LEARNING SPOONS

LEARNING SPOONS ONLINE

자바스크립트 기초

모듈과 기타 주요기능

LEARNING SPOONS

Unit

1. 모듈 시스템 이해하기
2. ES6 모듈 시스템 살펴보기
3. this 키워드 집중 분석하기
4. [실습] To-do 앱 클론코딩 3차

Unit 4-1

모듈 시스템 이해하기

왜 모듈module 이 필요할까?

추상적인 개념 혹은 패턴으로 60년대 이후 많은 프로그래밍 언어에서부터 사용해왔다. 특정 파일이나 코드 블록단위로 모듈화를 한다. 모듈의 이점은 다음과 같다.

- 코드의 추상화
- 코드의 캡슐화
- 코드의 재사용
- 의존성 관리

```
const myModule = {
 version: "v1.0.0",
 config: {
   useCaching: true,
 },
 saySomething() { // 기본 메소드
   console.log("hello there.");
 },
 getConfig() { // 설정 조회
   console.log(`Caching - ${this.config.useCaching ? "ok" : "no"}`);
 },
 updateConfig(newConfig ) { // 설정 업데이트
   this.config = newConfig;
 }
};
```

모듈 패턴

- 즉각함수 호출을 이용하여 scope을 제한하여 전역 scope의 오염을 방지

```
const counterModule = (function () {
 let counter = 0;
 return {
   incrementCounter: function () {
     return counter;
   },
   resetCounter: function (){
     console.log( "counter value prior to reset: " + counter );
     counter = 0;
   }
 };
})();
```

모듈 포맷

모듈 포맷은 모듈을 정의하는 문법으로 ES6이전엔 공식적인 문법이 없었어 다양한 문법이 존재하게 됨

- Asynchronous Module Definition (AMD)
- CommonJS
- Universal Module Definition (UMD)
- System.register
- ES6 module format

LEARNING SPOONS

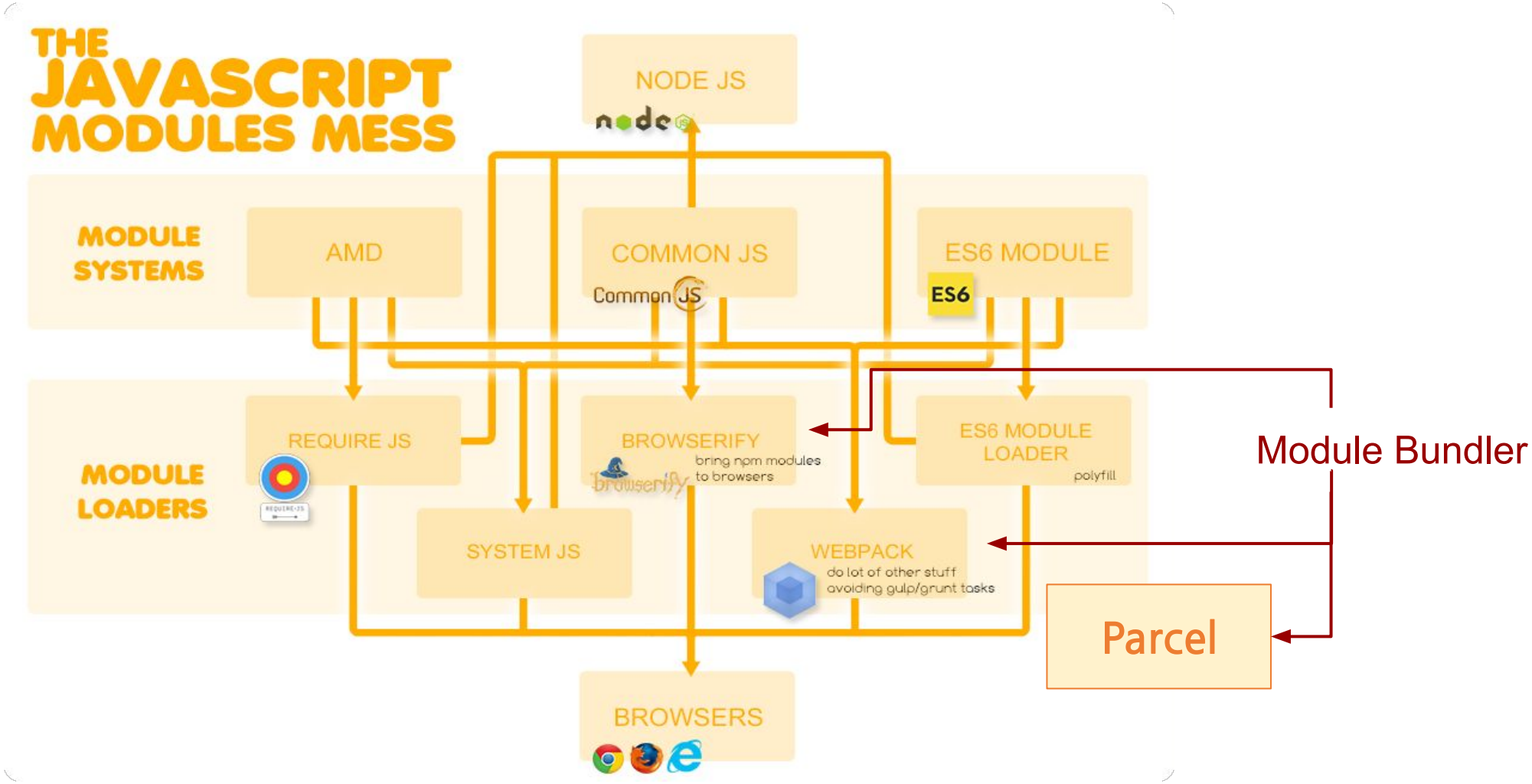
모듈 로더module loader ⚔️ 모듈 번들러module bundler

모듈 로더는 특정 모듈 포맷을 런타임에서 읽고 실행.

- 모듈 로더는 메인 파일을 로드하고 의존된 다른 모듈파일을 다운로드하고 실행함
- require.js
  - AMD 포맷 모듈 로더
- system.js
  - AMD, CommonJS, UMD 포맷 모듈 로더

모듈 번들러는 빌드 타임에서 모듈을 읽고 하나의 파일로 만듬

- 모듈 번들러는 메인 파일을 로드하고 의존된 다른 모듈파일을 하나의 번들파일로 만들어줌
- browserify
  - CommonJS용 번들러
- webpack
  - AMD, CommonJS, ES6 Module 번들러



출처: http://bertrandg.github.io/the-javascript-module-mess/

LEARNING SPOONS

AMD

- define 메소드를 통해 의존 모듈을 기술하고 factory 함수로 모듈을 정의

```
define(["jquery"], function($) {
    let counter = 0;
    return {
        incrementCounter: function () {
            return counter++;
        },
        resetCounter: function () {
            $('#display').html( "counter value prior to reset: " + counter );
            counter = 0;
        }
    };
});
```

CommonJS

- node.js에서 사용하는 모듈 포맷으로 require와 module.exports를 통해 모듈을 정의

```
var jquery = require('jquery');
var counter = 0;

module.exports = {
    incrementCounter: function () {
        return counter++;
    },
    resetCounter: function () {
        $('#display').html( "counter value prior to reset: " + counter );
        counter = 0;
    }
}
```

LEARNING SPOONS

UMD Universal Module Definition

- 브라우저와 node.js에서 모두 사용할 수 있는 포맷

```
(function (root, factory) {
 if (typeof define === 'function' && define.amd) { // AMD.
   define(['jquery'], factory);
 } else if (typeof module === 'object' && module.exports) { // Node.
   module.exports = factory(require('jquery'));
 } else { // Browser globals (root is window)
   root.returnExports = factory(root.$);
 }
}(this, function (jquery) {
  return {
    incrementCounter: function () { },
    resetCounter: function () { }
  };
}));
```

LEARNING SPOONS

Unit 4-2

ES6 모듈 시스템 살펴보기

ES6 모듈 정의 (default)

- 하나의 파일이 하나의 모듈이고 하나의 모듈은 하나의 default 를 가진다.
- 동적 할당은 안된다.

```
export default { //module.exports = {}
   incrementCounter: function () {},
   resetCounter: function () {}
}
export default 'version' //module.exports = 'version'
export default function getVer() {} //module.exports = function getVer() {}

function setup() {
 export default 'version' // SyntaxError
}
setup()
```

ES6 모듈 정의 (named export)

- 하나의 모듈은 여러 export가 가능함

```
// module.exports.version = 'v1.0.0';
export let version = 'v1.0.0';
// module.exports.getVer = () => version;
export const getVer = () => version;

let version = 'v1.0.0';
const getVer = () => version;
export version // SyntaxError 선언이나 문장이 요구된다.
export getVer // SyntaxError 선언이나 문장이 요구된다.

export class Ver {
   ver = 'v1.0.0';
   getVer() { this.ver; }
}
```

ES6 모듈 정의 (named export list)

- 여러 선언들을 하나의 문장으로 export 할 수 있다.

```
class Ver {
    ver = 'v1.0.0';
    getVer() {
        this.ver;
    }
}
let version = 'v1.0.0';
const getVer = () => version;

export { Ver, version as ver, getVer }
```

LEARNING SPOONS

ES6 모듈 사용하기①

- import 문을 이용하여 다른 모듈을 사용할 수 있다.

```
// version.js
export default 'v1.0.0';
// app.js
import version from './version.js'
```

- named export는 import 시 { } 를 사용 (Destructuring과 흡사)

```
// version.js
export let version = 'v1.0.0';
export const getVer = () => version;
// app.js
import { version, getVer } from './version.js'
```

LEARNING SPOONS

ES6 모듈 사용하기②

- 이름만 작성하는 경우와 경로를 작성하는 경우에 따라 로드되는 파일이 다르다.

```
import myVal from './src/file'
// 다음 두 파일 모두 해당된다.
// ./src/file/index.js
// ./src/file.js

import myVal from './src/file.js'
// 오직 아래 파일만 해당된다.
// ./src/file.js

import * as _ from 'lodash'
// node_modules 폴더 아래에서 찾는다.
```

LEARNING SPOONS

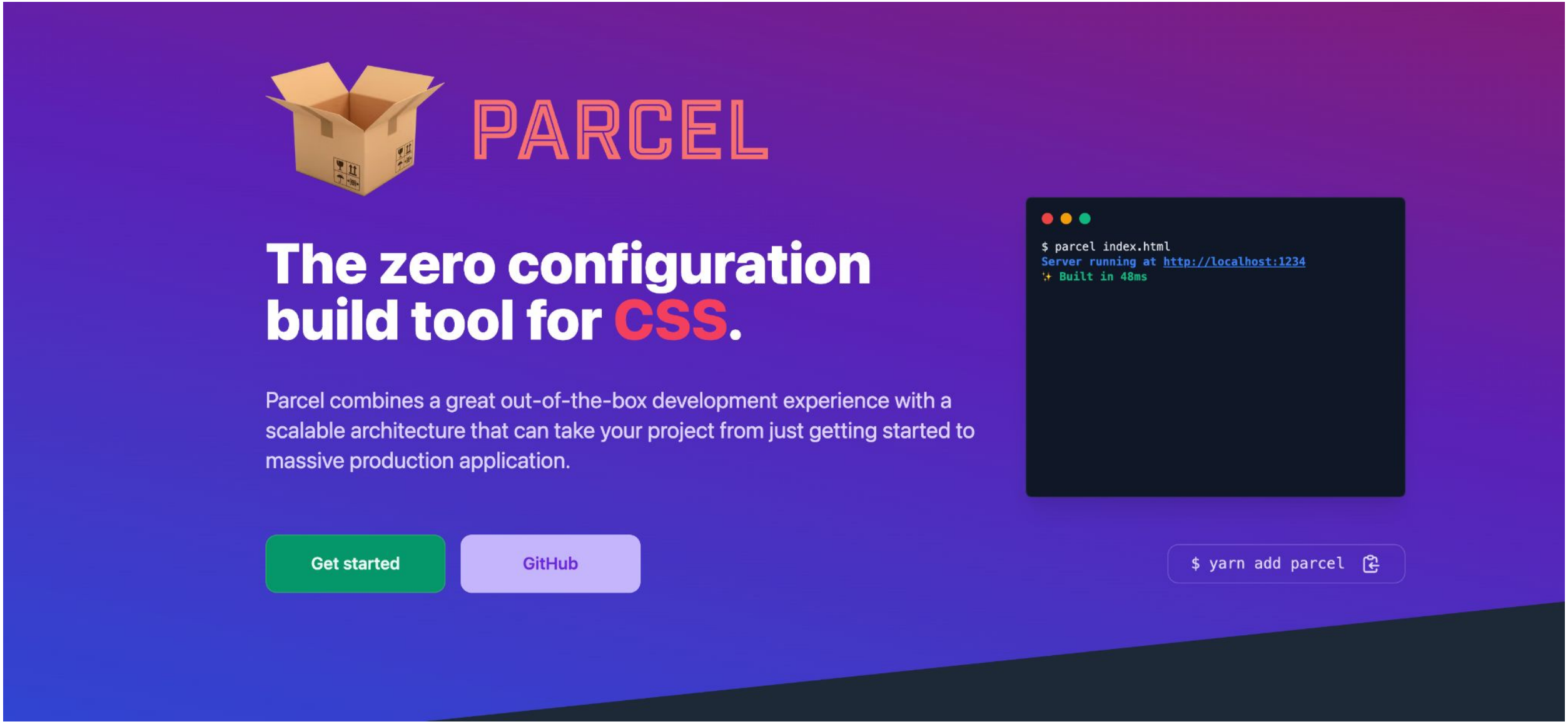
ES6 모듈 사용하기③

- import 문을 as 키워드로 이름을 미러링 할 수 있다.

```
// version.js
export { Ver, version as default, getVer as getV}
// app.js
import version, { Ver, getV as getVV } from './version.js';
```

- *를 이용하면 전체를 import하게되고 꼭 as가 뒤에 붙어야 한다.

```
// version.js
export { Ver, version}
// app.js
import * as version from './counter.js'
version.Ver; // class Ver
```


PARCEL
The zero configuration build tool for CSS.
Parcel combines a great out-of-the-box development experience with a scalable architecture that can take your project from just getting started to massive production application.
Get started
GitHub
$ parcel index.html
Server running at http://localhost:1234
✨ Built in 48ms
$ yarn add parcel

LEARNING SPOONS

Unit 4-3

this 키워드 집중 분석하기

this 키워드

- this는 실행 컨텍스트의 코드로부터 접근가능한 동적이고 암묵적인 컨텍스트 객체입니다. 같은 코드를 여러 객체에 적용하고자 할 때 사용됩니다.
- this는 어떠한 실행 컨텍스트에서 작성하는 것에 따라 다른 객체를 가리킵니다.
  - 전역에서 작성될 때
  - 함수 안에서 작성될 때
  - 생성자 함수 안에서 작성될 때
  - .call과 .apply
  - 화살표 함수 안에서 작성될 때
  - 클래스 안에서 작성될 때

this가 전역에서 작성될 때

- import 문을 as 키워드로 이름을 미러링 할 수 있다.

```
this.a = 10;
console.log(a);
b = 20;
console.log(this.b);

var c = 30;
console.log(this.c);
console.log(this === window);
```

this가 함수안에서 작성될 때

- 함수안에서 this는 전역객체를 가리키지만 엄격한 모드에서는 undefined가 됩니다.

```
function func() {
 "use strict";
 console.log(this === undefined);
}
func();

function Person(firstName, lastName) {
 this.firstName = firstName;
 this.lastName = lastName;
}

var person = Person("jeado", "ko");
console.log(person);
console.log(window.firstName);
console.log(window.lastName);
```

LEARNING SPOONS

this가 생성자 함수 안에서 작성될 때

```
function Person(firstName, lastName) {
 this.firstName = firstName;
 this.lastName = lastName;
 setTimeout(function() {
   console.log(this.firstName);
   console.log(this);
 }, 1000);
}

var person = new Person("jeado", "ko");
console.log(person);
```

LEARNING SPOONS

this가 메소드 안에서 작성될 때

```
var person = {
 firstName: "jeado",
 sayHi() {
     console.log(`hello, I am ${this.firstName}!`);
 }
};

person.sayHi();

var greet = person.sayHi;
greet(); // 여기서 this는 글로벌객체가 된다.
greet.bind(person)() // 정상 동작

setTimeout(person.sayHi, 1000) //여기서도 this가 글로벌 객체가된다.
setTImeout(function() {
  person.sayHi(); // 정상동작
}, 1000);
setTimeout(person.sayHi.bind(person), 1000) // 정상동작
```

call, apply 그리고 bind로 this 할당

- apply: 인자로 주어진 this와 arguments 로 함수를 호출.
- call: 인자로 주어진 this와 arguments 로 함수를 호출.
- bind: 해당 함수가 호출될때 인자로 넘긴 this 를 전달.

```
renderTodos: function() {
    for (var i = 0; i < this.todos.length; i++) {
     var todoEl = this.createTodoEl(this.todos[i]);
     this.todoContainerEl.append(todoEl);
     todoEl.find('input[type="checkbox"]').on('click',  (function() {
       var num = i;
       return function(evt) {
         this.todos[num].done = evt.target.checked;
         this.renderTitle();
       }.bind(this);
     }.bind(this))());
   }
},
```

this가 화살표 함수에서 작성될 때

- 화살표 함수안의 this는 화살표 함수 밖의 this와 같다.

```
const util = {
  threshold: 5,
  filter: function (...numbers) {
    return numbers.filter(n => (n < this.threshold) ? true : false);
  }
}

util.filter(2, 5, 10); //2
```

실습

To-do 앱 클론코딩 3차