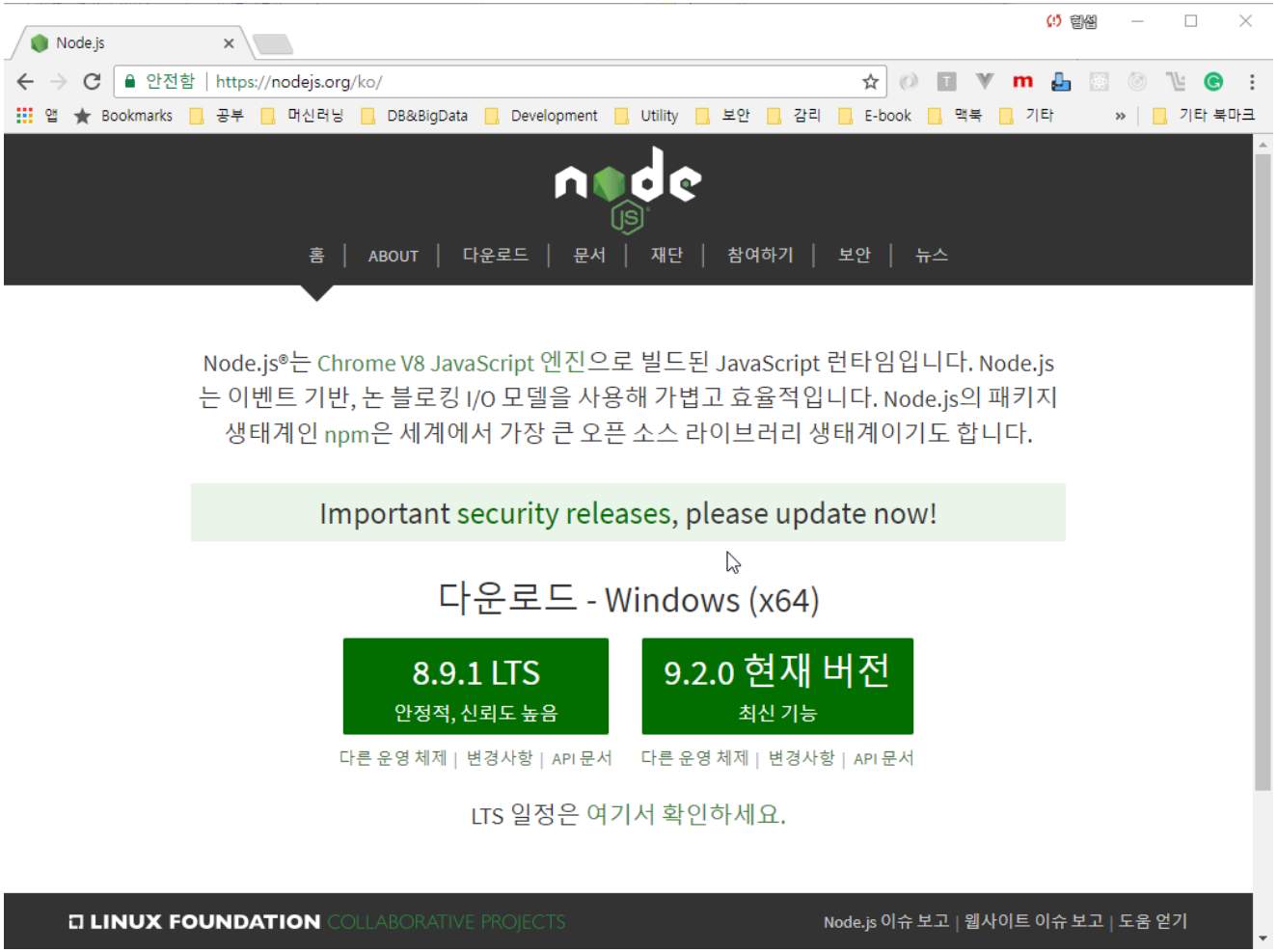
1. 환경설정

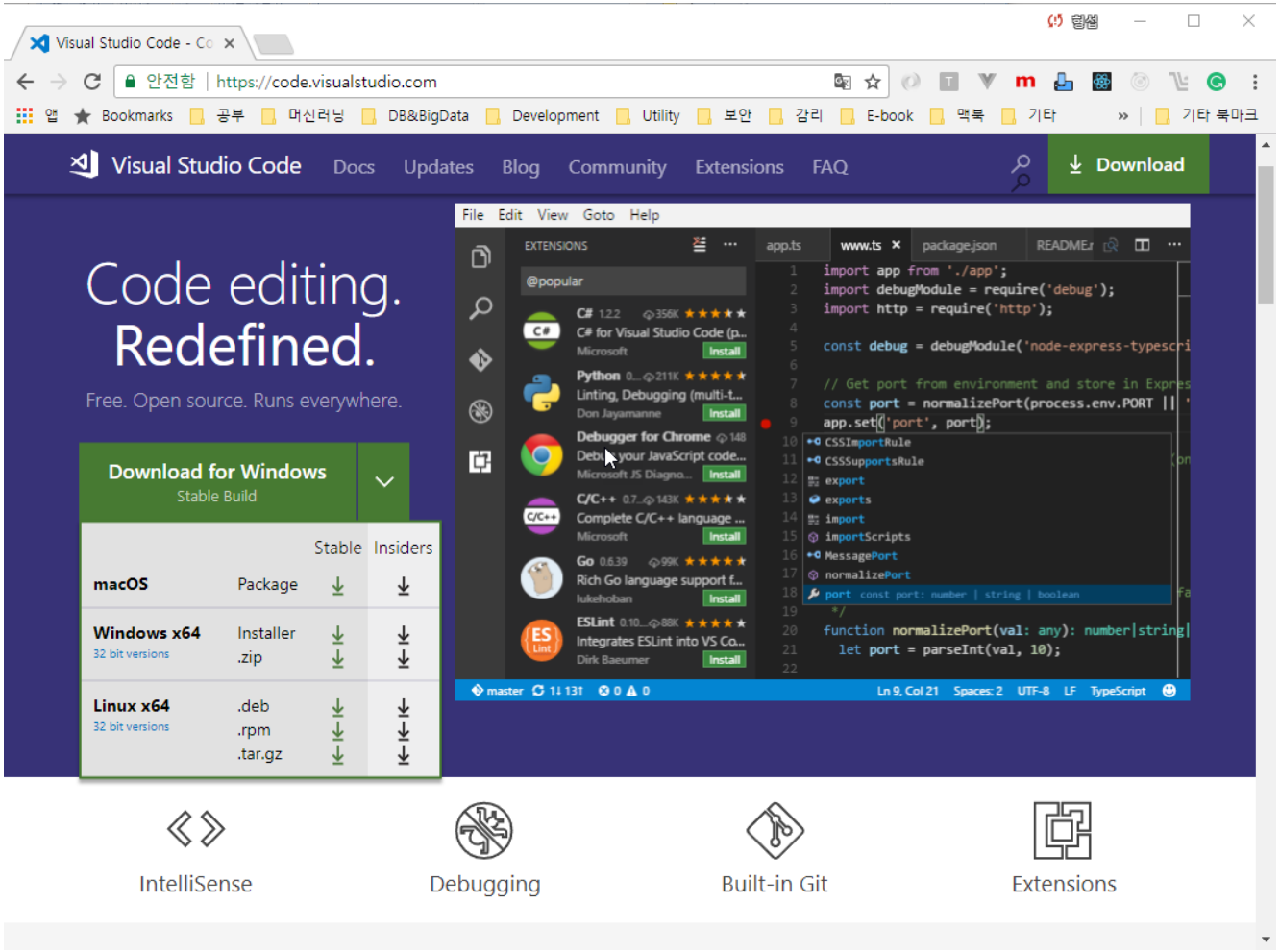
- Node.js 설치
- Visual Studio Code 설치
  - 플러그인 설정

1.1 Node.js 설치

- Node.js 설치



1.2 Visual Studio Code 설치(1)



1.2 Visual Studio Code 설치(2)

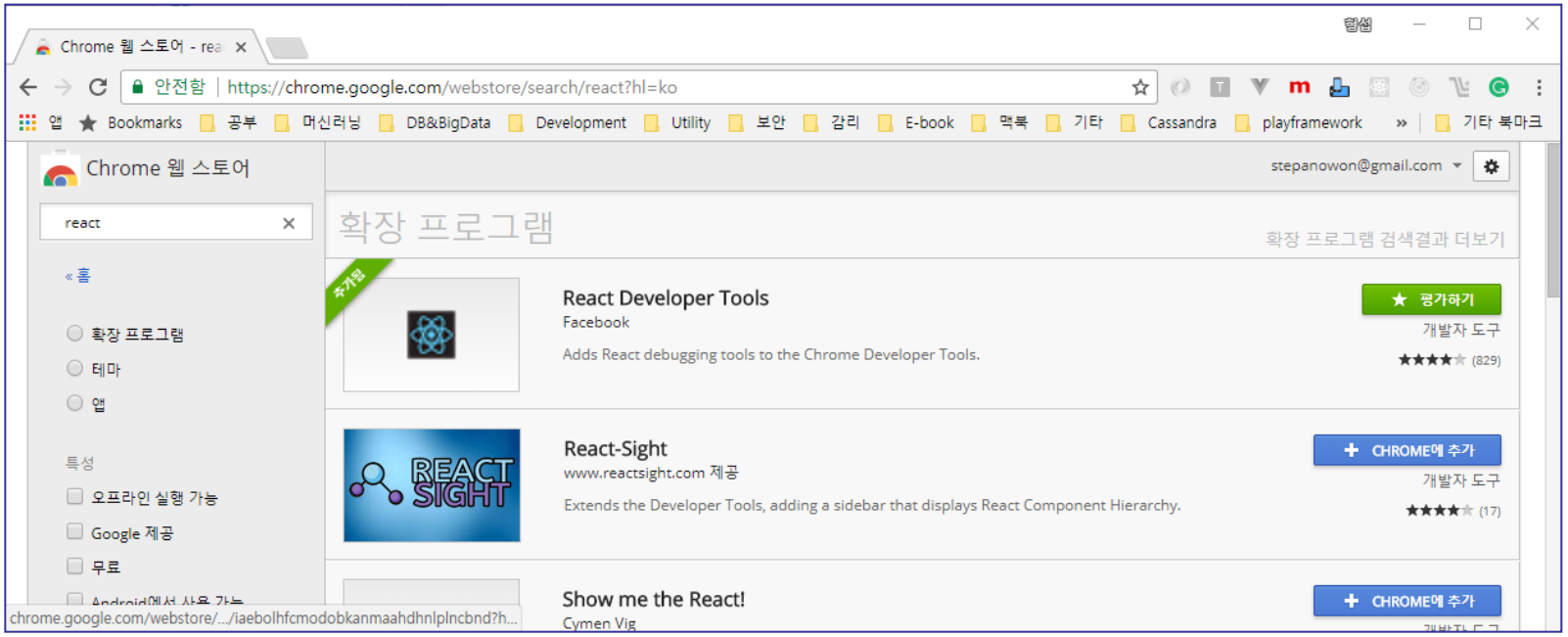
Visual Studio Code 실행 후 필요한 플러그인 설치

- Studio Code 실행 후 Ctl + Shift + X 를 누르거나, "보기" – "명령 팔레트"를 클릭하고 "Extensions: Install Extensions"를 입력하고 엔터키를 누르면 플러그인 확장 기능이 나타난다.
- 여기에서 필요한 플러그인을 설치할 수 있음.
- 이 과정에서 사용할만한 플러그인 리스트
  - Babel ES6/ES7
  - Reactjs code snippets
- 이밖에도 추가적인 플러그인을 설치하여 사용할 수 있음

1.3 Chrom 브라우저 플러그인(1)

React Developer Tools

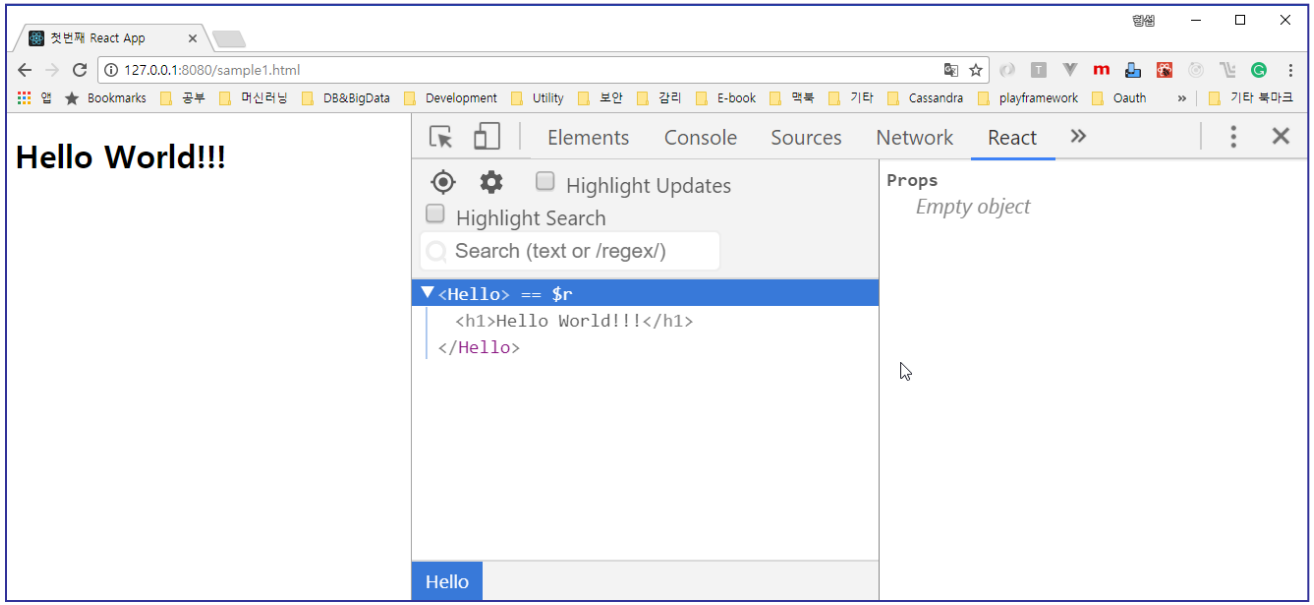
- Chrome Web Store에서 설치함.



1.3 Chrom 브라우저 플러그인(1)

React Developer Tools를 이용한 디버깅

- 컴포넌트의 속성과 상태를 직접 확인할 수 있음.



2. ES 2015

- ECMAScript 2015

2.1 ES 2015 개요

ES2015 개요

- A.K.A. ES6
- 새로운 문법이 추가되었음
- ES2015는 ES5에 대해 하위 호환성을 가짐
- 현재 ES2015를 완벽하게 지원하는 환경은 없음
  - 크롬 : 97%, Edge : 92%, IE11 : 11%, FF: 92%
  - Node.js 6 : 92%
  - http://kangax.github.io/compat-table/es6/
- 대표적인 트랜스 파일러
  - Babel : 71%
  - Typescript : 59%
  - React는 주로 babel을 사용함.

2.2 babel

babel 개요

- ES2015 코드를 ES5 코드로 트랜스파일함.
- 온라인 도구(http://babeljs.io/repl)

2.2 babel(2)

babel 설치

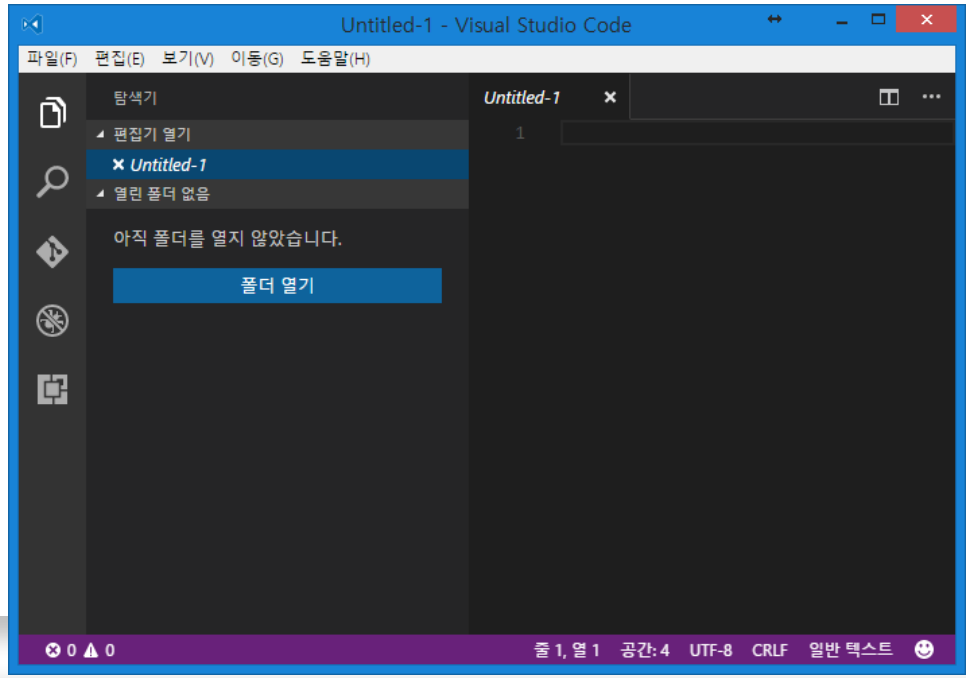
- node.js 설치 후 babel을 설치함.
  - npm install --global babel-cli babel-preset-env

ES 2015 테스트를 위한 프로젝트 생성

- 프로젝트를 위한 디렉토리 생성
- mkdir testapp

Visual Studio Code 실행

- 실행 후 통합 생성한 폴더 열기
  - 파일 메뉴 – 폴더 열기



2.2 babel(3)

프로젝트 초기화

- 보기 – 통합 터미널 실행
- npm init 실행
  - 기본 값으로 입력하거나 적절한 값을 입력함.
  - 아래는 입력한 사례

```
터미널                                                    1: cmd.exe
keywords:
license: (MIT)
About to write to c:\_dev\workspace\testapp\package.json:

{
  "name": "testapp",
  "version": "1.0.0",
  "description": "testapp 입니다.",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "stephen",
  "license": "MIT"
}


Is this ok? (yes) yes
```

2.2 babel(4)

babel을 사용하기 위한 설정

- npm을 이용한 기본 설정
  - npm install babel-cli babel-core babel-preset-env --save-dev
  - --global 옵션은 전역, --save-dev 옵션은 개발 의존성으로 설치
- npm 명령어 수행 후 package.json 확인

```
{
  "name": "test1",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "babel-cli": "^6.26.0",
    "babel-core": "^6.26.0",
    "babel-preset-env": "^1.6.1"
  }
}
```

2.2 babel(5)

.babelrc 파일 작성

- .babelrc 파일은 babel 실행을 위한 기본 설정 파일
- Visual Studio Code에서 .babelrc 파일 추가후 다음과 같이 작성

```
{
   "presets" : [ "env" ]
}
```

테스트 코드 작성

- src 폴더 생성 후 sample.js 파일 추가

```
let name = "world";
console.log(`hello ${name}!!`);
```

- 작성후 통합 터미널에서 babel src -d build 명령어 실행
- build 디렉토리의 sample.js 파일 확인

2.3 source map(1)

- source map?
  - 트랜스파일된 코드로 실행하지만 디버깅은 원본 코드로……
  - source map 스펙은 이미 coffeescript 등에서도 널리 사용하던 것임
    - http://www.html5rocks.com/en/tutorials/developertools/sourcemaps/?redirect_from_locale=ko
    - https://docs.google.com/document/d/1U1RGAehQwRypUTovF1KRlpiOFze0b-_2gc6fAH0KY0k/edit?hl=en_US&pli=1&pli=1
- babel을 이용해 트랜스파일 할 때 source map 생성하기
  - npm install babel-plugin-source-map-support --save-dev
  - babel src/test1.js -o build/test1.js --source-maps
  - babel src -d build --source-maps

2.3 source map(2)

source map 테스트

- src/sample.js

```
let name = "world";
console.log(`hello ${name}!!`);
```

- babel src -d build --source-maps
- build/sample.js

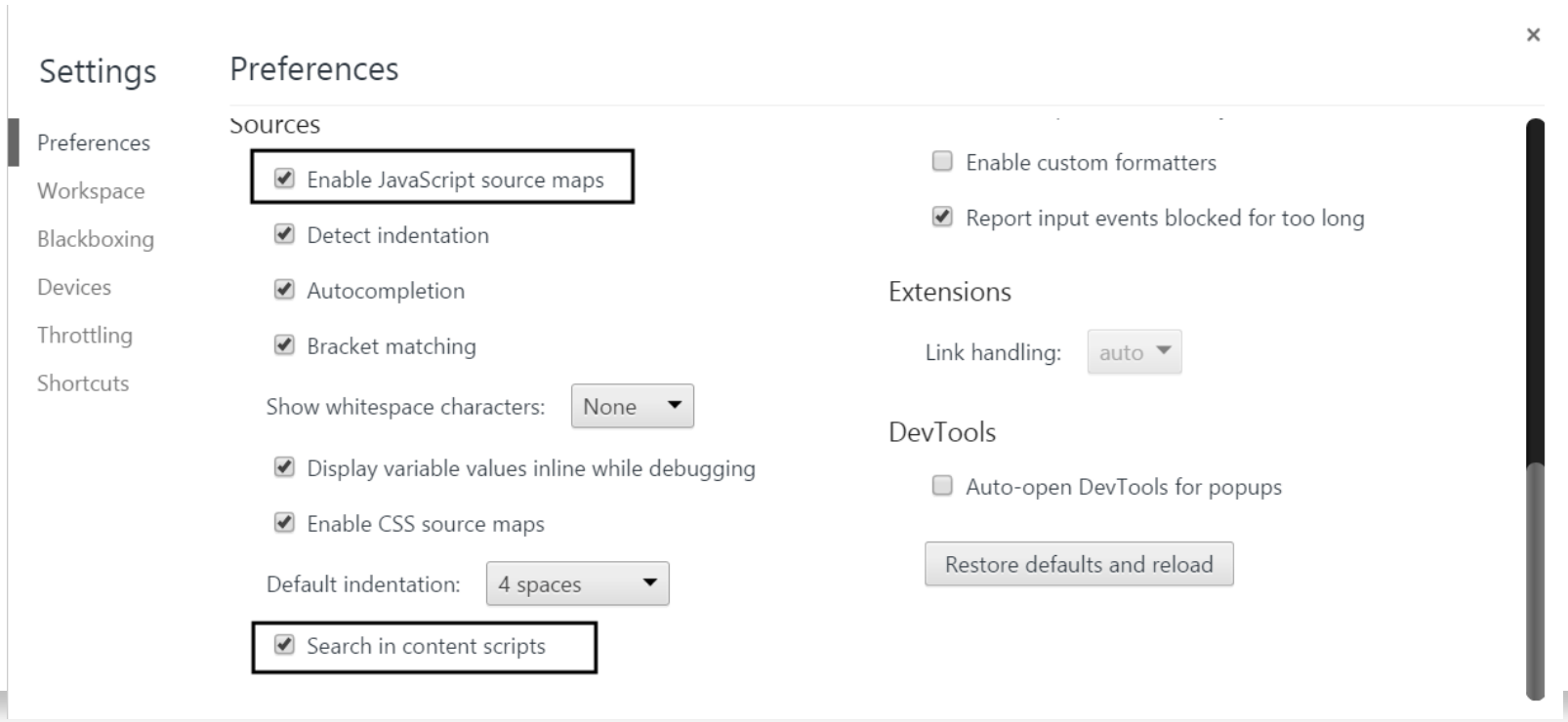
```
"use strict";
var name = "world";
console.log("hello " + name + "!!");
//# sourceMappingURL=sample.js.map
```

- build/sample.js.map
  - 아래 참조

2.3 source map(3)

크롬 브라우저에서 디버깅하기

- 브라우저에서 개발자 도구 실행(ctl+ shift+ i)
  - MacOSX에서는 cmd+ option+ i
- 개발자 도구 메뉴에서 Settings 클릭!!

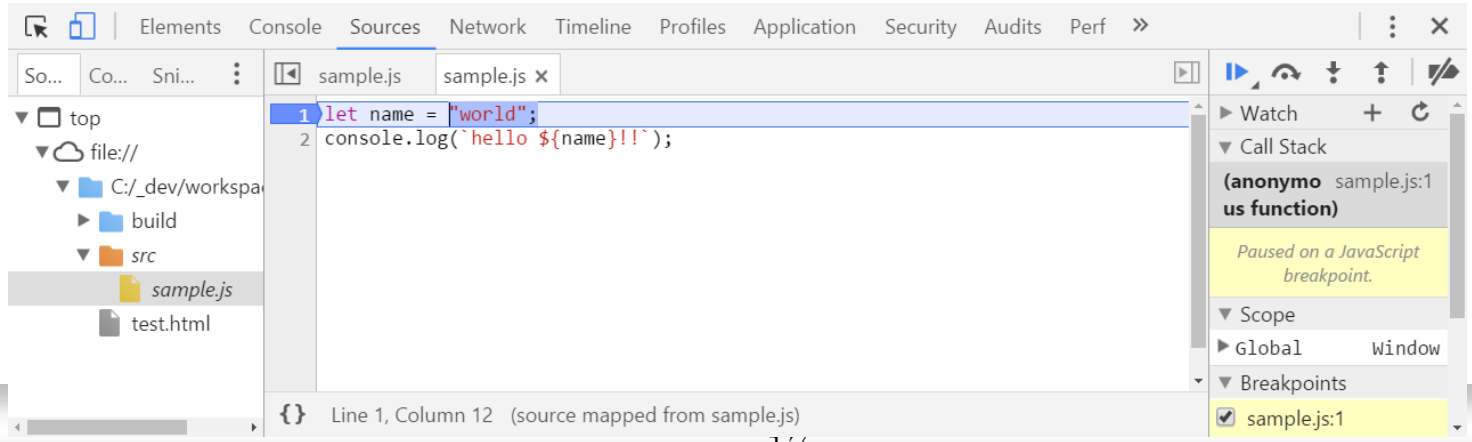


2.3 source map(4)

크롬 브라우저에서 디버깅하기(이어서)

- build/sample.js를 참조하는 페이지 작성 & 실행(test.html)

```
<html>
    <head>
        <title>test!!</title>
        <script type="text/javascript" src="build/sample.js"></script>
    </head>
    <body>
    </body>
</html>
```



3. ES2015 상세

- ES2015에서 추가된 내용 다룸
- Babel을 통해서 사용하게 될 문법 중심

3.1 let, const(1)

var

- hoisting : 개발자들에게 이해하기 어려운 부분
- 함수단위 scope만 제공함
- var 중복 선언을 허용함으로써 혼란 야기

let

- var와 선언하는 방법은 유사하지만...
- 중복 선언을 허용하지 않음

```
> let a = 100;
< undefined
> let a = "hello";
✖ ▶Uncaught TypeError: Identifier 'a' has already been declared(…)    VM78:1
```

3.1 let, const(2)

let (이어서)

- block scope를 지원함.

ES2015

```
let msg= "GLOBAL";
function outer(a) {
    let msg = "OUTER";
    console.log(msg);
    if (true) {
        let msg = "BLOCK";
        console.log(msg);
    }
}
```

ES5

```
"use strict";
var msg = "GLOBAL";
function outer(a) {
   var msg = "OUTER";
   console.log(msg);
   if (true) {
       var _msg = "BLOCK";
       console.log(_msg);
   }
}
```

- 대부분의 var는 let으로 대체가 가능함.

3.1 let, const(3)

- const
  - 상수
  - 값이 한번 초기화되면 변경이 불가능하다.
  - block scope를 지원함.
- 기존의 var는?
  - hoisting!! – 변수를 미리 생성!
  - block scope 지원하지 않음

```
//에러 안남
console.log(A1);
var A1 = "hello";
```

```
var msg = "hello";
function test() {
    console.log(msg);
    if (false) {
        var msg = "world";
    }
    console.log(msg);
}
test();
```

```
undefined
undefined
```

3.2 Default Parameter

- 파라미터 값을 전달하지 않았을 때의 기본값을 정의

```
function addContact(name, mobile,
                    home="없음",
                    address="없음",
                    email="없음") {
    var str = `name=${name}, mobile=${mobile}, home=${home},
              address=${address}, email=${email}`;
    console.log(str);
}

addContact("홍길동", "010-222-3331")
addContact("이몽룡", "010-222-3331", "02-3422-9900", "서울시");
```

```
name=홍길동, mobile=010-222-3331, home=없음, address=없음, email=없음
name=이몽룡, mobile=010-222-3331, home=02-3422-9900, address=서울시, email=없음
```

3.3 Rest Operator

가변 파라미터

- 마지막에 배치해야 함.
- Rest Operator를 지원하지 전에는 arguments를 이용해 가변인자를 처리하였음 → 더이상 arguments를 이용하지 않아도 됨.

```
function foodReport(name, age, ...favoriteFoods) {
    console.log(name + ", " + age);
    console.log(favoriteFoods);
}

foodReport("이몽룡", 20, "짜장면", "냉면", "불고기");
foodReport("홍길동", 16, "초밥");
```

```
이몽룡, 20
[ '짜장면', '냉면', '불고기' ]
홍길동, 16
[ '초밥' ]
```

3.4 Destructuring(1)

구조 분해 할당

- 배열, 객체의 값들을 여러 변수에 추출하여 할당할 수 있도록 하는 새로운 표현식

```
let arr = [10,20,30,40];
let [a,b,c] = arr;
console.log(a, b, c);
```

```
10 20 30
```

```
let p1 = {name:"홍길동", age:20, gender:"M"};
let { name:n, age:a, gender } = p1;
console.log(n,a,gender);
```

```
홍길동 20 M
```

3.4 Destructuring(2)

구조 분해 할당(이어서)

```
function addContact({name, phone, email="이메일 없음", age=0}) {
    console.log("이름 : " + name);
    console.log("전번 : " + phone);
    console.log("이메일 : " + email);
    console.log("나이 : " + age);
}

addContact({
    name : "이몽룡",
    phone : "010-3434-8989"
})
```

```
이름 : 이몽룡
전번 : 010-3434-8989
이메일 : 이메일 없음
나이 : 0
```

3.5 Arrow Function Expression(1)

화살표 함수 표현식

- function에 비해 짧은 코드

```
//var test = function(a,b) {
//    return a+b;
//}

//let test = (a,b) =>{
//    return a+b;
//};

let test = (a,b) => a+b;
console.log(test(3,4));     //7
```

3.5 Arrow Function Expression(2)

Lexical this binding

- function 표현은 함수를 호출할 때 this가 바인딩됨.
- 따라서 어떻게 호출하느냐에 따라 this가 Global이 될 수 있음

```
function Person(name, yearCount) {
    this.name = name;
    this.age = 0;
    for (var i=1; i <= yearCount; i++) {
        incrAge();
    }
    function incrAge() {
        this.age++;
    }
}
var p1 = new Person("홍길동",20);
console.log(p1.name + "님의 나이 : " + p1.age);
//--age는 00이 출력됨.
```

3.5 Arrow Function Expression(3)

Lexical this binding(이어서)

- Arrow function expression을 이용하면 this 바인딩 문제 해결!!

```
function Person(name, yearCount) {
    this.name = name;
    this.age = 0;
    let incrAge = ()=> this.age++;

    for (var i=1; i <= yearCount; i++) {
        incrAge();
    }
}
var p1 = new Person("홍길동",20);
console.log(p1.name + "님의 나이 : " + p1.age);
```

3.6 Object Literal(1)

새로운 객체 리터럴

- 객체 속성 표기

```
var name = "홍길동";
var age = 20;
var email = "gdhong@test.com";
var obj = { name, age, email };

console.log(obj);
```

- 속성명과 변수명이 같은 경우는 생략 가능

```
var obj = { name: name, age: age, email: email };
```

3.6 Object Literal(2)

새로운 객체 리터럴(이어서)

- 새로운 메서드 표기법

```
let p1 = {
    name : "아이패드",
    price : 200000,
    quantity : 2,
    order : function() {
        if (!this.amount) {
            this.amount = this.quantity * this.price;
        }
        console.log("주문금액 : " + this.amount);
    },
    discount(rate) {
        if (rate > 0 && rate < 0.8) {
            this.amount = (1-rate) * this.price * this.quantity;
        }
        console.log((100*rate) + "% 할인된 금액으로 구매합니다.");
    }
}
p1.discount(0.2);
p1.order();
```

3.7 Symbol

Symbol이란?

- 고유성이 보장되는 불변의 값을 나타내기 위한 primitive data type
- 프로그램 내부에서 이름의 충돌 위험 없이 속성의 키로 사용할 수 있음

생성

- let s = Symbol(description);
- new 키워드를 사용하지 않음

용도

- 고유 식별자의 키로 이용함
- var obj[s] = "hello";
- dot(.) 기호로 접근할 수 없음. 반드시 [s] 형태로만 접근 가능

3.7 Template Literal(1)

backtick(`)으로 묶여진 문자열

- 템플릿 대입문(${}) 로 문자열 끼워넣기 기능 제공
  - 템플릿 대입문에 수식 구문, 변수, 함수 호출 구문 등 모든 표현식이 올 수 있음.
  - 템플릿 문자열을 다른 템플릿 문자열 안에 배치하는 것도 가능
  - ${ 을 나타내려면 $ 또는 {을 이스케이프시킴

```
var d1 = new Date();
var name = "홍길동";
var r1 = `${name} 님에게 ${d1.toDateString() }에 연락했다.`;
```

- 여러줄도 표현가능

```
var product = "갤럭시S7";
var price = 199000;
var str = `${name}의 가격은
    ${price}원 입니다.`;
console.log(str);
```

3.7 Template Literal(2)

Tagged Template Literal

```
var getPercent = function(str, ...values) {
      //str : [ '첫번째 값은 ', '이고, 두번째 값은 ', '이다.' ]
      //values : [ 0.222, 0.78999 ]
}

var v1 = 0.222;
var v2 = 0.78999;
var r2 = getPercent`첫번째 값은 ${v1}이고, 두번째 값은 ${v2}이다.`;
```

- tagged template 함수 뒤에 template literal이 따라오면…
- tagged template 함수
  - 첫번째 인자 : 대입 문자열이 아닌 나머지 문자열들의 배열
  - 두번째 이후 인자 : 대입 문자열에 할당될 값들..

3.8 Collections(1)

- 자바스크립트 배열도 Collection 성격이긴 하나…
  - 순서있는 값의 나열일뿐…
  - ES2015에서 다양한 컬렉션을 지원함.
- ES2015 지원 컬렉션
  - Set, Map, WeakSet, WeakMap
- Set : 집합
  - 고유값들의 집합. 중복 요소를 허용하지 않음.
  - add(), has(), delete(), clear(), values() 메서드
  - size 속성

```
var s1 = new Set();
s1.add("사과");   s1.add("배");
s1.add("사과");   s1.add("포도");
console.log(s1);
```

```
Set { '사과', '배', '포도' }
```

3.8 Collections(2)

Set(이어서)

- 합집합, 교집합, 차집합

```
var john = new Set(["사과", "포도", "배"]);
var susan = new Set(["파인애플", "키위", "배"]);
//합집합
var union = new Set([...john.values(), ...susan.values()]);
console.log(union);
//교집합
var intersection = new Set([...john.values()].filter(e => susan.has(e)));
console.log(intersection);
//차집합
var diff = new Set([...john.values()].filter(e => !susan.has(e)));
console.log(diff);
```

```
Set { '사과', '포도', '배', '파인애플', '키위' }
Set { '배' }
Set { '사과', '포도' }
```

3.8 Collections(3)

Map

- 키/값의 쌍. 키는 고유한 값이어야 함.
- set(key, value), get(key) 메서드
- has(key), delete(key), clear() 메서드
- keys(), values() 메서드
- size 속성

```
let teams = new Map();
teams.set('LG', '트윈스');     teams.set('삼성', '라이온스');
teams.set('NC', '다이노스');   teams.set('기아', '타이거스');
teams.set('한화', '이글즈');   teams.set('롯데', '자이언츠');

console.log(teams.has("SK"));       //false
console.log(teams.get("LG"));       //트윈스
```

3.9 Class(1)

- ES5
  - 유사 클래스 : 함수를 이용해 클래스 기능을 만들어냄
  - 작성이 힘듬
    - 상속 : Prototype으로 구현
    - 캡슐화 : Closure로 구현
- ES2015
  - class 키워드 사용

```
class Polygon {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
}
```

  - 함수는 호이스팅(Hoisting)하지만 class는 그렇지 않음

3.9 Class(2)

class 예

```
class Person {
    constructor(name, tel, address) {
        this.name = name;
        this.tel = tel;
        this.address = address;
        if (Person.count) { Person.count++; } else { Person.count = 1; }
    }
    static getPersonCount() {
        return Person.count;
    }
    toString() {
        return `name=${this.name}, tel=${this.tel}, address=${this.address}`;
    }
}
var p1 = new Person('홍길동', '010-222-3331', '서울시');
var p1 = new Person('이몽룡', '010-222-3332', '경기도');
console.log(p1.toString());
console.log(Person.getPersonCount());
```

```
name=이몽룡, tel=010-222-3332, address=경기도
2
```

3.9 Class(3)

상속

```
class Person {
    ……
}

class Employees extends Person {
    constructor(name, tel, address, empno, dept) {
        super(name,tel,address);
        this.empno = empno;
        this.dept = dept;
    }
    toString() {
        return super.toString() + `, empno=${this.empno}, dept=${this.dept}`;
    }
    getEmpInfo() {
        return `${this.empno} : ${this.name}은 ${this.dept} 부서입니다.`;
    }
}

let e1 = new Employees("이몽룡", "010-222-2121", "서울시", "A12311", "회계팀");
console.log(e1.getEmpInfo());
console.log(e1.toString());
console.log(Person.getPersonCount());
```

3.9 Class(4)

캡슐화

- ES5에서는 클로저 활용함 => ES2015에서는 Symbol 활용
- Symbol : 고유의 불변값을 만들어 속성의 키로 사용

```
"use strict";

let _name = Symbol("name key");
let _tel = Symbol("tel key");
let _address = Symbol("address key");
let _count = Symbol("count key");

class Person {
    constructor(name, tel, address) {
        this[_name] = name;
        this[_tel] = tel;
        this[_address] = address;
        if (Person[_count]) { Person[_count]++; }
        else { Person[_count] = 1; }
    }
    static getPersonCount() {
        return Person[_count];
    }
    //이름은 getter만
    get name () {    return this[_name];    }
    get tel() { return this[_tel]; }
    set tel(tel) { this[_tel] = tel; }
    get address() { return this[_tel]; }
    set address(address) { this[_address] = address;    }

    toString() {
        return `name=${this[_name]}, tel=${this[_tel]},
                address=${this[_address]}`;
    }
}

var p1 = new Person('홍길동', '010-222-3331', '서울시');
//p1.age = "이몽룡";     //에러발생
p1.tel = "010-9999-2222";
console.log(p1.toString());
```

3.9 Class(5)

즉시 실행 함수를 사용하여 모듈 패턴 적용

```
var Person = (function() {
    "use strict";

    let _name = Symbol("name key");
    let _tel = Symbol("tel key");
    let _address = Symbol("address key");
    let _count = Symbol("count key");

    class Person {
        constructor(name, tel, address) {
            this[_name] = name;
            this[_tel] = tel;
            this[_address] = address;
            if (Person[_count]) { Person[_count]++; }
            else { Person[_count] = 1; }
        }
        static getPersonCount() {
            return Person[_count];
        }
        //이름은 getter만
        get name () {    return this[_name];    }
        //getter, sette!!
        get tel() { return this[_tel]; }
        set tel(tel) { this[_tel] = tel; }
        get address() { return this[_tel]; }
        set address(address) { this[_address] = address; }

        toString() {
            return `name=${this[_name]}, tel=${this[_tel]},`
                + `address=${this[_address]}`;
        }
    }

    return Person;
})();
```

3.9 Class(6)

ES5 와 ES2015 비교

- Function
  - ES5 : function
  - ES2015 : function 또는 Arrow Function
- Class
  - ES5 : constructor function
  - ES2015 : class 키워드 사용
- Method
  - ES5 : function을 prototyp에 작성
  - ES2015 : class 내부에 method로 작성
- Constructor
  - ES5 : constructor function
  - ES2015 : class 내에 constructor 작성

3.10 Promise(1)

비동기 처리를 위한 콜백 처리

- Callback Hell : 콜백함수들이 중첩되어 지옥을 경험함
  - 디버깅 어려움.
  - 예외처리 어려움

```
doSomething(param1, param2, function(err, paramx){
    doMore(paramx, function(err, result){
        insertRow(result, function(err){
            yetAnotherOperation(someparameter, function(s){
                somethingElse(function(x){
                    ……
                });
            });
        });
    });
});
```

3.10 Promise(2)

Promise 패턴

- 자바스크립트 비동기 처리를 수행하는 추상적인 패턴

```
var p = new Promise(function(resolve, reject) {
    setTimeout(function() {
        var num = Math.round(Math.random()*20);
        var isValid = num % 2;
        if (isValid) { resolve(num); }
        else { reject(num); }
    }, 1000);
});

p.then(function(num) {
    console.log("SUCCESS : " + num);
}).catch(function(num) {
    console.log("FAIL : " + num);
});

console.log("Hello!!");
```

3.10 Promise(3)

Promise Chaining

- then 메서드의 리턴값은 다시 Promise 객체 리턴 가능 → 연속적인 작업 처리시에 유용함
- Promise 객체를 직접 생성하여 리턴할 수도 있음

```
var p = new Promise(
    function(resolve, reject) {
    setTimeout(function() {
        var num = Math.round(Math.random()*20);
        var isValid = num % 2;
        if (isValid) { resolve(num); }
        else { reject(num); }
    }, 1000);
});
```

```
p.then(function(num) {
    console.log("SUCCESS1 : " + num);
    return num*2;
}).then(function(num) {
    console.log("SUCCESS2 : " + num);
    return num*2;
}).then(function(num) {
    console.log("SUCCESS3 : " + num);
})
```

```
SUCCESS1 : 9
SUCCESS2 : 18
SUCCESS3 : 36
```

3.10 Promise(4)

Promise를 이용한 Callback Hell 문제 해결1

```
var job1 = function(owner){
  return new Promise(function(resolve,reject) {
    setTimeout(function(){
      console.log("작업1 : " + owner);
      resolve(owner);
    },1000);
  });
}

var job2 = function(owner){
  return new Promise(function(resolve,reject) {
    setTimeout(function(){
      console.log("작업2 : " + owner);
      resolve(owner);
    },1000);
  });
}
```

```
var job3 = function(owner){
  return new Promise(function(resolve,reject) {
    setTimeout(function(){
      console.log("작업3 : " + owner);
      resolve(owner);
    },1000);
  });
}

var promise = job1('이몽룡');
promise
  .then(job2)
  .then(job3)
  .then(console.log);
```

```
작업1 : 이몽룡
작업2 : 이몽룡
작업3 : 이몽룡
이몽룡
```

3.10 Promise(5)

Promise를 이용한 Callback Hell 문제 해결2

```
(function(owner){
  return new Promise(function(resolve,reject) {
    setTimeout(function(){
      console.log("작업1 : " + owner);
      resolve(owner);
    },1000);
  });
})("이몽룡")
.then(function(owner){
  return new Promise(function(resolve,reject) {
    setTimeout(function(){
      console.log("작업2 : " + owner);
      resolve(owner);
    },1000);
  });
})
```

```
.then(function(owner){
  return new Promise(function(resolve,reject) {
    setTimeout(function(){
      console.log("작업3 : " + owner);
      resolve(owner);
    },1000);
  });
})
.then(console.log);
```

작업1 : 이몽룡
작업2 : 이몽룡
작업3 : 이몽룡
이몽룡

3.10 Promise(6)

Promise.all()

- 모든 작업이 완료되면 resolve promise를 리턴함.
- 작업 중 하나라도 reject이 되면 reject promise를 리턴함

```
var req1 = new Promise(function(resolve, reject) {
            setTimeout(function() { resolve('작업1'); }, 3000);
});
var req2 = new Promise(function(resolve, reject) {
            setTimeout(function() { resolve('작업2'); }, 1000);
});

Promise.all([req1, req2]).then(function(results) {
            console.log('Then: ', results);
}).catch(function(err) {
            console.log('Catch: ', err);
});
```

3.10 Promise(7)

Promise.race()

- 주어진 promise 들 중에서 하나라도 완료되면 resolve하는 메서드

```
var req1 = new Promise(function(resolve, reject) {
            setTimeout(function() { resolve('작업1'); }, 3000);});
var req2 = new Promise(function(resolve, reject) {
            setTimeout(function() { resolve('작업2'); },1000);});

Promise.race([req1, req2]).then(function(results) {
            console.log('Resolve : ', results);
}).catch(function(err) {
            console.log('Reject : ', err);
});
```

3.11 Module(1)

Module

- 여러 디렉토리와 파일에 나눠서 코드를 작성할 수 있도록 함.
- 자바스크립트 파일은 모듈로써 임포트 될 수 있음

Export

- 모듈안에서 선언된 모든 것은 local
- 모듈 내부의 것들을 public으로 선언하고 다른 모듈에서 이용할 수 있도록 하려면 export 해야 함.
- export 대상 항목
  - let, const, var, function, class
- export let a= 1000;
- export function f1(a) { … }
- export { n1, n2 as othername, … }

3.11 Module(2)

- Import
  - 다른 모듈로부터 값, 함수, 클래스들을 임포트할 수 있음
  - import * as obj from '모듈 경로'
  - import { name1, name2 as othername, … } from '모듈 경로'
  - import default-name from '모듈 경로'

3.11 Module(3)

Basic Example

utility.js

```
function generateRandom() {
    return Math.random();
}
function sum(a,b) {
    return a+b;
}
export { generateRandom, sum }
```

import_test.js

```
import { generateRandom, sum } from './utility';

console.log(generateRandom());
```

3.11 Module(4)

Default export

- default export를 사용해 단일 값을 익스포트, 임포트 할 수 있음

```
var utils = {
    generateRandom : function() {
        return Math.random();
    },
    sum : function(a,b) {
        return a+b;
    }
};
export default utils;
```

```
import utils from './utility';
console.log(utils.generateRandom());
console.log(utils.sum(2,3));
```