KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

자바스크립트 기초 문법과 모듈

[KB] IT's Your Life

KB 국민은행

1 화살표 함수

함수의 정의

```
function 함수명(인수, ...) {
   // 함수 로직

   return 반환값;
}
```

chapter02/sec01/func01.js

```
function getTriangle(base, height) {
  return base * height / 2;
}

console.log('삼각형의 면적:' + getTriangle(5, 2));
```

```
삼각형의 면적:5
```

함수 리터럴 표현으로 정의

- 익명함수
- 변수에 배정해서 사용

chapter02/sec01/func02.js

```
var getTriangle = function(base, height) {
  return base * height / 2;
};

console.log('삼각형의 면적:' + getTriangle(5, 2));
```

```
삼각형의 면적:5
```

1 화살표 함수

자바에서는 ->, js에서는 =>

화살표 함수 익명 함수의 간편한 표현식

람다 함수==익명함수==간단한 익명함수

- 다른 언어에서는 람다 함수라고 함

```
(인수, ...) => { 함수 본체 }
```

형식 : (매개변수 나열) => { 코드 };

- 인수가 1개인 경우 괄호 생략 가능
  - 인수 => { 함수 본체 }
- 함수 본체가 1줄인 경우 { } 생략 가능, return 키워드 생략 가능

```
let getTriangle = (base, height) => {
  return base * height / 2;
};

console.log('삼각형의 면적:' + getTriangle(5, 2));
```

값을 바로 리턴하는 람다 함수

```
let getTriangle = (base, height) => base * height / 2;

let getCircle = radius => radius * radius * Math.PI;
```

화살표 함수와 일반 함수 큰 차이점은
this

chapter02/sec01/func03.js

```
let getTriangle = (base, height) => (base * height) / 2;

let getCircle = (radius) => radius * radius * Math.PI;

console.log('삼각형의 면적:' + getTriangle(5, 2));
console.log('원의 면적:' + getCircle(5));
```

```
삼각형의 면적:5
원의 면적:78.53981633974483
```

익명함수는 콜백함수를 등록할 때 많이 쓰이고

번외로
함수를 반환하면 클로저

비동기는 복잡도가 올라가지만 성능이 좋아진다

동기 처리와 비동기 처리

js는 다른 언어와 다르게
비동기가 디폴트다.

동기가 디폴트인 언어는 멀티 스레딩 기법을
사용하는데
js는 비동기가 디폴트이므로 싱글쓰레드만
사용하는 언어이다!

동기 처리 방식과 비동기 처리 방식

비동기 :
함수 호출 후 끝날 때까지 기다리지 않고
호출만하고 바로 다음 흐름으로 진행하는 방식
혹은
원하는 순서로 함수 호출 제어.

그럼 호출 완료는 어떻게 알아
콜백함수 방식으로 비동기 처리하면

매개변수로 함수를 넘겨줘서 순서 제어

chapter02/sec02/sync.js

```
function displayA() {
  console.log('A');
}

function displayB() {
  console.log('B');
}

function displayC() {
  console.log('C');
}

displayA();
displayB();
displayC();
```

동기식 처리

호출 순서에 맞게 처리 됐죠>

```
A
B
C
```

chapter02/sec02/async-1.js

정확히는 동기와 비동기가 섞여있는

```
function displayA() {
  console.log('A');
}

function displayB() {
  setTimeout(() => {
    console.log('B');
  }, 2000);
}

function displayC() {
  console.log('C');
}

displayA();
displayB();
displayC();
```

호출 순서와 다르게 수행하죠?

b가 종료되어도 함수가 작동하죠?

비동기적이죠?

```
A
C
B
```

chapter02/sec02/async-2.js

```
function displayA() {
  console.log('A');
}

function displayB(callback) {
  setTimeout(() => {
    console.log('B');
    callback();
  }, 2000);
}

function displayC() {
  console.log('C');
}

displayA();
displayB(displayC);
```

b가 종료되어도
등록된 콜백함수가
실행되죠?
해당 종료된 함수의
스코프가 살아있어서 그래요

이번 비동기 처리는

콜백 함수 형태를 이용했죠?

원하는 순서로 함수를 호출시키고 싶은데?

하지만 이번처럼 간편하지 않고
여러 순서들이 유지되어야 한다면?
계속 콜백 시키게??
그럼 콜백 지옥인데

순서를 어덯게 잡는게 좋은 방법일까??

```
A
B
C
```

자바스크립트의 비동기 처리 방법

언어 차원에서 순서 제어에 대한 방법들

방법	버전	기능
콜백 함수	기존부터 사용	함수 안에 또 다른 함수를 매개변수로 넘겨서 실행 순서를 제어합니다. 콜백 함수가 많아지면 가독성이 떨어질 수 있습니다.
프라미스	에크마스크립트 2015부터 도입	프라미스 객체와 콜백 함수를 사용해서 실행 순서를 제어합니다.
async/await	에크마스크립트 2017부터 도입	async와 await 예약어를 사용해서 실행 순서를 제어합니다.

초반

콜백 함수 지옥 에바

최신

이것도 복잡 하더라

표현도 동기 처럼 할 수 있지만
실제 실행은 비동기 처리하는 방법

콜백 함수

방법 1 : 콜백함수 활용하여 비동기 처리(순서 제어)해보기

함수 이름을 콜백으로 사용하기

```
const order = (coffee) => {
   console.log(`${coffee} 주문 접수`);
   // 시간이 3초 걸리는 작업
}

const display = (result) => {
  console.log(`${result} 완료!`);
}
```

```
function order(coffee) {
      // 커피 주문
      // 3초 기다린 후 완료 표시
}
function display(result) {
      // 커피 완료 표시
}
```

예상하는 프로그램 흐름

콜백 함수

함수 이름을 콜백으로 사용하기

display 함수를 order 함수의 매개변수로 넘기기

order(어떤 커피인지, 어떤 활동을 나중에 할건지){...}

chapter02/sec02/callback-1.js

```
const order = (coffee, callback) => {
  console.log(`${coffee} 주문 접수`);
  setTimeout(() => {
    callback(coffee);
  }, 3000);
};

const display = (result) => {
  console.log(`${result} 완료!`);
};

order('아메리카노', display);
```

setTimeout(( )=>callback(coffee), 3000);

return 이 명시되지 않았다면
return undefined다.

```
아메리카노 주문 접수
아메리카노 완료!
```

익명으로 콜백 함수 작성하기

```
function displayLetter() {
  console.log("A");
  setTimeout( () => {
    console.log("B");
    setTimeout( () => {
      console.log("C");
      setTimeout( () => {
        console.log("D");
        setTimeout( () => {
          console.log("stop!");
        }, 1000);
      }, 1000);
    },1000);
  }, 1000);
}

displayLetter();
```

프라미스 Promise

- ES2015(ES6)에서 도입
- Callback Hell을 피할 수 있는 방법

미래에 완료디ㄴ면 비동기적 처리를 하겠다고 약속

- Promise 객체를 생성할 때 비동기 작업 함수를 전달
  - 비동기 작업함수의 매개변수
    - resolve 함수: 작업 성공시 호출할 함수
    - reject 함수: 작업 실패시 호출할 함수
  - Promise 객체의 메서드
    - then(result): // 작업성공시 resolve 함수의 매개변수가 전달
    - catch(err): // 작업 실패시 reject 함수의 매개변수가 전달

프라미스 Promise

콜백함수에 비동기 작업을 미리 넘겨줘

```
let likePizza = true;
const pizza = new Promise((resolve, reject) => {
  if(likePizza)
    resolve('피자를 주문합니다.');
  else
    reject('피자를 주문하지 않습니다.');
});
```

비동기 작업들 등록

나중에 then과 catch 키워드를 이용.
성공시 then
실패시 catch

```
pizza
 .then(result => console.log(result))
 .catch(err => console.log(err));
```

등록한 형식에 맞게
콜백함수 등록

then은 연속으로 연결할 수 있다. 그러므로
순서가 많은 비동기적인 처리를 동기적으로 표현할 수 있다.
콜백지옥 이 아닌 then의 나열로 표현 가능
pizza.then(...).then(...).then(...) ...
앞의 then함수의 리턴값이 뒤 then함수의 매개변수로
들어감

chapter02/sec02/async-2.js

```
let likePizza = true;
const pizza = new Promise((resolve, reject) => {
  if (likePizza)
    resolve('피자를 주문합니다.');
  else
    reject('피자를 주문하지 않습니다.');
});

pizza
  .then(result => console.log(result))
  .catch(err => console.log(err));
// 피자를 주문합니다.
```

피자를 주문합니다.

최신

async/await

- prmise의 then을 여러 번 사용하는 경우 복잡해짐
- 비동기 처리를 동기 스타일로 표현할 수 있는 기법

- **주의! async가 선언된 함수 내에서만 await 사용 가능**

```
async function() {
...
await 함수
}
```

await는 비동기 작업을 기다리게 하는 키워드ㅡ

await키워드는 전역에서는 당연히 사용 안되겟죠?

실무에서는 promise하고 async/await를 섞어 쓴다

chapter02/sec02/withoutAwait.js

가짜 api 서버.
백엔드 역할을 할 수 있는 가짜 서버.
프론트만 만들었을 때 테스트로 많이 사용됨.

```
fetch('https://jsonplaceholder.typicode.com/users')
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((err) => console.log(err));
```

fetch js 표준함수 : 요청하는 함수
json문자열을 객체화 한 것이 필요함. json함수는 비동기 작업
첫번째 비동기 작업의 결과를 매개변수로 받아서 출력하는 함수

```
[
  {
    id: 1,
    name: 'Leanne Graham',
    username: 'Bret',
    email: 'Sincere@april.biz',
    address: {
      street: 'Kulas Light',
      suite: 'Apt. 556',
      city: 'Gwenborough',
      zipcode: '92998-3874',
      geo: [Object]
    },
    ...
]
```

chapter02/sec02/await.js

비동기적인 처리를 하는 함수라는 뜻

```
async function init() {
  try {
    const response = await fetch('https://jsonplaceholder.typicode.com/users');
    const users = await response.json();
    console.log(users);
  } catch (err) {
    console.error(err);
  }
}

init();
```

비동기 함수의 완료를 기다리는 키워드

비동기 함수 fetch

처리 결과를 기다림

비동기 처리하다가 에러가 발생하면 동기식 에러처리로

js가 async 붙은 함수를 가공한다. 비동기적인 표현으로

비동기 함수의 리턴값은 promise객체다.

await 키워드가 promise객체의 resolve함수의 반환 값이 대입되는 변수로.

fetch함수의 반환값의 resolve함수의 반환값이 response변수로

res= await fetch();
res = await promise객체;
res= promise객체.resolve();

```
[
  {
    id: 1,
    name: 'Leanne Graham',
    username: 'Bret',
    email: 'Sincere@april.biz',
    address: {
      street: 'Kulas Light',
      suite: 'Apt. 556',
      city: 'Gwenborough',
      zipcode: '92998-3874',
      geo: [Object]
    },
    ...
]
```

프로그래밍에서 가장 기본적인 개념, 모듈

모듈 module

- 기능별로 만들어 놓은 함수
- 파일 형태로 저장하고 필요할 때마다 가져와서 사용

모듈은 크게 3가지로 나눠짐

1 노드가 자체적으로 제공해주는 내장 모듈
2 내가 만드는 모듈
3 다른 사람이 만들어서 제공된느 서드파티 모듈

CommonsJS 모듈 시스템과 ES 모듈 시스템

- 노드가 출시될 당시 모듈 운영에 대한 표준이 없었음
- CommonsJS
  - require() 함수를 통해 모듈을 사용

어떤 웹브라우저는 표준 모듈이 없음.
모듈에 따라서 특정 브라우저에서는 정상 작동이 안됨
어떻게 해야해

- ES 모듈 시스템 해당 모듈을 이용하여 개발
  - 자바스크립트 표준 모듈 시스템
  - 노드 13.2 이후 버전부터 지원

일단 모듈로 작성을 개발하면

변환기프로그램(webpack,vite등이 있음)를 통해서 배포할 때는

비묘듈된 통합된 코드로 변환시키는 작업이 필요함.

모듈이 아닌 코드면 모든 브라우저에서 실행 가능

chapter02/sec03/greeting.js

```
const user = '홍길동';

// 인사하는 함수
const hello = (name) => {
  console.log(`${name}님, 안녕하세요?`);
};

hello(user);
```

데이터 보관

모듈화, 분리

데이터 가공

운영

```
홍길동님, 안녕하세요?
```

chapter02/sec03/user.js

```
const user = '홍길동';
```

chapter02/sec03/hello.js

```
const hello = (name) => {
  console.log(`${name}님, 안녕하세요?`);
};
```

모듈 내보내기 – module.exports

자동으로 node가 활용함.
module.exports = { };//초기값이 비어잇는 객체

```
module.exports = 외부로 내보낼 함수 또는 변수
```

해당 객체에 포함된 속성들이 내보내진다.

chapter02/sec03/user.js

```
const user = '홍길동';

module.exports = user; // user 변수 내보내기
```

변수 하나를 내보내

chapter02/sec03/hello.js

```
const hello = (name) => {
  console.log(`${name}님, 안녕하세요?`);
};

module.exports = hello; // hello 함수 내보내기
```

함수 하나를 내보내

해당 표현법으로는
하나씩 밖에
못 내보냄

여러개 내보내고 싶은데??
그럴 때 객체 형식에서
속성들로 넣어주면 된다!

```
exports.hello = (name) => {
  console.log(`${name}님, 안녕하세요?`);
};
```

이 형식이면
module.exports객체의 hello키의 해당 함수를 갖는 속성을 갖게 되죠

모듈 가져오기 – require 함수

```
require(모듈 파일 경로)
```

어떻게 경로를 지정하느냐에 따라 찾는 위치가 달라짐

모듈명만 쓰면 'ansi-colors'
표준 모듈 혹은 써드파티 모듈에 해당함.
이는 글로벌 전역객체인 node_modules라는 곳에서 검색함.

내가 만든 모듈은?
'./ ... '로 경로를 지정해야한다

현재 디렉토리(모듈)에서 검색을 시작해서
발견할 때까지
상위 디렉토리로 이동하며 검색

chapter02/sec03/app-1.js

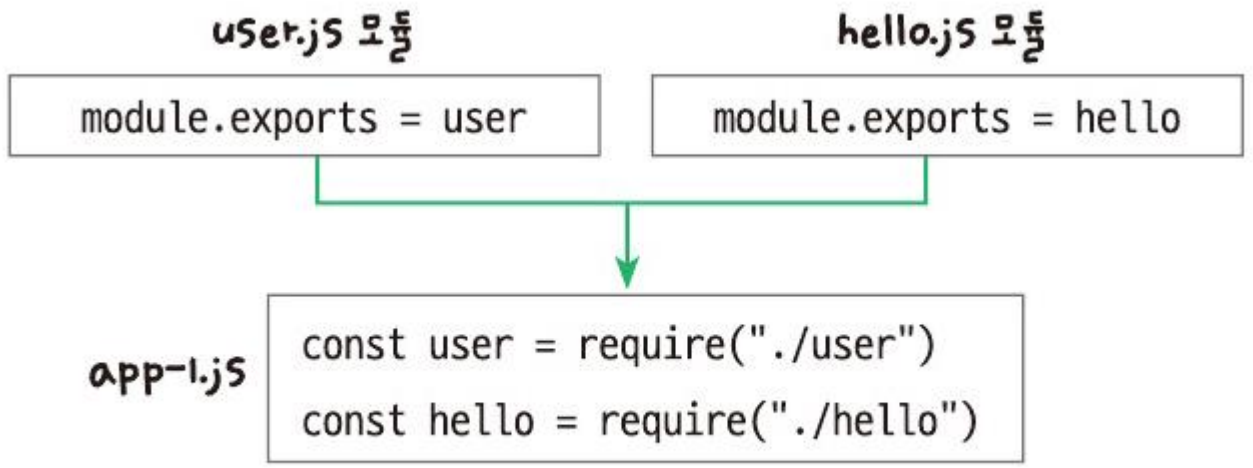
```
const user = require('./user'); // user.js에서 user 가져오기
const hello = require('./hello'); // hello.js에서 hello 가져오기

console.log(user);
console.log(hello);

hello(user); // 모듈에서 가져온 user 변수와 hello 함수 사용하기
```

상수로 외부 모듈객체을 가져오는 것이 좋다?

let으로 받아 다른 값을 대입하더라도
참조 방향이 달라지는 것이지
모듈 자체에 대한 데이터는 바뀌지 않는다.

```
홍길동
[Function: hello]
홍길동님, 안녕하세요?
```

require되면
module cahe테이블에서
캐싱한다.



모듈을 활용한 프로그램

두개 이상의 변수 내보내기 및 가져오기

chapter02/sec03/users-1.js

```
const user1 = 'Kim';
const user2 = 'Lee';
const user3 = 'Choi';

module.exports = { user1, user2 };
```

동일한 표현으로는
exports.user1=user1
exports.user2=user2

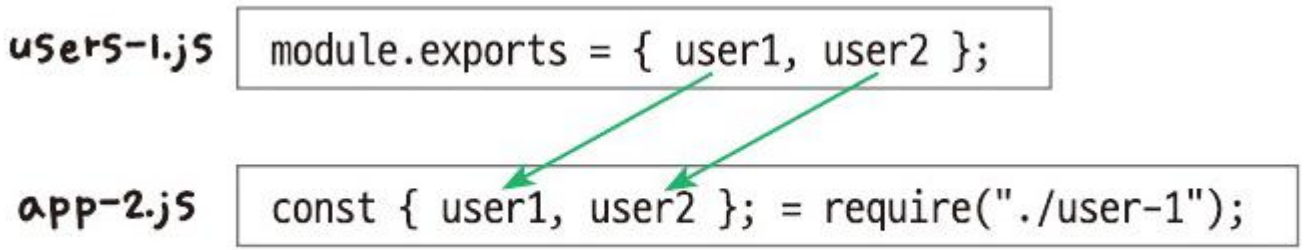
exports는 외부 객체를
참조하는 또다른 이름이다

exports에는
대입 표현을 사용하면 안된다

chapter02/sec03/app-2.js

```
const { user1, user2 } = require('./users-1');  js의 분해 할당 기법이다. 조건은 이름이 같아야 함.
const hello = require('./hello');

hello(user1);
hello(user2);
```

```
Kim님, 안녕하세요?
Lee님, 안녕하세요?
```



변수 2개 내보내기

기능	모듈명	설명
파일 시스템	fs	파일이나 폴더에 접근할 수 있는 기능을 제공합니다. 예를 들어 파일 읽기/쓰기/삭제/이동/이름 변경이나 폴더 작업을 처리할 수 있습니다. 자세한 사용법은 03-2절에서 설명합니다.
HTTP	http	HTTP 서버를 만들고 요청을 처리하는 기능을 제공합니다. 익스프레스 같은 프레임워크 없이 HTTP 서버를 만드는 방법은 04-2절에서 설명합니다.
경로	path	파일 경로와 관련된 작업을 하는 기능을 제공합니다. 예를 들어 파일 경로를 지정하거나 상대 경로를 계산하는 작업을 할 수 있습니다. 자세한 사용법은 03-1절에서 설명합니다.
스트림	streams	데이터 스트림을 처리하는 기능을 제공합니다. 예를 들어 파일이나 네트워크와 같은 스트림에서 데이터를 읽거나 쓰는 작업을 할 수 있습니다. 스트림과 streams 모듈을 사용하는 방법은 03-5절에서 설명합니다.
암호화	crypto	암호화와 관련된 기능을 제공합니다. 해시 함수, 암호화 알고리즘, 암호화 및 복호화 등을 지원합니다.
운영체제	os	운영체제와 상호 작용하는 기능을 제공합니다. 예를 들어 운영체제의 정보를 알아내거나 시스템 리소스 정보를 확인할 수 있습니다.
유틸리티	util	다양한 유틸리티 함수를 제공합니다. 예를 들어 객체 상속, 비동기 처리를 프라미스로 변환하는 등의 작업을 할 수 있습니다.
이벤트	events	이벤트 기반 프로그래밍을 지원하는 기능을 제공합니다. 이벤트 생성, 등록, 처리 등을 할 수 있으며, 커스텀 이벤트를 사용하여 비동기 처리를 할 수 있습니다.

글로벌 모듈

- __dirname
  - 현재 모듈의 디렉토리 경로
- __filename
  - 현재 모듈의 파일 경로

chapter02/sec04/here.js

```
console.log(`현재 모듈이 있는 폴더 경로: ${__dirname}`);
console.log(`현재 모듈이 있는 파일 경로: ${__filename}`);
```

```
현재 모듈이 있는 폴더 경로: c:₩fullstack₩02_Node.js₩chapter02₩sec03
현재 모듈이 있는 파일 경로: c:₩fullstack₩02_Node.js₩chapter02₩sec03₩14_here.js
```

5 ES 모듈 시스템 사용법

commonjs는 그냥 패키지를 사용하면 되지만

es모듈 시스템을 이용하려면 npm init을 하고
생성된 packagae.json의 내용에 밑ㅇ 내용을 가미해야함

package.json에 설정하는 방법

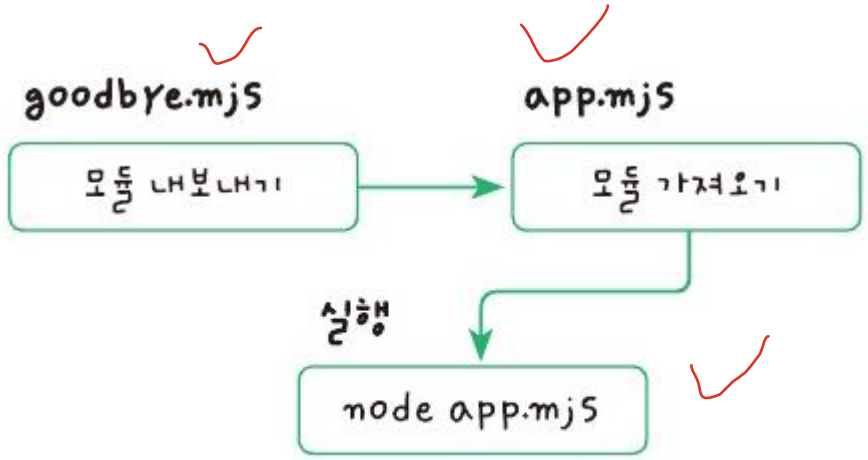
```
{
  "name": "src",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon app.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "ansi-colors": "^4.1.3"
  },
  "type": "module"
}
```

ES만 ㅅ써야할때

하지만 혼용해야 할 때는

파일 확장자를 .mjs로 지정

- CommonJS 모듈 시스템과 ES 모듈 시스템을 같이 사용할 때 사용
- package.json에 모듈 시스템 지정하면 안됨



ES 모듈 시스템에 사용하는 .mjs 확장자

ES 모듈 시스템에서 모듈 내보내기 – export, export default

- 변수나 함수 앞에 export를 붙임
  - 변수는 반드시 상수여야 함

```
export 대상
```

import할 때 대상을 진행해야 한다.

chapter02/sec05/goodbye-1.mjs

```
export const goodbye = (name) => {
  console.log(`${name} 님, 안녕히 가세요.`);
};
```

export는 여러번 여러개 가능``

기본으로 내보내기 – export default

- 모듈에서 내보낼 대상이 하나 뿐일 때 그리고 한번만

```
export default 대상
```

chapter02/sec05/goodbye-2.mjs

```
const goodbye = (name) => {
  console.log(`${name} 님, 안녕히 가세요.`);
};

export default goodbye;
```

여러 개 내보내기

- 객체로 묶어서 내보냄

```
export {대상1, 대상2, ...}
```

엥 객체를 왜 {..., ...} 처럼 표현하지
키 – 값 쌍이 아니고.

es6에서 간편하게 사용하라고

키와 값의 내용이 같다면

{... , ... , ...}로 표현함.
이는
{키 : 값, 키 :값 ...}랑 같음

chapter02/sec05/greeting-1.mjs

```
const hi = (name) => {
  console.log(`${name}님, 안녕하세요?`);
};

const goodbye = (name) => {
  console.log(`${name}님, 안녕히 가세요.`);
};

export { hi, goodbye };
```

취향에 맞게 해라
권장사항은 없다

common은 required함수겠지만

ES 모듈 시스템에서 모듈 가져오기 – import ~ from

```
import 변수명/함수명 from 모듈_파일
```

모듈 경로인데, 앞에서 말한 조건과 같다

여기서는 키워드로 해결

chapter02/sec05/app-5.mjs

```
import { goodbye } from './goodbye-1.mjs';

goodbye('홍길동');
```

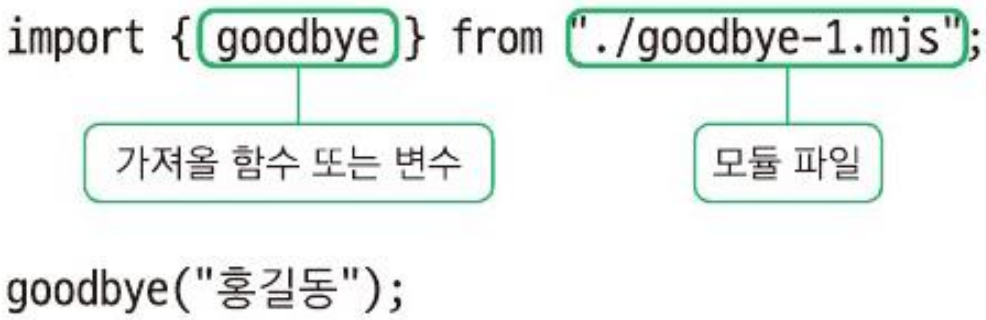
요런 형식이야

.mjs는

commonjs 모듈 방식과 표준 ES 모듈 방식 합쳐서 쓸 때

ES 표준 모듈 방식을 쓰면
.js가 패키지 파일이겠지. 이걸로 패키지 파일 지정

```
홍길동 님, 안녕히 가세요.
```



chapter02/sec05/app-6.mjs

```
import { hi, goodbye } from './greeting-1.mjs';
    나열 형식으로 여러개 갖고 오기

hi('홍길동');
goodbye('홍길동');
```

몇백개의 변수와 함수를 다 갖고 와야 할때는
as키워드 사용
이어서 뒤에

```
홍길동님, 안녕하세요?
홍길동님, 안녕히 가세요.
```

import ~ as

- 가져오는 함수나 변수의 이름을 바꿔서 받을 수 있음

chapter02/sec05/app-7.mjs

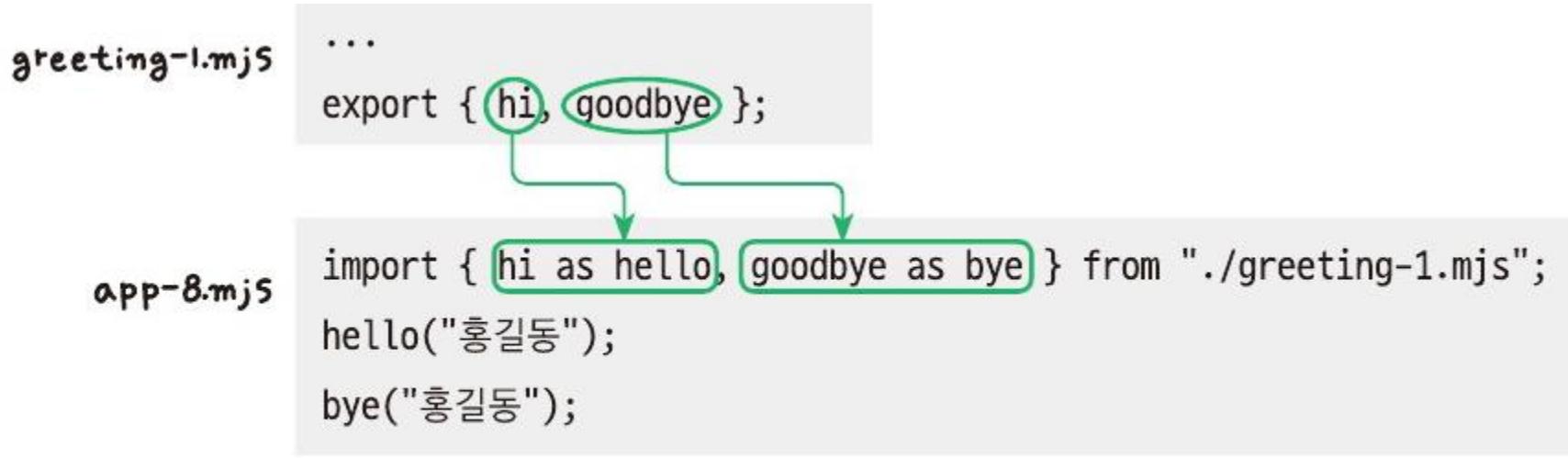
```
import { goodbye as bye } from './goodbye-1.mjs';

bye('홍길동');
```

```
홍길동님, 안녕히 가세요.
```

chapter02/sec05/app-8.mjs

```
import { hi as hello, goodbye as bye } from './greeting-1.mjs';

hello('홍길동');
bye('홍길동');
```

```
홍길동님, 안녕하세요?
홍길동님, 안녕히 가세요.
```



ES 모듈 시스템에서 둘 이상의 모듈 이름 바꾸기

- **import * as** 패키지내에 있는 거 다 받기
  - 모듈에서 가져와야 할 것이 너무 많은 경우 사용
  - 해당 이름의 객체로 묶어 줌

chapter02/sec05/app-9.mjs

```
import * as say from './greeting-1.mjs'; // greeting.mjs에서 내보낸 함수들을 한꺼번에 say로 받기

say.hi('홍길동');
say.goodbye('홍길동');
```

```
홍길동님, 안녕하세요?
홍길동님, 안녕히 가세요.
```

- **export default 가져오기**

한개만 내보냈으니

chapter02/sec05/greeting-2.mjs

```
const hi = (name) => {
  console.log(`${name}님, 안녕하세요?`);
};

const goodbye = (name) => {
  console.log(`${name}님, 안녕히 가세요.`);
};

export default { hi, goodbye };
```

export { ... ,...}
export default =...;

이면

import를 두줄로해

import { ... , ... } from ...
import ... from ...

chapter02/sec05/app-10.mjs

```
import say from './greeting-2.mjs';   그냥 단일식별자로 가져오기하면 됨

say.hi('홍길동');
say.goodbye('홍길동');
```