Ask Company

리액트와 함께 장고 시작하기 / 리액트

꼭 알아야할 ES6+ 문법

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

꼭 알아야할 문법들

상수/변수 선언 (const/let)

Object 선언, 단축 속성, key 계산, 객체 복사

Template Literals

배열/객체 비구조화 (Array/Object Destructuring)

전개 연산자 (Spread Operator)

함수와 인자 (Functions, Parameters, Named Parameters)

Arrow Functions

Promise와 async/await

클래스와 상속

모듈 시스템

고차 함수 (High Order Function)

상수/변수 선언

var 대신에 const 혹은 let을 사용 → block scope

- const : 재할당 불가. 내부 속성값은 수정 가능.
- let : Lexical Variable Scoping을 지원하는 변수 선언 문법

```
var div;
var container = document.getElementsByTagName('body')[0];

for(let i=0; i<5; i++) {
    div = document.createElement('div');
    div.onclick = function() {
        alert("clicked : #" + i);
    };
    div.innerHTML = "#" + i;
    container.appendChild(div);
}
```

var/let 사용여부에 따라
alert 메세지가 달라집니다.

Object 선언 (1/3)

아래 tom1/tom2는 동일

```
let tom1 = {
    name: "Tom",
    age: 10,
    region: "Seoul"
};

let tom2 = {
    "name": "Tom",
    "age": 10,
    "region": "Seoul"
};
```

파이썬

```
tom = {
    "name": "Tom",
    "age": 10,
    "region": "Seoul"
}
```

Object 선언 (2/3)

Key 계산이 필요한 경우

자바 스크립트

```
const tom2 = {
    "name": "Tom",
    "age": 10,
    "region": "Seoul",
    ["score" + "1"]: 100,
};
```

→ "score1"

Array가 아닙니다.

파이썬

```
tom = {
    "name": "Tom",
    "age": 10,
    "region": "Seoul",
    "score" + "1": 100,
}
```

자바 스크립트

```
const key1 = "location";

const tom = {
    name: "Tom",
    [key1]: "Seoul"
};
```

파이썬

```
key1 = "location"

tom = {
    "name": "Tom",
    key1: "Seoul"
}
```

Object 선언 (3/3)

단축 속성명

```
let name = "Tom";
let age = 10;

let tom1 = {
    name: name,
    age: age,
    print: function() {
        console.log(`name: ${this.name}, age: ${this.age}`);
    }
};


let tom2 = {
    name,
    age,
    print() {
        console.log(`name: ${this.name}, age: ${this.age}`);
    }
};
```

클래스 없이도 이렇게 객체를 만들수 있어요.
→ tom1.print()

위와 동일한 객체
→ tom2.print()

객체 복사

JS는 Object/Array에 대해서는 대입 시에 얕은 복사 (Shallow Copy)

```
const obj1 = { value1: 10 };
const obj2 = obj1;                  // 얕은 복사
const obj3 = JSON.parse(JSON.stringify(obj1))

obj1.value1 += 1;

console.log(`obj1:`, obj1);
console.log(`obj2:`, obj2);
console.log(`obj3:`, obj3);
```

```
❯ node t.js
obj1: { value1: 11 }
obj2: { value1: 11 }
obj3: { value1: 10 }
```

obj2.value1 도 변경되었습니다.

Template Literals

Multi-line string

String Interpolation

```
`string text ${expression}
string text`
```

비교) 파이썬

```
"""string text
string text"""

f"""string text {expression}
string text"""
```

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Template_literals

배열 비구조화 (Array Destructuring)

리액트에서 자주 쓰는 문법

```
let [name] = ["Tom", 10, "Seoul"];

let [,age,] = ["Tom", 10, "Seoul"];


let [name, age, region, height] = ["Tom", 10, "Seoul"]
```

undefined 할당

파이썬에서는 ValueError 예외

```
let [name, age, region, height=150] = ["Tom", 10, "Seoul"]
```

디폴트값 할당

```
function get_default_height() {
    console.log("get_default_height() 호출")
    return 150;
}

let [name, age, region, height=get_default_height()] = ["Tom", 10, "Seoul"]
```

실제 디폴트값 할당이 필요할 때, 호출됩니다.

```
> let [x, y] = [1, 2]
undefined
> [x, y]
[ 1, 2 ]
> [x, y] = [y, x]
[ 2, 1 ]
```

객체 비구조화 (Object Destructuring) (1/2)

리액트에서 정말 자주 쓰는 문법

```
const tom = {
    name: "Tom",
    age: 10,
    region: "Seoul"
};

const {age, name, height}= tom;
```

객체에서 필요한 값들만 뽑아냅니다.

height는 undefined

아래와 같이 할 수는 있지만 ...

```
const age = tom.age;
const name = tom.name;
```

```
const print_person1 = (person) => {
    console.log(person.name);
};

const print_person2 = ({ name }) => {
    console.log(name);
};

print_person1(tom);
print_person2(tom);
```

```
const people = [
    { name: 'Tom', age: 10, region: 'Seoul' },
    { name: 'Steve', age: 12, region: 'Pusan' }
];

for (const person of people) {
    console.log(person.name, person.age);
}

for (const {name, age} of people) {
    console.log(name, age);
}
```

객체 비구조화 (Object Destructuring) (2/2)

```
const person = {
    name: 'Tom',
    age: 10,
    region: {
        country: '서울',
        postcode: '06222',
    }
};

const { name, region: { postcode }} = person;

console.log(name, postcode);
```

region은 할당 X

전개 연산자 (Spread Operator)

```
let [name, ...rest] = ["Tom", 10, "Seoul"];

let names = ["Steve", "John"];
let students = ["Tom", ...names, ...names];
```

복사본

리액트에서는 수많은 값들을 불변객체로서 처리합니다.

Tip) 이때 "전개 연산자"를 많이 쓰며, 구조가 복잡할 경우 immer 라이브러리를 쓰는 것이 코드 가독성에 도움이 됩니다.

https://github.com/immerjs/immer

```
let printArgs = (...args) => {
    console.log(args);
}
```

```
let tom = {
    name: "Tom",
    age: 10,
    region: "Seoul"
};

let steve = {
    ...tom,
    name: "Steve"
};
```

속성명이 중복될 경우, 마지막 값이 남습니다.

nodejs

```
> const numbers = [1, 3, 7, 9];
undefined
> Math.max(numbers)
NaN
> Math.max(...numbers)
9
```

python

```
>>> numbers = [1, 3, 7, 9]
>>> max(numbers)
9
>>> max(*numbers)
9
```

함수 / Default Parameters

모든 타입의 값들을 디폴트 파라미터로 지정할 수 있습니다.

→ 파이썬에서는 Immutable 값들만 디폴트 파라미너로 지정 가능

```
function hello(name="Tom", age=10) {
    console.log(`나는 ${name}. ${age}살이야.`);
}


const get_default_age = () => 10

function hello(name="Tom", age=get_default_age()) {
    console.log(`나는 ${name}. ${age}살이야.`);
}

console.log(hello("Steve"))
```

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Functions/Default_parameters

[node/python 비교] 디폴트 값에 함수를 적용할 경우

nodejs

```nodejs
function get_default_height() {
    console.log("get_default_height() 호출");
    return 150;
}

function say_hello(name, height=get_default_height()) {
    console.log(`name: ${name}, height: ${height}`);
}

say_hello("Tom", 160);
say_hello("John");
say_hello("Steve", 170);
say_hello("Suji");
```

```
❯ node t.js
name: Tom, height: 160
get_default_height() 호출
name: John, height: 150
name: Steve, height: 170
get_default_height() 호출
name: Suji, height: 150
```

python

```python
def get_default_height():
    print("get_default_height() 호출")
    return 150

def say_hello(name, height=get_default_height()):
    print(f"name: {name}, height: {height}")

say_hello("Tom", 160)
say_hello("John")
say_hello("Steve", 170)
say_hello("Suji")
```

```
❯ python t.py
get_default_height() 호출
name: Tom, height: 160
name: John, height: 150
name: Steve, height: 170
name: Suji, height: 150
```

함수 / Named Parameters

객체 비구조화를 활용

```
function print_person1(name, age, region) {
    console.log('1>', name, age, region)
}

print_person1('Tom', 10, 'Seoul');


function print_person2({ name, age, region }) {
    console.log('2>', name, age, region)
}

print_person2({ name: 'Tom', age: 10, region: 'Seoul' });
```

비교) 파이썬

```
def print_person(name, age, region):
    print(name, age, region)

print_person('Tom', 10, 'Seoul')

print_person(name='Tom', age=10, region='Seoul')
```

함수 / Arrow Function (1/2)

return을 사용하지 않아도, 계산된 값을 반환

인자가 1개일 경우, 소괄호 생략 가능

```
var hello1 = function(name, age) {
    return `안녕. 나는 ${name}. ${age}이야.`;
};

let hello2 = (name, age) => `안녕. 나는 ${name}. ${age}이야.`;

let hello3 = (name, age) => {
    return `안녕. 나는 ${name}. ${age}이야.`;
}
```

함수를 중괄호로 감싸지 않으면,
return 문을 쓰지 않아도
반환값으로 사용됩니다.

함수 / Arrow Function (2/2)

중요) this와 arguments를 바인딩하지 않습니다.

실행결과

```
[print1-1] name : Tom
[print1-2] name : undefined

[print2-1] name : Tom
[print2-2] name : Tom

[print3-1] name : Tom
[print3-2] name : Tom
```

this가 변경되죠. !!! 유념하세요.

```
var tom = {
    name: "Tom",
    print1: function() {
        console.log(`[print1-1] name : ${this.name}`);
        (function() {
            console.log(`[print1-2] name : ${this.name}`);
        })();
    },
    print2: function() {
        console.log(`[print2-1] name : ${this.name}`);
        var me = this;
        (function() {
            console.log(`[print2-2] name : ${me.name}`);
        })();
    },
    print3: function() {
        console.log(`[print3-1] name : ${this.name}`);
        (() => {
            console.log(`[print3-2] name : ${this.name}`);
        })();
    }
};

tom.print1();
tom.print2();
tom.print3();
```

함수 / 다양한 형태

```
const mysum1 = (x, y) => x + y;
const mysum2 = (x, y) => {x, y};
const mysum3 = (x, y) => ({x: x, y: y});
const mysum4 = (x, y) => {
  return {x: x, y: y};
}

const mysum5 = function(x, y) {
  return {x: x, y: y};
};

function mysum6(x, y) {
  return {x: x, y: y};
}
```

콜론으로 인해, 소괄호가 필요

콜백지옥 : callbackhell.com

비동기 프로그래밍을 위해
콜백(callback)을 많이 사용

```
fs.readdir(source, function (err, files) {
    if (err) {
        console.log('Error finding files: ' + err)
    } else {
        files.forEach(function (filename, fileIndex) {
            console.log(filename)
            gm(source + filename).size(function (err, values) {
                if (err) {
                    console.log('Error identifying file size: ' + err)
                } else {
                    console.log(filename + ' : ' + values)
                    aspect = (values.width / values.height)
                    widths.forEach(function (width, widthIndex) {
                        height = Math.round(width / aspect)
                        console.log('resizing ' + filename + 'to ' + height + 'x' + height)
                        this.resize(width, height).write(dest + 'w' + width + '_' + filename, function(err) {
                            if (err) console.log('Error writing file: ' + err)
                        })
                    }.bind(this))
                }
            })
        })
    }
})
```

콜백 → Promise → async/await

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

fs.readdir('.', function (err, files) {
    if (err) {
        console.log('Error finding files: ' + err)
    }
    else {
        console.log(files);
    }
});

// 위 fs.readdir이 끝나기 전에 수행
console.log("ENDED");
```

콜백 → Promise → async/await

```
const fs = require('fs');
const fsPromises = fs.promises;

fsPromises.readdir('.')
    .then(files => {
        console.log(files);
    })
    .catch(err => console.error(err));

// 위 fsPromises.readdir이 끝나기 전에 수행
console.log("ENDED");
```

콜백 → Promise → async/await

ES8 (ECAM 2017) 부터 지원

```
const fs = require('fs');
const fsPromises = fs.promises;

async function fn() {
    try {
        let files = await fsPromises.readdir('.');
        console.log(files);
    }
    catch(err) {
        console.error(err);
    }
}

fn();  // async 함수 이기에, 완료 전에 다음 로직이 동작

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

클래스와 상속

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}

Person.prototype.print = function() {
    console.log(this.name + ", " + this.age);
}


var tom = new Person("Tom", 10);
tom.print();
```

```
class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }

    print() {
        console.log(this.name + ", " + this.age);
    }
}

const tom = new Person("Tom", 10);
tom.print();


class Developer extends Person {
    constructor(name, age, field) {
        super(name, age);
        this.field = field;
    }

    print() {
        super.print();
        console.log(`field : ${this.field}`);
    }
}
```

문법이 다를 뿐,
여전히 prototype을 사용합니다.

모듈 시스템

예전 웹 상의 자바스크립트에서는 script 태그를 통해서만이 로딩

- 모두 전역 객체에 바인딩

2가지 모듈 시스템

- CommonJS Module : nodejs에서 주로 활용
- ES6 Module : 리액트에서 주로 활용

모듈 / ES6 module

React를 쓰실 때, 사용할 모듈 시스템

IE를 포함한 구형 브라우저에서는 미지원

node 이후에 ES6 Module이 나왔기에, node에서는 왠만한 ES6 문법은 지원하지만, 모듈은 ES6 module을 지원하지 않고, CommonJS만을 지원

문법 : export, export default, import ... from

Using JavaScript code modules : https://developer.mozilla.org/en-US/docs/Mozilla/JavaScript_code_modules/Using

ES Modules와 Node.js: 쉽지 않은 선택 : https://nodejs.github.io/nodejs-ko/articles/2016/06/09/es-modules-and-node-js-hard-choices/

모듈 / CommonJS

node에서 지원하는 일반적인 모듈 패턴

my_module.js

```
const name = "tom";
const age = 10;

module.exports = {
  name,
  age,
  region: "Seoul"
};
```

in_nodejs.js

```
const my_module = require("./my_module");
const { name } = require("./my_module");

console.log(my_module);
console.log(name);
```

my_module_es6.js

```
const name = "tom";
const age = 10;

export default {
  name,
  age,
  region: "Seoul"
};

export {
  name,
};
```

in_react.js

```
import my_module from "./my_module"; // export default를 참조
import { name } from "./my_module";  // export를 참조
```

고차 함수 (High Order Function)

함수를 인자로 받거나 반환이 가능하고, 다른 함수를 조작하는 함수

함수/클래스 역시 모두 객체.

javascript #1

```javascript
function base_10(fn) {
    function wrap(x, y) {
        return fn(x, y) + 10;
    }
    return wrap;
}

function mysum(x, y) {
    return x + y;
}

mysum = base_10(mysum);

console.log(mysum(1, 2));
```

javascript #2

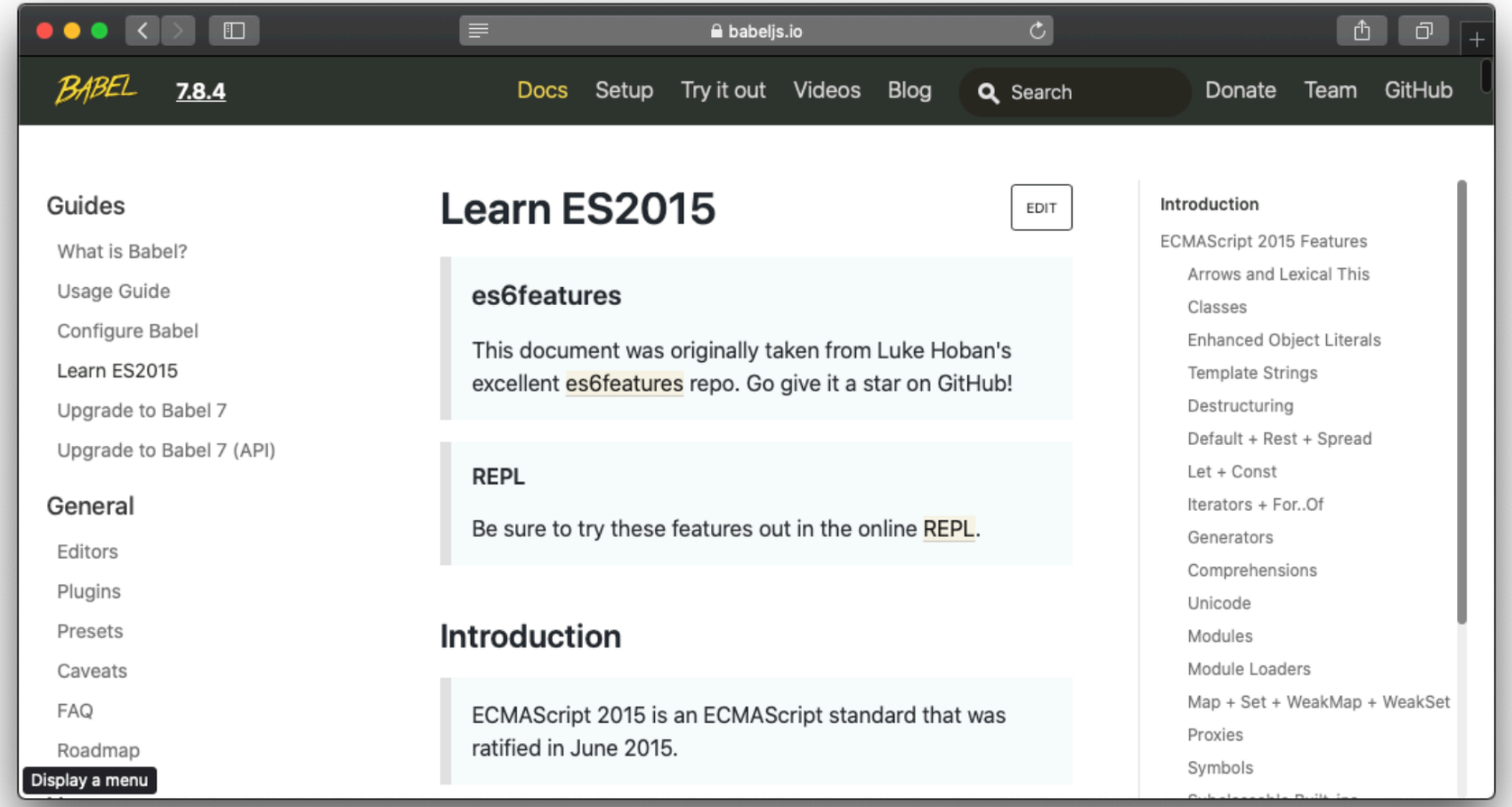
```javascript
const base_10 = fn => (x, y) => fn(x, y) + 10;

let mysum = (x, y) => x + y;
mysum = base_10(mysum);

console.log(mysum(1, 2));
```

비교) python

```python
def base_10(fn):
    def wrap(x, y):
        return fn(x, y) + 10
    return wrap

def mysum(x, y):
    return x + y

mysum = base_10(mysum)

print(mysum(1, 2))
```

참고) ES6 (ECMAScript2015) 문법 더 살펴보기

https://babeljs.io/docs/en/learn



Life is short.
You need Python and Django.

I will be your pacemaker.