

자바스크립트 프로그래밍

목 차

1. 자바스크립트 소개와 개발환경 설정
2. 변수 기본 컨셉
3. 타입 6가지
4. 기본 타입
5. 함수
6. 객체
7. 배열
8. 프로토타입

6차시

자바스크립트 시작하기

자바스크립트 소개

- ☑ 웹 브라우저에서 동작하는 언어로 시작
 - 웹 페이지의 모든 상호작용은 자바스크립트 때문
 - 1995년에 시작되어 현재 가장 널리 사용되는 언어로 간주
 - 2009년 웹 표준 이후로 다양한 프레임워크 등장
 - React, Vue.js, Angular, Svelte 등..

자바스크립트 활용 범위

☑ 웹 개발

- 브라우저의 자바스크립트 성능향상
- 프론트엔드 개발은 가장 활발한 개발 영역 중 하나

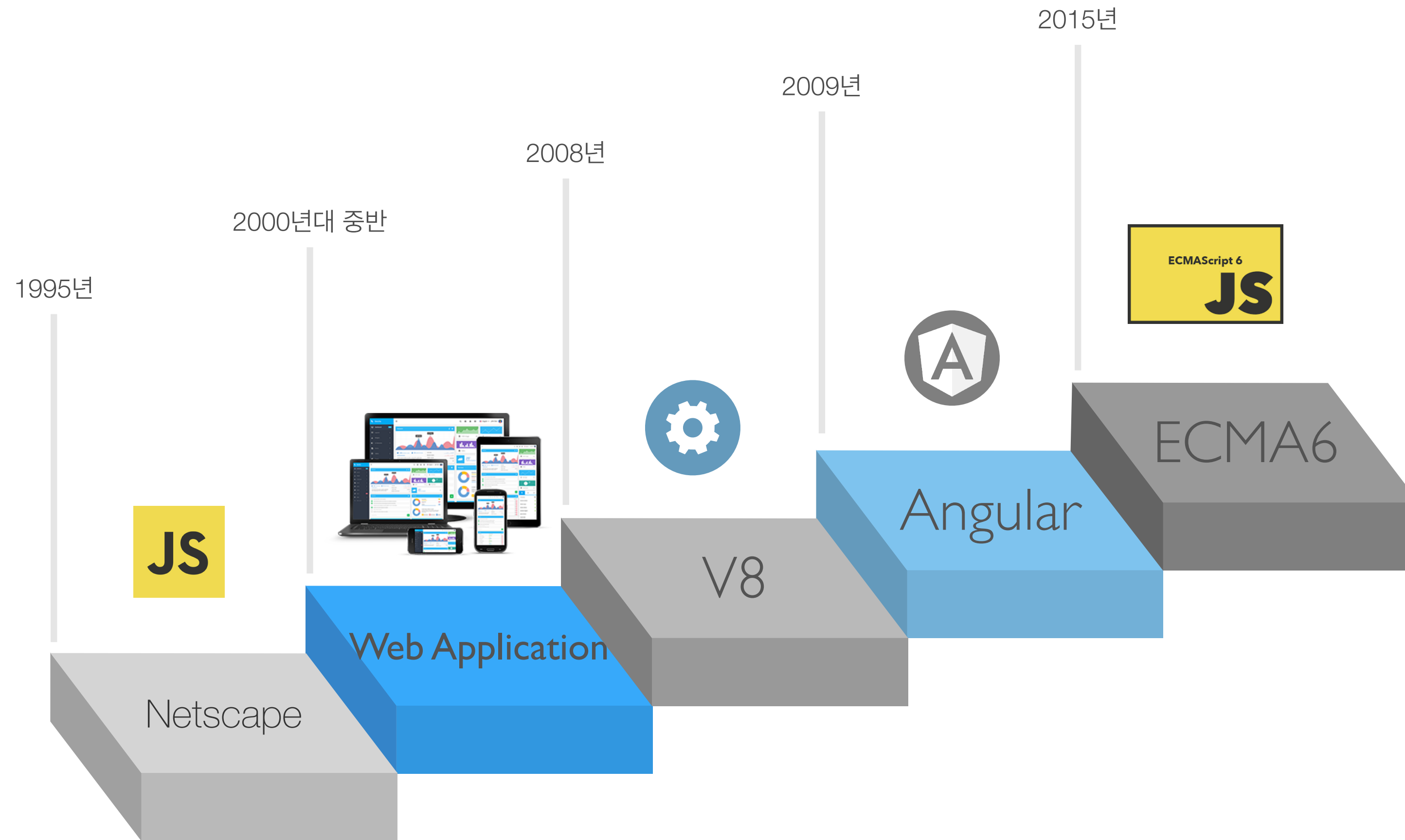
☑ 서버 개발

- 2009년 Node.js 등장
- 브라우저의 자바스크립트를 OS 레벨로 포팅
- 백엔드 개발이 가능해지고 성능도 뛰어남

☑ 애플리케이션 개발

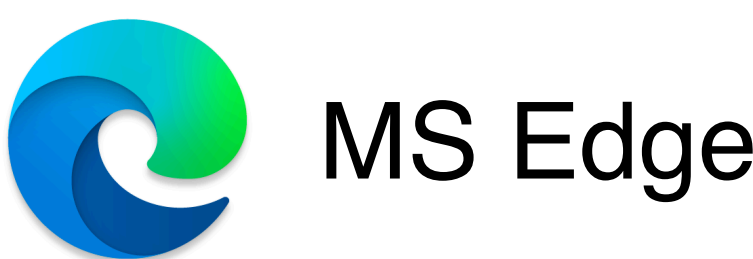
- 크롬 OS
- 데스크탑 애플리케이션 : Electron
- 모바일 애플리케이션 : 하이브리드 앱 (Cordova, phonegap..)

자바스크립트의 발전

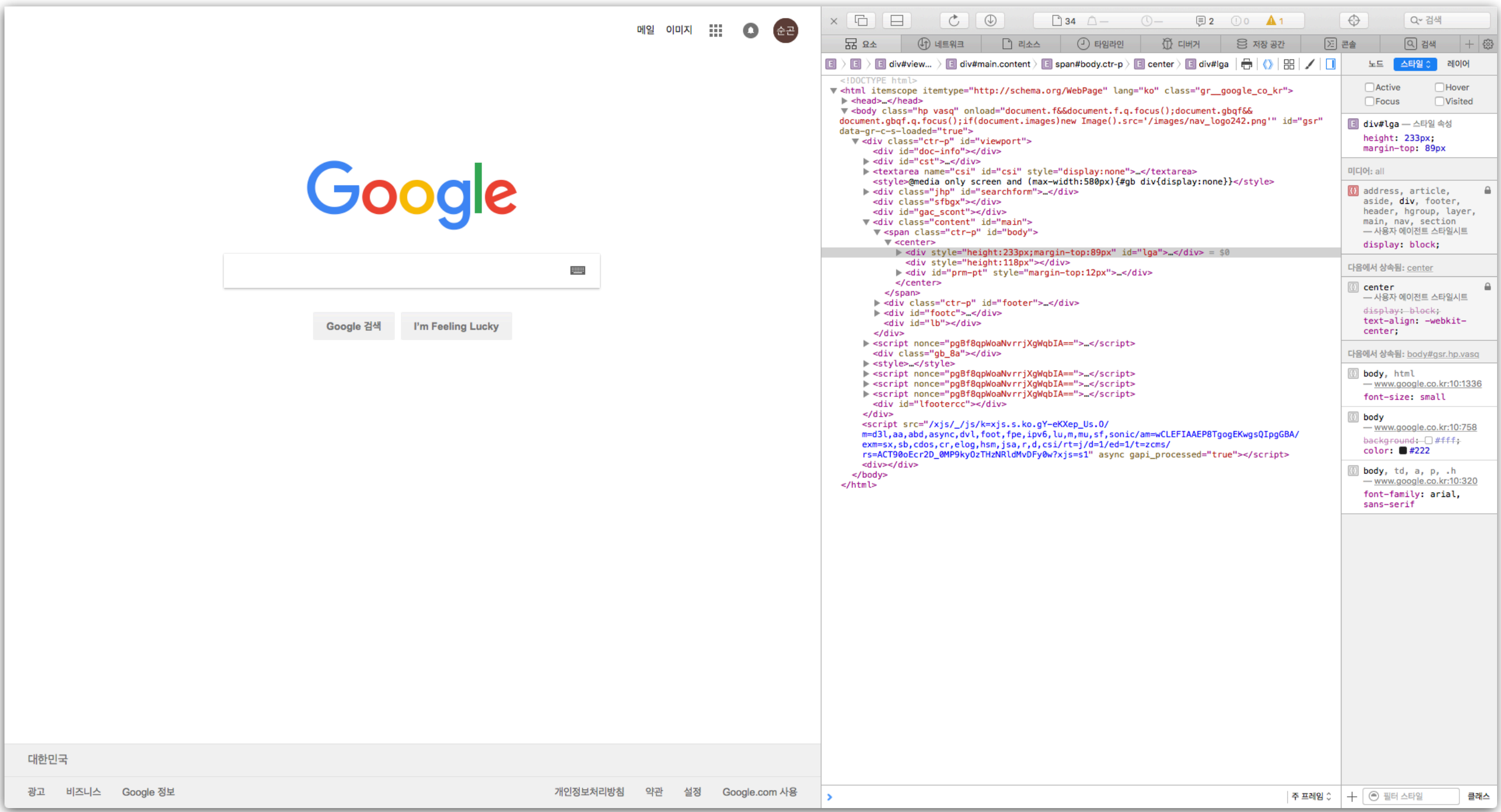


자바스크립트 개발 환경

☑ 웹 브라우저



○ Web Browser 의 Developer Tool 사용



웹 페이지에 JavaScript 추가

☑ HTML 에서 직접

```
<html>
  <script type="text/javascript">
    alert("Hi there!");
  </script>
</html>
```

☑ 외부 파일을 웹 페이지에 연결

```
<html>
  <script type="text/javascript" src="index.js"></script>
</html>
```


자바스크립트 코드 작성

☑ 들어쓰기와 공백

☑ 세미콜론

☑ 코드 주석

☑ 브라우저 개발자 툴 사용

변수

- ☑ 대부분 언어를 배울 때 처음 접하는 빌딩 블록
- ☑ 변수는 값(value, data)
- ☑ 변수를 처음 만드는 것은 —> 선언
 - var, let, const 사용

```
var number1 = 8;  
let number2 = 12;  
const PI = 3.14159;
```

기본 데이터 타입

☑ 문자 : string

```
let singleString = 'Hi there!';  
let doubleString = "How are you?";  
let language = "JavaScript";  
let message = `Let's learn ${language}`;  
console.log(message);
```

☑ 숫자 : number

☑ 정의되지 않은 타입 : undefined, null

기본 데이터 타입

☑ 문자 : string

☑ 숫자 : number

```
let intNr = 1;  
let decNr = 1.5;  
let expNr = 1.4e15;  
let octNr = 0o10; //8진수  
let hexNr = 0x3E8; //16진수  
let boolType = true; //[true | false]
```

☑ 정의되지 않은 타입 : undefined, null

기본 데이터 타입

- ☑ 문자 : string
- ☑ 숫자 : number
- ☑ 정의되지 않은 타입 : undefined, null

```
let unassigned;  
console.log(unassigned);  
let terribleThingToDo = undefined;  
  
let empty = null;
```

기본 데이터 타입

☑ 데이터 타입 알아내기

○ typeof

```
testVariable = 1;  
variableTypeTest1 = typeof testVariable;  
variableTypeTest2 = typeof(testVariable);  
console.log(variableTypeTest1);  
console.log(variableTypeTest2);
```

기본 데이터 타입

☑ 데이터 타입 변환하기

```
let nrToStr = 6;  
nrToStr = String(nrToStr);  
console.log(nrToStr, typeof nrToStr);
```

```
let strToNr = "12";  
strToNr = Number(strToNr);  
console.log(strToNr, typeof strToNr);
```

```
let strToBool = "any string will return true";  
strToBool = Boolean(strToBool);  
console.log(strToBool, typeof strToBool);
```

연산자

☑ 산술 연산자

○ + - * / % **

○ ++ --

☑ 비교 연산자

○ > < >= <= == !=

☑ 논리 연산자

○ && || !

7차시

컬렉션 타입

다중값 데이터 타입

☑ 기본 데이터 타입은 변수에 값을 하나만 저장 가능

☑ 여러 값을 저장 하려면 컬렉션 데이터 타입이 필요

○ 자바스크립트에서는 두 가지 형태로 제공

○ Array : []

○ Object : { }

배열 (Arrays)

☑ 배열 생성

```
const arr1 = new Array("purple", "green", "yellow");  
const arr2 = ["black", "orange", "pink"];
```

☑ 배열의 요소(element) 에 접근

```
const cars = ["Hyundai", "Tesla", "Volkswagen"];  
console.log(cars[0]);  
  
cars[3] = "Kia";  
  
// length 속성  
console.log("Length of cars:", cars.length);
```

배열 (Arrays)

☑ 배열을 다루는 메소드들

- 요소를 추가 : `push()`
- 특정 인덱스에 요소를 추가 : `splice()`
- 요소 삭제 : `pop()`
- 요소 검색 : `find()`
- 정렬과 뒤집기 : `sort()` `reverse()`

객체 (objects)

☑ 속성(property) 을 여러 개 포함

○ 속성은 속성이름과 값으로 구성되어 있음

```
const dog = {  
  dogName: "JavaScript",  
  weight: 2.4,  
  color: "brown",  
  breed: "chihuahua",  
  age: 3,  
};
```

○ 속성 값 접근

```
dog["age"] = "three";
```

8차시

IF 문과 FOR 문

if & for

☑ 논리와 로직을 표현하기 위해서 사용됨

☑ IF 문

- 조건 분기 : 어떤 조건에 따라 다른 코드를 수행하는 것

☑ FOR 문

- 반복적인 작업을 수행

- 주로 컬렉션 데이터 타입에서 사용됨

if 및 if else 문

- ☑ 조건이 참이면 특정 코드를 수행하고 그렇지 않으면 다른 코드를 수행한다.

```
const rain = true;
if(rain) {
  console.log("** 외출할 때 우산을 들고 나간다. **");
} else {
  console.log("** 우산을 두고 외출한다. **");
}
```

```
if(age < 18) {
  console.log("미성년자 관람불가!");
} else {
  console.log("Welcome!");
}
```


if 및 if else 문

☑ 여러 블록이 있는 if 문

```
if(age < 3) {  
    console.log("유아는 무료입니다.");  
} else if(age < 12) {  
    console.log("어린이 할인 5,000원 입니다.");  
} else if(age < 65) {  
    console.log("성인 일반은 10,000원 입니다.");  
} else if(age >= 65) {  
    console.log("경로 할인 7,000원 입니다.");  
}
```

조건부 삼항 연산자

☑ ? :

```
let access = age < 18 ? "denied" : "allowed";
```

```
age < 18 ? console.log("denied") : console.log("allowed");
```

switch 문

☑ if else 문의 조건이 많아 질때 사용

```
if(activity === "Get up") {  
    console.log("It is 6:30AM");  
} else if(activity === "Breakfast") {  
    console.log("It is 7:00AM");  
} else if(activity === "Drive to work") {  
    console.log("It is 8:00AM");  
} else if(activity === "Lunch") {  
    console.log("It is 12.00PM");  
} else if(activity === "Drive home") {  
    console.log("It is 5:00PM")  
} else if(activity === "Dinner") {  
    console.log("It is 6:30PM");  
}
```

```
switch(activity) {  
    case "Get up":  
        console.log("It is 6:30AM");  
        break;  
    case "Breakfast":  
        console.log("It is 7:00AM");  
        break;  
    case "Drive to work":  
        console.log("It is 8:00AM");  
        break;  
    case "Lunch":  
        console.log("It is 12:00PM");  
        break;  
    case "Drive home":  
        console.log("It is 5:00PM");  
        break;  
    case "Dinner":  
        console.log("It is 6:30PM");  
        break;  
}
```

반복문 - for

☑ 특정 횟수만큼 블록을 실행

```
let names = ["Chantal", "John", "Maxime", "Bobbi", "Jair"];  
for (let i = 0; i < names.length; i ++){  
  console.log(names[i]);  
}
```

○ for - of

```
let names = ["Chantal", "John", "Maxime", "Bobbi", "Jair"];  
for (let name of names){  
  console.log(name);  
}
```

반복문 - for

☑ Object 를 for 로 반복

○ for - in

for - in : 키를 반환, for - of : 값을 반환

```
let car = {  
  model: "Golf",  
  make: "Volkswagen",  
  year: 1999,  
  color: "black",  
};  
  
for (let prop in car){  
  console.log(car[prop]);  
}
```

continue & break

- ☑ continue : 현재 코드를 중단하고 다음 반복을 수행
- ☑ break : 반복문을 중단하고 블록을 빠져나감

```
for (let i = 0; i < 10; i++) {  
  console.log(i);  
  if (i === 4) {  
    break;  
  }  
}
```

9차시

함수

자바스크립트 함수

☑ 함수란 무엇인가?

- 일련의 작업을 수행하는 코드의 집합
- 코드를 재사용할 수 있게 하는 강력한 기능

자바스크립트 함수

☑ 자바스크립트 함수의 정의

```
function functionName(parameters) {  
    // 실행될 코드  
}
```

☑ 함수 호출

- 함수를 호출하려면 함수 이름 뒤에 괄호 사용

```
functionName();
```

함수의 종류

☑ 일반함수, Named Function

```
function greet(name) {  
  return `Hello, ${name}!`;  
}
```

☑ 익명함수, Anonymous Function

```
const greet = function(name) {  
  return `Hello, ${name}!`;  
};
```

☑ 화살표 함수, Arrow Function

```
const greet = (name) => `Hello, ${name}!`;
```

자바스크립트 함수의 특징

☑ 일급 객체

- : 함수는 다른 변수에 할당되거나,
- 다른 함수의 인자로 전달되거나,
- 다른 함수의 결과로서 반환될 수 있음

☑ 클로저:

- 함수는 자신이 생성될 때의 환경을 '기억', 이런 특성을 '클로저'라고 합니다.

☑ 호이스팅:

- 함수 선언은 해당 스코프의 최상단으로 '끌어올려' 짐

ES6+ 자바스크립트 기능

☑ 기본 파라미터

```
function greet(name = 'World') {  
  console.log(`Hello, ${name}!`);  
}
```

```
greet(); // "Hello, World!"  
greet('John'); // "Hello, John!"
```

ES6+ 자바스크립트 기능

☑ 기본 파라미터

○ 함수 파라미터에 기본값을 부여 가능

```
function greet(name = 'World') {  
  console.log(`Hello, ${name}!`);  
}
```

```
greet(); // "Hello, World!"  
greet('John'); // "Hello, John!"
```

ES6+ 자바스크립트 기능

☑ 화살표 함수

- 함수 표현식을 더 짧고 간결하게 만들어 줌

```
const numbers = [1, 2, 3, 4, 5];
```

```
// ES5
```

```
const squaresES5 = numbers.map(function (n) {  
  return n * n;  
});
```

```
// ES6
```

```
const squaresES6 = numbers.map(n => n * n);
```

ES6+ 자바스크립트 기능

☑ Rest 파라미터

- 파라미터를 배열로 그룹화할 수 있음
- 항상 마지막 매개변수여야 하며, 전달된 "나머지" 인수들을 모두 포함

```
function sum(...numbers) {  
  return numbers.reduce((previous, current) => {  
    return previous + current;  
  });  
}  
  
console.log(sum(1, 2, 3, 4)); // 10
```

ES6+ 자바스크립트 기능

☑ 스프레드(Spread) 연산자

- 배열이나 이터러블한 객체를 개별 요소로 확장 가능하게 해줌
- 함수 호출 시 매개변수를 전달하는 데 유용함

```
const numbers = [1, 2, 3];  
  
function sum(a, b, c) {  
  return a + b + c;  
}  
  
console.log(sum(...numbers)); // 6
```


배열 데이터 처리

배열 처리 방식 두 가지

☑ 배열을 처리하는 방식은 다음 두 가지로 분류 가능

- Mutable : 원본 배열이 변경되는 방식

- 원본데이터를 보존해야 하는 상황에서는 주의해서 사용
- 메모리 효율성이 좋고, 동작이 직관적임

- Immutable : 원본을 변경하지 않고 변경된 복사본을 반환

- 리액티브 프로그래밍
- 함수형 프로그래밍
- 복잡한 상태 관리에서 주로 사용

배열 처리 방식 두 가지

☑ Mutable : 원본 배열이 변경되는 방식

- push(), pop(), reverse(), splice()

```
let arr = [1, 2, 3, 4, 5];  
  
arr.push(6); // arr: [1, 2, 3, 4, 5, 6]  
arr.pop(); // arr: [1, 2, 3, 4, 5]  
arr.shift(); // arr: [2, 3, 4, 5]  
arr.unshift(1); // arr: [1, 2, 3, 4, 5]  
arr.reverse(); // arr: [5, 4, 3, 2, 1]
```

☑ Immutable : 원본을 변경하지 않고 변경된 복사본을 반환

- concat(), join(), slice()
- map(), filter(), reduce(), forEach()

Spread Operator

Destructuring

펼침 연산자, 구조분해

펼침 연산자, Spread 연산자

☑ 펼침 연산자(Spread operator)

- 배열이나 객체를 쉽게 복사하거나 합칠 수 있게 해줌
- 기호는 세 개의 점(...)

☑ 종류

- 배열 펼침 / 객체 펼침
- 배열 복사 / 객체 복사
- 배열에서의 함수 파라미터
- 객체 속성 오버라이드 / 배열을 문자열로 펼치기

펼침 연산자, Spread 연산자

☑ 배열 펼침

```
const arr1 = [1, 2, 3];  
const arr2 = [...arr1, 4, 5, 6]; // arr2 is now [1, 2, 3, 4, 5, 6]
```

☑ 객체 펼침

```
const obj1 = { a: 1, b: 2 };  
const obj2 = { ...obj1, c: 3 }; // obj2 is now { a: 1, b: 2, c: 3 }
```

펼침 연산자, Spread 연산자

☑ 배열 복사

```
const arr = [1, 2, 3];  
const arrCopy = [...arr]; // 모든 배열 복사
```

☑ 객체 복사

```
const obj = { a: 1, b: 2 };  
const objCopy = { ...obj }; // 모든 객체 복사
```

펼침 연산자, Spread 연산자

☑ 객체의 속성 오버라이드

```
const obj1 = { a: 1, b: 2, c: 3 };  
const obj2 = { ...obj1, b: 20 }; // obj2 is now { a: 1, b: 20, c: 3 }
```

☑ 배열을 문자열로 펼치기

```
const str = 'Hello';  
const chars = [...str]; // chars is now ['H', 'e', 'l', 'l', 'o']
```

☑ 배열 에서의 함수 파라미터

```
const nums = [1, 2, 3];  
Math.max(...nums); // Returns 3
```


구조분해, Destructuring

☑ 구조 분해 할당 (Destructuring Assignment)

- 배열이나 객체의 속성을 해체하여 그 값을 개별 변수에 할당
- 코드를 더 깔끔하게 만들고, 변수 할당을 보다 간결하게 해줌

☑ 종류

- 배열 구조분해
- 객체 구조분해
- 함수 파라미터에서의 구조분해

구조분해, Destructuring

- ☑ 배열 구조분해 - 배열에서 값을 쉽게 추출할 수 있는 방법을 제공

```
const arr = [1, 2, 3, 4, 5];
```

기본 구조분해

```
const [a, b, c, d, e] = arr;
```

```
console.log(a, b, c, d, e); // Outputs: 1 2 3 4 5
```

```
const arr = [1, 2, 3, 4, 5];
```

```
const [a, b, ...rest] = arr;
```

```
console.log(a, b); // Outputs: 1 2
```

```
console.log(rest); // Outputs: [3, 4, 5]
```

나머지 구문 사용하기

구조분해, Destructuring

- ☑ 배열 구조분해 - 배열에서 값을 쉽게 추출할 수 있는 방법을 제공

```
const arr = [1, 2];
```

기본값 사용하기

```
const [a = 'default', b = 'default', c = 'default'] = arr;
```

```
console.log(a, b, c); // Outputs: 1 2 default
```

```
const arr = [1, 2, 3, 4, 5];
```

```
const [a, , c] = arr;
```

```
console.log(a, c); // Outputs: 1 3
```

일부 값 무시하기

구조분해, Destructuring

- ☑ 객체 구조분해 - 객체의 속성을 쉽게 추출할 수 있는 방법을 제공

```
const obj = { a: 1, b: 2, c: 3 };  
  
const { a, b, c } = obj;  
  
console.log(a, b, c); // Outputs: 1 2 3
```

객체 기본 구조분해

```
const obj = { a: 1, b: 2, c: 3 };  
  
const { a: newA, b: newB, c: newC } = obj;  
  
console.log(newA, newB, newC); // Outputs: 1 2 3
```

새로운 변수이름으로 할당하기

구조분해, Destructuring

- ☑ 함수 파라미터 구조분해 - 함수 내부로 객체 속성 간편하게 전달 가능

```
const obj = { a: 1, b: 2, c: 3 };

function print({ a, b, c }) {
  console.log(a, b, c); // Outputs: 1 2 3
}

print(obj);
```

prototype

class

클래스와 프로토타입

클래스, Class

- ☑ 클래스는 객체 지향 프로그래밍(OOP)에 있어 중요한 개념
 - 프로토타입 기반 프로그래밍을 좀 더 쉽고 직관적으로 접근할 수 있게 도와줌
- ☑ 클래스 특징
 - 클래스 선언
 - 생성자
 - 메소드
 - 인스턴스 생성
 - 상속

클래스, Class

☑ 클래스 선언

```
class MyClass {  
    // ...  
}
```

☑ 생성자

```
class MyClass {  
    constructor(param1, param2) {  
        this.param1 = param1;  
        this.param2 = param2;  
    }  
}
```

☑ 메소드

```
class MyClass {  
    myMethod() {  
        console.log('Hello, world!');  
    }  
}
```


클래스 상속, Inheritance

```
// 기본 클래스
class Person {
  constructor(name) {
    this.name = name;
  }

  greet() {
    console.log(`Hello, my name is ${this.name}.`);
  }
}

// Person 클래스를 상속받는 Employee 클래스
class Employee extends Person {
  constructor(name, jobTitle) {
    super(name); // 부모 클래스의 constructor를 호출
    this.jobTitle = jobTitle;
  }

  introduce() {
    console.log(`Hello, I'm ${this.name} and ${this.jobTitle}.`);
  }
}
```

```
const john = new Employee('John Doe', 'Software Developer');
john.greet(); // 출력: Hello, my name is John Doe.
john.introduce(); // 출력: Hello, I'm John Doe and Software Developer.
```

프로토타입, Prototype

☑ 자바스크립트는 프로토타입 기반 언어

- 프로토타입을 통해 객체 간에 속성과 메서드를 상속
- 자바스크립트에서, 모든 객체는 다른 객체로부터 속성을 상속받음
- 이 때, 상속받는 객체를 프로토타입이라고 함

☑ 객체의 프로토타입은 __proto__ 속성을 통해 접근 가능

- 객체가 특정 속성을 갖고 있지 않을 경우, 해당 객체의 프로토타입에서 그 속성을 찾음
- 이를 프로토타입 체인이라고 함
- 만약 프로토타입 체인을 따라가도 해당 속성이 발견되지 않는다면 undefined를 반환

프로토타입, Prototype

- ☑ 자바스크립트는 프로토타입 기반 언어
- ☑ 객체의 프로토타입은 `__proto__` 속성을 통해 접근 가능

```
const obj = {  
  property1: 'value1'  
};  
  
// obj2는 obj를 프로토타입으로 갖게 됩니다.  
const obj2 = Object.create(obj);  
  
// 'value1' - obj2 자신은 property1을 갖지 않지만,  
//프로토타입인 obj에서 해당 속성을 찾을 수 있습니다.  
console.log(obj2.property1);
```

프로토타입, Prototype

☑ 함수 객체의 경우,

- prototype 속성을 통해 프로토타입 객체를 설정할 수 있음
- 해당 함수를 생성자로 사용해 객체를 생성할 때 그 객체의 프로토타입이 됨

```
function MyConstructor() {  
  // ...  
}  
MyConstructor.prototype.myMethod = function() {  
  console.log('Hello, world!');  
};  
// MyConstructor의 인스턴스 생성  
const myObj = new MyConstructor();  
// 'Hello, world!' - myMethod는 myObj의 프로토타입인  
// MyConstructor.prototype에서 찾을 수 있습니다.  
myObj.myMethod();
```

프로토타입, Prototype

- ☑ 프로토타입 체인의 특성은 코드 재사용에 유용
 - 효과적인 메모리 관리를 가능하게 함
 - 모든 인스턴스가 같은 프로토타입을 공유하므로,
 - 각 인스턴스가 동일한 메서드에 대한 복사본을 가질 필요가 없기 때문
- ☑ 프로토타입 체인이 너무 길어지면 성능에 영향을 줄 수 있으므로,
 - 적절한 프로토타입 체인 길이를 유지하는 것이 중요
 - 프로토타입 체인을 이해하지 못하면 예상치 못한 결과를 초래할 수 있으므로,
 - 해당 개념에 대한 이해가 중요

HTML5 APIs

브라우저 내장 객체

HTML5 API

☑ HTML Geolocation

- 사용자 현재 위치를 알려준다.
- 모바일 디바이스에서 정확
- `getCurrentPosition()` 메서드로 사용자 위치 반환

```
<script>
var x = document.getElementById("demo");
function getLocation() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(showPosition);
    } else {
        x.innerHTML = "Geolocation is not supported by this browser.";
    }
}
function showPosition(position) {
    x.innerHTML = "Latitude: " + position.coords.latitude +
    "<br>Longitude: " + position.coords.longitude;
}
</script>
```

HTML5 API

☑ HTML Web Storage

○ local storage




- window.localStorage
- 키/밸류 데이터 저장
- 브라우저 메모리에 저장

○ session storage

- window.sessionStorage
- 로컬 스토리지와 저장방식 동일
- 브라우저(탭)가 종료되면 데이터 삭제됨

Browser Support

The numbers in the table specify the first browser version that fully supports Web Storage.

API					
Web Storage	4.0	8.0	3.5	4.0	11.5

HTML5 API

☑ HTML 로컬 스토리지

- 데이터 유효기간 없음
- `setItem(key, value)` : 데이터 저장
- `getItem(key)` : 데이터 조회

```
// Store
localStorage.setItem("lastname", "Smith");
// Retrieve
document.getElementById("result").innerHTML =
    localStorage.getItem("lastname");
```

☑ HTML 세션 스토리지

- 해당 세션에서만 데이터를 유지
- 사용법은 로컬스토리지와 유사

HTML5 API

☑ HTML5 Web Workers

- 자바 스크립트를 백그라운드에서 동작시킴
- 현재 페이지의 성능에 영향을 주지 않음

PWA 를 만들기 위한 핵심 기술

☑ Web Worker 란?

- HTML 페이지 내의 스크립트가 수행될 때는 해당 페이지의 동작이 중지됨
- 백그라운드에 동작되는 자바스크립트
- 웹 워커가 동작되는 동안 원하는 작업을 수행 할 수 있음
- 버튼 클릭, 아이템 선택 등..

자바스크립트로 웹 브라우저의 HTML DOM 을 다루는 방법

HTML DOM WITH JAVASCRIPT

DOM 다루기 목차

☑ DOM 다루기

- CSS 셀렉터를 이용
- DOM 의 노드 찾기와 만들기 그리고 붙이기

☑ 이벤트 처리

- `addEventListener('event', 콜백함수)`

☑ CSS 스타일

- `.classList.add('클래스 명')`
- `add()` 대신에 `remove()`, `toggle()`, `contains()` 메소드 제공

DOM 선택하기

☑ HTML 코드

```
<div class="container" id="root"></div>
```

☑ ID 로 노드 찾기

```
document.getElementById('root');  
document.querySelector('#id');
```

DOM 선택하기

☑ HTML 코드

```
<div class="container" id="root"></div>
```

☑ 클래스 로 노드 찾기

```
document.getElementsByClassName('container');  
document.querySelector('.container');  
document.querySelectorAll('.container');
```

DOM 선택하기

☑ HTML 코드

```
<div class="container" id="root">Hello</div>
```

☑ 태그 이름으로 노드 찾기

```
document.getElementsByTagName('div');  
document.querySelector('div');  
document.querySelectorAll('div');
```

DOM 에서 데이터 가져오기

☑ HTML 코드

```
<div class="container" id="root">Hello</div>
```

☑ DOM 콘텐츠 접근/변경

```
var root = document.querySelector('#root');  
console.log(root.textContent);  
  
// 노드를 찾아서 내용을 새롭게 교체한다.  
document.querySelector('#root').textContent = '안녕하세요';
```


새로운 DOM 노드 만들기

☑ HTML 코드

```
<div class="container" id="root">Hello</div>
```

☑ 새로운 노드를 만들고 기존 노드에 붙인다.

```
// 루트 노드
var root = document.querySelector('#root');

// 새로운 노드를 만든다.
var newNode = document.createElement('h2');
newNode.textContent = '이것은 H2 엘리먼트 입니다.';

// 루트노드에 새로운 노드를 붙인다.
root.appendChild(newNode);
```

이벤트 처리

☑ HTML 코드

```
<button type="button">Contact us</button>
```

☑ 버튼에 클릭 이벤트 부여

```
const theButton = document.querySelector('...');  
  
theButton.addEventListener('click', function() {  
    alert('hello');  
});
```

노드에 스타일 동적으로 부여하기

☑ HTML 코드

```
<div class="container" id="root">Hello</div>
```

☑ HTMLElement 는 classList 라는 DOMTokenList 를 반환

```
document.querySelector('#root').classList.add('active');
```

```
// add() 이외에 remove(), toggle(), contains() 사용 가능
```

```
document.querySelector('#root').className += ' active';
```

API 와 비동기 프로그래밍

API 란?

☑ Application Programming Interface

- 고유한 기능을 가진 두 애플리케이션 간의 서비스 계약
- 요청과 응답을 사용 애플리케이션 통신 방법 정의
- API 문서에 관련 정보 포함
 - 예) 기상청 소프트웨어 시스템에는 일일 기상데이터가 있음
 - 날씨앱은 API 를 통해 이 시스템과 ‘대화’ 하여
 - 최신 날씨 정보를 표시

API 작동방식

☑ API 아키텍처는 클라이언트와 서버 측면에서 설명 가능 (요청과 응답)

- SOAP API
- RPC API
- Websocket API
- REST API

REST API 는 오늘날 웹에서 볼 수 있는 가장 많이 사용되고 유연한 API

REST API 란?

☑ Representational State Transfer

- 클라이언트가 서버 데이터를 액세스 할 때,
 - GET, POST, PUT, DELETE 등의 함수집합을 정의
 - HTTP 를 사용해서 데이터 교환
 - 주요 특징은 상태없음 (Stateless)
 - 서버의 응답은 그래픽 렌더링이 없는 일반데이터(주로 JSON 사용)

REST API 장점

☑ 통합

- 기존 시스템과 통합, 기존 코드 활용 가능

☑ 확장

- 무료 또는 유료 API 를 통해 서비스를 확장 가능
- 매쉬업, API 통합

☑ 유지관리 용이성

- API 는 시스템 간 게이트웨이 역할 수행
- API 가 영향을 받지 않도록 시스템을 내부적으로 변경

API 개발 및 생성 단계

☑ API 계획

☑ API 빌드

☑ API 테스트

- 서버 응답 검증에 초점을 둠, 시스템 공격에 대비한 보안테스트 포함

☑ API 문서화

- 도구를 사용하여 자동 또는 수동으로 생성
- 간단하고 읽기 쉬운 영어로 작성
- 정확하고 최신 상태로 유지

☑ API 마케팅

- API 를 통한 수익창출 가능

API 사용 방법

☑ API 키 확보

- API 서비스 공급자에 의해 API 키가 발급됨

☑ HTTP API 클라이언트 사용

- API 요청을 구성

☑ API 구문에 익숙해 지면 코드에서 사용

API 를 찾을 수 있는 곳

☑ RapidAPI

- 최대 규모의 글로벌 API 마켓플레이스

☑ Public APIs

- API 탐색을 쉽게 할 수 있는 40여개 범주로 그룹화

☑ 공공 API 포털 등..

- data.go.kr ..

API 요청을 위한 환경 구성

☑ API 서버 필요

- Fake server 사용 - http, restful api server, json
- <https://jsonplaceholder.typicode.com/>

{JSON} Placeholder

Free fake API for testing and prototyping.

Powered by [JSON Server](#) + [LowDB](#)

As of Dec 2020, serving ~1.8 billion requests each month.

☑ HTTP 클라이언트 툴 - HTTP 테스트 클라이언트

- Postman



비동기 메소드와 데이터 가져오기

☑ **async/await** : 비동기 코드를 동기적으로 보이게 작성하는 또 다른 방법

- 코드의 가독성을 높이고, 비동기 로직을 더 쉽게 이해하고 작성
- `async` 키워드는 함수 앞에 위치, `await` 키워드는 `async` 함수 내에서만 사용하며
- 결과 값이 반환될 때까지 함수의 실행을 일시 중단

```
async function exampleFunction() {  
  try {  
    const response = await fetch('https://api.example.com/data');  
    const data = await response.json();  
    console.log(data);  
  } catch (error) {  
    console.error("An error occurred: ", error);  
  }  
}  
  
exampleFunction();
```

자바스크립트에서 사용하는 Http client

☑️ **Axios** <https://github.com/axios/axios>

🕒 브라우저 자바스크립트와 노드에서 모두 사용가능 - universal library

```
// async ~ await 방식  
const res = await axios.get('https://jsonplaceholder.typicode.com/todos/1');  
console.log(res);
```

```
// Promise 방식  
axios.get('https://jsonplaceholder.typicode.com/todos/1')  
  .then(res => console.log(res))  
  .catch(error => {  
    console.error(error);  
  });
```

Weather App

날씨 앱

enter city name



22°C
Seoul



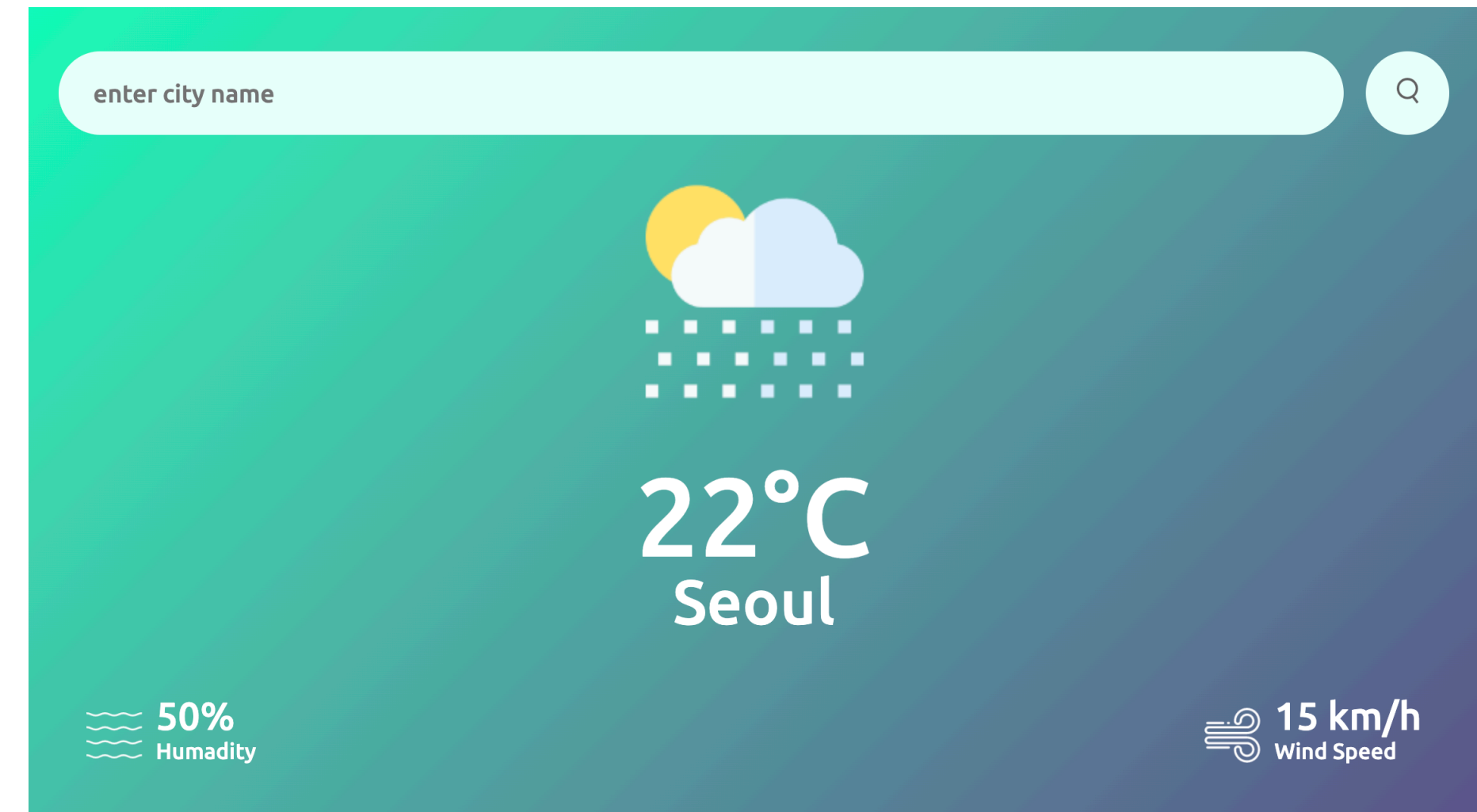
50%
Humadity



15 km/h
Wind Speed

날씨 앱 작성

- ☑ weather API 사용
- ☑ HTML/CSS 로 화면 작성



수고하셨습니다.

김순곤

soongon@huccloud.co.kr