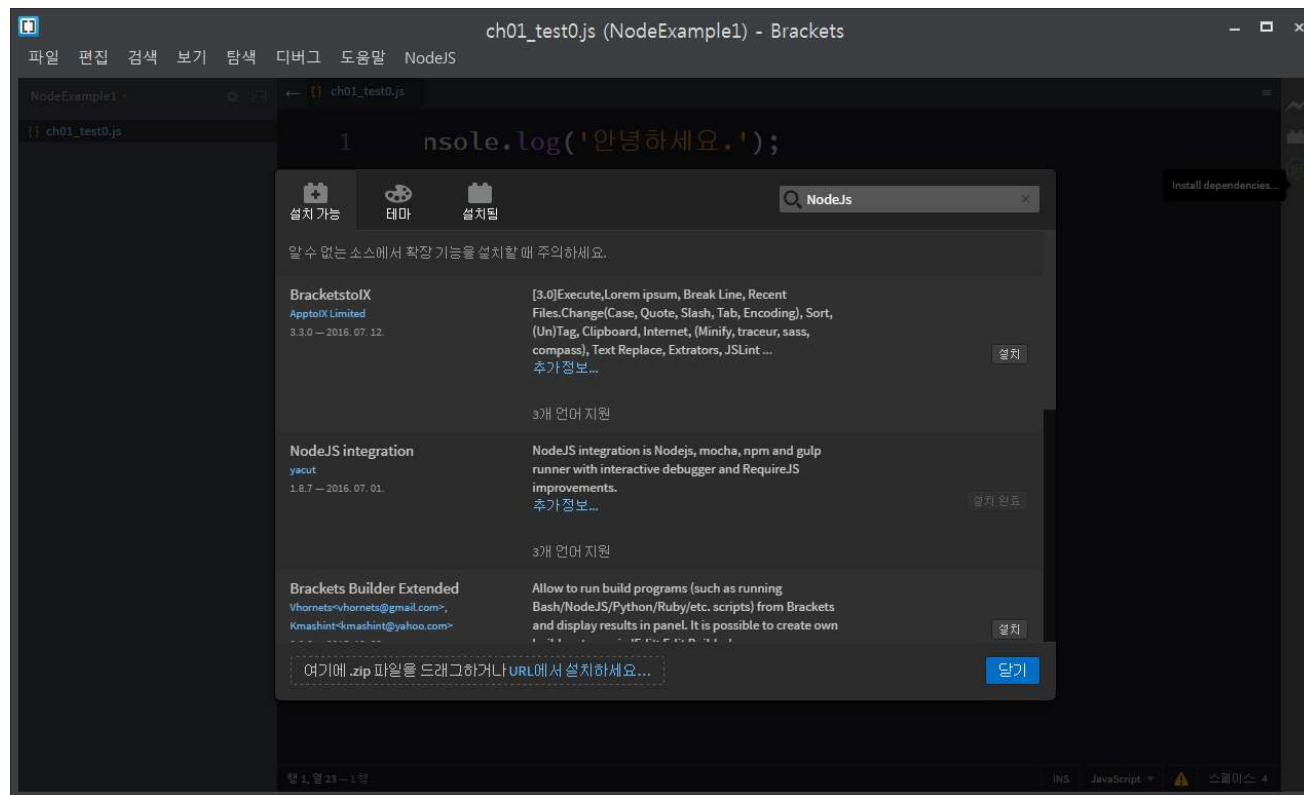


Ch 02

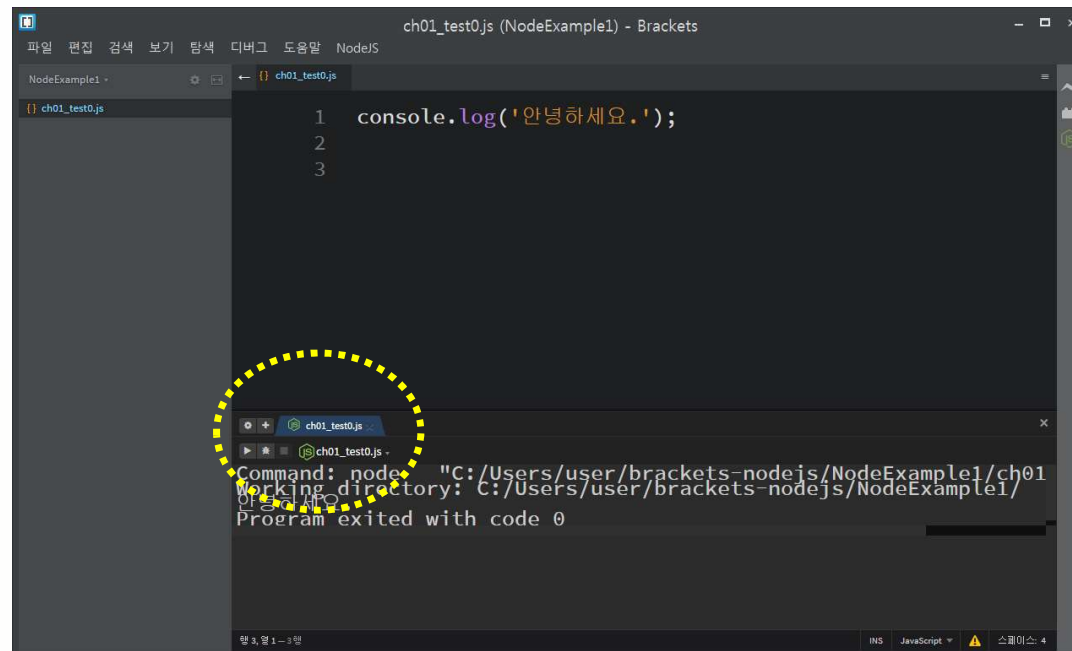
02-1 첫 번째 노드 프로젝트 만들기

- 파일>폴더 열기 메뉴를 누르고 brackets_nodejs 폴더 아래의 NodeExample 폴더 지정
(기존에 지정되어 있으면 그대로 사용)
- 확장 기능 중에서 NodeJs Integration 설치
 - brackets에서 바로 실행

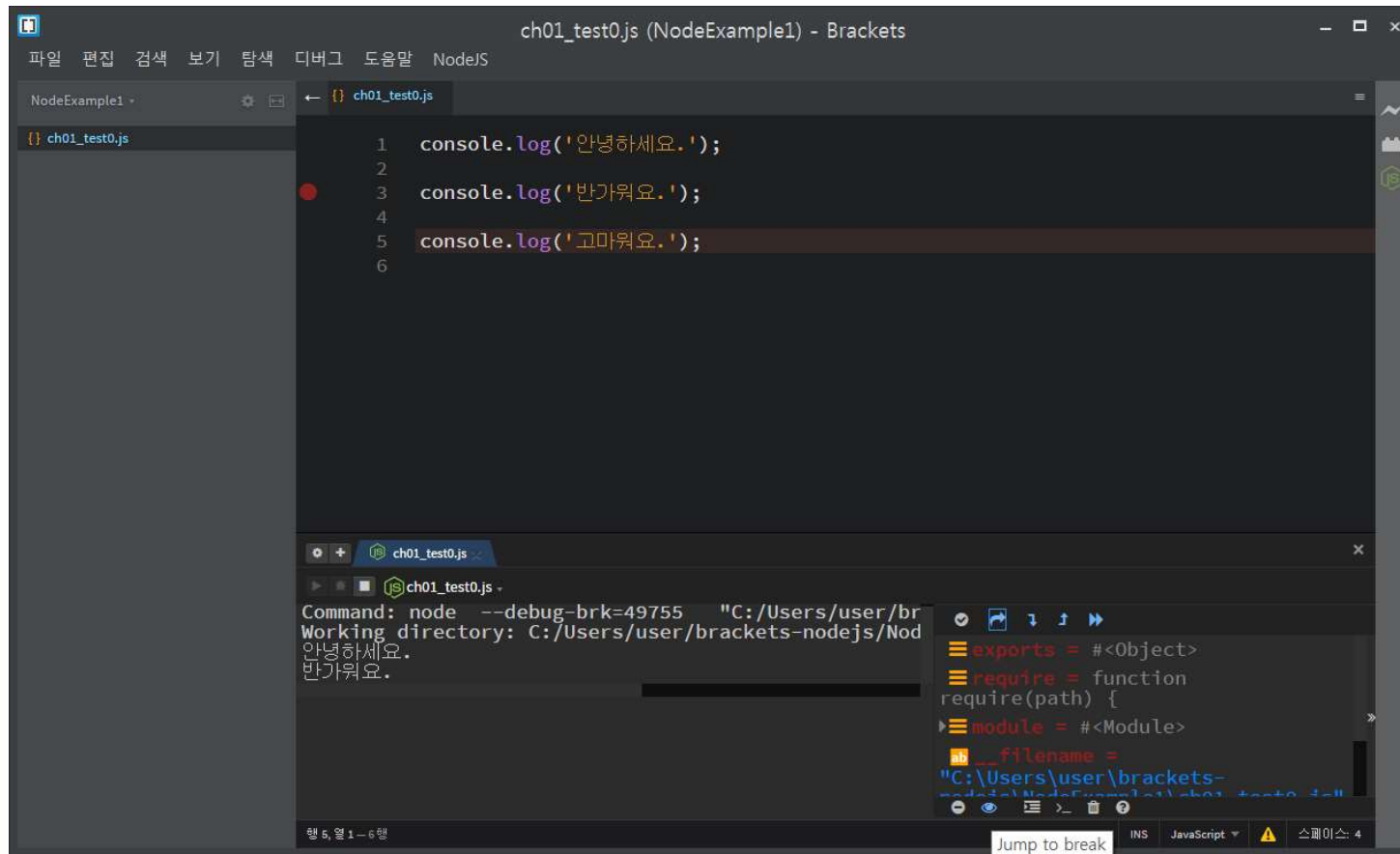


- node 실행 방법

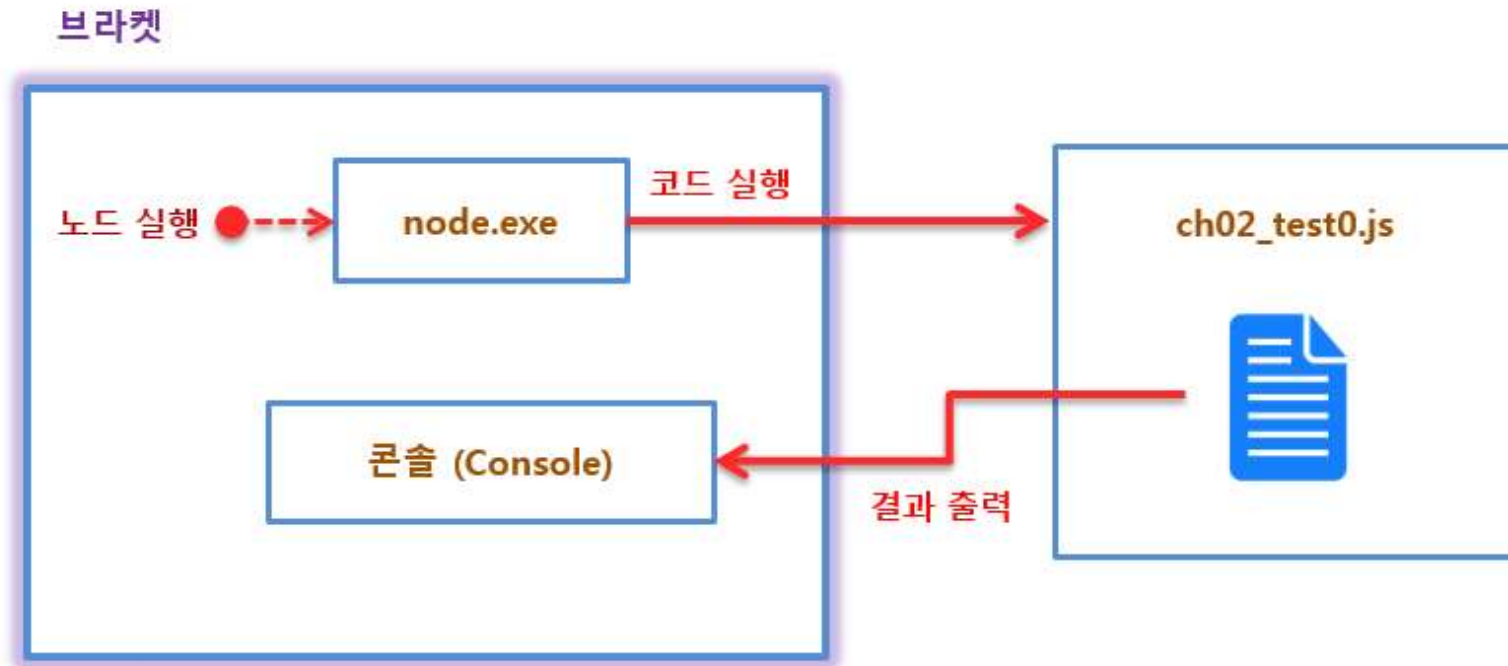
- ① 자바스크립트 파일은 명령프롬프트를 열고 node 실행 파일을 이용해 실행
- ② 브라켓에서 NodeJs Integration 확장기능을 이용해 실행 가능
 - 왼쪽 프로젝트 영역에서 파일 선택 후 오른쪽 마우스 버튼 눌러 Add to Node.js 메뉴 선택
 - 아래쪽에 탭 보이면 새로 만든 파일 지정 후 실행



- 디버깅 버튼을 눌러 디버깅 가능



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



02-2 콘솔에 로그 뿌리기

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

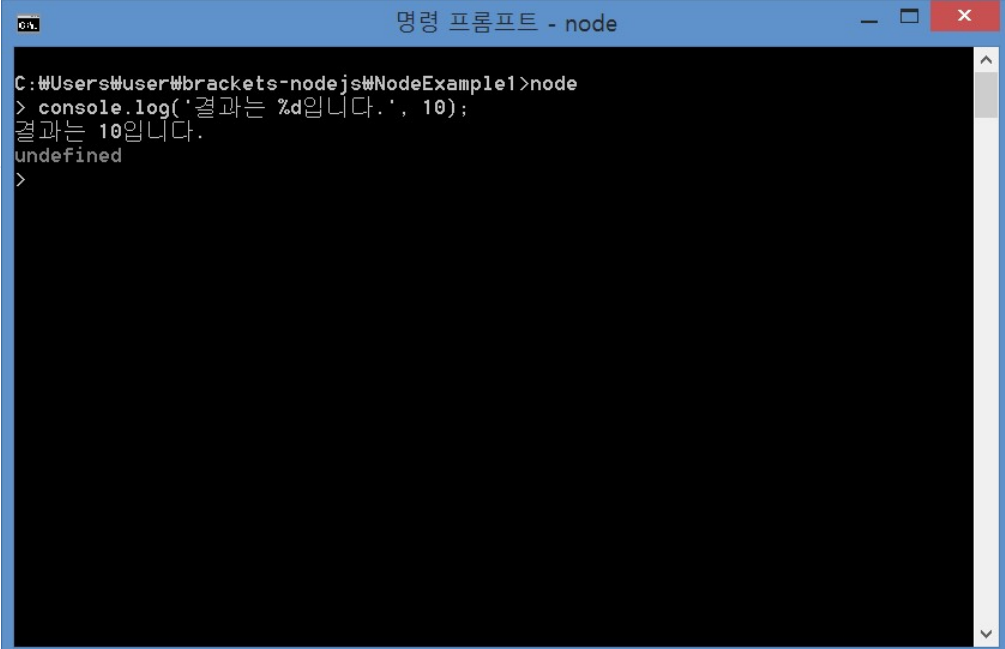
```
% cd brackets_nodejs\NodeExample
```

```
% node ch02_test1.js
```


- 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
 - 어디서든 사용할 수 있는 전역 객체(global object)
 - 소스코드의 어디에서도 사용 가능

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

- log() 메소드
 - 로그를 출력할 수 있도록 함
 - %d, %s, %j : 숫자, 문자열, JSON 객체를 문자열로 변환하여 출력

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

- dir, time, timeEnd 메소드

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.

- 출력값 undefined와 null

- undefined
 - type 또는 value
 - 값이 정의되지 않은 경우 즉, 존재하지 않는 경우
- null
 - 값이 의도적으로 비어있는 경우

코드 실행한 시간 체크하기(예제)

12/37

- 왼쪽 [NodeExample] 프로젝트 선택 후 오른쪽 마우스 버튼 눌러 [파일 만들기] 메뉴 선택
- 파일 이름으로 ch02_test2.js 입력
- 새로 만들어진 파일 안에 아래 내용 코드 입력
- 실행

```
var result = 0;
```

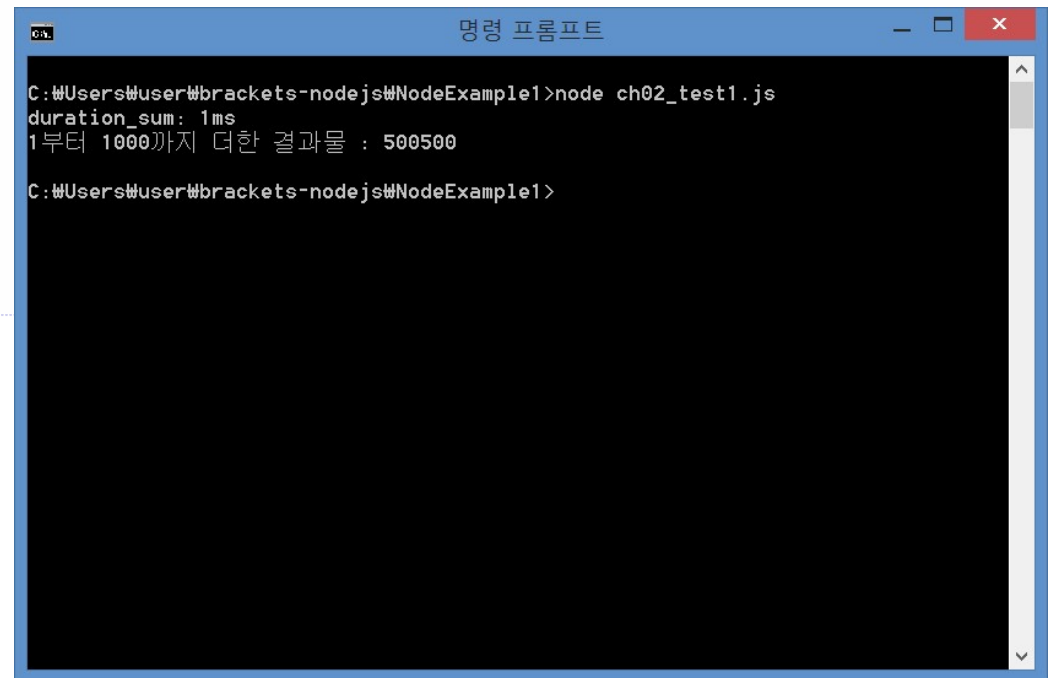
```
console.time('duration_sum');
```

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

```
}
```

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

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



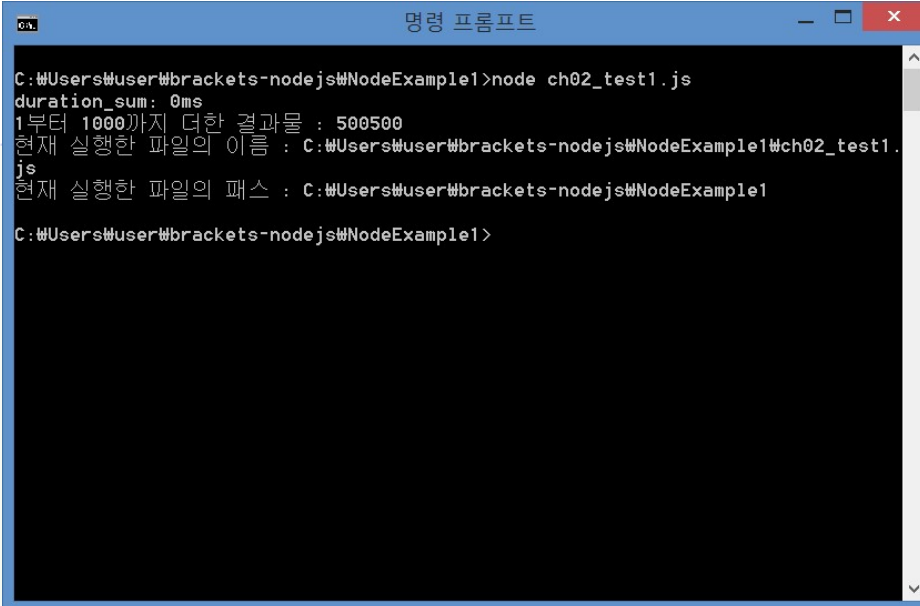
```
명령 프롬프트  
C:\Users\User\brackets-nodejs\NodeExample1>node ch02_test1.js  
duration_sum: 1ms  
1부터 1000까지 더한 결과물 : 500500  
C:\Users\User\brackets-nodejs\NodeExample1>
```

- `__filename`, `__dirname`
 - 전역 변수
 - `__filename`: 실행한 파일의 이름
 - `__dirname`: 실행한 파일이 있는 폴더 이름
- `console.dir` 함수 이용하여 객체 정보 출력
- 자바스크립트의 객체 생성
 - 총 3가지 방식
 - 객체 리터럴(Object Literal) 방식
 - `{ }` 를 이용해서 만듦
 - 형태: `{ 속성이름 : 속성값, 속성이름 : 속성값 }`

... 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>
```

02-3 PROCESS 객체 살펴보기

- process 객체
 - 프로그램 실행할 때 생성되는 프로세스 정보를 다루는 객체

속성/메소드 이름	설명
argv	<ul style="list-style-type: none">• 프로세스를 실행할 때 전달되는 파라미터(매개변수) 정보• 배열 객체
env	<ul style="list-style-type: none">• 사용자 정의 환경 변수 정보• 시스템 환경 변수 정보는 없음
exit()	<ul style="list-style-type: none">• 프로세스를 끝내는 메소드

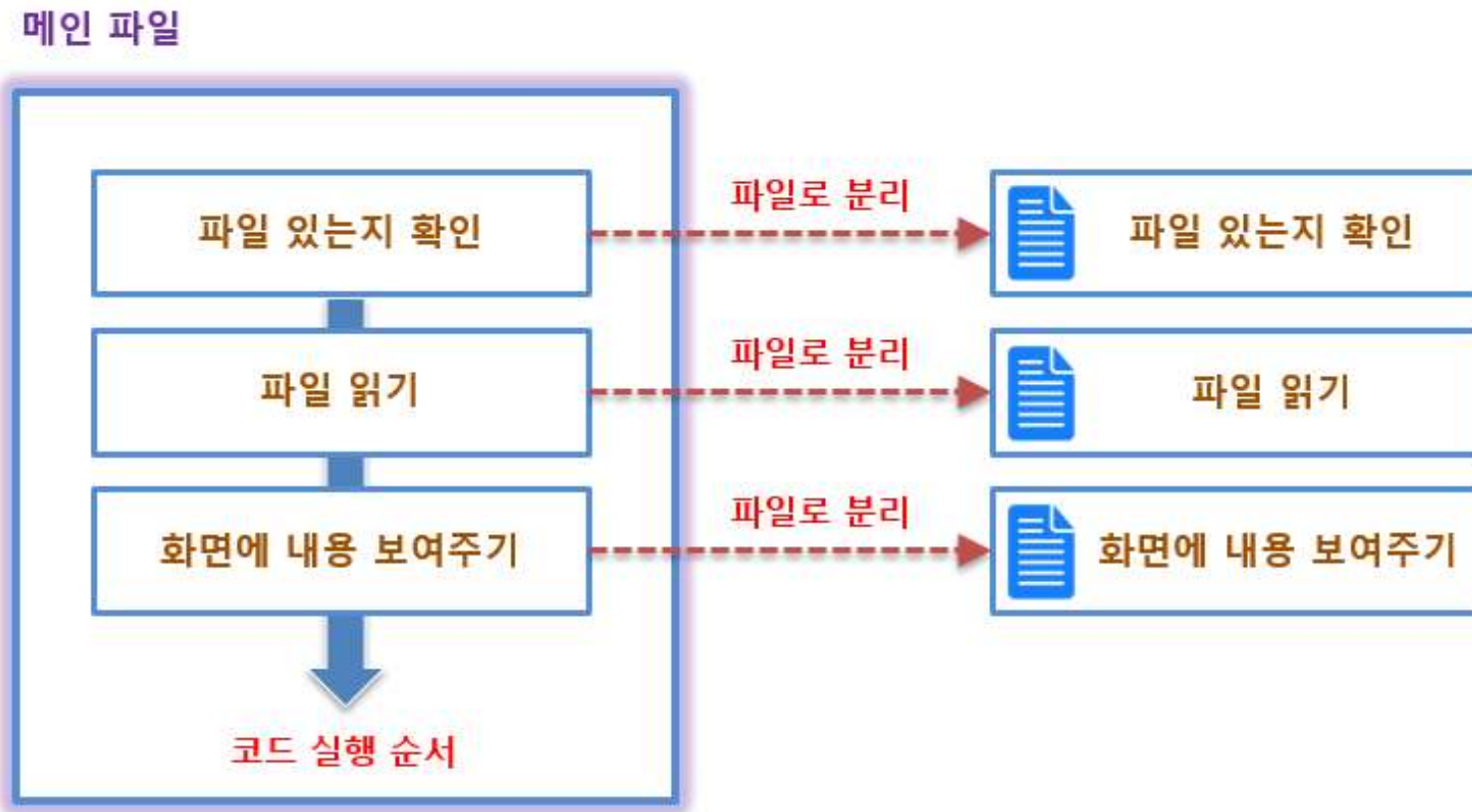
```
console.log('argv 속성의 파라미터 수 : ' + process.argv.length);
console.dir(process.argv);

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

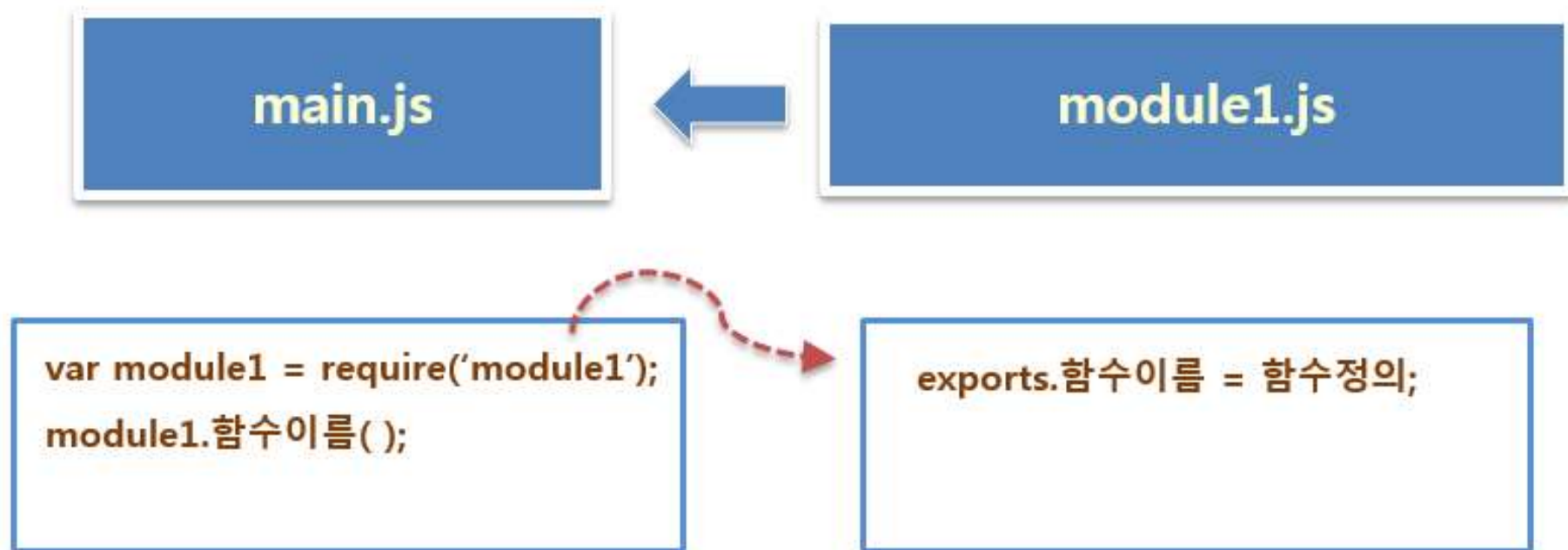

02-4 노드에서 모듈 사용하기

기능을 각각의 파일로 분리하기

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



- 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



- 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.

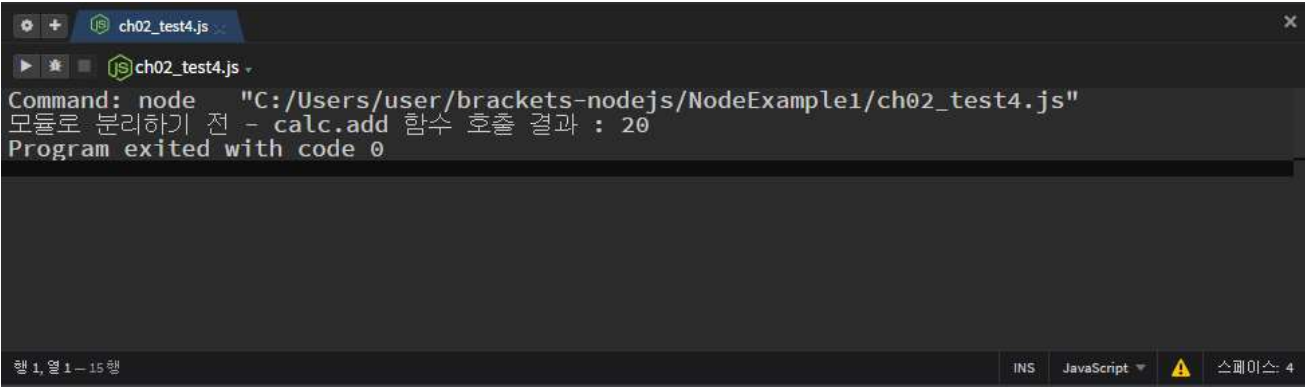
module2.js

```
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(예제)

- 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));
```



```
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
```

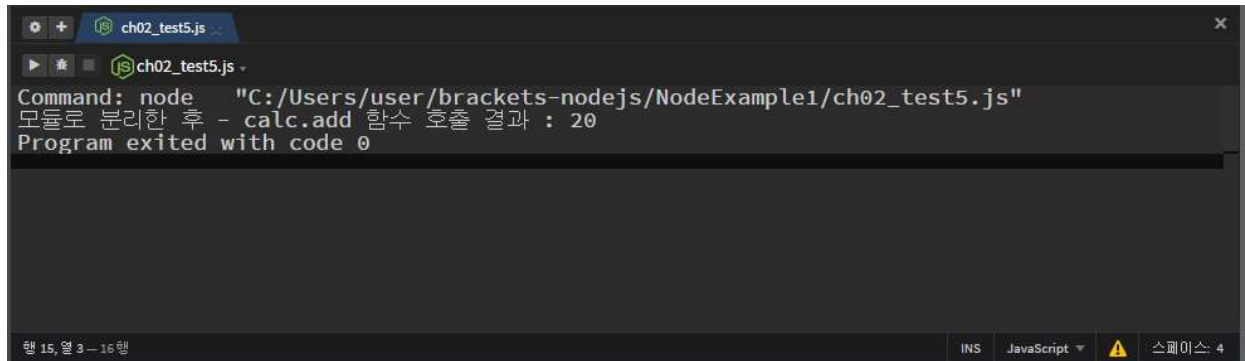
- 모듈 파일 생성, calc.js
- 함수를 exports 객체의 add 속성으로 추가

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

- 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

사용자가 만든 모듈은 상대패스 지정 필요함

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



```
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
```

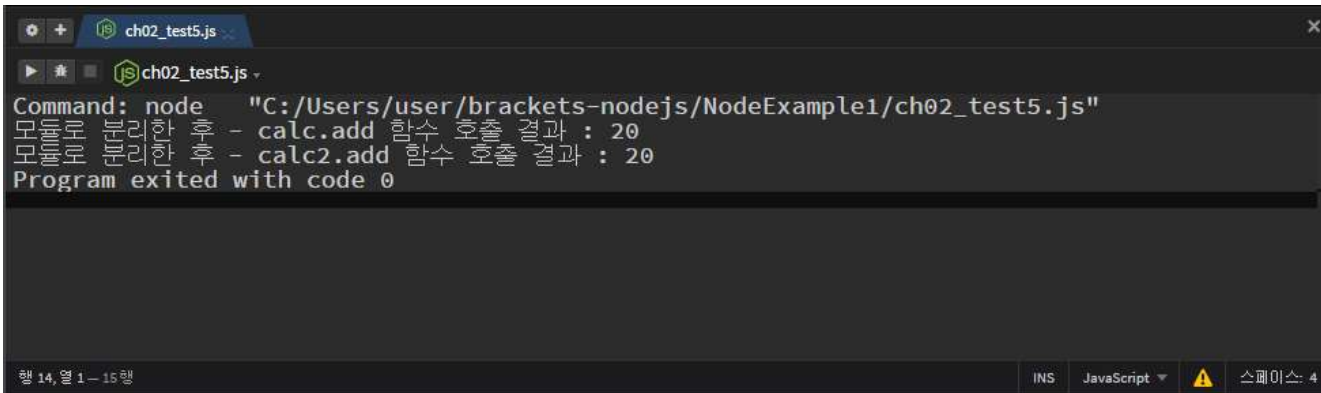
모듈 파일에서 module.exports 사용 (예제)

- 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;
```


- 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));
```



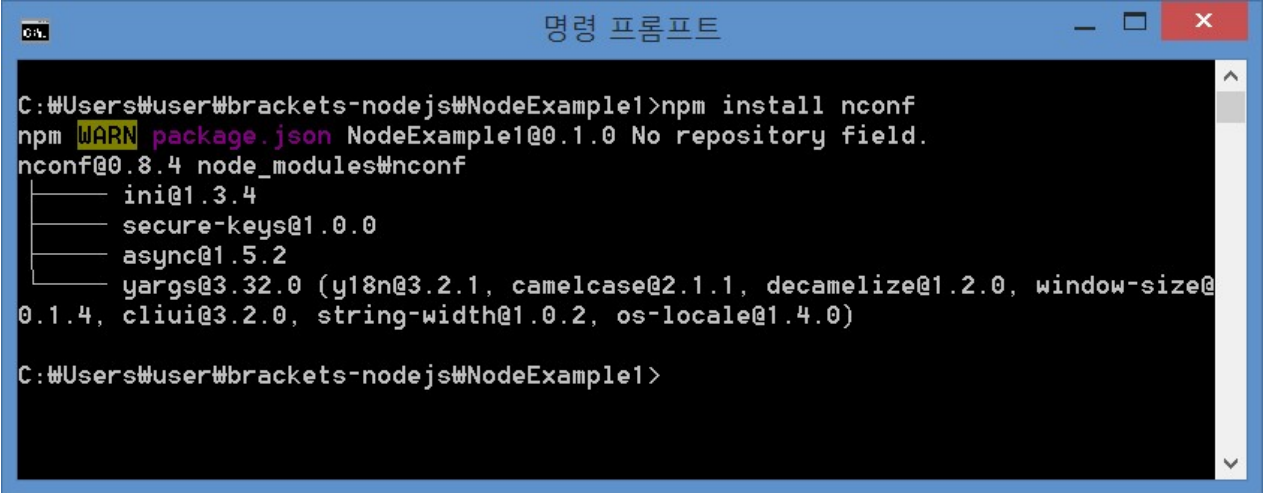
```
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch02_test5.js"  
모듈로 분리한 후 - calc.add 함수 호출 결과 : 20  
모듈로 분리한 후 - calc2.add 함수 호출 결과 : 20  
Program exited with code 0
```

- 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'));
```

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

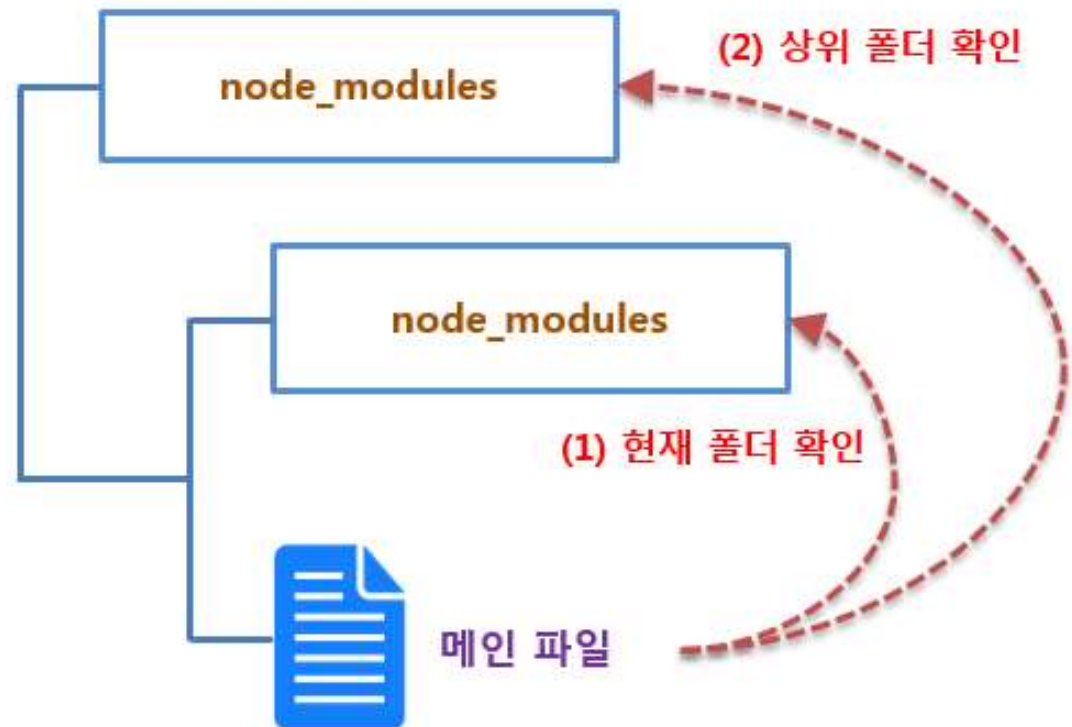
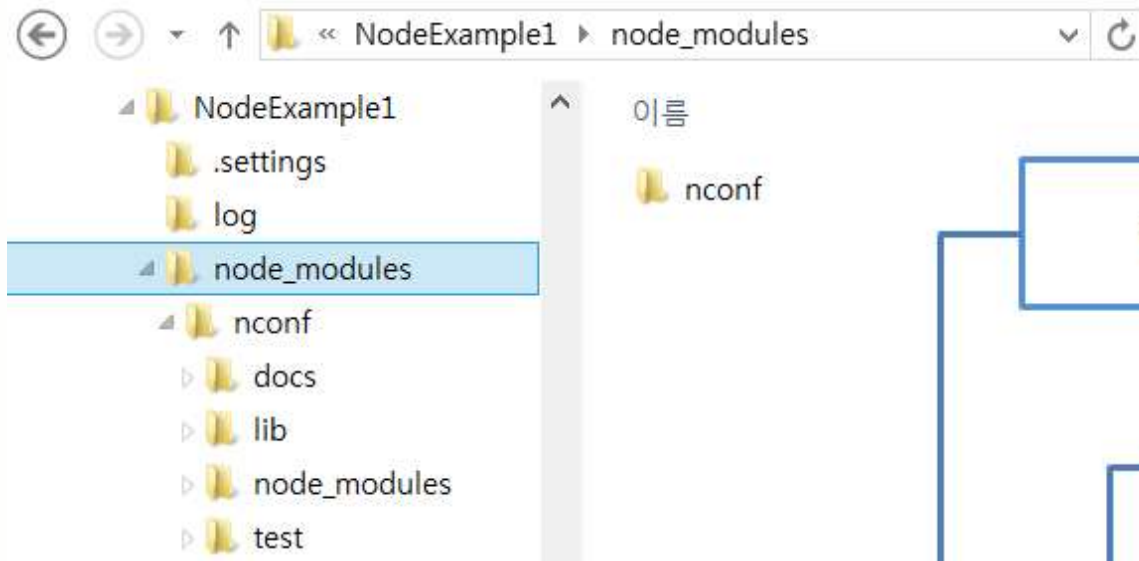
% npm install nconf



```
명령 프롬프트

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>
```

- node_modules 폴더 안에 패키지가 설치됨
- 프로젝트별로 사용 가능하나 프로젝트 상위 폴더에 있어도 사용 가능



- npm으로 설치한 패키지 정보를 확인할 수 있음

```
{
  "name": "NodeExample",
  "version": "0.1.0",
  "description": "NodeExample1",
  "main": "hello-world-server.js",
  "scripts": {
    "test": "echo W\"Error: no test specified! Configure in package.jsonW\" && 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
 - package.json를 복사했을 경우 npm install 명령만으로 package.json 안에 들어있는 패키지 정보를 이용해 패키지 일괄 설치

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

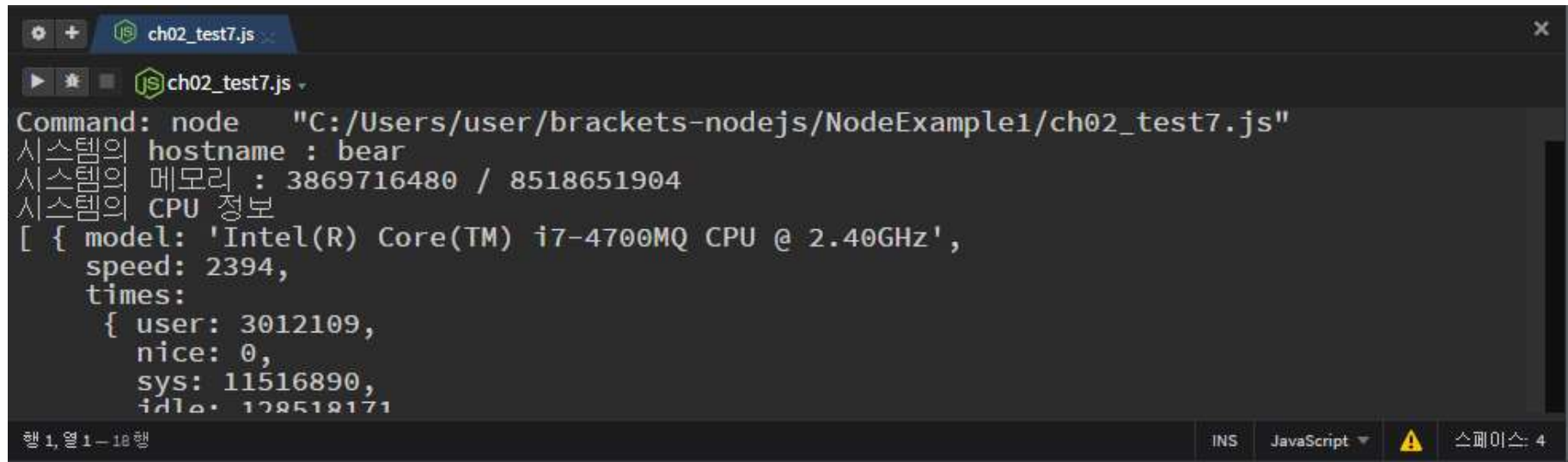
02-5 간단한 내장모듈 사용하기

- 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.

- 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 titled 'ch02_test7.js'. The command 'node "C:/Users/user/brackets-nodejs/NodeExample1/ch02_test7.js"' has been executed. The output displays system information in Korean and English: '시스템의 hostname : bear', '시스템의 메모리 : 3869716480 / 8518651904', and '시스템의 CPU 정보'. The CPU information is a JSON array containing an object with 'model', 'speed', and 'times' properties. The 'times' property is an object with 'user', 'nice', 'sys', and 'idle' properties. The status bar at the bottom indicates '행 1, 열 1-18 행', 'INS', 'JavaScript', a warning icon, and '스페이스: 4'.

```
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch02_test7.js"
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: 128518171 }
}]
```

행 1, 열 1-18 행 INS JavaScript ⚠ 스페이스: 4

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

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

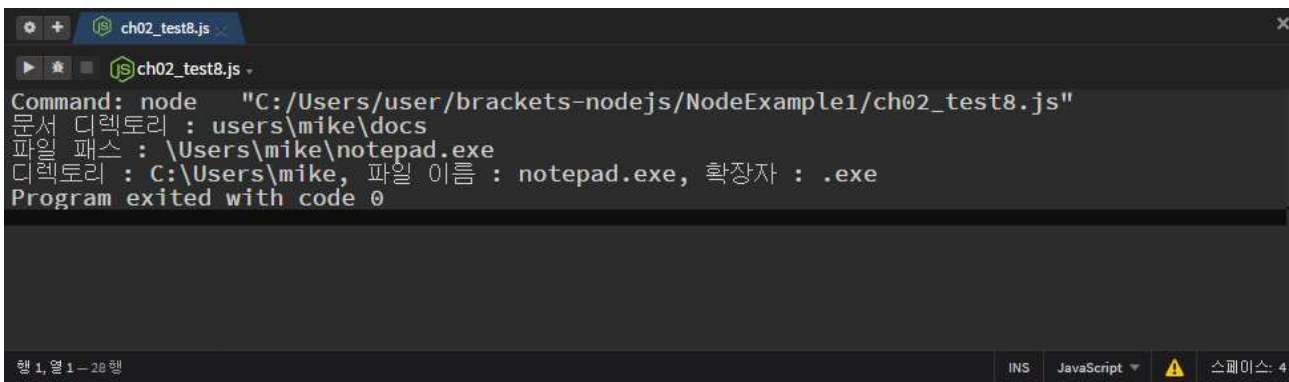
```
var filename = "C:\\Users\\mike\\notepad.exe";
```

```
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);
```



```
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
```

행 1, 열 1 — 28 행

INS JavaScript 스페이스: 4