Blockchain Applications

2019.09.26

From Last Class

- Blockchain State
  - Blockchain is a state machine changes over transactions
- Transaction
  - Transactions trigger changes of a blockchain state
  - Changes entail token transfers and smart contract deploy/execution
- Smart Contract
  - Smart contracts are programs that can be referenced via addresses
- Using Klaytn SDK

Today’s Agenda

- Blockchain Application
- Frontend Development
- Backend Development
- Introduction to Javascript Programming

Blockchain Application (BApp)

- 블록체인 어플리케이션(BApp)은 블록체인을 사용하는 어플리케이션
  - 기존의 기술로 풀기 어려운 문제들을 블록체인의 특성을 활용하여 풀어내는 것이 목적
- **불변성**과 **투명성**이 대표적인 블록체인의 특성
  - 한번 기록된 정보는 변경할 수 없으며
  - 정해진 규칙(e.g., 블록생성 등 프로토콜이 가진 규칙, 컨트랙트로 구현된 규칙)에 따라 상태를 변경
  - 기록의 내역이 블록에 공개되어 있으므로 누구든지 정보의 진실여부를 확인 가능

BApp들이 블록체인을 사용하는 유형

- As a Payment Channel
  - 토큰을 사용한 결제
- As a Storage
  - 블록체인을 안전한 저장소로 인식
- As a World Computer
  - 모든 노드가 동일한 연산을 수행
  - 어느 한 노드에 의존하지 않는 탈중앙화된 실행 환경

BApp의 유형

- Fully decentralized
  - 사용자(클라이언트)가 직접 블록체인과 통신

- Semi-decentralized with centralized proxy
  - 클라이언트가 블록체인과 통신하기 위해 중개 서버와 통신
  - 블록체인 기반으로 만들어진 서비스가 있고 그 서비스를 사용자들이 사용하는 형태
  - 클라이언트 ⇔ 중개서버 ⇔ 블록체인

Fully Decentralized

- 장점:
  - 높은 투명성
  - 신뢰형성에 필요한 비용 없음
  - 경우에 따라 사용자의 익명성 보장 가능
  - (설치형 BApp의 경우) 관리 비용 낮음
- 단점:
  - 사용자 책임 증가, 어려운 UX
  - 로직 변경 어려움
  - 경우에 따라 사용자가 블록체인에 상시 접속할 필요 및 블록을 복제할 필요 있음

Semi-Decentralized with Centralized Proxy

- 장점:
  - (기존의 서비스들과 동일한) 높은 수준의 UX
  - 사용자가 블록체인과 직접 통신할 필요 없음
  - 로직 변경 비교적 쉬움
- 단점:
  - 신뢰비용 발생
  - 서비스가 Single Point of Failure (SPoF)가 됨
  - 관리 비용 높음

BApp 개발

- 프론트엔드(Frontend)
  - 사용자가 직접 사용하는 프로그램 (e.g., mobile app, web page/app)
  - User Interface (UI), 통신, 이벤트 처리 등을 사용자 환경을 고려하여 개발
  - TX 생성, 서명, 전송 등을 프론트엔드에서 처리
- 백엔드(Backend)
  - 사용자에 눈에 보이지 않는 상시 동작하는 서비스
  - 프론트엔드가 사용자 요청을 전달하면 백엔드가 처리하는 구조
  - 블록체인 동기화 등 컴퓨팅 리소스가 많이 필요한 일을 처리하는데 적합
  - 블록체인 동기화, 블록 파싱(parsing), TX 전달, 가스비 대납 등을 백엔드에서 처리

BApp 개발

- Fully decentralized = Frontend + Blockchain
  - e.g., Web + Klaytn
  - e.g., Android + Klaytn
  - e.g., Windows + Klaytn

- Semi-decentralized = Frontend + Server + Blockchain
  - e.g., Web + Java Server on AWS + Klaytn
  - e.g., Android + Node.js Server on Azure + Klaytn

프론트엔드 개발

- BApp이 실행되는 환경에 따라 개발방법이 달라짐
  - 실행환경: Web, Mobile (Android, iOS), Native (Windows, Linux, macOS)
  - 어느 환경에서 개발하느냐에 따라 개발 언어와 UI/UX 디자인, 사용 SDK가 달라짐
  - Klaytn은 Javascript, Java (Android, Native) SDK를 제공
  - Javascript는 Web, Native (with Node.js support)에서 사용
  - Java는 Android, Native (with JRE)에서 사용
- 프론트엔드 개발에 영향을 끼치는 실행환경 중 하나가 지갑
  - 지갑의 존재유무에 따라 개발방법이 변경
  - 특정 지갑을 사용할 경우 해당 지갑이 제공하는 라이브러리를 사용

지갑 (Wallet)

- TX를 서명하려면 키가 필요
  - 키 → 어카운트
  - 서로 다른 키는 다른 어카운트에 매핑
  - 하나의 어카운트로 여러 BApp을 사용하려는 사용자의 니즈가 존재
- 지갑 = 키를 관리하는 프로그램
  - 키를 보관하고 BApp이 요청할 때마다 보관 중인 키로 TX를 서명
  - 여러 유형의 지갑이 존재
    - 브라우저 플러그인, Dapp 브라우저 내장 지갑, 클라우드 지갑, 디바이스 지갑

지갑을 고려한 BApp 개발

- 어떤 지갑을 사용하느냐에 따라 사용자 환경이 변화
  - 어떤 형태로 BApp을 만들 것인지? 웹앱? 모바일 웹? 모바일 네이티브? 데스크탑?
  - BApp에 지갑을 내장할 것인지 아니면 외부 지갑을 사용할 것인지
  - 외부 지갑을 쓸 경우 개발하려는 BApp 형태와 맞는 지갑이 어떤 것인지?
  - e.g., 웹앱의 경우 Metamask를 사용 가능 (Ethereum)
  - e.g., 모바일 웹 또는 모바일 네이티브의 경우 삼성 블록체인 키스토어를 사용 가능 (Klaytn)

BApp의 목적 및 타겟 사용자를 분석하여 어느 형태로 키를 관리할지 결정

백엔드

- 블록체인 프로토콜 이외의 정보를 관리할 경우 필요
  - UX 향상 및 서비스 구현을 위해 TX 외 다른 정보가 필요할 경우 백엔드를 운영
  - e.g., 문서공증 BApp
    - 사용자가 문서공증을 위해 BApp을 사용
    - 문서의 해시를 포함한 공증기록은 블록체인에 기록
    - 정보의 누설을 원치 않을 경우 문서 원본은 블록체인에 기록하기 어려움
    - 문서를 클라이언트에 보관할 수 있으나 분실의 위험, 여러 장치에서 접근할 수 없는 점 등이 문제
    - 백엔드에서 안전하게 원본 문서를 관리하여 편리한 서비스 구현이 가능
- 서비스 제공자가 실행환경을 결정
  - 개발방법 선택이 비교적 자유로운편이나 대부분의 경우 플랫폼 SDK 존재유무에 따라 방향을 결정

실습 1에서 개발할 BApp

- Fully decentralized BApp을 개발
- Klaytn Docs에 소개된 Count Bapp을 직접 구현
  - https://docs.klaytn.com/bapp/tutorials/count-bapp
  - 지갑을 사용하지 않고 사용자가 키스토어(= 암호화된 키)를 입력하는 방식으로 키를 관리
  - 기초적인 웹 프로그래밍 (HTML, Javascript/React)

Web Programming Crash Course

- 실습 1을 위해 알아야할 자바스크립트, HTML, CSS 프로그래밍 기본
- 개발환경
  - 에디터: Visual Studio Code
  - 브라우저: Chrome 76.0.x
  - Node.js 10.16.3 LTS
- 예제는 [url]에 공개

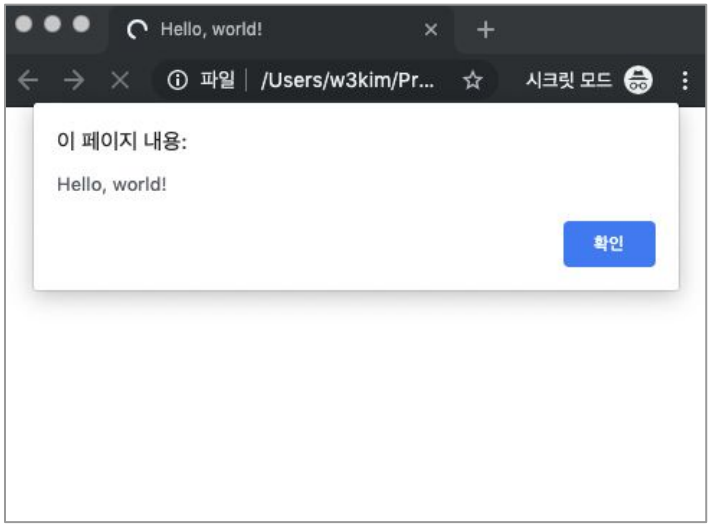
Javascript

- 자바스크립트(JavaScript)는 웹페이지를 동적으로 제어하기 위해서 고안된 언어
  - 웹브라우저에서 동작하도록 설계된 프로그래밍 언어
  - 웹페이지를 표현하는 HTML이 정적(static)이었기 때문에 동적인 요소를 부여하기 위해 고안됨
  - 초기 버전은 HTML로 표현된 요소(element)들의 정보를 읽거나 조작하는데 그쳤으나
  - HTML5의 도입과 함께 브라우저에서 할 수 있는 것들이 많아짐에 따라 활용도가 높아짐
  - ECMAScript 5 표준이 도입됨에 따라 자바스크립트로 구현 가능한 것들이 더 많아질 것

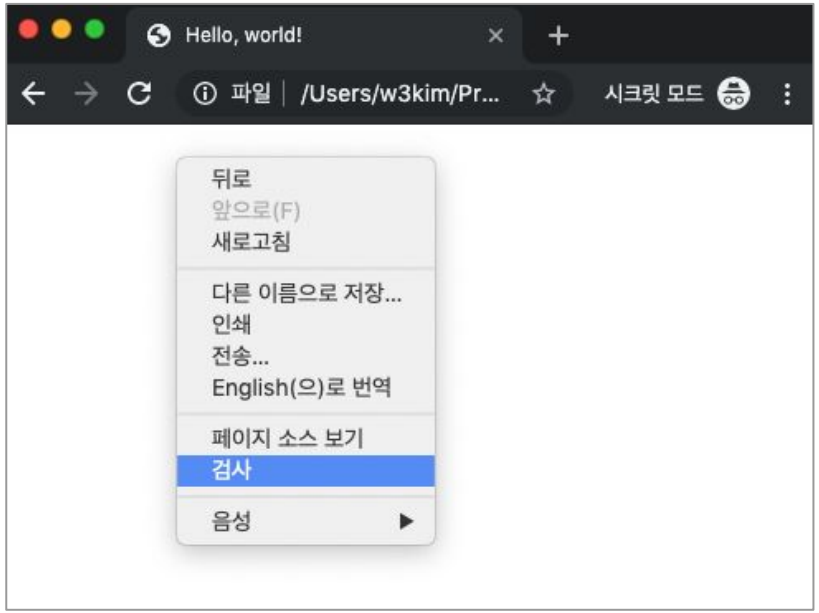
Hello, World

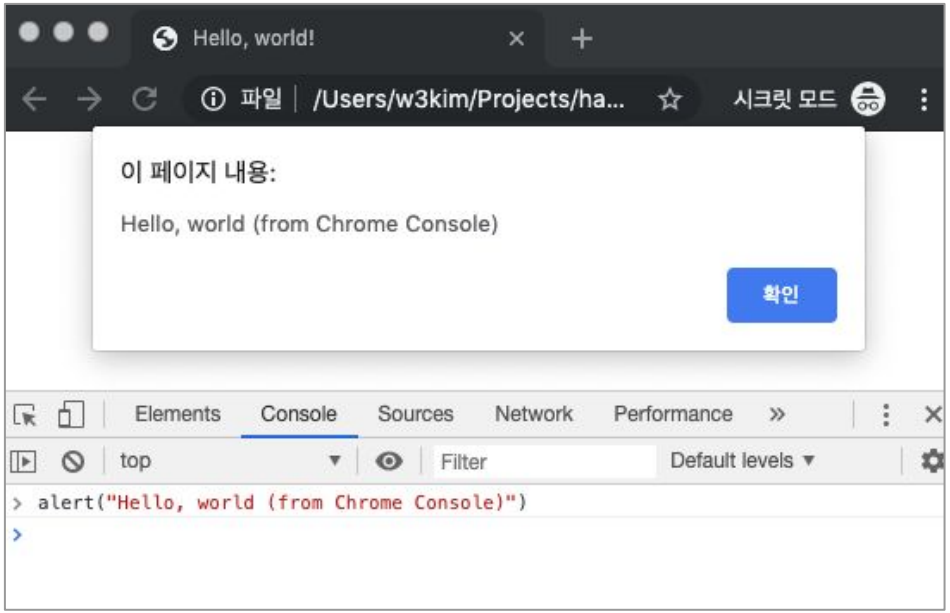
- 자바스크립트는 HTML을 제어 + 동적인 요소를 부여하기 위한 언어
  - 다음 코드는 “Hello, World”라는 문자열과 함께 알림창을 표시

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hello, world!</title>
        <meta charset="utf-8"/>
    </head>
    <body>
        <script>
            alert("Hello, world!");
        </script>
    </body>
</html>
```



Hello, World from Chrome Console





Numbers & Arithmetic Operations

- 자바스크립트에서 숫자와 사칙연산은 다음과 같이 수행

```
<!DOCTYPE html>
<html>
    <body>
        <script>
            console.log(1 + 1);     // should output 2
            console.log(1.5 + 1);   // 2.5
            console.log(5 - 3);     // 2
            console.log(2 * 4);     // 8
            console.log(10 / 2);    // 5
            console.log(10 / 3);    // 3.3333333333333335
            console.log(-3);        // -3
        </script>
    </body>
</html>
```

Math

- 자바스크립트는 수학연산을 위해 Math 객체를 제공

```
<!DOCTYPE html>
<html>
    <body>
        <script>
            console.log(Math.pow(2, 3));      // should output 8
            console.log(Math.abs(-3));        // 3
            console.log(Math.ceil(10 / 3));   // 4
            console.log(Math.floor(10 / 3));  // 3
            console.log(Math.round(10 / 3));  // 3
            console.log(Math.sqrt(4));        // square root; 2
            console.log(Math.random());       // some number between [0, 1.0) *unsafe*
        </script>
    </body>
</html>
```

String

- 자바스크립트의 문자열은 single-quote, double-quote 두가지 모두 허용

```
<!DOCTYPE html>
<html>
    <body>
        <script>
            console.log('single works');
            console.log("double works too");
            console.log("here's one method of expressing apostrophe");
            console.log('and here\'s another');
            console.log('multiple lines...\nline 1\nline 2');
            console.log('plus concats' + ' strings');
            console.log('check the length of this string', 'check the length of this string'.length); // 31
        </script>
    </body>
</html>
```

Variables and Constants

- 자바스크립트는 상태를 저장하기 위해 변수와 상수를 사용

```
<!DOCTYPE html>
<html><body>
    <script>
        var x = 10;     // declaring and initializing a variable
        var y;          // declaration
        y = 20;         // and initialization can be separated
        console.log(x + y); // should output 30
        x = 20;         // updating a variable is okay
        const c = 10;   // constant declaration
        console.log(x + c); // resulting 30
        c = 20;         // ERROR, TypeError: Assignment to constant variable.
    </script>
</body></html>
```

Arithmetic with Variables

- 변수에 연산을 수행; short-hand는 사칙연산만 가능

```
<!DOCTYPE html>
<html><body>
    <script>
        var x = 10;
        console.log(x + 1);     // should output 11
        x = x - 5;              // changes x to be (x - 5)
        console.log(x);                 // 5
        console.log(Math.pow(x, 2));    // 25
        x -= 10;                // translated as x = x - 10
        console.log(x);         // -5
        x += 10;                // translated as x = x + 10
        console.log(x);         // 5
    </script>
</body></html>
```

String Operations with Variables

```
<!DOCTYPE html>
<html>
    <body>
        <script>
            var x = "first";
            console.log(x, x.length);   // should output as ‘first’ 5
            x += " second";
            console.log(x);             // ‘first second’
            x += 10;                    // treated as string; 10 is appended at the end of x
            console.log(x);             // ‘first second10’
            x = "abcde";
            console.log(x[0]);          // ‘a’
            console.log(x.charAt(0));   // ‘a’
            console.log(x.substring(1, 3)); // extracts a substring of x; returning ‘bc’
        </script>
    </body>
</html>
```

Comparison 1: ==

```
<!DOCTYPE html>
<html>
    <body>
        <script>
            var x = 1;
            var y = 1;
            // == evaluates both LHS and RHS and returns `true` if the outcomes are equal to each other
            // otherwise `false`
            console.log(x == 1);        // true
            console.log(x == y);        // true
            console.log(x == 2);        // false
            console.log("a" == "a");    // true; works for strings too
            console.log("a" == "b");    // false
        </script>
    </body>
</html>
```

Comparison 2: ===

```
<!DOCTYPE html>
<html>
    <body>
        <script>
            // Recal == *evaluates* both LHS and RHS before comparison
            // === on the other hand literally compares LHS and RHS including the type
            console.log(eval('1'));     // `eval` evaluates input; returning 1
            console.log(1 == '1');      // true because eval('1') is 1
            console.log(1 === '1');     // false because 1 is number and '1' is string
        </script>
    </body>
</html>
```

Comparison 3: inequalities

```
<!DOCTYPE html>
<html><body>
    <script>
        var x = 1; var y = 2;
        // LHS < RHS returns `true` iff eval(LHS) is strictly less than eval(RHS); otherwise false
        console.log(x < y);         // true
        console.log(y < x);         // false
        // LHS > RHS returns `true` iff eval(LHS) is strictly greater than eval(RHS); otherwise false
        console.log(x > y);         // false
        console.log(y > x);         // true
        // LHS != RHS returns `true` iff eval(LHS) is not equal to eval(RHS); otherwise false
        console.log(x != y);        // true
        console.log(x != '1');      // false
        console.log(x !== '1');     // true; !== works similarly with ===
    </script>
</body></html>
```

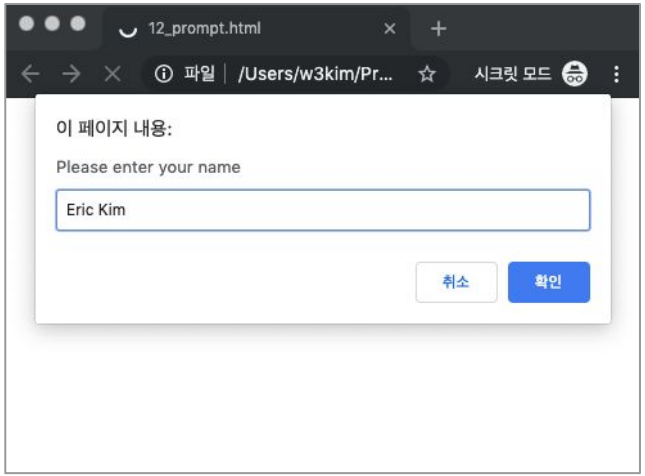
Literal, null, undefined, NaN

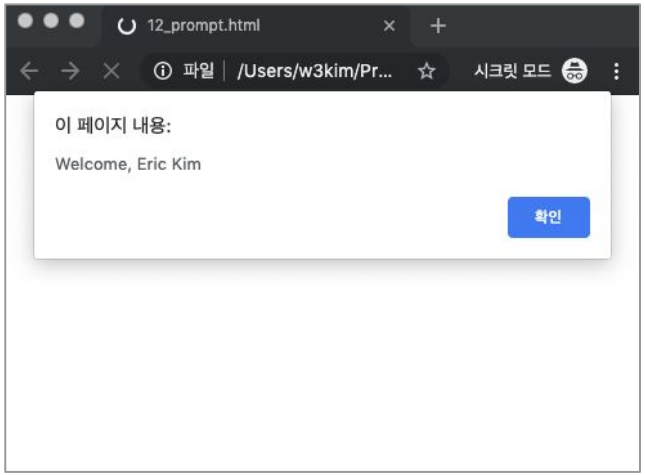
- 숫자(e.g., 1, -8, 1.45), 문자열(e.g., ‘a’, ‘hello’) 등 언어에서 부여한 고정값을 리터럴(literal)이라고 정의
- null은 비어있는 값을 의미하며 실제로 값으로 인식
- undefined는 값이 저장되어 있지 않은 상태를 표현
- NaN은 ‘Not a Number’ 즉 숫자가 아님을 표현

prompt 함수

- 사용자에게 입력을 받아 프로그램에 전달

```
<!DOCTYPE html>
<html>
<body>
    <script>
        var userInput = prompt("Please enter your name");
        alert("Welcome, " + userInput);
        console.log(userInput);
    </script>
</body>
</html>
```





Boolean Condition

- Boolean은 true와 false 두가지 값을 가지는 자료형
- Boolean은 AND(&&)와 OR(||) 연산이 가능
  - true AND true → true && true === true
  - true AND false → true && false === false
  - false AND true → false && true === false
  - true OR false → true || false === true
  - false OR true → false || true === true
  - false OR false → false || false === false
- Boolean으로 연산 가능한 statement를 조건문(conditional statement)이라 함

Condition 1: if

- if 블록은 주어진 조건이 true로 연산될 때만 실행되는 코드

```
if (condition1) {
    // if-block
    // condition1 === true
    console.log(1);
}
console.log(2);
```

Condition 2: if-else

- if-else는 하나의 조건이 주어질 때 조건결과에 따라 다른 실행이 필요할 때 사용

```
if (condition1) {
    // if-block
    // condition1 === true
} else {
    // else-block
    // condition1 === false
}
```

Condition 3: if, else if

- if, else if는 여러 조건이 주어질 때 조건결과에 따라 다른 실행이 필요할 때 사용

```
if (condition1) {
    // if-block
    // condition1 === true
} else if (condition2) {
    // else-if-block
    // condition2 === true
}
```

Condition 4: if, else if, else

- 여러 조건 + 조건이 모두 충족되지 않을 때의 실행을 위해 다음과 같이 표현

```
if (condition1) {
    // condition1 === true
} else if (condition2) {
    // condition1 === false && condition2 === true
} else {
    // condition1 === false && condition2 === false
}
```

Condition 4: nested conditions

- 복잡한 조건을 표현하기 위해 if 문을 중첩(nesting)사용

```
if (condition1) {
      // condition1
      if (condition2) {
            // condition1 && condition2
            if (condition3) {
                  // condition1 && condition2 && condition3
            }
      } else {
            // condition1 && !condition2
      }
}
```

Condition Example 1

```
<!DOCTYPE html>
<html>
<body>
    <script>
        var cond = true; // change this value to see different results
        if (cond) {
            alert("if block");
        } else {
            alert("else block");
        }
    </script>
</body>
</html>
```

Condition Example 2

```
<!DOCTYPE html>
<html><body>
    <script>
        var x = Number(prompt('Please enter a number')); // Number(x) returns a number iff x is a number; NaN otherwise
        if (!x) { // NaN is interpreted as `false`; negating NaN results `true`; this block gets executed if x is NaN
            alert('Not a number!');
        } else {
            if (x < 10) {
                alert('Small');
            } else {
                alert('Big');
            }
        }
    </script>
</body></html>
```

Arrays

- 배열(어레이, Array)은 여러 데이터를 하나의 자료로 묶어주는 자료형
  - 대괄호([])로 표현되며 순서가 결정되어 있음
  - [] → 비어있는 배열
  - [1, 2, 3] → 3개의 숫자가 들어있는 배열
  - 배열 안의 자료를 원소(element)라 부르며 원소의 인덱스는 0부터 시작
- 배열 조작은 다음 링크를 참조
  - https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array

Array Example

```
<!DOCTYPE html>
<html><body>
    <script>
        var array = []; // an empty array
        console.log(array.length);      // length of an array; returning 0
        array.push(7);                  // adding an element
        console.log(array.length);      // 1
        console.log(array[0]);          // 7
        array.push(9);                  // adding another
        console.log(array.length);      // 2
        console.log(array[1]);          // 9
        array = [1, 2, 3, 4, 5];        // initializing an array with 5 elements
        console.log(array[3]);          // returning fourth item - 4
    </script>
</body></html>
```

Functions 1

- 함수(Function)란 코드를 재실행 할 수 있도록 정의한 것

```
<!DOCTYPE html>
<html>
<body>
    <script>
        // function is a keyword indicating a function definition
        function add1(num) {    // add1 is function name; num is the only parameter add1 accepts
            return num + 1;     // function body
        }
        console.log(add1(10));  // 11
        console.log(add1(19));  // 20
        console.log(add1(99));  // 100
    </script>
</body>
</html>
```

Functions 2

```
<!DOCTYPE html>
<html>
<body>
    <script>
        function add(a, b) {        // accepting two parameters
            return a + b;
        }
        console.log(add(1, 1));     // 2
        console.log(add(2, 8));     // 10
    </script>
</body>
</html>
```

Functions 3

```
<!DOCTYPE html>
<html>
<body>
    <script>
        function quadratic(a, b, c) {   // something complex
            var bsqr = Math.pow(b, 2);
            var less = 4 * a * c;
            var sqrt = Math.sqrt(bsqr - less);
            var top1 = (-1 * b) + sqrt;
            var top2 = (-1 * b) - sqrt;
            var bottom = 2 * a;
            return [top1 / bottom, top2 / bottom];
        }
        console.log(quadratic(1, 2, 1));    // [-1, -1]
        console.log(quadratic(1, 4, 3));    // [-1, -3]
    </script>
</body>
</html>
```

Functions 4

```
<!DOCTYPE html>
<html><body>
    <script>
        var count = 0;
        function increment() {
            count += 1;
            // notice there is no return statement
        }
        console.log(count);     // 0
        increment();
        increment();
        increment();
        console.log(count);     // 3
    </script>
</body></html>
```

Functions 5

```
<!DOCTYPE html>
<html>
<body>
    <script>
        var add1 = function(num) {
            return num + 1;
        }
        console.log(add1(1));   // 2
        console.log(add1(2));   // 3
        console.log(add1(3));   // 4
        console.log(add1(4));   // 5
    </script>
</body>
</html>
```

Objects

- 자바스크립트 객체는 key → value 묶음으로 이해될 수 있음
  - 객체는 키를 문자열로 가지며 키 또는 접근자(Accessor)로 값을 가져올 수 있음
  - 중괄호({})를 사용하여 생성하거나 new 키워드를 사용하여 생성 가능

Object Example 1

```
<!DOCTYPE html>
<html>
<body>
    <script>
        var point = {};         // an empty object representing a cartesian point (x, y)
        console.log(point);     // {}
        point['x'] = 10;
        point['y'] = 15;
        console.log(point);     // {'x': 10, 'y': 15}

        point = {'x': 33, 'y': 7};
        console.log(point);     // {'x': 33, 'y': 7}
        console.log(point.x);   // 33
        console.log(point.y);   // 7
    </script>
</body>
</html>
```

Object Example 2

```
<!DOCTYPE html>
<html><body>
    <script>
        function dist(x1, x2) { … } // a function calculating difference between points (on the same axis)
        function Point(x, y) {       // constructor for Point object
            this.x = x; this.y = y;  // initialize states
            this.getDistanceFrom = function (point) {   // a function calculating 2D distance
                var dx = dist(this.x, point.x);         // between this point and the input point
                var dy = dist(this.y, point.y);
                return Math.sqrt(Math.pow(dx, 2) + Math.pow(dy, 2));
            }
        }
        var p1 = new Point(0, 0);
        var p2 = new Point(1, 1);
        console.log(p1.getDistanceFrom(p2));
    </script>
</body></html>
```

End of Document