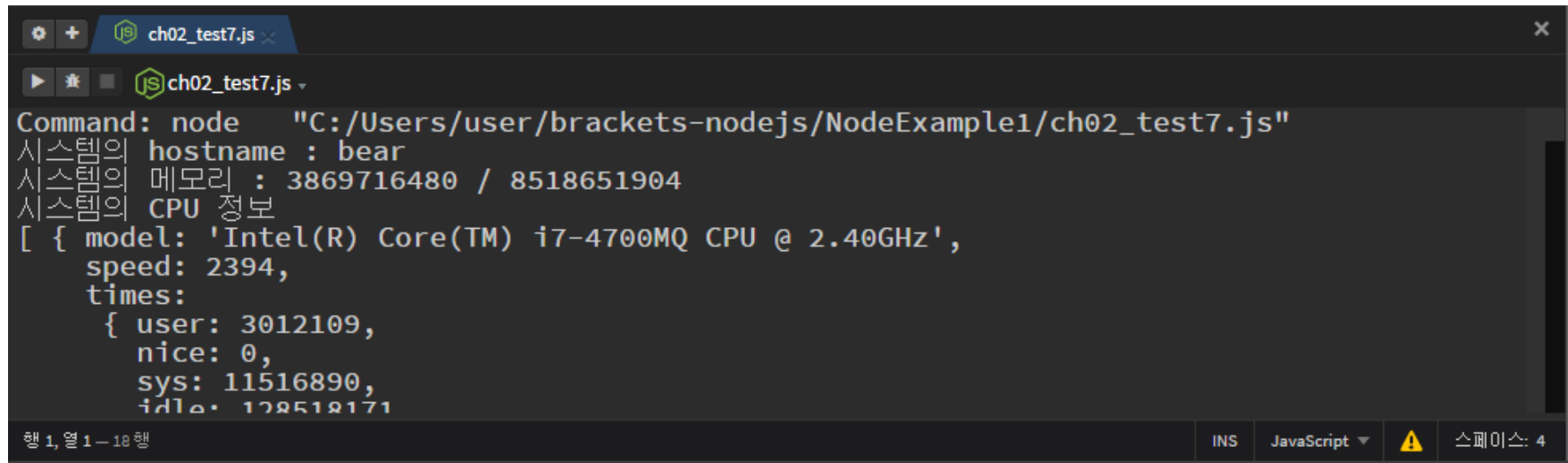
# 패키지 삭제와 설치 시 package.json에 정보 추가

- npm uninstall nconf 를 이용해 삭제
- npm install nconf --save 옵션을 주면 package.json 파일에 패키지 정보 추가
- 프로젝트 폴더를 복사했을 경우 npm install 명령만으로 package.json 안에 들어있는 패키지 정보를 이용해 패키지 일괄 설치

```
{  
  ...중략  
  "dependencies": {  
    "nconf": "^0.7.1"  
  }  
}
```

- require 함수를 이용해 모듈을 불러온 후 사용

```
var os = require('os');  
console.log('시스템의 hostname : %s', os.hostname());  
console.log('시스템의 메모리 : %d / %d', os.freemem(), os.totalmem());  
console.log('시스템의 CPU 정보\n');  
console.dir(os.cpus());  
console.log('시스템의 네트워크 인터페이스 정보\n');  
console.dir(os.networkInterfaces());
```



The screenshot shows a Node.js REPL window with the file 'ch02\_test7.js' open. The command 'node "C:/Users/user/brackets-nodejs/NodeExample1/ch02\_test7.js"' has been executed, resulting in the following output:

```
시스템의 hostname : bear
시스템의 메모리 : 3869716480 / 8518651904
시스템의 CPU 정보
[ { model: 'Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz',
  speed: 2394,
  times:
    { user: 3012109,
      nice: 0,
      sys: 11516890,
      idle: 122518171 }
```

The status bar at the bottom indicates '행 1, 열 1 - 18 행', 'INS', 'JavaScript', a warning icon, and '스페이스: 4'.

## ch02\_test7.j os 내부 모듈 사용하기

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

console.log('시스템의 hostname : %s', os.hostname());
console.log('시스템의 메모리 : %d / %d', os.freemem(), os.totalmem());
console.log('시스템의 CPU 정보\ n');
console.dir(os.cpus());
console.log('시스템의 네트워크 인터페이스 정보\ n');
console.dir(os.networkInterfaces());
```

- 호스트, 메모리의 여유량의 내용

---

5.

간단한 내장 모듈 사용하기



# 내장 모듈

- 노드 설치 시 미리 제공하는 모듈
- <http://nodejs.org/api> 사이트에서 확인 가능
- 시스템 정보를 알려주는 os 모듈

| 메소드 이름               | 설명                              |
|----------------------|---------------------------------|
| hostname( )          | 운영체제의 호스트 이름을 알려줍니다.            |
| totalmem( )          | 시스템의 전체 메모리 용량을 알려줍니다.          |
| freemem( )           | 시스템에서 사용 가능한 메모리 용량을 알려줍니다.     |
| cpus( )              | CPU 정보를 알려줍니다.                  |
| networkInterfaces( ) | 네트워크 인터페이스 정보를 담은 배열 객체를 반환합니다. |

- 파일을 다룰 때 파일 패스에서 파일 이름을 구별하는 등의 기능 제공

| 메소드 이름                   | 설명   |
|--------------------------|--|
| <code>join( )</code>     | 여러 개의 이름들을 모두 합쳐 하나의 파일 패스로 만들어 줍니다.<br>파일 패스를 완성할 때 구분자 등을 알아서 조정합니다. |
| <code>dirname( )</code>  | 파일 패스에서 디렉터리 이름을 반환합니다.  |
| <code>basename( )</code> | 파일 패스에서 파일의 확장자를 제외한 이름을 반환합니다.  |
| <code>extname( )</code>  | 파일 패스에서 파일의 확장자를 반환합니다.  |

```
var path = require('path');  
// 디렉터리 이름 합치기  
var directories = ["users", "mike", "docs"];  
var docsDirectory = directories.join(path.sep);  
console.log('문서 디렉터리 : %s', docsDirectory);  
// 디렉터리 이름과 파일 이름 합치기  
var curPath = path.join('/Users/mike', 'notepad.exe');  
console.log('파일 패스 : %s', curPath);
```

...중략

// 패스에서 디렉터리, 파일 이름, 확장자 구별하기

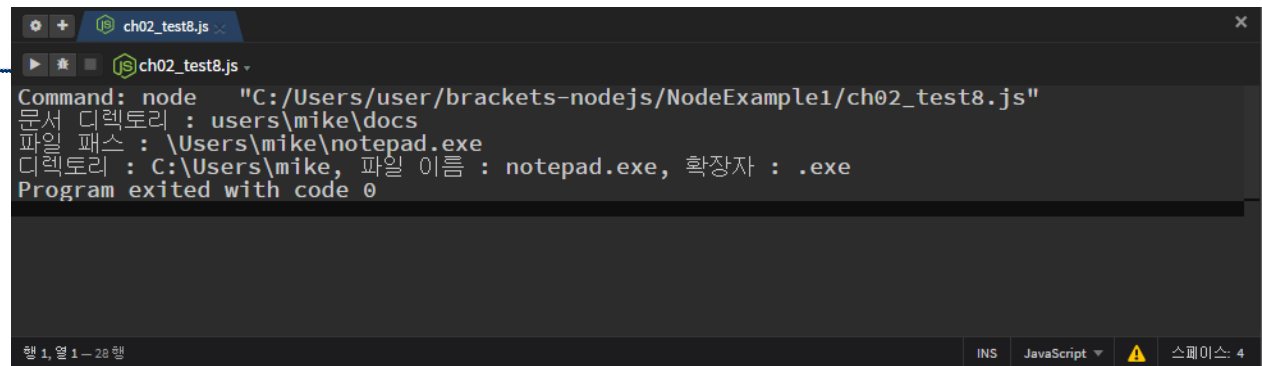
```
var filename = "C:\\Users\\mike\\notepad.exe";
```

```
var dirname = path.dirname(filename);
```

```
var basename = path.basename(filename);
```

```
var extname = path.extname(filename);
```

```
console.log('디렉터리 : %s, 파일 이름 : %s, 확장자 : %s',  
            dirname, basename, extname);
```



The screenshot shows a terminal window with the following content:

```
ch02_test8.js  
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch02_test8.js"  
문서 디렉토리 : users\mike\docs  
파일 패스 : \Users\mike\notepad.exe  
디렉토리 : C:\Users\mike, 파일 이름 : notepad.exe, 확장자 : .exe  
Program exited with code 0
```

The terminal window has a title bar with "ch02\_test8.js" and a close button. The status bar at the bottom shows "행 1, 열 1 - 28 행", "INS", "JavaScript", and "스페이스: 4".

## ch02\_test8.js path 내부 모듈 사용하기

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

// 디렉토리 합치기
var directories = ["users", "sunny", "docs"];
var docsDirectory = directories.join(path.sep);
console.log('문서 디렉토리 : %s', docsDirectory);

//디렉토리명과 파일명 합치기
var curPath = path.join('/Users/sunny', 'notepad.exe');
console.log('파일 패스 : %s', curPath);

// 패스에서 디렉토리, 파일명, 확장자 구분하기
var filename = "C:\\\\Users\\sunny\\notepad.exe";
var dirname = path.dirname(filename);
var basename = path.basename(filename);
var extname = path.extname(filename);
console.log('디렉토리 : %s, 파일 이름 : %s, 확장자 : %s', dirname, basename, extname);
```

