10/11 강의자료

1교시

01. Exports vs Module.exports

▼ REPL란?

JS는 스크립트 언어로 미리 컴파일 하지 않아도 즉성에서 코드를 실행할 수 있습니다. 스크립트 언어로 대표적인게 python이 있죠?

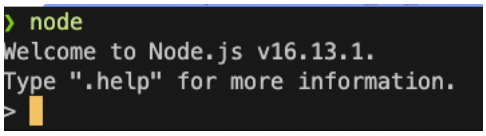
Node도 콘솔을 제공하는데 읽고(Read), 해석하고(Eval), 결과물을 반환하고(Print), 종료할때까지 반복한다(Loop)고 해서 REPL라고 부릅니다.

여러분은 지금까지 code runner를 통해 편리하게 사용했었죠?

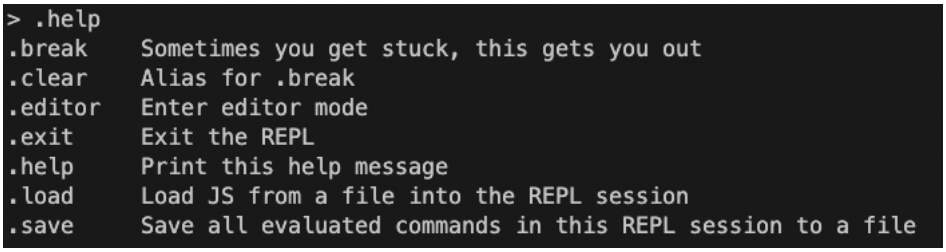
원래는 명령어 창에서 node를 실행하려면 node를 치시고 js코드를 실행해야 합니다.

실습해보겠습니다.

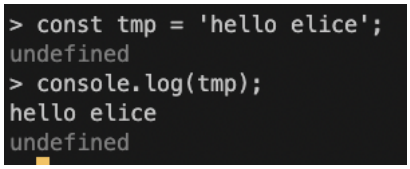
$node



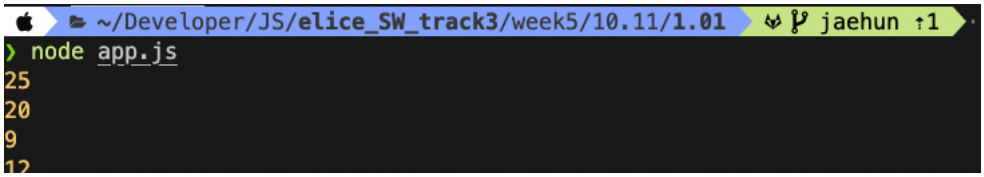
.help를 치면 node에 어떤 명령어가 있는지 알 수 있습니다.



JS를 입력해보겠습니다. 우리가 개발자도구에서 보던 콘솔창과 동일하게 사용하실 수 있습니다.



JS파일을 실행해 보겠습니다.



▼ 모듈이란?

모듈이란 특정한 기능을 하는 함수나 변수들의 집합입니다. 모듈 자체로도 하나의 프로그램이면서 다른 프로그램의 부픈으로도 사용할 수 있죠.

이렇게 기능들과 변수들을 모듈로 만들어 놓으면 재사용에 용이하기 때문에 JS에서 코드를 재사용하기 위해 함수를 만드는 이유와 동일합니다.

모통은 파일 하나가 모듈 하나가 됩니다.

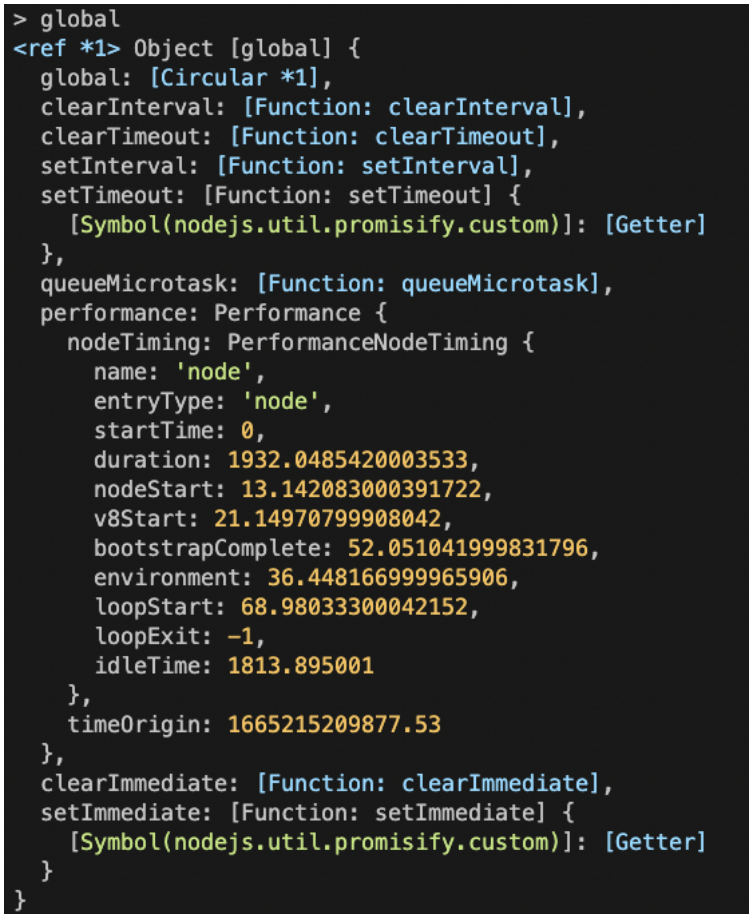
▼ Node 내장 객체 알아보기

브라우저에 window라는 내장객체로부터 뻗어나가는것처럼 node에도 기본적인 내장객체들이 있습니다.

node에서 많이 사용하는 내장객체에 대해서 알아보겠습니다.

global

global객체는 window와 같은 전역 객체로 모든 파일에서 접근할 수 있습니다. window를 생략할 수 있는 것처럼 global도 생략할 수 있죠 위에 require함수도 사실은 global.require에서 global이 생략된 것입니다. 이전에 배웠던 것들과 다른 점은 node에는 DOM, BOM이 없기 때문에 window와 documnt 객체를 사용할 수는 없습니다.



console

우리가 자주사용하던 console.log말고도console객체에는 유용한 메서드들이 있습니다. 대표적인 예시로 시간을 측정하기 위한 console.time('key'), console.tineEnd('key') console.err(e), console.table(arr) 등이 있습니다.

실습을 통해 알아보겠습니다.

```
const str = "abc";
const num = 1;
const bool = true;
const obj = {
    outside: {
        inside: {
            key: "value",
        },
    },
};

// 전체시간 측정 시작
console.time("전체시간");
console.log("로그");
console.log(str, num, bool);
//  error내용 콘솔에 표시
console.error("Error 메시지");

// 배열 요소 테이블 형태로 시각화
console.table([
    { name: "elice", age: 3 },
    { name: "토깽이", age: 20 },
]);

// console.dir(객체, 옵션) : 객체를 콘솔에 표시하며 옵션값으로 colors를 넣으면 색갈이 표시되고, depth을 주면 객체 안 몇단계까지 보여줄 것인지를 결정합니
console.dir(obj, { colors: false, depth: 2 });
console.dir(obj, { colors: true, depth: 1 });

// 함수 시간 측정
```

```
console.time("시간측정");
for (let i = 0; i < 10000; i++) {}
// 함수 시간 측정 끝
console.timeEnd("시간측정");

function b() {
    // Error 위치 찾기 error발생시 에리발생한 위치를 알려줍니다. (자주쓰지는 않지만 error가 어디서 나는 지 모를때 사용해보세요!)
    console.trace("error 위치 추적");
}

function a() {
    b();
}

a();

// 전체시간 측정 끝
console.timeEnd("전체시간");
```

이외에도 setTimeout 같은 브라우저API에 있던 타이머 기능들도 node의 내장 객체로 존재합니다.

▼ exports와 Module.exports의 차이

moduel.exports는 한번에 대입하는 대신

exports는 각각의 변수를 exports객체에 하나씩 넣습니다.

하지만 동일하게 작동하는데요 그 이유는 둘 다 같은 객체를 참조하기 때문입니다.

```
console.log(module.exports === exports); // true
```

▼ 구조분해할당(Destructuring Assignment) 이용

ES6문법으로 배웠었던 구조분해할당를 사용한다면 어떻게 바뀔까요?

02. File System

▼ File System 접근하기

fs모듈은 파일 시스템에 접근하는 모듈입니다. 파일을 생성 / 삭제 / 읽기 / 쓰기 를 할 수 있습니다.

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

// 주의할 점 ; 파일의 경로가 node 명령이 실행하는 콘솔 기준입니다.
fs.readFile("./test.txt", (err, data) => {
    if (err) {
        throw err;
    }
    console.log(data);
    console.log(data.toString());
});

// <Buffer 74 65 73 74 20 74 65 78 74>
// test text

console.log(process.cwd()); // 실행 경로 찾기
```

readFile은 버퍼 형식으로 반환합니다.

하지만 위의 fs는 콜백 형태의 모듈이므로 사용하기 힘들어 Promise 형식으로 바꾸어 사용을 권장합니다.

코드를 통해 알아보겠습니다.

▼ 동기 메서드 비동기 메서드

setTimeout, process.nextTick 외에도 노드는 대부분 메서드를 비동기 방식으로 처리합니다. 하지만 몇몇 메서드는 동기 방식으로도 처리할 수 있는데요 특히 fs 모듈이 그러한 메서드가 많습니다.

어떤 메서드가 동기 방식인지, 비동기 방식인지를 배우고

언제 어떤 메서드를 사용해야 하는지 생각해보세요 :)

```
const fs = require("fs").promises;

console.log("start");
fs.readFile("./test.txt")
    .then((data) => {
        console.log("1:", data);
    })
    .catch((e) => {
        console.log(e);
    });

fs.readFile("./test.txt")
    .then((data) => {
        console.log("2:", data);
    })
    .catch((e) => {
        console.log(e);
    });

fs.readFile("./test.txt")
    .then((data) => {
        console.log("3:", data);
    })
    .catch((e) => {
        console.log(e);
    });

console.log("end");
```

기존에 배운 readFile 메서드는 비동기 방식으로 작동합니다.

백그라운드에 해당 파일을 읽으라고 요청만하고 다음 작어븡로 넘어가죠 그래서 콘솔에 'end'가 먼저 찍히게 됩니다.

파일 읽기가 완료되면 메인 스레드에 등록된 콜백함수가 실행됩니다. 그래서 순서가 우리가 생각하는 대로 안나 올 수도 있는 것이죠

순서대로 출력을 하고싶다면 비동기가 아닌 동기 메서드를 사용하면 됩니다.

```
// 동기 메서드
const fs_sync = require("fs");

console.log("start sync");
let data = fs_sync.readFileSync("./test.txt");
console.log("1: ", data.toString());

data = fs_sync.readFileSync("./test.txt");
console.log("2: ", data.toString());

data = fs_sync.readFileSync("./test.txt");
console.log("3: ", data.toString());

data = fs_sync.readFileSync("./test.txt");
console.log("4: ", data.toString());

data = fs_sync.readFileSync("./test.txt");
console.log("5: ", data.toString());

console.log("end");
```

▼ 버퍼와 스트림 이해하기

파일을 읽거나 쓰는 방식에는 크게 두가지 방식이 있습니다. 버퍼를 이용하는 방식과 스트림을 이용하는 방식이 있는데요.

버퍼링 , 스트리밍 이라는 용어는 많이 접해보셨을 겁니다. 영상을 로딩할 때는 버퍼링 걸린다. 영상을 실시간으로 송출할때는 스트리밍한다 하라고 사용하시죠?

좀 더 들어가보면

버퍼링은 영상을 재생 할 수 있을 때까지 데이터를 모으는 동작이고,

스트리밍은 송출하는 곳에서 받는 곳까지 영상 데이터를 조금씩 전송하는 동작입니다.

스트리밍 하는 과정에서 버퍼링을 할 수도 있죠 전송이 너무 느리면 화면을 내보내기까지 최소한 데이터를 모아야 하고 영상 데이터가 재생 속도보다 빠르게 전동되어도 미리 전송받은 데이터를 저장할 공간이 필요하니깐요

노드의 버퍼, 스트림도 비슷한데요

노드가 파일을 읽을 때 메모리에 파일 크기만큼 공간을 마련해두고 파일 데이터를 저장한 뒤 사용자가 조작할 수 있게 합니다. 이때 메모리에 저장된 데이터가 버퍼입니다.

하지만 버퍼는 파일 크기만큼 메모리가 사용되기 때문에 메모리를 많이 잡아 먹는다는 단점이 있습니다.

많은 이용자가 동시에 사용하거나 한번에 많을 것을 읽을 때 문제가 발생하겠죠?

이러한 단점을 해결하려고 버퍼의 크기를 작게 만든 후 여러번에 나누어서 전송하는 방식이 등장했습니다.

100MB의 버퍼를 전송하던 것을 10MB 버퍼로 나누어 10번에 걸쳐 전송하는 것이죠

이 방식이 바로 스트림입니다. 조금씩 나눠서 보내느 조각을 chunk이라고 부릅니다.

03. Node.js로 간단한 웹서버 만들기

▼ 요정과 응답 이해하기

클라이언트에서 서버로 요청(Request)을 보내고, 서버에서는 요청의 내용을 읽고 처리한 뒤 클라이언트에 응답(response)를 보냅니다.

따라서 서버는 요청이 없으면 아무것도 하지 않아요. 요청과 응답은 이벤트 방식처럼 클라이언트로부터 요청이 왔을 때 어떤 작업을 수행할지 미리 이벤트 리스너를 등록해놔야 합니다.

이때 http 서버가 있어야 웹 브라우저의 요청을 처리할 수 있기 때문에 http 모듈을 사용해서 node.js 서버를 만들어 줄 수 있습니다.

어떻게 만드는지는 실습을 통해 알아보겠습니다.

▼ (error)만약 포트가 사용중이라면?

Error: listen EADDRINUSE: address already in use :::8080

이러한 error를 만나실 수 있습니다. 이것은 다른 프로세스가 포트를 사용하고 있다는 뜻이니

누가 사용하고 있는지를 먼저 찾아보고

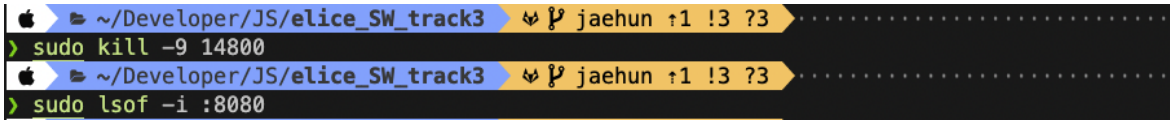
mac : ❯ **sudo lsof -i :[포트번호]**

window : ❯ **netstat -ano | findstr [포트번호]**

필요없다면 삭제 후 실행 해주시면 됩니다.

mac : ❯ **sudo kill -9 [PID]**

window : ❯ **taskkill /f /pid [PID]**



▼ http 모듈이란?

http웹 서버와 관련된 모든 기능을 담은 모듈입니다.

createServer()메서드를 사용해서 server객체를 생성할 수 있습니다.

server객체의 메서드로는

가 있습니다.

server에 사용되는 이벤트로는

가 있습니다.

request 이벤트 같은 경우 createServer()메서드 안에 매개변수로서 사용할 수 있습니다.

request이벤트 리스너의 두번째 객채로 response가 전달 됩니다. 이 response를 통해 요청에 대한 응답을 작성해줄 수 있습니다.

response의 메서드로는

헤더 정보 내보내기

.writeHead() : response 객체의 메소드에서 헤더 정보를 응답에 작성해서 내보내는 것입니다.

첫번째 인자는 상태 코드를 지정하고 두번째인수에 헤더 정보를 연관 배열로 정리한 입니다.

컨텐츠 내보내기

.write() : 바디 부분이 되는 콘텐츠를 작성

컨텐츠 종료

.end() : 내보낼 내용이 완료되면 콘텐츠 출력을 완료한다는 end 메서드를 새용합니다.

end안에도 마지막으로 내보낼 콘텐츠를 쓸 수 있습니다.

▼ 콘솔에 왜 두번 찍히죠?

크롬같은 브라우저에서 파비콘이 뭔지 html에서 유츄할 수 없다면 서버에 파비콘 정보에 대한 요청을 한 번 더 보내기 때문입니다.

파비콘이란? : 파비콘이란 인터넷 웹 브라우저의 주소창에 표시되는 웹사이트나 웹페이지를 대표하는 아이콘입니다.

2교시

04. Node.js html 요소 렌더링하기

▼ HTML 렌더링 하기

우리는 http 모듈을 통해 웹서버를 생성하는 방법과 그 안에 콘텐츠를 넣어서 요청에 응답하는 방법을 배웠습니다.

이전에는 fs 모듈을 통해 파일을 읽고 쓰는 방법에 대해서 배웠었죠?

그렇다면 응답으로 html을 보내주려면 어떻게 해야할까요?

먼점 http 모듈을 통해 웹서버를 생성하고

fs 모듈을 통해 보내줄 html 파일을 읽어오고 response안에 콘텐츠로써 넣어서 요청에대한 응답을 보내주면 되겠죠?

실습 코드를 통해 배워보겠습니다.

▼ 쿠키와 세션 이해하기

클라이언트에서 보내는 요청에는 한 가지 큰 단점이 있습니다. 바로 누가 요청을 보내는지 모른다는 것입니다. IP 주소나 브라우저의 정보를 가져올 수는 있지만 여러 컴퓨터가 IP를 공유하거나 한 컴퓨터를 여러사람에서 사용할 수도 있습니다. 그래서 로그인 기능이라는 것이 생긴것이죠

로그인 기능을 제대로 이해하려면 쿠키와 세션에 대해서 알고 있어야 합니다.

여러분이 사용하는 대부분 웹사이트에서 로그인 하고 새로고침을 해도 로그아웃이 되지않죠? 그 이유는 클라이언트에서 서버에게 여러분이 누구인지 알려주고 있기 때문입니다.

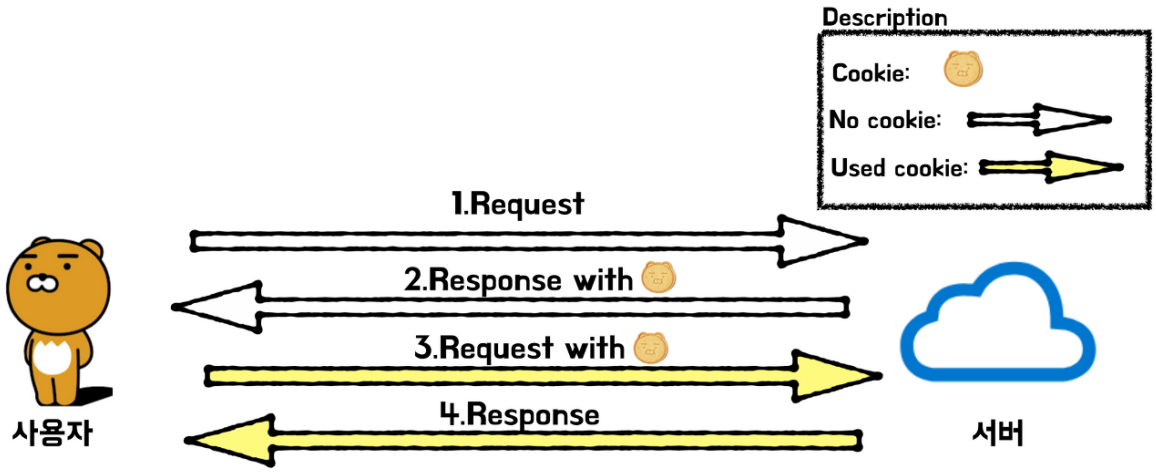
클라이언트는 어떻게 서버에게 알려줄까요? 이때 사용하는 것이 쿠키입니다.

쿠키는 key:value 쌍으로 이루어진 작은 데이터 파일입니다.

로그인할 때 서버는 클라이언트에게 쿠키를 Header에 담아 함께 응답을 보내줍니다.

그리고 클라이언트는 이 쿠키를 저장해두었다가 다음 요청에 header에 동봉해서 같이 보냅니다.

서버는 요청에 들어있는 쿠키를 읽고 사용자를 파악합니다.



즉, 쿠키가 여러분의 정보를 가지고 있는거죠 그래서 개인정보 유출 방지를 위해 주기적으로 쿠키를 지우라고 권고하지만 저도 잘 안 합니다,,ㅎ,,,(귀찮,,,,,,)

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

http.createServer((req, res) => {
    console.log(req.url, req.headers.cookie);
    res.writeHead(200, {
        cookie: "mycookie=test",
        "Content-Type": "text/html; charset=utf-8",
    });
    res.write("<h1>쿠키는 브라우저에게 양보하세요</h1>");
    res.end();
}).listen(8080, () => {
    console.log("8080대기중");
});
```

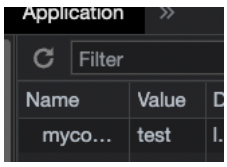
```
/ undefined
/favicon.ico mycookie=test
```

이렇게 응답을 받는 것을 아실 수 있습니다.

첫번째 요청에는 어떠한 쿠키도 없다가 서버가 응답헤더에 쿠키를 심으라고 브라우저에게 말해주었으므로 브라우저는 mycookie=test라는 쿠키를 가지고 있습니다. 그래서 다음 번 요청에는 쿠키를 확인할수 있죠.

실제 로그인 과정은 단순 쿠키를 저장하는게 아닌 쿠키로 사용자를 식별할 수 있게 해야하고, 쿠키의 유효기간을 둬야하기 때문에 조금 더 복잡합니다. 차차 배워가실 겁니다 🙂

하지만 쿠키는 보안에 취약합니다.



바로 노출이 되기 때문이죠..

이러한 단점을 보안하고자 세션이 나오게 된 것입니다.

세션은 서베에 사용자 정보를 저장하고 클라이언트와는 세션 아이디로만 소통합니다. 세션을 위해 사용하는 쿠키를 세션쿠키라고 합니다.

사용자정보를 브라우저에 저장하는 쿠키와달리 세션은 사용자 정버를 웹서버에 저장 한 후 세션 쿠키를 통해서만 사용자를 식별하죠

쿠키, 세션, 토큰 모두 비슷한 동작을 하는 개념들이니 헷갈리실겁니다. 하지만 백엔드 트랙을 계속 수강하시다보면 많이 접하실 개념입니다 너무 걱정하지 마세요 😊

05. Process 객체

▼ Process객체

Process 객체는 현재 실행되고 있는 노드 프로세스에 대한 정보르 담고 있습니다.

하나씩 알아보면

```
process.env;           // 컴퓨터 환경과 관련된 정보를 가진 객체
process.version;       // node.js의 버전
process.versions;      // node.js와 연관된 프로그램들의 버전을 가진 객체
process.arch;          // 프로세서의 아키텍처(arm/ia32/x64)
process.platform;      // 플랫폼(win32/linux/sunos/freebsd/darwin)
process.memoryUsage(); // 메모리 사용 정보를 가진 객체
process.uptime();      // 현재 프로그램이 실행된 시간

process.pid; // 현재 프로세스의 아이디
process.execPath // 노드의 경로
process.cwd(); // 현재 프로세스가 실행되고 있는 위치
process.cpuUsage() // 현재 CPU 사용량

process.nextTick() // 콜백함수 우선 처리
```

이러한 기능들이 있고

이 중 process.nextTick()를 추가설명하면

```
setTimeout(() => {
    // 3등
    console.log("timeout");
}, 0);

process.nextTick(() => {
    // 1등
    console.log("nextTick");
});

Promise.resolve().then(() => console.log("promise")); // 2등
```

nextTick를 다른 콜백함수보다 더 우선적으로 처리합니다.

순서가 nextTick → promise → 다른 콜백함수 순으로 처리됩니다.

그 이유는 nextTick과 Promise.resolve()로 받은 콜백함수는 마이크로태스크 큐에 들어가서 이들을 우선적으로 처리하고 태스크 큐를 처리하기 때문입니다.