require(), exports

- 모듈을 불러오기 위해 **require() 함수** 사용
- 연결하고자 하는 모듈의 특정 값을 **다른 모듈로 넘겨주고 싶을 때** 사용

```
// foo.js
const a = 10

// bar.js
console.log(a); // 오류 발생, a is not defined

// bar_2.js
const foo = require('./foo') // require 함수 사용, foo.js 가져옴
console.log(foo.a); // 여전히 오류 발생, a is not defined

// foo_2.js
exports.a = 10 // exports 사용

// bar_3.js
const foo = require('./foo')
console.log(foo.a) // 정상 실행
```

- module.exports 사용
  - 주로 객체, 함수형태의 **모듈**을 연계할 때 사용
  - 먼저 모듈로 연계할 함수를 정의
  - 파일 끝에 module.exports에 모듈로 만들 함수를 지정

```
// bar.js
const foo = require('./foo')
const checknum = require('./fun')

console.log(foo.a);
checknum(10);

// foo.js
exports.a = 10

// fun.js
function check(num) {
    if (num % 2) {
        console.log('홀수');
    }
    console.log('짝수');
}

module.exports = check; // 해당 함수를 export
```

모듈로 만들기

- Node.js로 자바스크립트 코드를 모듈로 만드는 것이 가능
  - 특정한 기능을 하는 함수나 변수들의 집합
  - 모듈로 만들면 여러 프로그램에서 재사용 가능

- 한 코드를 부분으로 나눈다
    - 변수를 정의해놓은 모듈
    - 함수를 정의해놓은 모듈

HTTP 모듈로 웹서버 만들기

HTTP 모듈

- Node.js에서 가장 기본적인 웹 모듈, HTTP 웹 서버를 생성하는 것과 관련된 모든 기능 담당
- Server 객체
    - http 모듈에서 가장 중요한 객체
    - http 모듈의 **createServer()** 메서드를 사용해 **server 객체 생성**
    - 메서드
        - **listen**(port[, callback]) : 서버를 실행
        - close() : 서버를 종료
    - 이벤트
        - request : 클라이언트가 요청할 때 발생
        - connection : 클라이언트가 접속할 때 발생
        - close : 서버가 종료될 때 발생
        - checkContinue : 클라이언트가 지속적인 연결을 하고 있을 때 발생
        - upgrade : 클라이언트가 HTTP 업그레이드를 요청할 때 발생
        - clientError : 클라이언트에서 오류가 발생할 때 발생

요청과 응답

- 서버와 클라이언트의 관계
    - 클라이언트가 서버로 **요청(request)**을 보냄
    - 서버는 요청을 처리
    - 처리 후 클라이언트로 **응답(response)**을 보냄

노드로 HTTP 서버 만들기

createServer() 메서드

- http 요청에 응답하는 노드 서버
    - createServer로 요청 이벤트에 대기
    - **req 객체**는 요청에 관한 정보가, **res 객체**는 응답에 관한 정보가 담겨 있음

```
// createServer.js
const http = require('http');

http.createServer((req, res) => {
    // 어떻게 응답할지 작성
})
```

8080 포트에 연결

- res 메서드로 응답 보냄

    - write 응답 내용을 적고
    - end로 응답 마무리(내용을 넣어도 됨)
- listen(포트) 메서드로 **특정 포트**에 연결

```
//server1.js
const http = require('http');

http.createServer((req,res) => {
    res.write('<h1>Hello Node!</h1>');
    res.end('<p>Hello Server!</p>');
}).listen(8080, () => {
    console.log('8080번 포트에서 서버 대기 중입니다!');
})
```

8080 포트로 접속하기

- 스크립트를 실행하면 8080 포트에 연결됨
    - localhost:8080 또는 http://127.0.0.1:8080에 접속

파일 시스템 접근하기

- fs : 파일 시스템에 접근하는 모듈
    - 파일/폴더 생성, 삭제, 읽기, 쓰기 가능
    - 웹 브라우저에서는 제한적이었으나 노드는 권한을 가지고 있음
    - ex) fs.readFile(filename, [options], callback);

```
// fs_readFile.js
var fs = require('fs');

fs.readFile('./fs_readFile.js', function (err, data) {
    if (err) {
        throw err;
    }
    console.log(data.toString());
});

// readFile.js
const fs = require('fs');

fs.readFile('./readme.txt', (err, data) => {
    if (err) {
        throw err;
    }
    console.log(data);
    console.log(data.toString());
})
```

FS로 파일 만들기

```
// writeFile.js
const fs = require('fs');

fs.writeFile('./writeme.txt', '글이 입력됩니다', (err) => {
```

```
    if (err) {
        throw err;
    }
    fs.readFile('./writeme.txt', (err, data) => {
        if (err) {
            throw err;
        }
        console.log(data.toString());
    })
})
```

HTML 읽어서 전송하기

- write와 end에 문자열을 넣는 것은 비효율적
  - fs 모듈로 html을 읽어서 전송
  - write가 버퍼도 전송 가능
  - server2.html 문서 작성 후 fs 모듈을 이용해 HTML 문서를 웹서버로 전송

```
<!-- server2.html -->
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "utf-8" />
        <title> Node.js 웹 서버 </title>
    </head>
    <body>
        <h1>
             Node.js 웹 서버
        </h1>
        <p>
            만들 준비되셨나요?
        </p>
    </body>
</html>
```

- 새로운 방식

```
//server2.js
const http = require('http');
const fs = require('fs');

http.createServer((req, res) => {
  fs.readFile('./server2.html', (err, data) => {
    if (err) {
      throw err;
    }
    res.write(data);
    res.end();
  });
}).listen(8080, () => {
  console.log('8080번 포트에서 서버 대기중입니다');
});
```