JavaScript

1. JavaScript 시작하기

자바스크립트는 1995년 당시 넷스케이프 커뮤니케이션즈에 근무하던 브렌든 아이크(Brendan Eich)에 의해 개발되었다. 처음에는 모카(Mocha)라는 이름으로 개발되었으나, 그 후에 라이브스크립트(LiveScript)라는 이름으로 변경되었고, 당시 썬마이크로시스템즈와 협력으로 인해서 자바스크립트(JavaScript)라는 이름으로 변경되어 오늘날 까지 사용되고 있다.
자바스크립트는 주로 웹브라우저에서 실행되는 객체기반 스크립트 언어로 1995년 당시 가장 많이 사용되던 웹브라우저인 네스케이프 2.0에 탑재되었고 1996년 인터넷 익스플로러(IE) 3.0에 탑재된 이후 웹브라우저 표준 스크립트 언어로 발전하게 되었으며 현재는 거의 모든 웹브라우저에 자바스크립트 인터프리터(JavaScript Interpreter)가 내장되어 있어 자바스크립트를 지원하는 핵심 스크립트 언어로 자리매김 하였다.
자바스크립트가 썬마이크로시스템즈와의 협력으로 개발된 프로그래밍 언어이고 자바와 문법적으로 비슷한 부분이 있어서 자바와 아주 밀접한 관계가 있는 것처럼 오해(?)를 할 수 있겠지만 자바스크립트와 자바는 C 언어의 기본 구문을 바탕으로 만들어져서 문법이 약간 닮은 것과 둘 다 웹브라우저에서 실행 될 수 있다는 공통점을 가지고 있지만 두 언어는 완전히 관련이 없는 언어이다.
자바스크립트(JavaScript)는 객체(object) 기반의 스크립트 언어로 동적 타입핑을 지원하여 하나의 변수에 다양한 유형의 데이터를 저장할 수 있으며 객체지향 프로그래밍 기법과 함수형 프로그래밍 기법을 모두 사용할 수 있다. 또한 자바스크립트는 주로 웹 브라우저에서 사용되지만 Node.js와 같은 프레임워크를 이용하면 자바스크립트로 서버 프로그래밍을 구현할 수 있다.
HTML로는 웹페이지의 구조를 작성하고 CSS로는 웹페이지에 스타일을 적용해 디자인하며 자바스크립트로는 DOM(Document Object Model) 제어와 이벤트 처리 등과 같은 웹페이지에서 필요한 동작을 구현할 수 있다. 다시 말해 자바스크립트를 활용해서 HTML의 내용과 속성 등을 설정하고 변경할 수 있으며 스타일을 설정하고 변경할 수도 있다. 이외에도 웹브라우저에서 수행해야할 다양한 동작들을 자바스크립트로 구현할 수 있다.

▶ 예제 1-1 웹 문서에서 자바스크립트 사용하기

- JavaScriptStudyCh01/javascript01_01.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>웹 문서에서 자바스크립트 사용하기</title>
<style>
  /* css 주석 - 한 줄 또는 여러 줄 주석 */
</style>
<!--
  여기는 HTML 주석 - 자바스크립트 첫 번째 예제(내부 자바스크립트)
  이전에는 어떤 스크립트 언어를 사용하는지 type 속성에 명시해야 했으나
  HTML5에서는 자바스크립트가 공식 스크립트 언어이므로 type을 기술할 필요가 없다.
-->
<script type="text/javascript">

  /* 자바스크립트 여러 줄 주석
```

```
   * author : 홍길동
   * date : 2021년 01월 15일
   **/
  // 경고 창 띄우기 - 자바스크립트 한 줄 주석
  alert('안녕 하세요 첫 번째 자바스크립트 예제 입니다.');
</script>

<!--
  여기는 HTML 주석 - 아래는 외부에 있는 자바스크립트(제이쿼리)를 참조하는 예이다.
  HTML5에서는 자바스크립트가 공식 스크립트 언어이므로 type을 기술할 필요가 없다.
-->
<script src="http://code.jquery.com/jquery-3.3.1.mim.js"></script>
<!-- <script src="./js/jquery-3.3.1.mim.js"></script> -->
</head>
<body>
  <!-- HTML 주석 - 아래는 브라우저 화면에 출력되는 내용 -->
  자바스크립트 첫 번째 예제

  <!--
    내부 자바스크립트
    script 요소는 HTML 문서에서 여러 곳에 중복적으로 사용할 수 있다.
  -->
  <script>alert("body의 마지막에 기술된 스크립트 메시지...")</script>
</body>
</html>
```

▶ 예제 1-2 자바스크립트 출력 함수

- JavaScriptStudyCh01/javascript01_02.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>자바스크립트 출력 함수</title>
<script>
  // 경고 창 띄우기 - window 객체의 alert() 함수
  window.alert('Hello JavaScript!');

  // HTML 문서에 출력하기 - document 객체의 write() 함수
  window.document.write('여기는 body에 출력되는 write() 메소드')

  // 브라우저의 콘솔 창에 출력하기 - console 객체의 log() 함수
  console.log('10 + 20 : ' + (10 + 20));
</script>
```

```
</head>
<body></body>
</html>
```

2. 자료형과 연산자

자료형(Data Type)은 프로그래밍에서 사용되는 숫자 자료, 문자 자료 등과 같은 자료의 형태를 의미하며 프로그램 언어에서 가장 기본이 되는 단위가 자료형 이다. 그러므로 프로그래밍 언어를 공부할 때는 해당 언어에 대한 자료형을 충분히 이해하고 있어야 하며 아래 표는 자바스크립트에서 사용되는 자료형에 대한 설명이다.

분류	데이터형	설 명
기본형	수치형(number)	±4.94065645841246544x100-324 ~ ±1.79769313486231570 x 10308 사이의 정수 또는 실수의 숫자 데이터
	문자열형(string)	작은따옴표 또는 큰따옴표로 감싼 문자 집합
	논리형(boolean)	true(참) 또는 false(거짓) 값
	특수형 (null/undefined)	값이 정의되지 않았거나 할당되지 않은 것을 나타냄
참조형	배열(array)	데이터의 집합(각 요소에 인덱스 번호로 접근)
	객체(object)	데이터의 집합(각 요소에 이름(키)으로 접근)
	함수(function)	일련의 처리를 정의한 코드의 집합

▶ 예제 2-1 자바스크립트 변수 선언과 할당

- JavaScriptStudyCh02/javascript02_01.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>자바스크립트 변수 선언과 할당</title>
<script>
  /* 변수(Variable)는 데이터를 저장하는 메모리 공간의 이름
   * 변수는 프로그램 수행 중에 데이터를 담아두는 상자라 할 수 있다. 어떤
   * 프로그램이 실행되면 그 프로그램 안의 변수는 컴퓨터의 메인 메모리에
   * 만들어지는데 이렇게 만들어진 메모리 공간의 이름이 바로 변수인 것이다.
   * 프로그램에서는 변수를 통해 메모리에 접근하여 데이터를 저장하거나
   * 가져올 수 있고 변수는 하나의 값만 저장할 수 있으며 프로그램 수행 중에
   * 여러 가지 값으로 변경될 수 있기 때문에 변수(Variable)라고 부른다.
   *
   * 상수(Constant)도 데이터를 저장하는 메모리 공간의 이름
   * 자바스크립트에서 변수와 더불어 데이터를 저장할 수 있는 상수(Constant)라는 것도
   * 사용되는데 상수는 변하지 않고, 항상 동일한 값을 갖는 고정된 수를 의미한다.
   * 상수도 변수와 마찬가지로 데이터가 저장되는 메모리 공간의 이름이지만 프로그램이
   * 실행되는 동안 여러 가지 값으로 변경될 수 있는 변수와 달리 상수는 값이 한 번
   * 정해지면 그 값을 변경할 수 없다.
```

```
 *
 * 변수와 같이 데이터를 구분하기 위해 사용하는 이름을 식별자(Identifier)라
 * 하며 자바스크립트에서 식별자를 지정할 때 꼭 지켜야 하는 규칙이 있다.
 * 식별자는 문자, 숫자, 2개의 특수문자($, _)를 사용해 명명할 수 있으며 식별자의
 * 첫 문자는 숫자를 사용할 수 없고 반드시 문자나 2개의 특수문자로 시작해야 하며
 * 공백을 가질 수 없다. 변수도 식별자에 해당하므로 이 규칙에 따라서 이름을 지어야
 * 한다. 또한 자바스크립트에서 사용하는 예약어는 식별자로 사용할 수 없다.
 *
 * 자바스크립트는 유니코드를 지원하는 프로그래밍 언어로 여러 나라의 다양한 문자를
 * 식별자로 사용할 수 있어서 한글을 사용해 변수 명을 지정할 수도 있지만 대부분은
 * 관례적으로 영문을 사용하기 때문에 실무에서는 식별자로 한글을 사용하지 않는다.
 * 또한 대문자와 소문자를 구분하므로 num, Num, NUM은 모두 다른 식별자로
 * 구분되기 때문에 대소문자 사용에 주의를 기울여야 한다.
 *
 * 적합한 변수명 : $harp, _7up, seven11, $ession, _percent
 * 부적합한 변수명 : s#arp, 7up, 7eleven, ?uestion, &percent
 **/

/* 변수 선언 후에 초기화
 * 변수를 선언하고 그 변수에 값이 최초로 저장(할당)될 때 변수를 초기화 한다고 말한다.
 * 자바스크립트는 변수를 선언할 때 변수 앞에 let 예약어를 사용해 변수를 선언하며
 * 변수에 특정 값이 저장(할당) 되는 순간 변수의 타입(자료형)이 결정된다.
 *
 * 아래와 같이 val = 100 이라고 기술하면 정수 100을 val이라는 변수에 저장하게
 * 되는데 이런 동작을 "정수 100을 변수 val에 대입(할당) 한다."라고 말 한다.
 * 그래서 "="를 대입 연산자 또는 할당 연산자라고 부른다. 우리 생활에서 "="은
 * 같음을 의미하지만 대부분의 프로그래밍 언어에서는 대입 연산자로 사용되며
 * 우측의 데이터를 좌측의 변수에 대입하는 역할을 한다.
 **/
let val;
val = 100;

// 변수 선언과 동시에 초기화
let value = 200;

// HTML 문서에 출력 - 화면에 출력됨
document.write('val : ' + val + '<br/>');
document.write('value : ' + value + '<br/>');

/* 자바스크립트는 어떤 값이 변수에 저장(할당) 되는 순간 그 변수의 타입(자료형)이
 * 결정되는데 이렇게 변수에 값이 저장될 때 마다 변수의 자료형이 결정되는 방식을
 * 동적으로 자료의 종류가 결정되기 때문에 동적타입핑이라고 부른다.
 * 자바와 같은 프로그래밍 언어는 변수를 선언할 때 그 변수가 다룰 데이터의 타입을
 * 미리 결정해서 변수를 선언할 때 데이터 타입을 미리 지정해야 하고 이 후에 그 변수에는
 * 이미 결정한 데이터 타입만을 대입할 수 있지만 자바스크립트는 동적타입핑을 지원하기
```

```
    * 때문에 변수 하나에 여러 가지 데이터 타입을 대입할 수 있다.
    **/
    value = "자바스크립트";

    // 여러 변수를 먼저 선언한 후에 변수를 초기화
    let str, pi;
    str = "안녕하세요", pi = 3.141592;

    // HTML 문서에 출력 - 화면에 출력됨
    document.write('str : ' + str + ', pi : ' + pi + '<br/>');

    /* 상수의 선언과 사용
    * 자바스크립트에서 상수(Constant)는 아래와 같이 const 예약어를 사용해 선언한다.
    * 다음과 같이 상수를 선언하고 상수에 값이 정해지면 그 값을 변경할 수 없다.
    **/
    const num = 10;
    document.write('const num : ' + num);

    // 상수는 값을 변경할 수 없으므로 아래 코드가 실행되면 오류가 발생한다.
    //num = 100;
</script>
</head>
<body>
</body>
</html>
```

▶ 예제 2-2 변수와 자료형

- JavaScriptStudyCh02/javascript02_02.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>변수와 자료형</title>
<script>

    /* 리터럴(Literal)
    * 프로그램에서 리터럴(Literal)이란 변수에 저장되는 값 자체를 의미하거나
    * 어떤 자료형에 대한 값을 표현하는 방법을 말한다. 다시 말해서 숫자 153이라는
    * 값을 프로그램 코드에서 표현한다면 아래의 숫자 리터럴 코드와 같이 표현하는데
    * 이 코드에서 153 자체를 숫자 리터럴이라고 부른다. 이외에도 자바스크립트에서
    * 사용하는 다양한 데이터를 표현하기 위해서 아래 코드들과 같이 문자열 리터럴,
    * 논리 리터럴, 함수 리터럴, 배열 리터럴, 객체 리터럴 등이 있다.
    **/
```

```
// 숫자 리터럴 - 정수 리터럴
let num = 153;

// 문자열 리터럴
let str = '문자열 데이터 저장';

// 논리 리터럴
let booleanVal = false;

// 함수 리터럴
let func = function() {
   document.write('자료형에 따른 변수에 값 할당하기<br/>');
};

/* 배열 리터럴 - 배열은 대괄호([ ... ])로 감싼 형태로 표현한다.
 * 자바스크립트에서 배열에 아래와 같이 여러 데이터 타입을 저장할 수 있다.
 **/
let arr = ['배열데이터 1', 123, true];

/* 객체 리터럴
 * 배열안의 하나의 데이터를 요소라고 하지만 객체는 프로퍼티라고 부른다.
 * 배열은 인덱스를 이용해 요소에 접근하지만 객체는 프로퍼티의 이름을 통해
 * 접근할 수 있다. 이렇게 이름을 키로 접근할 수 있는 배열을 연관배열이라 한다.
 * 아래와 같이 객체의 프로퍼티에 함수가 지정되면 이를 메소드(Method)라고 부른다.
 **/
let obj = {
     x: 1,
     y: 2,
     name: '홍길동',
     func: function() { document.write("나는 obj 메소드 + <br/>"); }
};

document.write("arr : " + arr + '<br/>');
func();
document.write('obj.x : ' + obj.x + ', obj.y : ' + obj.y
     + ', obj["name"] : ' + obj['name'] + '<br/>');

// obj 객체의 메소드 호출
obj.func();

// 변수를 선언하고 초기화 하지 않음 - 변수에 값을 할당 하지 않음
let data;

/* 아래 코드의 실행 결과는 모두 undefined(미정의 값)이 출력됨
```

```
   * undefined(미정의 값)는 변수의 값이 정의되어 있지 않음을 의미하며
   * 변수가 초기화 되지 않은 경우와 객체에서 아직 초기화 되지 않은 프로퍼티를
   * 사용할 경우 undefined가 반환된다.
   **/
  document.write("data : " + data + ", obj.age : " + obj.age);
</script>
</head>
<body>
</body>
</html>
```

▶ 예제 2-3 문자열 따옴표 처리와 템플릿 문자열

- JavaScriptStudyCh02/javascript02_03.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>문자열 따옴표 처리와 템플릿 문자열</title>
<script>
  /* 자바스크립트에서 문자열은 홑 따옴표(' ') 또는 쌍 따옴표(" ")를 사용해 표현한다.
   * 쌍 따옴표가 문자열에 포함되어 있다면 홑 따옴표로 감싸서 문자열을 표현하면 되고
   * 한 가지 따옴표만 사용하려면 역 슬래쉬(\)를 사용해 문자열을 표현하면 된다.
   * 홑 따옴표(' ') 또는 쌍 따옴표(" ")는 문자열을 표현하는데 사용되는 특정 의미를
   * 지닌 문자로 메타 문자(meta character)라고 하며 이렇게 프로그램 코드에서
   * 특정 의미를 지닌 문자가 일반 문자처럼 그 문자 자체로 사용할 될 수 있도록 지원하는
   * 역 슬래쉬(\)와 같은 문자를 이스케이프(escape) 문자라고 부른다.
   **/
  document.write('그는 나에게 "조심해서 잘 가~"라고  말했다.<br>');
  document.write("그는 나에게 \"조심해서 잘 가~\"라고 말했다.<br>");
  document.write("You're My Sunshine<br>");

  /* 다음과 같이 문자열에 홑 따옴표(' ') 또는 쌍 따옴표(" ")가 여러 번 나오는 경우
   * 역 슬래쉬(\)를 사용해 작성하는 것도 번거롭고 가독성도 좋지 못하게 되는데 이 때
   * 백틱(`)을 사용하면 문자열 작성도 간편하고 가독성도 높게 작성할 수 있어 편리하다.
   * 참고로 백틱(`)은 키보드에서 ESC 키 바로 아래에 위치한 키이다.
   **/
  document.write(`영화 "기생충"은 한국 영화 역사상 '아카데미상'을 받은 최초의 영화이다.<br>`);

  /* 문자열과 변수를 연결해 문자열로 출력하는 경우 플러스(+) 연산자를 사용하게 되면
   * 아래와 같이 출력 코드가 길어지고 가독성도 떨어지게 되는데 이 때 템플릿 문자열
   * (Template strings)을 사용하면 가독성도 뛰어나고 편리하게 문자열을 만들 수 있다.
   * 템플릿 문자열은 백틱으로 출력할 문자열을 감싸고 그 안에서 ${변수}와 같이 사용한다.
   **/
```

```
  let name = "홍길동";
  let year = 2023
  let month = 7
  let day = 28
  let hour = 10
  let minute = 30
  document.write(name + "님의 예약 시간은 " + year + "년 " + month + "월 "
      + day + "일 " + hour + "시 " + minute + "분 입니다.<br>");
  document.write(`${name}님의 예약 시간은 ${year}년 ${month}월
      ${day}일 ${hour}시 ${minute}분 입니다.`);
</script>
</head>
<body></body>
</html>
```

▶ 예제 2-4 산술 연산자

- JavaScriptStudyCh02/javascript02_04.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>산술 연산자</title>
<script>
  //변수 a, b를 선언하고 각각 7과 3으로 초기화
  let a = 7, b = 3;

  /* 사칙 연산자
   * 더하기 : + , 빼기 : -, 곱하기 : *, 나누기 : /, 나머지 : %, 거듭제곱 : **
   **/
  document.write("7 * 3 = " + a * b  + ", 7의 3승 = " + a ** 3 + "<br>");
  document.write("7 / 3 = " + a / b + ", 7 % 3 = " + a % b + "<br>");

  /* 증감 연산자
   * 증감 연산자는 연산 대상이 되는 피연산자가 하나인 단항 연산자로 피연산자의 값을
   * 1 증가 또는 1 감소하는 연산자이다. 증감 연산자는 아래와 같이 피연산자의 앞에
   * 지정할 수도 있고 뒤에 지정할 수도 있다. 앞에 지정하는 경우를 선 증감 연산자라고
   * 하며 뒤에 지정하는 경우를 후 증감 연산자라고 한다. 선 증감 연산자는 다음 연산을
   * 수행하기 전에 피연산자의 값을 먼저 1 증감 시키고 다음 연산을 수행하며 후 증감
   * 연산자는 다음 연산을 수행한 후에 피연산자의 값을 1 증감 시킨다.
   *
   * 아래 코드에서 a++은 앞에 있는 문자열 데이터 "a++ = "과 더하기(+) 연산을
   * 수행한 후에 a의 값을 1 증가시키고 ++b는 먼저 b의 값을 1 증가 시킨 후에 앞에
   * 있는 문자열 데이터 "++b = "과 더하기(+) 연산을 수행한다.
```

```
    **/
    document.write("a++ = " + a++ + ", a : " + a + "<br>");
    document.write("++b = " + ++b + ", b : " + b + "<br>");

    /* 연결 연산자
     * 다음과 같이 문자열과 문자열의 더하기(+) 연산은 앞뒤의 문자열을 연결해 주는 연산을
     * 수행하며 문자열과 숫자의 더하기(+) 연산도 문자열을 우선하여 숫자를 문자열로 변환한
     * 후에 앞뒤의 문자열을 연결하는 연산을 수행한다.
     **/
     document.write("Hello + JavaScript : " + "Hello " + "JavaScript" + "<br>");
     document.write("20 + line : " + 20 + "line" + "<br>");
</script>
</head>
<body>
</body>
</html>
```

▶ 예제 2-5 비교 연산자와 논리 연산자

- JavaScriptStudyCh02/javascript02_05.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>비교 연산자와 논리 연산자</title>
<script>
    /* 비교 연산자
     * 비교 연산자는 좌변과 우변의 값을 비교하여 그 결과를 논리 값으로 반환하는 연산자이다.
     * 비교 연산자는 좌변을 기준으로 좌변의 값이 우변의 값 보다 큰지(>), 크거나 같은지(>=)
     * 작은지(<), 작거나 같은지(<=), 같은지(==), 같지 않은지(!=) 등등을 비교할 수 있다.
     **/
    document.write("10 > 30 = " + (10 > 30) + "<br>");
    document.write("10 <= 30 = " + (10 <= 30) + "<br>");
    document.write("10 != 30 = " + (10 != 30) + "<br><br>");

    // 문자열을 비교 연산자로 비교하면 사전적인 위치에 따라서 크고 작음을 비교한다.
    document.write("안드로이드 > 자바 : " + ("안드로이드" > "자바") + "<br><br>");

     /* 논리 연산자
      * 논리 연산자는 논리 값을 비교하여 그 결과를 논리 값으로 반환하는 연산자로 연산
      * 결과를 논리 값으로 반환하는 비교 연산자와 같이 많이 사용된다. 논리 연산자의 종류는
      * &&(AND), ||(OR), !(NOT) 3가지 종류가 있으며 && 연산자는 좌변과 우변을 비교해
      * 모두 참(true)이면 참(true)을 그렇지 않으면 거짓(false)을 반환하며 || 연산자는
      * 좌변과 우변을 비교해 하나만 참(true)이면 참(true)을 반환하는 연산자이다. 또한
```

```
    * ! 연산자는 연산 대상이 되는 피연산자가 하나인 단항 연산자로 피연산자의 논리 값이
    * 참(true)이면 거짓(false)을 거짓(false)이면 참(true)으로 반전하는 연산자이다.
    **/
  // 다음과 같이 비교 연산자와 논리 연산자를 조합해 사용하는 경우가 많다.
  document.write("7 > 5 && 3 > 1 : " + (7 > 5 && 3 > 1) + "<br>");
  document.write("3 > 1 || 7 < 5 : " + (3 > 1 || 7 < 5) + "<br><br>");

  // 부정 연산자는 피연산자의 논리 값을 반전하는 역할을 한다.
  document.write("!true : " + !true + "<br><br>");

  /* 문자열과 수치형 데이터의 연산에서 플러스(+) 연산은 문자열을 우선하여
   * 문자열 연결 연산을 수행하고 비교 연산이나 나머지 사칙연산은 문자열을
   * 수치형으로 변환 후 연산을 수행한다. 등가연산(==)은 변수의 값이 같은지만
   * 비교하고 동치연산(===)은 변수의 자료형과 값이 같은지를 비교한다.
   **/
  document.write('15 == "15" : ' + (15 == "15") + "<br>");
  document.write('15 === "15" : ' + (15 === "15") + "<br><br>");

  // 문자열이 0으로 시작하면 진수 표현으로 인식한다.
  document.write('"0xF" == 15 : ' + ("0xF" == 15) + "<br>");
  document.write('"0xF" === 15 : ' + ("0xF" === 15) + "<br><br>");

  // 논리 값을 비교 연산이나 플러스(+) 연산을 하면 true=1, false=0으로 변환되어 계산된다.
  document.write("true > false : " + (true > false) + "<br>");
  document.write("false + 0 = " + (false + 0) + "<br><br>");
</script>
</head>
<body>
</body>
</html>
```

▶ 예제 2-6 복합 대입연산자

- JavaScriptStudyCh02/javascript02_06.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>복합 대입연산자</title>
<script>
  // 다양한 방법으로 변수를 선언하고 초기화
  let num1 = 10, num2 = 20, num3 = 30, str;

  /* 아래의 식에서 사용한 "+=", "/=", "%=" 연산자 들은 모두 먼저 더하기(+),
```

```
    * 나누기(/), 나머지(%) 연산을 수행한 후에 좌측의 변수에 대입하는 두 가지
    * 동작(연산)을 수행하는 연산자로 이런 유형의 대입 연산자를 복합 대입연산자라고
    * 부른다. 아래에서 num1 += 30;의 동작은 먼저 num1 + 30의 연산을 수행한
    * 후에 그 결과를 다시 num1에 대입하게 된다.
    **/
    num1 += 30;
    num2 /= 5;
    num3 %= 7;

    // 문자열과 수치 데이터의 + 연산은 문자열 연결 연산이 된다.
    document.write('num1 : ' + num1 + ', num2 : '
            + num2 + ', num3 : ' + num3);
    document.write('<br/><br/>');

    str = '<ul>';
    str += ' <li>자바스크립트 완벽가이드</li>';
    str += ' <li>JavaScript Cookbook</li>';
    str += ' <li>자바스크립트 마스터북</li>';
    str += '</ul>';

    document.write(str);
</script>
</head>
<body>
</body>
</html>
```

▶ 예제 2-7 형 변환과 typeof 연산자

- JavaScriptStudyCh02/javascript02_07.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>형 변환과 typeof 연산자</title>
<script>
let num10 = 10;
let strNum10 = '10'
let boolTrue = true;

/* 문자열 데이터와 다른 형 데이터의 + 연산은 문자열 연결 연산을 한다.
 * 문자열을 + 연산하면 문자열 연결 연산이 되는데 이때 문자열과 + 연산을 하는
 * 다른 형의 데이터는 자동으로 문자열로 변환되어 문자열 연결 연산을 한다.
 * 다음과 같이 문자열 + 숫자, 문자열 + 불린 데이터를 연산을 하면 문자열이
```

```
 * 우선되어 숫자나 불린 데이터는 자동으로 문자열 데이터로 변환 된 후 문자열
 * 연결 연산이 이루어진다.
 **/
document.write("strNum10 + num10 = " + (strNum10 + num10) + "<br>");
document.write("strNum10 + boolTrue = " + (strNum10 + boolTrue) + "<br>");

/* 만약 문자열이나 불린 데이터를 + 연산 이외의 사칙연산을 시도하면 숫자가
 * 우선되어 문자열이나 불린 데이터를 숫자로 변환 후에 사칙연산이 이루어진다.
 * 이렇게 자동으로 데이터 형이 변환되는 것을 묵시적(implicit) 형 변환이라고 한다.
 **/
document.write("strNum10 - num10 = " + (strNum10 - num10) + "<br>");
document.write("strNum10 - boolTrue = " + (strNum10 - boolTrue) + "<br>");

/* Number() 함수는 아래와 같이 괄호 안에 지정한 데이터를 숫자로 변환해 준다.
 * 이렇게 데이터 형을 강제로 변환하는 것을 명시적(Explicit) 형 변환이라고 한다.
 **/
document.write(`Number(true) : ${Number(true)}, Number("3") : ${Number("3")}<br>`);
document.write(`Number("") : ${Number("")}, Number('034.56') : ${Number('034.56')}<br>`);

// window 객체의 parseInt(), parseFloat() 함수는 문자열을 숫자로 변환해 주는 함수이다.
document.write(`parseInt("") : ${parseInt("")}, parseInt("034.56") : ${parseInt("034.56")}<br>`);
document.write(`parseFloat("") : ${parseFloat("")}, parseFloat("034.56") :
${parseFloat("034.56")}<br>`);
document.write(`parseInt("12A") : ${parseInt("12A")}, parseFloat("123.4A5") :
${parseFloat("123.4A5")}<br>`);

/* toString() 함수는 null과 undefined 자료형을 제외한 나머지 모든 자료형에서
 * 사용할 수 있는 함수로 괄호에 지정한 데이터를 문자열로 변환해 주는 함수이다.
 * String() 함수는 괄호에 지정한 데이터를 문자열로 변환해 주는 함수이다.
 **/
document.write(`String(20) + num10.toString() : ${String(20) + num10.toString()}<br>`);
document.write(`String(null) + String(undefined) : ${String(null) + String(undefined)}<br>`);

// Boolean() 함수는 괄호에 지정한 데이터를 불린 데이터로 변환해 주는 함수이다.
document.write(`Boolean(1) : ${Boolean(1)}, Boolean(0) : ${Boolean(0)}<br>`);
document.write(`Boolean("a") : ${Boolean("a")}, Boolean("") : ${Boolean("")}<br>`);

function func() { }
let arr = []
let obj = {}

/* typeof 연산자는 지정한 변수나 리터럴의 데이터 형의 이름을 소문자로 반환한다.
 * 숫자형은 number, 논리형은 boolean, 문자열은 string, 함수형은 function,
 * 객체 자료형은 object가 반환된다.
 **/
```

```
document.write(`typeof num10 : ${typeof num10}, typeof true : ${typeof true}<br>`);
document.write(`typeof "a" : ${typeof "a"}, typeof func : ${typeof func}<br>`);
document.write(`typeof arr : ${typeof arr}, typeof obj : ${typeof obj}<br>`);
document.write(`typeof null : ${typeof null}, typeof undefined : ${typeof undefined}<br>`);
</script>
</head>
<body>
</body>
</html>
```

3. 조건문

▶ 예제 3-1 if 조건문

- JavaScriptStudyCh03/javascript03_01.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>if 조건문</title>
<script>
  let x = 10, score = 79;

  document.write("배수 구하기<br>");
  if(9 % 3 == 0) {
    document.write("9 는 3의 배수 입니다.<br><br>");
  }

  document.write("홀/짝수 구하기<br>");
  if(x % 2 == 0) {
    document.write(x + "는 짝수<br><br>");
  } else {
    document.write(x + "는 홀수<br><br>");
  }

  document.write("합격 여부<br>");
  if(score > 80) {
    document.write("축하 합니다. 합격 입니다.<br><br>");
  } else {
    document.write("아쉽네요 다음 기회를...<br><br>");
  }
</script>
</head>
<body>
</body>
</html>
```

▶ 예제 3-2 if else 조건문

- JavaScriptStudyCh03/javascript03_02.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
```

```
<title>if else 조건문</title>
<script>
  function closeWindow() {

    /* Confirm 대화상자는 사용자가 창을 강제로 닫거나 취소 버튼을
     * 클릭하면 false를 반환하고 확인 버튼을 클릭하면 true를 반환한다.
     * confirm() 함수는 window 객체에 정의되어 있는 함수이다.
     **/
    let input = confirm('프로그램을 종료 하시겠습니까?')

    /* if 문은 조건식이 참일 때 실행되는 조건문 이다.
     * if(조건식) { 조건식이 참일 때 실행 }
     * if(조건식) { 조건식이 참일 때 실행 } else { 조건식이 거짓일 때 실행 }
     **/
    if(input) {
      // 현재 브라우저 창을 닫는다.
      window.close();
      //this.close();

    } else {
      alert('그럴 줄 알았어요..^_^');
    }
  }
</script>
</head>
<body>
  <!--
    버튼에 onclick 속성에 호출할 자바스크립트 함수를 지정하여 버튼이 클릭되었을 때
    필요한 기능을 동작시킨다. 아래와 같이 요소의 이벤트 속성에 이벤트 처리함수를
    등록하여 이벤트를 처리하는 방식을 인라인 이벤트 모델이라고 한다.
  -->
  <input type="button" value="현재창 닫기" onclick="closeWindow()"/>
</body>
</html>
```

▶ 예제 3-3 if else if 조건문

- JavaScriptStudyCh03/javascript03_03.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>if else if 조건문 - 배수 구하기</title>
<script>
```

```
  let num = 14;

  /* 사용자가 입력한 값이 3의 배수인지 확인
   * 어떤 수를 3으로 나눠서 나머지가 0이면 그 수는 3의 배수이다.
   **/
  if(num % 3 == 0) {
     document.write(num + "은 3의 배수 입니다.");

  /* 사용자가 입력한 값이 7의 배수인지 확인
   * 어떤 수를 7로 나눠서 나머지가 0이면 그 수는 7의 배수이다.
   **/
  } else if(num % 7 == 0) {
     document.write(num + "은 7의 배수 입니다.");

  } else {
     document.write(num + "은 3과 7의 배수 아닙니다.");
  }
</script>
</head>
<body>
</body>
</html>
```

▶ 예제 3-4 if else if 조건문

- JavaScriptStudyCh03/javascript03_04.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>if else if 조건문 - 숫자를 입력 받아서 배수 구하기</title>
<script>
  function inputMultiple() {

     /* html 문서에서 name 속성의 값이 form1인 폼 안에서 name 속성의
      * 값이 inputNum인 input 요소를 찾아서 이 입력상자에 입력된 값을 읽어온다.
      * 폼 안의 컨트롤들은 value 속성을 이용해 값을 설정하거나 읽어올 수 있다.
      **/
     let inputNum = document.form1.inputNum.value;

     /* html 문서에서 document 객체의 querySelector() 함수를 이용해서
      * 결과를 출력할 id가 result인 문서 객체를 구한다.
      **/
     let result = document.querySelector("#result");
```

```
    /* 사용자 입력한 값이 숫자가 아닌지 확인
     * isNaN() 함수는 인수의 값이 숫자가 아니면 true 숫자면 false를 반환한다.
     **/
    if(isNaN(inputNum)) {
      alert('3과 7의 배수를 정수로 입력해주세요')

    /* 사용자가 입력한 값이 3의 배수인지 확인
     * 어떤 수를 3으로 나눠서 나머지가 0이면 그 수는 3의 배수이다.
     **/
    } else if(inputNum % 3 == 0) {

      /* 결과를 출력할 div 요소의 innerHTML 속성을 이용해 strong 요소를
       * 포함하여 결과를 출력한다.
       **/
      result.innerHTML =
        "<strong>입력 값 : " + inputNum + "은 3의 배수 입니다.</strong>";

    /* 사용자가 입력한 값이 7의 배수인지 확인
     * 어떤 수를 7로 나눠서 나머지가 0이면 그 수는 7의 배수이다.
     **/
    } else if(inputNum % 7 == 0) {
      result.innerHTML =
        "<strong>입력 값 : " + inputNum + "은 7의 배수 입니다.</strong>";

    } else {
      result.innerHTML =
        "<strong>입력 값 : " + inputNum + "은 3과 7의 배수가 아닙니다.</strong>";
    }

    /* 다음 입력을 위해서 이전에 입력한 input 요소의 값을 지운다.
     * html 문서에서 name 속성의 값이 form1인 폼 안에서 name 속성의
     * 값이 inputNum인 input 요소를 찾아서 이 입력상자에 입력된 값을 지운다.
     * 폼 안의 컨트롤들은 value 속성을 이용해 값을 설정하거나 읽어올 수 있다.
     **/
     document.form1.inputNum.value = "";
  }
</script>
</head>
<body>
  <form name="form1">
    3의 배수 또는 7의 배수를 입력해 주세요 : <br/><br/>
    <input type="text" name="inputNum"/>
    <input type="button" value="배수 입력" onclick="inputMultiple()"/>
  </form>
```

```
  <div id="result">
  </div>
</body>
</html>
```

▶ [연습문제 3-1] 조건문을 이용한 학점 출력

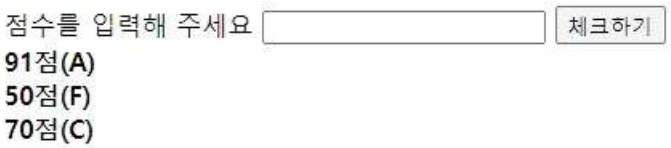
다음 그림과 같이 폼을 만들고 사용자로부터 점수를 입력받아 HTML 문서의 DIV 박스에 점수와 학점을 출력하는 프로그램을 작성해보자.

```
<body>
  <form name="inputForm">
    점수를 입력해 주세요
    <input type="text" name="score" id="score"/>
    <input type="button" value="체크하기" onclick="choose()"/>
  </form>
  <div id="result"></div>
</body>
```

1. 조건문을 사용하여 입력된 점수에 따라서 각각 아래와 같이 출력되게 작성하시오.
   90 ~ 100점 사이 : 입력된 점수(A)
   80 ~ 89점 사이 : 입력된 점수(B)
   70 ~ 79점 사이 : 입력된 점수(C)
   60 ~ 69점 사이 : 입력된 점수(D)
   이외 점수 : 입력된 점수(F)

2. 입력된 데이터가 숫자가 아닌 경우 적절한 경고 창을 띄우고 데이터는 출력되지 않게 하시오.

3. 입력된 점수를 학점으로 출력한 기존 데이터에 줄을 바꿔가며 다음과 같이 출력되게 하시오.



▶ 예제 3-5 switch 조건문

- JavaScriptStudyCh03/javascript03_05.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>switch 조건문</title>
<script>
  function choose() {

    // 문서에서 id가 result인 문서 객체를 선택한다.
    let result = document.getElementById('result');

    // 문서에서 이름이 inputForm 하위의 score에 입력된 값을 읽어온다.
    let inputNum = document.inputForm.score.value;

    // 입력된 학점을 대문자로 변환해 조건을 체크한다.
    switch(inputNum.toUpperCase()) {

    /* switch 문의 () 안의 값이 A 이면 아래의 case 'A' 부분이 실행되고 switch
     * 문이 종료된다. 이렇게 switch 문의 () 안의 값에 따라서 해당하는 case 문 안의
     * 코드가 실행되고 해당하는 부분이 없을 때는 default 문의 코드가 실행된다.
     * default가 필요하지 않으면 생략할 수 있다.
     **/
    case 'A' :
      // id가 result인 문서 객체의 하위에 html 태그를 포함해 데이터를 추가한다.
      result.innerHTML = '90점 이상<br/>';
      break;
    case 'B' :
      result.innerHTML = '80 ~ 89점<br/>';
      break;
    case 'C' :
      result.innerHTML = '70 ~ 79점<br/>';
      break;
    case 'D' :
      result.innerHTML = '60 ~ 69점<br/>';
      break;
    default :
      result.innerHTML = '60점 미만<br/>';
    }
  inputForm.score.value = '';
  }
</script>
</head>
<body>
```

```
  <form name="inputForm">
    학점을 입력해 주세요(예 : A, B, C)<br/>
    <input type="text" name="score" id="score"/>
    <input type="button" value="체크하기" onclick="choose()"/>
  </form>
  <div id="result"></div>
</body>
</html>
```

▶ 예제 3-6 삼 항 연산자

- JavaScriptStudyCh03/javascript03_06.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>삼 항 연산자</title>
<script>
  /* Prompt 대화상자는 사용자로부터 데이터를 입력 받을 수 있는 대화상자로
   * 사용자가 Prompt 대화상자의 창을 강제로 닫거나 취소 버튼을 클릭하면
   * null을 반환하고 확인 버튼을 클릭하면 대화상자에 입력된 문자열을 반환한다.
   * 참고로 prompt() 함수는 window 객체에 정의되어 있는 함수이다.
   **/
  let input = prompt("숫자를 입력해주세요", 0);

  /* 삼 항 연산자는 조건 연산자라고도 하면 피연산자가 3개라 삼 항 연산자라고 부른다.
   * 조건 ? 조건이 참일 때 반환 값 : 조건이 거짓일 때 반환 값
   **/
  let result = input % 2 == 0 ? "짝수" : "홀수";
  console.log(input + "은 " + result);

  // 삼 항 연산자는 다음과 같이 여러 번 중첩하여 사용할 수 있다.
  result = input > 0 ? "양수임" :
           input == 0 ? "0임" :
             input < 0 ? "음수임" : "값이 입력되지 않음"
  console.log(input + "은 " + result);
</script>
</head>
<body></body>
</html>
```

▶ [연습문제 3-2] 중첩 if문을 이용한 주사위 게임

다음 그림과 같이 폼을 만들고 사용자로부터 숫자를 입력받아 아래 설명과 같이 동작하는 주사위

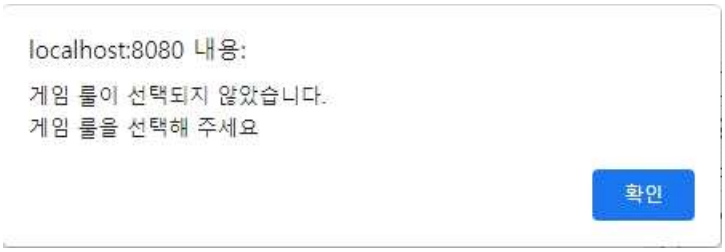
게임을 만들어 보자.

```
<body>
  <form name="inputForm">
    게임 룰 선택 : <label><input type="radio" name="rule" value="1"/>높은 수 승</label>
      <label><input type="radio" name="rule" value="0"/>낮은 수 승</label>
    <input type="button" value="게임시작" onclick="diceGame()"/>
  </form>
</body>
```

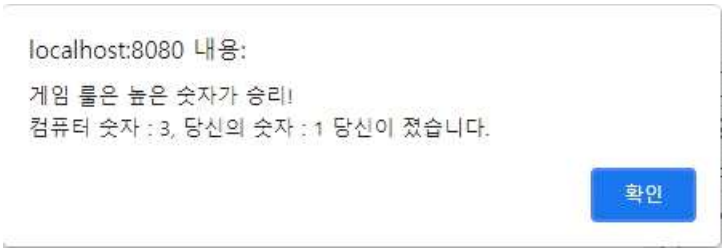
게임 룰 선택 : ○높은 수 승 ○낮은 수 승 게임시작

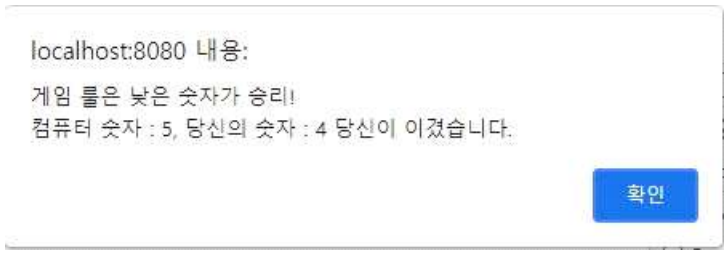
1. 컴퓨터와 사용자 주사위는 다음 코드를 사용해 1 ~ 6까지의 난수를 발생 하시오.
   let num = parseInt(Math.random() * 6) + 1;

2. 사용자가 게임 룰을 선택하지 않고 “게임시작” 버튼을 클릭하였다면 다음과 같은 경고 창을 띄워서 안내하시오.



3. 사용자로부터 게임 룰이 제대로 선택되고 “게임시작” 버튼이 클릭되면 라디오 버튼에 입력된 게임 룰을 읽어와 현재 게임 룰과, 컴퓨터 숫자, 사용자 숫자 그리고 게임의 결과를 아래와 같이 알림 창으로 출력하시오.




localhost:8080 내용:
게임 룰은 낮은 숫자가 승리!
컴퓨터 숫자 : 5, 당신의 숫자 : 4 당신이 이겼습니다.
확인

4. 반복문

▶ 예제 4-1 for 문

- JavaScriptStudyCh04/javascript04_01.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>for 문</title>
<script>
  // for 문을 이용해 1 ~ 100까지 합 구하기
  let sum = 0;

  // for(초기식; 조건식; 증감식) 문을 이용해 1 ~ 100까지 합 구하기
  for(let i = 1; i <= 100; i++) {
    sum += i;
  }
  document.write("<h2>1 ~ 100까지 합 : " + sum + "</h2>");

  document.write("<h2>for 문을 이용해 구구단 7단 출력</h2>");
  let x = 7;
  for(let i = 1; i < 10; i++) {
    document.write(x + " x " + i + " = " +  x * i + "<br>");
  }
</script>
</head>
<body></body>
</html>
```

▶ 예제 4-2 for 문에서 if 문 사용하기

- JavaScriptStudyCh04/javascript04_02.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>for 문에서 if 문 사용하기</title>
<script>
  document.write("<h2>1 ~ 100까지 짝수와 홀수의 합 구하기ㅏ</h2>");
  let oddSum = 0;
  let evenSum = 0;

  // 반복문을 돌면서 if 문을 이용해 짝수의 합과 홀수의 합을 구한다.
```

```
  for(let i = 1; i <= 100; i++) {

    /* 반복문 안에서 특정 조건에 해당하는지 판단하기 위해서 if 문 사용
     * 현재 i를 2로 나누어 나머지가 0이면 짝수 그렇지 않으면 홀수로 판단
     **/
    if(i % 2 == 0) {
      evenSum += i;

    } else {
      oddSum += i;
    }
  }
  document.write("1 ~ 100까지 짝수의 합 : " + evenSum + "<br>");
  document.write("1 ~ 100까지 홀수의 합 : " + oddSum + "<br>");

  document.write("<h2>1 ~ 100까지 3의 배수와 개수 그리고 합계 구하기</h2>");
  let num = 3;
  let count = 0;
  let sum = 0;
  let nums = "";

  for(let i = 1; i <= 100; i++) {

    /* 반복문 안에서 if 문을 이용해 3의 배수인지 판단
     * 어떤 수를 3으로 나눠서 그 나머지가 0이면 3의 배수
     **/
    if(i % num == 0) {

      // 3의 배수를 콤마(,)로 구분해 문자열로 저장
      nums += (100 - num >= i) ? i + ", " : i;

      // 3의 배수를 sum에 누적하여 더함
      sum += i;

      // 3의 배수의 count 변수 값을 하나씩 증가
      count++; // count = count + 1;
    }
  }
  document.write("1 ~ 100까지 3의 배수 :<br>" + nums + "<br>");
  document.write("3의 배수의 개수 : " + count + ", 합계 : " + sum + "<br>");
</script>
</head>
<body></body>
</html>
```

▶ 예제 4-3 for 문과 if문을 이용한 약수 구하기

- JavaScriptStudyCh04/javascript04_03.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>for문과 if문을 이용한 약수 구하기</title>
</head>
<body>
  <h1>for문과 if문을 이용한 약수 구하기</h1>
  <script>
    /* 약수란 어떤 정수 n을 나누어떨어지게 하는 정수를 말한다.
     * 만약 정수 n의 약수가 i라면 n을 i로 나누었을 때 나머지가 0이 된다.
     * 다음은 정수 120의 약수와 그 개수를 구하여 출력하는 프로그램 이다.
     **/
    const N = 120;
    let cnt = 0;
    let divisor = "";

    for(let i = 1; i<= 120; i++) {
      if(N % i == 0) {
        cnt++;
        divisor += i + (i < 120 ? ", " : "");
      }
    }
    document.write(`${N}의 약수는 ${divisor}이며 모두 ${cnt}개 입니다.`);
  </script>
</body>
</html>
```

▶ [연습문제 4-1] 홀수의 개수와 짝수의 개수 구하기

정수 75부터 ~ 355까지 홀수와 짝수의 개수를 구하여 아래와 같이 출력하는 프로그램을 작성하시오.

[실행결과]
75부터 ~ 355까지 홀수의 개수 : 141
75부터 ~ 355까지 짝수의 개수 : 140

▶ [연습문제 4-2] 3의 배수이면서 6의 배수가 아닌 정수와 그 배수의 개수

아래와 같이 57부터 ~ 119까지 3의 배수이면서 6의 배수가 아닌 정수와 그 배수의 개수 그리고 그 배수의 합계를 출력하는 프로그램을 작성하시오.

[실행결과]

57부터 ~ 119까지 3의 배수이면서 6의 배수가 아닌 정수 구하기

3의 배수이면서 6의 배수가 아닌 수 : 57, 63, 69, 75, 81, 87, 93, 99, 105, 111, 117

3의 배수이면서 6의 배수가 아닌 정수의 개수 : 11

3의 배수이면서 6의 배수가 아닌 정수의 합 : 957

▶ 예제 4-4 중첩 for 문을 이용한 구구단 출력

- JavaScriptStudyCh04/javascript04_04.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>중첩 for 문을 이용한 구구단 출력</title>
<style>
  #gugudan {
    white-space: pre;
    padding: 15px;
  }
</style>
</head>
<body>
  <h1>구구단</h1>
  <div id="gugudan"></div>
  <script>
    let gugudan = document.querySelector("#gugudan");

    for(let i = 1; i <= 9; i++) {
      for(let j = 2; j <= 9; j++) {
        gugudan.innerHTML += j + " x " + i + " = " + i * j + "\t";
      }
      gugudan.innerHTML += "<br>";
    }
  </script>
</body>
```

▶ 예제 4-5 while 문과 do while 문

- JavaScriptStudyCh04/javascript04_05.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>while 반복문과 do while 반복문</title>
<script>
  let i = 0;

  // sum을 초기화 하지 않으면 NaN(Not a Number)
  let sum = 0;

  /* while 문은 조건식이 거짓일 때 단 한 번도 실행되지 않는다.
   * while(조건식) { 조건이 참인 동안 실행 }
   */
  while(i < 100) {

    /* while 문은 for 문과 같이 증감식 같은 것이 없으므로 while 문 안에서
     * 조건식을 거짓으로 만들 수 있는 코드를 작성하지 않으면 무한 반복문이 된다.
     **/
    i++;
    if(i % 2 == 0) {
      sum += i;
    }
  }
  document.write('while문의 1 ~ 100까지 짝수 합 : ' + sum + "<br/>");

  i = 0;
  sum = 0;

  /* do~while 문은 최소 한 번은 실행을 한다.
   * do { 먼저 실행된 후에 조건식이 참인지 확인  } while(조건식);
   **/
  do {

    /* do~while 문도 while 문과 마찬가지로 do~while 문의 실행 문장 안에서
     * 조건식을 거짓으로 만들 수 있는 코드를 작성하지 않으면 무한 반복문이 된다.
     **/
    i++;
    if(i % 2 == 1) {
      sum += i;
    }
  } while(i < 100);
  document.write('do while문의 1 ~ 100까지 홀수의 합 : ' + sum);
</script>
</head>
```

```
<body></body>
</html>
```

▶ 예제 4-6 break와 continue

- JavaScriptStudyCh04/javascript04_06.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>break와 continue</title>
<script type="text/javascript">
  let i = 0;
  while(true) {
    if(!confirm('계속 하시겠습니까? - while')) { // 사용자가 취소를 선택하였으면
      alert('while 문 ' + i + '번 실행 후 종료!');

      // 현재 실행 중인 반복문을 빠져 나간다.
      break;

    } else {  // 사용자가 확인을 선택하였으면

      /* continue를 만나면 while 문의 조건식으로 올라간다.
       * 그러므로 아래 i++은 실행되지 않기 때문에 i는 증가되지 않는다.
       **/
      continue;
    }
    i++;
  }
</script>
</head>
<body></body>
</html>
```

▶ [연습문제 4-3] 1 ~ 60까지 한 줄에 10단위씩 출력하기

반복문을 사용해 아래와 같이 1부터 ~ 60까지의 수를 한 줄에 10단위씩 출력하는 프로그램을 작성하시오.

[실행결과]

1, 2, 3, 4, 5, 6, 7, 8, 9, 10
11, 12, 13, 14, 15, 16, 17, 18, 19, 20
21, 22, 23, 24, 25, 26, 27, 28, 29, 30

31, 32, 33, 34, 35, 36, 37, 38, 39, 40
41, 42, 43, 44, 45, 46, 47, 48, 49, 50
51, 52, 53, 54, 55, 56, 57, 58, 59, 60

5. 함수

프로그램은 단순히 하나의 동작만 하는 것이 아니라 요구사항에 따라서 해결해야할 다양한 기능을 여러 가지 동작으로 구현해야 되며 이런 여러 가지 동작을 연결하여 실행하게 된다. 이렇게 여러 가지 동작을 연결하여 하나의 작업 단위로 실행될 수 있도록 묶어 놓은 소스 코드의 덩어리를 함수(function)라고 한다. 하나의 작업 단위 코드를 함수로 만들어 놓으면 필요할 때 마다 함수 안의 코드만 실행할 수도 있고 다른 곳에서 이 기능이 필요하면 함수를 호출해 코드를 재사용할 수도 있다.

▶ 예제 5-1 함수 정의하고 사용하기

- JavaScriptStudyCh05/javascript05_01.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>함수 정의하고 사용하기</title>
<script>
  /* 이름 없는(익명) 함수 정의 - 함수 리터럴 표현으로 정의
   *
   * 이름 없는 함수를 정의할 때 아래와 같이 변수에 함수를 할당해 놓고
   * 다른 곳에서 이 변수 이름으로 함수를 여러 번 호출 할 수 있다.
   *
   * 함수(function)는 아래와 같이 여러 가지 동작을 연결하여 하나의
   * 작업 단위로 실행될 수 있도록 묶어 놓은 소스 코드의 덩어리 이다.
   * 함수는 중괄호("{}")를 사용해 함수의 영역을 지정하고 그 안에 필요한
   * 여러 가지 코드를 작성하는데 이 함수가 다른 곳에서 사용(호출)될 때
   * 이 중괄호 안에 작성한 소스 코드가 위에서부터 순차적으로 실행된다.
   **/
  let docPrint = function() {
     document.write("<h2>함수 정의하기</h2>");
     document.write("<ul>");
     document.write("  <li>익명 함수 정의하기</li>");
     document.write("  <li>function 예약어로 함수 정의하기</h2>");
     document.write("  <li>매개변수가 있는 함수 정의하기</h2>");
     document.write("</ul>");
  }

  /* 아래와 같이 함수를 사용하는 것을 "함수를 호출한다."라고 말한다.
   * 함수를 호출할 때는 함수에 매개변수가 없다면 인수는 지정하지 않으면 된다.
   * 만약 함수의 매개변수가 있다면 매개변수의 순서에 맞게 값을 지정하면 된다.
   **/
   docPrint();

  /* function 예약어로 함수 이름을 부여해 정의하기
```

```
 *
 * 함수를 정의할 때 함수의 괄호는 그 함수가 실행되면서 필요한 데이터를
 * 외부로부터 받을 수 있는 통로이다. 외부에서 데이터를 받을 필요가 없으면
 * 괄호 안을 비워서 빈 괄호로 작성하면 되고 함수 실행에 필요한 데이터를
 * 외부로부터 받으려면 아래와 같이 괄호 안에 매개변수를 만들면 된다.
 **/
function add(num1, num2) {
  document.write(`<p>${num1} + ${num2} = ${num1 + num2}</p>`);

  /* 함수는 실행한 결과를 return 문을 사용해 반환할 수 있지만 반환할
   * 필요가 없다면 return 문을 생략하면 된다. return 문은 현재 실행되는
   * 함수의 실행을 종료하고 함수를 호출한 곳으로 돌아가라는 명령으로 현재
   * 실행되고 있는 함수의 종료를 의미하며 반환 값이 없으면 생략하면 된다.
   * 만약 호출한 곳으로 돌아갈 때 함수가 실행한 결과 값을 넘겨주려면
   * 아래와 같이 return 문 뒤에 넘겨주려는 값을 지정하면 된다.
   **/
  //return;
  //return `<p>${num1} + ${num2} = ${num1 + num2}</p>`;
}

/* 매개변수가 있는 함수를 호출할 때 아래와 같이 함수의 괄호 안에
 * 넣어주는 값을 함수 안으로 전달하는 실제 값이라는 의미로 전달인수
 * 또는 전달인자라고 하며 간단히 인수(인자, argument)라고 부른다.
 **/
add(10, 20);

/* 자바스크립트에서는 매개변수가 있는 함수를 호출할 때 아래와 같이 함수의
 * 괄호 안에 인수를 지정하지 않고 호출할 수 있지만 이렇게 인수를 지정하지
 * 않거나 누락시키게 되면 함수 안의 코드가 실행되면서 오류가 발생할 수도
 * 있다. 또한 오류는 발생하지 않더라도 원하는 결과를 얻을 수 없기 때문에
 * 함수를 제대로 활용하려면 함수를 호출할 때 인수를 정확하게 지정해야 한다.
 **/
add();

// function 예약어로 함수 이름을 부여해 정의하기
function multiply(num1, num2) {

  /* 함수를 종료하고 호출한 곳으로 돌아갈 때 함수의 실행 결과를
   * 반환하려면 아래와 같이 return 문 뒤에 반환하려는 값을 지정하면 된다.
   **/
  return `<p>${num1} x ${num2} = ${num1 * num2}</p>`;
}

// 함수에 인수를 지정해 호출하고 반환 값을 받아 문서에 출력하고 있다.
document.write(multiply(20, 30));
```

```
</script>
</head>
<body></body>
</html>
```

▶ 예제 5-2 함수의 매개변수와 반환 값

- JavaScriptStudyCh05/javascript05_02.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>함수의 매개변수와 반환 값</title>
<script>
  /* function 예약어를 사용해 함수의 이름을 circle10으로 정의
   *
   * 이 함수의 기능은 반지름이 10인 원의 넓이를 구해 출력해 주는 함수로
   * 반지름이 10으로 고정되어 있어서 함수가 실행되면서 필요한 데이터를
   * 함수 내부에서 만들어 사용하므로 외부로부터 값을 받을 필요가 없어서
   * 별도로 매개변수를 선언하지 않았다. 또한 함수의 실행 결과를 함수 안에서
   * 문서에 출력하면 되기 때문에 return 문을 사용해 별도로 값을 반환하지 않았다.
   **/
  function circle10() {
    document.write(`반지름이 10인 원의 넓이 : ${10 * 10 * 3.14}<br>`);
  }

  /* 아래와 같이 함수를 사용하는 것을 "함수를 호출한다."라고 말한다.
   * 함수를 호출할 때 함수의 매개변수가 없으면 인수는 지정하지 않으면 된다.
   **/
  circle10();

  /* 이름 없는(익명) 함수 정의 - 함수 리터럴 표현으로 정의하고 변수에 할당
   *
   * 이름 없는 함수를 정의할 때 아래와 같이 변수에 함수를 할당해 놓고
   * 다른 곳에서 이 변수 이름으로 함수를 호출 할 수 있다.
   **/
  let rectangle = function(width, height) {
    return width * height;
  }

  /* function 예약어를 사용해 함수 이름을 triangle로 정의
   *
   * 함수를 정의할 때 함수의 괄호는 그 함수가 실행되면서 필요한 데이터를
   * 외부로부터 받을 수 있는 통로이다. 외부에서 데이터를 받을 필요가 없으면
```

```
 * 괄호 안을 비워서 빈 괄호로 작성하면 되고 함수 실행에 필요한 데이터를
 * 외부로부터 받으려면 아래와 같이 괄호 안에 매개변수를 만들면 된다.
 *
 * 함수를 정의할 때 괄호 안의 변수는 함수를 실행하기 위해서 외부로 부터
 * 입력 받는 값으로 함수 외부에서 함수 안쪽으로 값을 전달하여 사용할 수
 * 있도록 연결해 주는 매개체라는 의미로 매개변수(parameter)라고 부른다.
 **/
function triangle(width, height) {
  return width * height / 2;
}

/* 아래와 같이 함수를 사용하는 것을 "함수를 호출한다."라고 말한다.
 * 함수를 호출할 때는 함수에 매개변수가 없다면 인수는 지정하지 않으면 된다.
 * 만약 함수의 매개변수가 있다면 매개변수의 순서에 맞게 값을 지정하면 된다.
 *
 * 매개변수가 있는 함수를 호출할 때 아래와 같이 함수의 괄호 안에
 * 넣어주는 값을 함수 안으로 전달하는 실제 값이라는 의미로 전달인수
 * 또는 전달인자라고 하며 간단히 인수(인자, argument)라고 부른다.
 *
 * 매개변수와 인수를 정리하자면 매개변수는 함수를 정의할 때 함수의 괄호
 * 안에 선언하는 변수를 의미하며 함수를 호출할 때 지정하는 값은 인수라고
 * 구분하면 된다. 참고로 자바스크립트에서는 인수를 지정하지 않고 함수를
 * 호출할 수 있지만 함수 안의 코드가 실행되면서 오류가 발생할 수도 있다.
 * 또한 오류는 발생하지 않더라도 함수로부터 원하는 기능을 얻을 수 없기 때문에
 * 함수를 제대로 활용하려면 함수를 호출할 때 인수를 정확하게 지정해야 한다.
 **/
document.write('사각형의 넓이 : ' + rectangle(10, 20) + '<br/>');
document.write('삼각형의 넓이 : ' + triangle(20, 5) + '<br/>');

/* function 예약어를 사용해 함수 이름을 multiply로 정의하고 기본 매개변수 사용
 *
 * ECMAScript 2015(ES6)부터는 매개변수가 있는 함수를 선언할 때 매개변수의
 * 기본 값을 다음과 같이 지정할 수 있는 기능이 추가되었다. 함수를 실행할 때
 * 인수가 부족하면 함수를 정의할 때 지정한 기본 값이 사용된다. 함수를 호출할
 * 때 매개변수의 개수에 맞게 인수가 지정되면 기본 값은 사용되지 않는다.
 **/
function multiply(x, y = 5, z = 10) {

  // 매개변수 z에 기본 값이 없을 때 값이 전달되지 않으면 z는 undefined가 된다.
  console.log("z : " + z);

  /* 모든 매개변수의 값이 숫자로 입력되지 않으면 아래의 수식은
   * 숫자를 곱셈하는 수식이므로 NaN(Not a Number)이 반환 된다.
   **/
  return x * y * z;
```

```
  }

  /* 함수의 매개변수에 기본 값이 지정된 함수 호출하기
   *
   * multiply(x, y, z)함수는 매개변수가 3개인 함수로 이 함수를 호출하면서
   * 아래와 같이 인수를 2개만 지정하면 함수의 첫 번째와 두 번째 매개변수에
   * 아래 지정한 값이 차례로 전달되고 세 번째 매개변수는 기본 값이 적용된다.
   **/
   document.write("<h2>매개변수에 기본 값이 있는 함수</h2>");
  document.write(multiply(10, 30) + '<br>');

  // 반환 값이 없는 함수를 호출하고 반환 값을 확인해 보면 undefined가 출력된다.
  document.write("<h2>반환 값이 없는 함수의 반환 값</h2>");
  let returnValue = circle10();
  document.write(returnValue + '<br>');
</script>
</head>
<body></body>
</html>
```

▶ [연습문제 5-1] 섭씨와 화씨온도 변환기 함수 정의하기

온도는 측정 방법에 따라서 기준을 다르게 하는 화씨(℉)와 섭씨(℃) 온도 단위가 있다. 섭씨는 0℃에서 화씨는 32℉에서 시작하며 온도가 올라가는 비율도 서로 다르다고 한다. 그래서 물이 어는점부터 끓는점까지의 온도 범위도 섭씨는 0 ~ 100℃이며 화씨는 32℉ ~ 212℉까지 서로 다르다.
다음을 참고해 섭씨온도를 입력으로 받아서 화씨온도로 변환하거나 화씨온도를 입력으로 받아서 섭씨온도로 변환해 반환하는 다음과 같이 실행되는 함수를 작성하고 테스트 해 보자.

화씨를 섭씨온도로 변환하는 식 : (86℉ - 32) / 1.8 = 30℃
섭씨를 화씨온도로 변환하는 식 : (30℃ * 1.8) + 32 = 86℉

화씨와 섭씨 온도 서로 변환하기

86℉는 30.0℃ 입니다.
30℃는 86.0℉ 입니다.
23℃는 73.4℉ 입니다.
74℉는 23.3℃ 입니다.

▶ [연습문제 5-2] 체질량지수로 비만을 진단하는 함수 정의하기

체질량지수(BMI = body mass index)는 세계적으로 비만을 평가하는 공통 표준 지수라고 하는데, 이 체질량지수를 계산하면 다음과 같은 기준으로 어느 정도 비만인지 간단하게 진단할 수 있다.

다음 자료를 참고해서 사용자가 입력한 값을 바탕으로 체질량지수를 구하고 어느 정도 비만인지를 진단해 아래 그림과 같이 진단 메시지를 반환해 주는 함수를 작성해 보자.

저체중 : 18.5 미만
정상체중 : 18.5 이상 23 미만
과체중 : 23이상 25미만
경도비만 : 25이상 30미만
중도비만 : 30이상 35미만
고도비만 : 35이상

체질량 지수를 구하는 식(단위: 체중 = kg, 신장 = cm) : **체중 / (신장 * 신장 / 10000)**

체질량지수로 비만을 진단하는 함수

홍길동님의 키는 172cm, 몸무게는 93kg이며
체질량지수는 31.44로 중도비만에 해당합니다.

왕호감님의 키는 185cm, 몸무게는 80kg이며
체질량지수는 23.37로 과체중에 해당합니다.

어머나님의 키는 167cm, 몸무게는 53kg이며
체질량지수는 19.00로 정상체중에 해당합니다.

▶ 예제 5-3 변수의 유효범위

- JavaScriptStudyCh05/javascript05_03.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>변수의 유효 범위</title>
<script>
  /* 자바스크립트에서 변수를 사용할 때 그 변수가 가지는 유효 범위(scope)를
   * 잘 살펴서 프로그래밍을 해야 한다. 변수의 유효 범위란 그 변수가 어디에서
   * 선언되었는가에 따라서 해당 변수에 접근할 수 있는 가능한 범위가 달라지는데
   * 이렇게 변수에 접근할 수 있는 가능한 범위를 유효 범위(scope)라고 한다.
   *
   * ECMAScript 2015(ES 6) 이전에는 var 예약어를 사용해 변수를 선언했는데
   * 이 var 예약어로 선언한 변수는 변수가 선언된 위치에 따라서 크게 전역 변수
   * (global variable)과 지역 변수(local variable)로 나누며 전역 변수는 전역 스코프
   * (local scope)를 가지고 지역 변수는 지역 스코프(global scope)를 가지게 된다.
   * 전역 스코프란 현재 웹 문서 어디에서든지 접근할 수 있는 스코프를 의미하며
   * 지역 스코프란 변수가 선언된 함수 안에서 유효한 함수 스코프를 의미하며 그
   * 변수가 선언된 함수 안에서만 해당 변수에 접근할 수 있어 지역 변수라고 한다.
   **/
```

```
// script 영역에 선언된 변수는 전역 변수가 되면 전역 스코프를 가진다.
var num = 10;

// 함수의 매개변수는 함수 안에서 선언되었기 때문에 지역 변수가 된다.
function add(num1, num2) {

  /* var 예약어를 사용하지 않고 함수 안에 선언된 변수는 전역 스코프를 갖는다.
   * 만약 num3을 선언할 때 변수 앞에 var 예약어를 사용하면 현재 함수 안에서만
   * 접근 가능한 지역 스코프(함수 스코프)를 가지기 때문에 함수 밖에서 이 변수에
   * 접근하면 "ReferenceError: num3 is not defined" 오류가 발생한다.
   **/
  num3 = 10;

  /* 함수 안에서 코드 블록에 var로 변수를 선언하면 그 변수가 선언된 블록 안에서
   * 유효한 블록 스코프(block scope)를 가지는 것이 아니라 현재 블록이 소속된 함수
   * 안에서 유효한 함수 스코프(function scope)를 가지게 된다. 블록 스코프란 중괄호
   * ("{}")로 둘러싸인 영역을 의미하며 이 블록 별로 변수의 유효 범위가 결정되는
   * 것을 블록 스코프(block scope)라고 한다. 블록에는 아래와 같이 코드 블록이
   * 될 수도 있고 if 문의 블록, for 블록, 함수의 블록이 될 수도 있다.
   **/
  {var num10 = 10}

  // 변수 num은 전역 스코프이므로 현재 페이지의 어디에서든지 접근 가능하다.
  return num1 + num2 + num + num10;
}
console.log(`add 함수 : ${add(10, 20)}`);
console.log(`num3 : ${num3}`);

// 자바스크립트에서는 아래와 같이 코드 블록을 사용할 수 있다.
{
  /* var로 선언된 변수는 블록("{}") 안에 선언되었더라도 함수를 기준으로
   * 스코프를 가지기 때문에 함수 블록이 아니면 이 블록 밖에서 접근할 수 있다.
   **/
  var num4 = 100;
}

// num4는 위의 블록에서 var로 선언되었기 때문에 블록 밖에서 접근할 수 있다.
console.log(`num4 : ${num4}`);

/* ECMAScript 2015(ES 6)부터 변수를 선언할 때는 let 예약어를 사용하고
 * 상수를 선언할 때는 const 예약어를 사용하도록 예약어가 추가 되었다.
 * let과 const 예약어로 선언된 변수도 var로 선언된 변수와 마찬가지로 변수가
 * script 영역에 선언되면 전역 변수(global variable)라고 하며 함수, 제어문의
 * 블록, 코드 블록 등에 선언되면 지역 변수(local variable)라고 한다. 하지만
 * let과 const 예약어로 선언된 전역 변수는 var 예약어로 선언된 전역 변수와
```

```
    * 마찬가지로 현재 페이지의 자바스크립트 전체에서 접근 가능하지만 유효 범위는
    * 전역 스코프(전역 스코프는 window 객체 영역으로 브라우저 전체 영역을 의미)가
    * 아니라 스크립트에서만 접근 가능한 스크립트 스코프(script scope)를 가지게
    * 되며 지역 변수의 스코프는 그 변수가 선언된 블록("{}") 범위 안에서만 접근
    * 가능하기 때문에 블록 스코프(block scope)라고 한다.
    **/
  let str1 = "스크립트 스코프";
  const SCRIPT_SCOPE_NUM = 100;

  function sum(num1, num2) {
    let sum = 0;
    for(let i = num1; i <= num2; i++) {
      sum += i;
    }

    /* let과 const 예약어로 선언된 변수는 블록 스코프를 가지기 때문에 for 문
     * 안에서 선언된 변수는 for 문의 블록("{}") 안에서 유효하므로 아래 코드를
     * 실행하면 "ReferenceError: i is not defined" 오류가 발생한다.
     **/
    //console.log(`sum 함수에서 for 문의 i에 접근 : ${i}`);

    /* script 영역의 최상위에 let과 const로 선언된 변수와 상수는 현재 페이지의
     * 어디에서든지 접근 가능한 스크립트 스코프를 가지기 때문에 아래와 같이
     * 상수 SCRIPT_SCOPE_NUM에 접근할 수 있다.
     **/
    return sum + SCRIPT_SCOPE_NUM;
  }
  console.log(`sum(1, 10) : ${sum(1, 10)}`);
</script>
</head>
<body></body>
</html>
```

▶ 예제 5-4 화살표 함수

- JavaScriptStudyCh05/javascript05_04.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>화살표 함수</title>
<script>
  /* 화살표 함수(arrow function)
   * ECMAScript 2015(ES6)부터는 화살표 표기법(=>)을 사용해 함수를 조금 더
```

```
 * 간단하게 정의할 수 있도록 지원하는데 이런 함수를 "화살표 함수" 또는
 * "애로우 펑션(arrow function)"이라고 한다. 화살표 함수는 앞에서 익명함수를
 * 선언할 때와 마찬가지로 함수를 화살표 표현식을 사용해 정의하며 함수의
 * 이름이 없기 때문에 아래와 같이 변수에 화살표 함수를 할당해 놓고 다른
 * 곳에서 이 변수 이름으로 함수를 호출 할 수 있다.
 *
 * let func = (매개변수) => { 함수 본문 };
 * func();
 *
 * 자바스크립트는 함수를 1급 시민(first-class citizen)으로 취급
 * 함수 정의를 변수에 저장하거나 간결하게 한 줄짜리 함수를 지원하는
 * 이유는 자바스크립트에서 함수를 활용해 보다 유연한 프로그래밍을 할 수
 * 있도록 지원하기 위해서이다. 자바스크립트에서는 함수를 변수에 할당
 * 할 수 있고, 함수를 다른 함수의 인수로 지정할 수 있으며 다른 함수의
 * 반환 값으로 사용할 수도 있다. 이는 프로그래밍 언어에서 함수를 1급
 * 시민으로 취급하는 기본 조건으로 자바스크립트는 이를 기본 지원한다.
 **/

// 매개변수가 없는 화살표 함수 정의
let circle10 = () => {
  document.write(`반지름이 10인 원의 넓이 : ${10 * 10 * 3.14}<br>`);
}

// 함수 본문의 코드가 한 줄일 경우 중괄호("{}") 생략 가능
let circle11 = () =>
  document.write(`반지름이 11인 원의 넓이 : ${11 * 11 * 3.14}<br>`);

// 기존 익명함수 호출과 같이 변수 이름으로 함수를 호출할 수 있다.
circle10();
circle11();

/* 매개변수가 1개인 화살표 함수 정의
 * 매개변수가 1개인 경우 매개변수의 소괄호를 생략할 수 있으며
 * 함수 본문이 한 줄일 경우 중괄호("{}")와 return도 생략할 수 있다.
 **/
let circle0 = radius => radius * radius * 3.14;
document.write(`매개변수 1개인 화살표 함수 : ${circle0(10)}<br>`);

// 매개변수가 2개 이상인 화살표 함수 정의
let rectangle = (width, height) => width * height;
document.write(`사각형의 넓이 : ${rectangle(10, 10)}<br>`)

// 매개변수에 기본 값이 있는 화살표 함수 정의
let triangle = (width, height = 5) => width * height / 2;
document.write(`삼각형의 넓이 : ${triangle(10)}<br>`);
```

```
</script>
</head>
<body></body>
</html>
```

▶ 예제 5-5 가변인자 함수 정의하기

- JavaScriptStudyCh05/javascript05_05.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>가변인자 함수 정의하기</title>
<script>
  function multiplyFunc(num1, num2) {

    /* 매개변수의 개수를 정확히 맞추기 위해 arguments 객체를 사용하였다.
     * 함수 안에서 매개변수 정보를 관리하는 arguments 객체를 사용할 수 있다.
     **/
    if(arguments.length != 2) {
      alert('인수의 수가 다릅니다. 인수는 2개만 입력 하세요');
      return;
    }
    alert(num1);
    // 인수 값이 undefined 이거나 숫자가 아니면 기본 값을 지정함
    if(num1 == 'undefined' || isNaN(num1)) { num1 = 1; }
    if(num2 == 'undefined' || isNaN(num2)) { num2 = 1; }

    console.log("multiplyFunc(num1, num2) : " + num1 * num2);
  }

  /* 함수를 선언할 때 함수가 몇 개의 인수를 받게 될 지 모르는 경우 아래와 같이
   * 전개구문을 이용해 매개변수를 만들면 인수의 개수와 상관없이 함수를 정의할
   * 수 있으며 함수 안에서 배열처럼 매개변수에 접근해 처리할 수 있다.
   * 물론 arguments 객체를 통해서 함수로 전달된 매개변수에 접근할 수도 있다.
   **/
  function multiplyParams(...nums) {
    console.log(nums);
    console.log(arguments);
    let result = 0;
    for(let num of nums) {
      if(isNaN(num)) {
        continue;
      }
```

```
        result += num;
      }
      console.log("multiplyParams() : " + result);
    }
</script>
</head>
<body>
  <div>
    <!--
      자바스크립트는 함수 정의와 다르게 인수의 개수를 다르게 호출할 수 있지만
      함수의 기능을 제대로 사용하기 위해서는 인수를 정확하게 지정해야 한다.
    -->
    <input type="button" value="multiplyFunc(1, 2) 호출"
      onclick="multiplyFunc(10, 30, 20)" />
    <input type="button" value="multiplyParams() 호출"
      onclick="multiplyParams(10, 20, 'NaN', 30, 40)" />
  </div>
</body>
</html>
```

▶ 예제 5-6 폼에서 함수 활용하기

- JavaScriptStudyCh05/javascript05_06.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>폼에서 함수 활용하기</title>
<script>
  /* 폼에서 "나이 입력" 버튼이 클릭되면 실행되는 이벤트 처리 함수
   *
   * 이 함수는 버튼이 클릭되었을 때 나이를 입력 받아서 폼 안의 나이
   * 입력상자에 입력해 주는 기능을 제공하는 함수이며 필요한 모든
   * 데이터는 함수 내부에서 처리하므로 외부에서 데이터를 받을 필요가
   * 없기 때문에 별도로 매개변수를 선언하지 않았다.
   **/
  let inputAge = function() {

    // prompt 창에서 나이를 입력 받는다.
    let age = prompt("나이를 입력해주세요");
    form1.age.value = age;
  }

  /* 폼에 있는 관심분야 선택 상자를 매개변수로 받아서
```

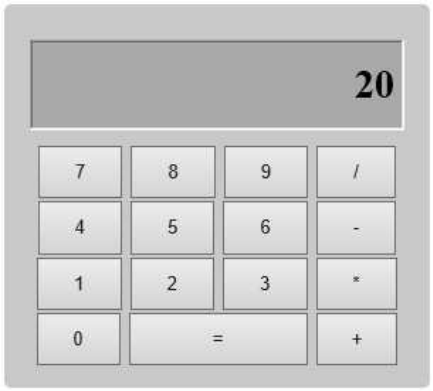
```
 * 현재 선택된 항목의 텍스트를 추출해 반환하는 함수
 *
 * 이 함수는 선택상자(select 박스)를 매개변수로 받아서 선택상자에서
 * 현재 선택된 텍스트를 추출해 반환하는 기능을 제공하는 함수로 선택
 * 상자를 매개변수로 받을 수 있도록 정의하였다.
 **/
function getSelectedValue(elem) {

  /* 매개변수로 받은 elem은 선택상자(select 요소)이며 이 요소의
   * selectedIndex 속성으로 현재 선택된 항목의 index 값을 알 수
   * 있으며 이 속성을 이용해 아래와 같이 options 속성에 현재 선택된
   * 항목의 index를 지정하면 value 또는 text를 읽어 올 수 있다.
   *
   * 함수는 return 문을 이용해 실행한 결과 값을 반환할 수 있다.
   * 만약 실행 결과를 반환할 필요가 없다면 return 문을 생략하면 된다.
   **/
  return elem.options[elem.selectedIndex].text;
}

/* 폼에 있는 관심분야 선택상자(select 박스)에서
 * 항목 선택이 변경될 때 마다 실행되는 이벤트 처리 함수
 *
 * 이 함수는 폼의 관심분야 선택상자에서 항목 선택이 변경될 때 마다
 * 호출되는 함수로 선택상자를 매개변수로 받을 수 있도록 정의하였다.
 **/
function changeSelect(elem) {

  /* 매개변수로 받은 elem을 다른 함수를 호출하면서 인수로 지정 했다.
   * getSelectedValue() 함수의 반환 값이 없으면 undefined가 반환된다.
   **/
  let val = getSelectedValue(elem) + ' 선택됨';
  alert(val);
}

/* 폼이 submit 될 때 호출되는 이벤트 처리 함수
 * 폼 안에서 submit 버튼이 클릭되면 폼에서 submit 이벤트가 발생하고
 * 폼의 onsubmit 속성에 지정한 이 함수가 호출되는데 이 때 폼 안에 있는
 * 폼 컨트롤에 입력된 값을 읽어와 alert() 함수를 통해 출력해 주는 함수이다.
 **/
function registSubmit() {
  let name = document.form1.name.value;
  let age = document.form1.age.value;
  let interest = form1.choice.options[form1.choice.selectedIndex].text;

  alert('입력된 정보\n'
```

```
            + '이름 : ' + name + "\n"
            + '나이 : '+ age + "\n"
            + '관심분야 : ' + interest);

        // 폼이 submit 되는 것을 막는다.
        return false;
    }
</script>

<!-- jQuery를 이용하면 보다 간편하게 DOM을 제어할 수 있다. -->
<script src="js/jquery-3.3.1.min.js"></script>
<script>
    $(function() {
        // select 박스의 선택이 변경될 때 발생하는 이벤트를 처리하는 코드
        $("#choice").change(function() {
            console.log(this    + " , " + $(this).val() + "선택됨");
        });
    });
</script>
</head>
<body>
<!--
    form 요소의 onsubmit 속성에 함수를 지정하면 해당 폼이 submit 될 때
    지정한 함수가 호출된다. 다시 말해 폼에서 submit 버튼이 클릭되면
    submit 이벤트가 발생하는데 onsubmit 속성에 지정한 함수가 호출된다.
-->
<form name="form1" onsubmit="return registSubmit();">
    이름 : <input type="text" name="name"/><br/>
    나이 : <input type="text" name="age" readonly/>
    <!--
        아래 버튼이 클릭될 때 onclick 속성에 지정한 inputAge() 함수가 호출된다.
     -->
    <input type="button" value="나이 입력" onclick="inputAge();"/><br/>
    <!--
        관심분야로 표시되는 선택 상자에서 선택된 항목이 변경될 때 마다
        onchange 속성에 저장한 changeSelect() 함수가 호출된다.
     -->
    관심분야 : <select name="choice" id="choice" onchange="changeSelect(this);">
            <option>자바</option>
            <option>JSP</option>
            <option>자바스크립트</option>
            <option>안드로이드</option>
            <option>스프링</option>
    </select><br/><br/>
    <input type="submit" value="등록하기" />
```

```
</form>
</body>
</html>
```

▶ [연습문제 5-3] 함수를 사용해 사칙연산을 수행하는 계산기 만들기

다음 그림과 같이 UI를 구성하여 사칙연산을 수행하는 계산기를 만들어 보자.



1. 버튼이 클릭되면 텍스트 입력상자에 클릭된 숫자가 표시되고 연산자가 클릭되기 전까지 연속적인 숫자로 표시되게 하시오.
2. 첫 번째 입력된 숫자가 '0'이면 텍스트 입력상자에 입력되지 않게 작성하시오.
3. 텍스트 입력상자에 입력된 숫자 없이 연산자 버튼이 먼저 클릭되면 "먼저 숫자를 입력해 주세요"라는 경고 창을 띄우고 이미 숫자가 입력된 상태라면 기존에 입력된 숫자를 지우고 새롭게 입력한 숫자가 텍스트 입력상자에 표시 되도록 하시오.
4. 연산할 숫자가 입력되지 않은 상태에서 "=" 클릭되면 아무 동작도 하지 않도록 하시오.
5. 모두 정상적으로 입력된 상태에서 '=' 버튼이 클릭되면 입력된 숫자와 연산자를 바탕으로 계산한 결과 값이 텍스트 입력상자에 출력되도록 하시오.

6. 객체

객체(Object)는 많은 프로그래밍 언어에서 사용되는 용어로 프로그램에서 사용될 수 있는 모든 대상을 가리키며 실세계에서 사용되는 특정 대상을 컴퓨터 세계에서 데이터로 관리하기 위해서 대상의 속성과 기능(동작, 행위)을 정의해 추상화(Abstration) 해 놓은 것이라 할 수 있다.
예를 들어 실세계에서 사용되는 특정 대상을 쇼핑몰의 "고객"이라고 가정한다면 고객이 가지는 속성은 이름, 생년월일, 성별, 주소, 가입일 등이 되고 기능(동작, 행위)은 주문, 리뷰작성 등이 될 것이며 이 정보들을 컴퓨터 세계에서 데이터로 관리하기 위해서 하나로 묶어서 정의한 것을 객체라 할 수 있을 것이다.
객체는 실제 눈에 보이는 것과 눈에는 보이지 않지만 실생활에서 사용되는 개념이 모두 포함될 수 있으며 우리가 보고 느끼고 인지할 수 있는 모든 대상이 객체가 될 수 있다. 이렇게 우리가 보고 인지할 수 있는 실세계의 사물을 흉내 내어 가장 기본적인 단위인 객체를 만들고 이 객체들 간의 유기적인 상호작용을 바탕으로 여러 객체를 연결해 하나의 완성된 프로그램을 만드는 프로그래밍 방법론을 객체지향 프로그래밍(OOP, Object Oriented Programming)이라고 한다.
자바스크립트로도 객체지향 프로그래밍(Object Oriented Programming) 기법을 지원하는 언어이며 자바스크립트에서 사용하는 객체에는 자바스크립트에서 기본적으로 제공하는 내장객체(Built in Object), 웹 브라우저를 계층 구조로 객체화한 웹 브라우저 객체(BOM, Browser Object Model), HTML 문서를 객체화한 문서 객체 모델(DOM, Document Object Model)이 있으며 이외에도 사용자가 필요에 의해 정의하는 사용자 정의 객체(Custom Object) 등이 있다.
자바스크립트에서 사용자 정의 객체를 만드는 방법은 크게 세 가지 정도로 나눌 수 있는데 그 첫 번째 방법으로 객체 리터럴로 객체를 생성하고 각각의 프로퍼티와 메소드를 추가하는 방법이 있고 두 번째로 생성자 함수(익명 함수, 생성자 함수)를 정의하고 이 함수를 통해 객체를 생성하는 방법이 있으며 마지막 세 번째로 클래스를 정의해 객체를 생성하는 방법이 있다. 참고로 클래스를 이용해 객체를 생성하는 방법은 ECMAScript 2015(ES6)에서 새롭게 도입된 방법이다.
객체지향에서 객체의 데이터는 객체가 가지고 있는 속성이며 객체의 동작은 객체가 수행할 수 있는 기능으로 자바스크립트는 객체의 데이터와 기능을 변수와 메서드로 표현하고 있다. 객체지향 프로그래밍에서는 객체 안의 변수를 속성(Property)이라 하고 객체의 기능을 메소드(Method)라 부른다.

6.1 객체 정의하고 사용하기

▶ 예제 6-1 객체를 정의하는 방법

- JavaScriptStudyCh06/javascript06_01.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>객체를 정의하는 방법</title>
<script>
/* 1. 객체 리터럴 방식으로 객체 정의하기
 * 자바스크립트에서 객체를 표기하는 방법을 JSON(JavaScript Object Notation)이라고 한다.
 * 참고 사이트 : https://www.json.org/json-ko.html
 **/
const Student = {
```

```
  /* 객체는 아래와 같이 데이터와 기능으로 구성되어 있으며 데이터는
   * 변수로 기능은 함수로 표현된다. 객체 안에 존재하는 변수를
   * 속성(Property)이라 하고 함수를 메소드(Method)라 부른다.
   * 다시 말해 객체는 속성과 메서드를 묶어서 하나의 데이터로 만든 것이다.
   **/
  이름: '홍길동',
  학년: 3,
  toString: function() {
    return '<p>'
          + '이름 : ' + this.이름 + '<br/>'
          + '학년 : ' + this.학년 + '<br/>'
        + '</p>';
  }
};

// 2. 익명함수 방식으로 객체 정의하기
const Member = function(name, id, address) {
  this.name = name;
  this.id = id;
  this.address = address;
  this.toString = function() {
    return '<p>'
          + '이 름 : ' + this.name + '<br/>'
          + '아이디 : ' + this.id + '<br/>'
          + '주 소 : ' + this.address + '<br/>'
        + '</p>';

  };
};

// 3. 생성자 함수 방식으로 객체 정의하기
function Human(name, age, gender) {
  this.name = name;
  this.age = age;
  this.gender = gender;
  this.toString = function() {
    return '<p>'
          + this.name + '(' + this.gender + '-' + this.age + ')<br/>'
        + '</p>';
  };
}

/* 생성자 함수와 익명 함수 방식으로 객체를 정의하면 아래와 같이
 * new 연산자를 사용해 서로 다른 속성을 가지는 객체를 무한정 생성할 수 있다.
```

```
 * new 연산자를 사용해 객체를 생성하는 것을 "인스턴스를 생성한다." 라고 말 한다.
 **/
let member = new Member("홍길동", "midas", "서울 구로구");
let human = new Human("김유신", 25, "남성");

// 아래와 같이 출력 함수에서 참조 변수를 사용하면 toString()이 자동으로 호출된다.
document.write("<h2>객체를 정의하기</h2>");
document.write(Student);
document.write(member);
document.write(human);
</script>
</head>
<body></body>
</html>
```

▶ 예제 6-2 객체 정의하고 사용하기

- JavaScriptStudyCh06/javascript06_02.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>객체 정의하고 사용하기</title>
<script>
  /* 객체 리터럴로 빈 객체를 생성하고 별도로 프로퍼티 설정하기
   * 아래 방법은 객체를 만드는 방법이 분산되어 있어 좋은 방법은 아니다.
   **/
  const objHuman1 = {};
  objHuman1.name = "홍길동";
  objHuman1.age = 30;
  objHuman1.toString = function() {
    return this.name + ", " + this.age;
  };

  /* 아래와 같이 프로퍼티와 메서드를 가진 객체를 객체 리터럴로 정의할 수 있다.
   * 객체 리터럴 방식은 객체를 정의할 때 마다 메서드가 계속 중복 정의된다.
   * 아래 객체 리터럴과 동일한 형식으로 objHuman3, objHuman4 등으로
   * 선언해 다른 프로퍼티 값을 가지게 정의할 수 있지만 객체를 정의할 때 마다
   * 계속해서 프로퍼티와 메서드가 중복 정의된다.
   **/
  const objHuman2 = {
      name: "장길산",
      age: 30,
      /* ECMAScript2015(ES6)에서 메서드를 정의할 때
```

```
        * 다음과 같이 function 예약어를 생략하고 간략하게 작성할 수 있다.
        **/
        toString() {
          return this.name + ", " + this.age;
        }
    }

    /* 생성자 함수를 이용해 다음과 같이 객체를 정의할 수 있다.
     * 생성자 함수는 일반 함수와 다르게 대문자로 시작하도록 정의한다.
     **/
    function Human(name, age) {
      this.name = name;
      this.age = age;
      this.toString = function() {
        return name + ", " + age;
      }
    }

    /* 생성자 함수와 익명 함수 방식으로 객체를 정의하면 new 연산자로
     * 객체를 생성할 수 있고 프로퍼티와 메서드가 중복 정의되지 않는다.
     **/
    let human1 = new Human("이순신", 35);
    let human2 = new Human("김유신", 25);

    /* 자바스크립트에서 객체는 key와 value 형태의 데이터로 이루져 있는데
     * 이런 형식의 데이터를 연관 배열(Associative Array)이라고 한다.
     * 연관 배열은 숫자로 된 index 대신에 key를 사용해 데이터를 저장하고
     * 관리하는 배열이다. 자바스크립트에서 객체에 접근하는 방법은 아래와 같이
     * 객체를 참조하는 변수와 닷 연산자(.)를 사용해 객체 안의 프로퍼티와 메서드에
     * 접근 할 수 있다. 또한 배열과 같이 대괄호([])를 이용해 접근할 수 있는데
     * 이때는 숫자로 된 index를 사용하는 것이 아니라 아래와 같이 key를 사용해
     * 프로퍼티와 메서드에 접근할 수 있다.
     **/
    document.write("<h2>객체에 접근하는 방법</h2>");
    document.write(`닷 연산자(.)를 이용해 접근 : ${human1.name}<br/>`);
    document.write(`객체의 속성(key)을 이용해 접근 : ${human1["name"]}<br/>`);
    document.write(human1["toString"]());
</script>
</head>
<body></body>
</html>
```

▶ 예제 6-3 클래스로 객체 만들기

- JavaScriptStudyCh06/javascript06_03.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>클래스로 객체 만들기</title>
<script>
  /* ECMAScript2015(ES6)에서 자바스크립트도 클래스라는 개념이 도입되었다.
   * 그러나 자바스크립트의 클래스는 다른 프로그래밍 언어의 클래스와 동작 방식이
   * 다르다. 자바스크립트에서 사용하는 클래스는 기존의 생성자 함수 방식을
   * 조금 더 쉽게 표현할 수 있도록 변경한 것으로 완전한 클래스 방식은 아니다.
   *
   * 클래스를 정의할 때는 다음과 같이 class 다음에 클래스명을 작성하면 된다.
   **/
  class Member {

    /* 클래스 안에서 생성자와 메서드를 분리해 작성한다.
     * 생성자는 클래스로 인스턴스를 생성할 때 호출하는 함수이다.
     * 생성자를 통해서 인스턴스가 가지는 속성을 초기화 할 수 있다.
     **/
    constructor(name, id, address) {
      this.name = name;
      this.id = id;
      this.address = address;
    }

    // 메소드는 다음과 같이 메소드 이름만 정의해 내용을 작성하면 된다.
    toString() {
      return '<p>'
          + '이 름 : ' + this.name + '<br/>'
          + '아이디 : ' + this.id + '<br/>'
          + '주 소 : ' + this.address + '<br/>'
        + '</p>';
    }
  }

  // 다음과 같이 클래스 표현식으로 클래스를 정의할 수 있다.
  const Student = class {
    constructor(name, grade) {
      this.name = name;
      this.grade = grade;
    }
    toString() {
      return '<p>'
```

```
            + '이름 : ' + this.name + '<br/>'
            + '학년 : ' + this.grade + '<br/>'
          + '</p>';
    }
  }

  /* 클래스도 생성자 함수방식과 마찬가지로 인스턴스를 생성할 때는 아래와 같이
   * new 연산자를 사용하며 서로 다른 속성을 가지는 객체를 무한정 생성할 수 있다.
   **/
  let member = new Member("홍길동", "midas", "서울 구로구");
  let s1 = new Student("김유신", 2);
  let s2 = new Student("어머나", 3);

  /* 자바스크립트에서 객체에 접근하는 방법은 아래와 같이 객체를 참조하는 변수와
   * 닷 연산자(.)를 사용해 객체 안의 프로퍼티와 메서드에 접근 할 수 있다. 또한
   * 배열과 같이 대괄호([])를 이용해 접근할 수 있는데 이때는 숫자로 된 index를
   * 사용하는 것이 아니라 key를 사용해 프로퍼티와 메서드에 접근할 수 있다.
   **/
  document.write("<h2>클래스로 객체 만들기</h2>");
  document.write(`닷 연산자(.)를 이용해 접근 : ${member.name}<br/>`);
  document.write(`객체의 속성(key)을 이용해 접근 : ${s1["name"]}<br/>`);
  document.write(s2["toString"]());
</script>
</head>
<body></body>
</html>
```

▶ 예제 6-4 객체의 타입 체크 instanceof 연산자

- JavaScriptStudyCh06/javascript06_04.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>객체의 타입 체크 - instanceof 연산자</title>
<script>
  // 생성자 함수방식으로 객체정의
  function Member(name, id) {
    this.name = name;
    this.id = id;
  }

  /* 생성자 함수를 통해 객체 생성
   * 자바스크립트의 모든 객체는 Object 객체를 묵시적으로 상속받으므로
```

```
    * 아래와 같이 new 연산자와 생성자 함수를 호출하면 Member 객체가
    * 생성되고 Member가 상속받는 Object 객체의 생성자 함수도 호출된다.
    **/
   const member = new Member('이순신', 'midastop');

   /* instanceof 연산자는 어떤 생성자 함수를 통해서 그 객체가 생성되었는지
    * 확인할 때 사용하는 연산자로 모든 객체는 Object 객체를 묵시적으로 상속하므로
    * new 연산자와 그 객체의 생성자 함수를 호출해 객체를 생성하게 되면 상속 관계에
    * 있는 모든 부모 객체의 생성자 함수가 호출되기 때문에 현재 객체와 부모 객체를
    * instaceof 연산을 하면 true가 반환된다.
    **/
   document.write("<h2>객체의 타입 체크</h2>");
   document.write(`typeof member : ${typeof member}<br>`);
   document.write(`member instanceof Member : ${member instanceof Member}<br/>`);
   document.write(`member instanceof String : ${member instanceof String} <br/>`)
   document.write(`member instanceof Object : ${member instanceof Object} <br/>`);
</script>
</head>
<body></body>
</html>
```

6.2 내장객체

자바스크립트 엔진에 기본 내장되어 있는 객체를 내장객체(Built-in Object)라고 하며 이 객체들은 자바스크립트가 사용되는 곳이면 어디서든 사용할 수 있다.
자바스크립트도 객체지향 프로그래밍을 지원하는 프로그래밍 언어이므로 자바스크립트에서도 상속 기법을 사용할 수 있다. 또한 자바스크립트에서 사용되는 모든 객체의 최상위에는 Object 객체가 존재하며 자바스크립트에서 모든 객체는 묵시적(Implicit)으로 Object 객체를 상속받게 된다.
내장객체에는 최상위 객체인 Object를 비롯해 문자(String), 날짜(Date), 배열(Array), 수학(Math), 정규표현(RegExp)과 관련된 객체를 지원하고 있다.

▶ 예제 6-5 자바스크립 최상위 객체 Object

- JavaScriptStudyCh06/javascript06_05.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>자바스크립트의 최상위 내장객체  - Object</title>
<script>
   /* Object 객체는 자바스크립트에서 최상위 객체이며 기본이 되는 내장객체이다.
    * Object 객체도 new 연산자와 생성자 함수를 호출해 인스턴스를 생성한다.
    **/
```

```
let object = new Object();

// 객체 리터럴 방식으로 Obejct를 묵시적으로 상속하는 객체를 생성할 수 있다.
let score = {
   kor : 87,
   math : 85,
   english : 91
}

/* 객체를 참조하는 참조 변수를 출력 메서드에 지정하면
 * Object 객체에 정의되어 있는 toString() 메서드가 자동으로 호출된다.
 **/
document.write(score + ", " + object.toString()+ "<br/>");
document.write((score instanceof Object) + "<br/>");

/* Object 객체의 프로토타입에 메서드를 등록하게 되면 자바스크립트의
 * 모든 객체가 Object를 묵시적으로 상속하기 때문에 모든 객체에서
 * 이 메서드를 사용할 수 있게 된다.
 **/
Object.prototype.everyFunc = function() {
   alert("Object everyFunc 호출됨");
}

/* Number 객체와 String 객체를 생성하고 아래와 같이 Object 객체의
 * 프로토타입에 정의한 everyFunc() 메서드를 호출하면 정상적으로 호출된다.
 **/
const num = new Number();
let str = "안녕 자바스크립트";
num.everyFunc();
str.everyFunc();

/* for in 반복문은 객체 안에 있는 속성을 가지고 반복 수행을 해야 할 때 사용한다.
 * for in 반복문은 객체가 가지고 있는 속성의 개수 만큼 반복하며 아래에서 key를
 * 가변수라고 부르며 반복문이 실행되면 가변수에는 객체의 속성명이 차례로 할당 된다.
 **/
for(key in score) {
   console.log(key + " : " + score[key]);
}

/* Object 객체의 메서드
 * keys()    : 객체가 가지고 있는 키 리스트를 배열로 반환한다.
 * values()  : 객체가 가지고 있는 값의 리스트를 배열로 반환한다.
 * entries() : 객체의 속성과 값 한 쌍이 배열로 저장된 2차원 배열을 반환한다.
 **/
console.log(Object.keys(score));
```

```
  console.log(Object.values(score));
  console.log(Object.entries(score));
</script>
</head>
<body>
</body>
</html>
```

▶ 예제 6-6 수학과 숫자관련 객체(Math, Number)

- JavaScriptStudyCh06/javascript06_06.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>수학과 숫자관련 내장객체(Math, Number)</title>
<script>
  // 아래에 사용된 메서드 외에 삼각함수, 로그관련 함수를 제공한다.
  console.log("-10의 절댓값 : " + Math.abs(-10));
  console.log("-10, 3, 10, 5, 7에서 최댓값 : " + Math.max(-10, 3, 10, 5, 7));
  console.log("0 ~ 1 사이의 난수 : " + Math.random());
  console.log("123.456의 첫째자리 반올림 : " + Math.round(123.456));
  console.log("10의 3승 : " + Math.pow(10, 3));
  console.log("2의 제곱근 : " + Math.sqrt(2));
  console.log("원주율 : " + Math.PI);
  console.log("123.987 소수 첫째자리 버림 : " + Math.floor(123.987));
  console.log("123.123 소수 첫째자리 올림 : " + Math.ceil(123.123));

  // Number 객체의 toFixed()를 이용해 소수 셋째자리에서 반올림하여 둘째자리까지 표시
  console.log("123.123 소수 둘째자리까지 표시 : " + (123.123).toFixed(2));
  console.log("123.789 소수 둘째자리까지 표시 : " + (123.789).toFixed(2));

  console.log("1 ~ 45 사이의 난수 : " + Math.floor(Math.random() * 45) +1);
</script>
</head>
<body>
</body>
</html>
```

▶ 예제 6-7 날짜관련 객체(Date)

- JavaScriptStudyCh06/javascript06_07.html

```
<!DOCTYPE html>
<html>
```

```
<head>
<meta charset="UTF-8">
<title>날짜 내장객체(Date)</title>
<script>
  // 현재 날짜와 시간 정보를 저장하는 Date 객체 생성
  let date = new Date();

  // 지정한 날짜(2021년 12월 25일)와 시간 정보를 저장하는 Date 객체 생성
  let date1 = new Date(2021, 11, 25);

  console.log("date : " + date);
  console.log("date1 : " + date1);

  let week = "";

  /* Date 객체는 날짜와 시간정보를 다루를 객체로 날짜와 시간 정보를 가져오는
   * getter 메서드와 날짜와 시간 정보를 설정할 수 있는 setter 메서드가 있다.
   *
   * getDay()는 요일 정보를 일(0) ~ 토(6)까지 반환한다.
   **/
  switch(date.getDay()) {
    case 0:
      week = "일요일";
      break;
    case 1:
      week = "월요일";
      break;
    case 2:
      week = "화요일";
      break;
    case 3:
      week = "수요일";
      break;
    case 4:
      week = "목요일";
      break;
    case 5:
      week = "금요일";
      break;
    case 6:
      week = "토요일";
      break;
  }
  console.log(date.getFullYear() + "년 " + (date.getMonth() + 1) + "월 "
      + date.getDate() + "일(" + week+ ")");
```

```
</script>
</head>
<body>
</body>
</html>
```

▶ 예제 6-8 배열관련 객체(Array)

- JavaScriptStudyCh06/javascript06_08.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>배열 내장객체(Array)</title>
<script>
  let arr = ["하나", "둘", "셋", "넷", "다섯"];
  let objArr = new Array("홍길동", 27, "남자");
  let numArr = new Array(3, 10, 5, 9, 7);

  console.log(arr + " - " + objArr);
  for(let i = 0; i < objArr.length; i++) {
    console.log(objArr[i]);
  }

  // objArr이 배열이므로 가변수 i에는 현재 인덱스가 저장된다.
  document.write("<h2>for in 반복문</h2>");
  for(let i in objArr) {
    document.write(objArr[i] + "<br>");
  }

  /* for in 반복문은 객체의 속성을 읽어오도록 객체에 최적화된 반복문으로
   * 간혹 배열 객체의 length와 같은 속성을 읽어올 수 있기 때문에 배열에는
   * 잘 맞지 않으며 배열에서는 다음과 같이 for of 반복문을 사용하는 것이 좋다.
   * 반복문이 실행되면서 가변수 num에는 배열 요소의 내용을 차례로 읽어온다.
   **/
  document.write("<h2>for of 반복문</h2>");
  for(let num of numArr) {
    document.write(num + "<br>");
  }

  // sort() 메서드를 이용해 numArr을 오름차순으로 정렬한다.
  console.log("정렬 전 : " + numArr);
  numArr.sort();
  console.log("정렬 후 : " + numArr);
```

```
// 배열의 첫 번째에 있는 데이터를 읽어오고 배열에서는 삭제한다.
//let first = arr.shift();
//console.log("first : " + first);
//console.log("shift : " + arr);

// 배열의 마지막에 있는 데이터를 읽어오고 배열에서는 삭제한다.
//let last = arr.pop()
//console.log("last : " + last);
//console.log("arr : " + arr);

// 배열의 첫 번째에 요소를 추가한다.
//arr.unshift("첫번째");
//console.log(arr);

// 배열의 마지막에 데이터를 추가한다.
//arr.push("마지막");
//console.log(arr);

// 두개의 배열을 결합해 하나의 배열로 반환하다.
//let newArr = arr.concat(objArr)
//console.log(newArr);

// 배열의 순서를 거꾸로 바꾼 후 반환한다.
//console.log(newArr.reverse());

// 현재 배열의 요소를 지정한 문자로 연결해 하나의 문자열로 반환한다.
//console.log(newArr.join("-"));

/* splice() 메서드 활용한 요소의 삭제, 수정, 추가하기
 * splice(startIndex, deleteCount, item)
 **/
//console.log(arr);

// arr 배열의 2번 인덱스부터 모든 요소를 삭제
//arr.splice(2)
//console.log(arr);

// arr 배열의 2번 인덱스부터 1개만 삭제
//arr.splice(2, 1)
//console.log(arr);

// arr 배열의 2번 인덱스부터 4, 5, 6 3개의 요소를 중간에 추가
//arr.splice(2, 0, 4, 5, 6);
//console.log(arr);
```

```
  // arr 배열의 2번 인덱스부터 2개를 치환
  //arr.splice(2, 2, "three", "four");
  //console.log(arr)
</script>
</head>
<body>
</body>
</html>
```

▶ 예제 6-9 문자열 객체(String)

- JavaScriptStudyCh06/javascript06_09.html

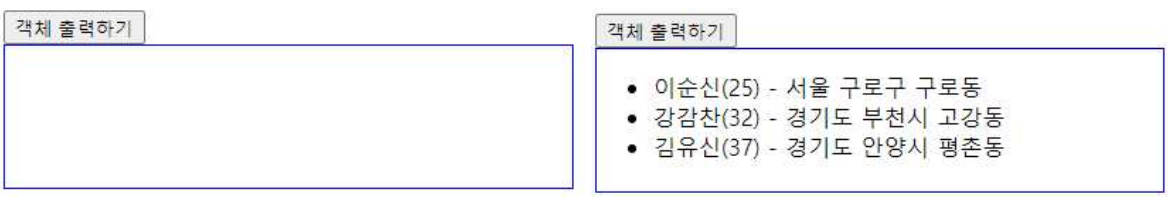
```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>문자열 내장객체(String)</title>
<script>
  let str = "     Hello JavaScript     ";
  let kStr = "안녕 자바스크립트";
  let path = "http://www.localhost:8080/JavaScriptStudy/javascript.html";

  /* length 속성은 문자열의 개수를 반환하고
   * trim() 메서드는 좌우측 끝의 공백을 제거한 문자열을 반환한다.
   **/
  console.log(str.length + ", " + str.trim().length);

  /* lastIndexOf() 메서드는 문자열에서 인수로 지정한 문자열을
   * 뒤에서부터 찾아서 그 위치를 index로 반환한다.
   * indexOf() 메서드는 지정한 문자열을 앞에서부터 찾아 그 위치를 index로 반환한다.
   **/
  let index = path.lastIndexOf("/");

  /* substring() 메서드는 문자열에서 인수로 지정한 구간의 문자열을 추출해서
   * 반환한다. substring(10) : index 10부터 끝까지 추출해 반환한다.
   * substring(10, 15) : index 10부터 15 이전까지 추출해 반환한다.
   **/
  let fileName = path.substring(index + 1);
  console.log(fileName);

  /* substr() 메서드는 문자열에서 첫 번째 인수로 지정한
   * index부터 두 번째 인수로 지정한 개수를 추출해 반환한다.
   **/
```

```
  console.log(fileName.substr(4, 6));

  // charAt() 메서드는 문자열에서 지정한 index에 위치한 문자를 반환한다.
  console.log(kStr.charAt(3));

  // split() 메서드는 문자열을 지정한 문자를 기준으로 나눠서 배열로 반환한다.
  console.log(path.substring(7).split("/"));

  /* toUpperCase() 메서드는 문자열을 모두 대문자로 변경해 반환한다.
   * toLowerCase() 메서드는 문자열을 모두 소문자로 변경해 반환한다.
   **/
  console.log(path.toUpperCase());
  console.log(path.toLowerCase());

  /* replace() 메서드는 문자열에서 첫 번째 인수로 지정한
   * 문자열을 두 번째 지정한 문자열로 치환하여 반환한다.
   **/
  console.log(kStr.replace("안녕", "Hi"));
</script>
</head>
<body>
</body>
</html>
```

▶ [연습문제 6-1] 객체 정의하고 출력하기

생성자 함수 방식으로 Person 객체를 정의하고 다음 요구사항에 따라서 웹페이지에 출력하시오.

1. Person 객체는 이름, 나이, 주소를 저장할 수 있는 속성을 가지고 있다.
2. Person 객체의 현재 상태를 문자열로 출력해 주는 toString 메서드를 가지고 있다.
3. 문서가 실행되면 아래 그림에서 좌측 그림과 같이 실행되도록 화면을 구성하시오
4. 객체 출력하기 버튼이 클릭되면 Person 객체를 생성하여 Array 객체에 저장한 후에 아래 그림 우측과 같이 출력되도록 프로그램을 작성하시오.

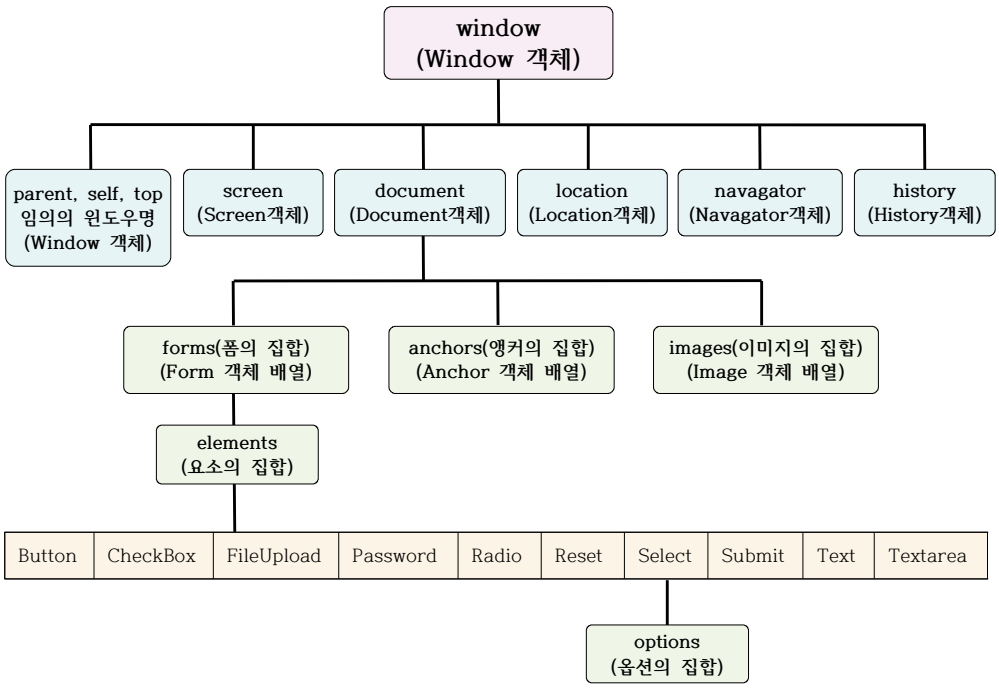


7. 브라우저 객체 모델(Browser Object Model)

브라우저 객체 모델(BOM - Browser Object Model)은 웹 브라우저에서 사용할 수 있는 모든 객체를 의미하며 아래 그림과 같이 웹 브라우저에서 사용되는 객체를 계층 구조로 만들어 놓은 것이 바로 BOM 이다. 이 BOM의 최상위 객체는 window 객체이며 모든 브라우저 객체는 최상위 객체인 window 객체에 포함되어 있다. 그러므로 window 객체를 통해 그 안에 포함된 나머지 객체에 접근할 수 있다. window 객체는 브라우저를 의미하며 브라우저 안에서는 어디에서든 접근할 수 있는 전역객체(global object) 이다.

window 객체 참고 : https://developer.mozilla.org/ko/docs/Web/API/Window

브라우저 객체 모델의 주요 객체



위의 그림에서 소문자로 표기한 것은 브라우저 객체 모델에서 각각의 객체를 참조하는 프로퍼티(Property) 이름이며 괄호 안에 표기한 영문 대문자로 시작하는 이름이 실제 객체 이름이지만 자바스크립트에서 브라우저 객체에 접근할 때는 실제 객체 이름은 사용하지 않고 각 객체를 참조하고 있는 소문자로 된 프로퍼티로 접근하기 때문에 일반적으로 소문자로 표기한 프로퍼티 이름을 해당 객체라고 사용하는 경우가 많다.

모든 브라우저 객체는 window 객체에 포함되어 있는 하위 객체이기 때문에 원칙적으로 window 객체를 통해서 접근해야 하지만 최상위 객체인 window 객체는 생략이 가능하기 때문에 하위 객체로 바로 접근하는 방법이 가장 일반적으로 사용되고 있다.

아래 코드와 같이 각 객체를 참조하고 있는 프로퍼티를 사용해 접근해야 하며 최상위 객체인 window 객체는 생략할 수 있다.

```
window.document.write("Hello JavaScript");
document.write("Hello JavaScript");
```

▶ 예제 7-1 window 객체의 속성

- JavaScriptStudyCh07/javascript07_01.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>window 객체의 속성</title>
<script>
  function showProperty() {
    let result = document.getElementById("result");
    let str = "<h2>window 객체의 속성</h2>";
    str += "윈도우 이름 : " + window.name + "<br/>"
    str += "윈도우가 닫혔는지 여부 : " + window.closed + "<br/>"
    str += "상태 표시줄의 문자열 : " + window.status + "<br/>"
    str += "윈도우 내용 영역의 폭과 너비 : "
        + window.innerWidth + "px, " + window.innerHeight + "px<br/>";
    str += "툴바와 스크롤바를 포함한 윈도우의 폭과 너비 : "
        + window.outerWidth + "px, " + window.outerHeight + "px<br/>";
    str += "화면에 대한 문서의 좌표 : "
        + window.screenLeft + "px, " + window.screenTop + "px<br/>";
    str += "화면에 대한 윈도우 좌표 : "
      + window.screenX + "px, " + window.screenY + "px<br/>";
    str += "스크롤된 좌표 : "
      + window.pageXOffset + "px, " + window.pageYOffset + "px<br/>";
    str += "현재 윈도우를 연 윈도우 : " + window.opener + "<br/>";
    str += "현재 윈도우의 부모 : " + window.parent + "<br/>";
    str += "현재 윈도우 객체 : " + window.self + "<br/>";
    str += "최상위 브라우저  : " + window.top + "<br/>";
    result.innerHTML = str;
  }

</script>
</head>
<body>
<input type="button" onclick="showProperty();"
    value="window 객체의 속성 보기" /><br/>
<div id="result"></div>
</body>
```

```
</html>
```

▶ 예제 7-2 새 창 띄우기

window.open(URL, name, specs, replace)		
파라미터	**매 개 변 수 설 명**	
URL	새로운 창에 표시할 페이지의 URL(필수 속성) 생략시 about:blank라는 빈 화면이 나타난다.	
name	새 창의 이름이나 target을 지정하는 속성(선택 속성) _blank : URL에 지정한 페이지가 새 창으로 표시된다.(default) _parent : 부모 창에 URL에 지정한 페이지가 표시된다. _self : 현재 창에 URL에 지정한 페이지가 표시된다. _name : 새 창의 이름	
specs	콤마로(,)로 구분하여 여러 속성을 지정할 수 있다. 브라우저에 따라 지원하는 속성이 약간씩 다르다.(선택 속성)	
	height=pixels 값	창의 높이를 설정. 최솟값은 100
	width=pxels 값	장의 너비를 설정. 최솟값은 100
	top=pxels 값	모니터 화면에서 윈도우 창 상단의 위치를 설정
	left=pixels 값	모니터 화면에서 윈도우 창 좌측 위치를 설정
	scrollbar=yes|no|1|0	스크롤바를 표시할지 여부 설정. default=yes
	fullscreen=yes|no|1|0	전체화면 모드로 열기 설정. default=no. 윈도우를 닫으려면 alt+F4 키를 사용한다. (IE에서만 작동함)
	location=yes|no|1|0	주소 필드를 표시할지 여부 설정 default=yes, (Opera에서만 작동)
	menubar=yes|no|1|0	메뉴 바를 표시할지 여부 설정. default=yes
	resizable=yes|no|1|0	창 크기 조절 여부 설정 default=yes (Chrome에서는 작동 안함)
	status=yes|no|1|0	상태 바를 표시할지 여부 설정. default=yes
	titlebar=yes|no|1|0	타이틀 바를 표시할지 여부 설정. default=yes
	toolbar=yes|no|1|0	브라우저의 툴바를 표시할지 여부 설정 default=yes, (IE, FireFox에서만 작동)
replace	history의 현재 주소를 어떻게 관리할지 여부 설정(선택 속성) true : history 목록에서 현재 주소를 대체 false : history 목록에 새로운 주소를 저장	

- JavaScriptStudyCh07/javascript07_02.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>새 창 띄우기 - Main Window</title>
```

```
<script>
  let subwin;

  function subOpen() {
    /* subwin에 자식 창의 참조가 저장된다.
     * 부모 창에서 subwin 변수를 이용해 자식 창에 접근할 수 있다.
     **/
    subwin = window.open('javascript07_0201.html', 'sub',
          'width=600, height=300, left=300, top=300');
  }

  function subClose() {
    // 아래와 같이 subwin 변수를 이용해 자식 창을 닫을 수 있다.
    subwin.close();
  }

  function parentClose() {
    /* 처음에 실행되었을 때 아래 명령을 수행하면 크롬과 Edge 모두 정상적으로
     * 브라우저 창이 잘 닫히는데 브라우저를 실행한 후에 주소를 입력하여 현재
     * 페이지에 접속하게 되면 창이 닫히지 않는다. 예전에는 close() 함수가 잘
     * 동작했으나 현재에는 보안상의 이유로 링크로 열린 창이거나 window.open()
     * 함수로 열린 창이 아니라면 window.close() 함수가 제대로 동작하지 않는다.
     **/
    self.close();
    // window.close();
    // this.close();
  }
</script>
</head>
<body>
<body>
  <form name="parentForm">
    주소 : <input type="text" name="address" readonly>
  </form><br>
  <input type="button" value="서브 윈도우 열기" onclick="subOpen()" />
  <input type="button" value="서브 윈도우 닫기" onclick="subClose()" />
  <input type="button" value="부모 윈도우 닫기" onclick="parentClose()" />
</body>
</body>
</html>
```

- javascript07_0201.html

```
<!DOCTYPE html>
```

```
<html>
<head>
<meta charset="UTF-8">
<title>새 창 띄우기 - SubWindow</title>
<script>
  /* window 객체의 onload 속성에 이벤트를 처리할 이벤트 핸들러 함수를 아래와
   * 같이 등록하면 load 이벤트가 발생했을 때 자바스크립트 시스템이 핸들러 함수를
   * 호출해 준다. load 이벤트는 현재 html 문서의 모든 구성 요소 즉 문서 객체,
   * 외부 연결파일(css, javascript, 이미지) 등이 모두 메모리에 로드되었을 때
   * 발생하는 이벤트 이다.
   **/
  window.onload = function() {

    // html 문서에서 id가 btnInput인 문서 객체를 구한다.
    let btnInput = document.querySelector("#btnInput");

    /* 표준 이벤트 모델을 이용해 이벤트 처리함수 등록
     * 입력하기 버튼이 클릭되었을 때 처리할 이벤트 핸들러 함수를 등록
     **/
    btnInput.addEventListener("click", function() {
      // 주소 입력상자에 입력된 값을 읽어온다.
      let address = document.querySelector("#address").value;

      // 현재 창을 열어준 부모 창 폼의 주소 입력 상자에 주소를 설정한다.
      opener.parentForm.address.value = address;

      // 현재 창을 닫는다.
      self.close();
    });
  }
</script>
</head>
<body>
  주소 입력 : <input type="text" name="address" id="address">
  <input type="button" id="btnInput" value="입력하기">
</body>
</html>
```

▶ 예제 7-3 window 객체의 메소드

- JavaScriptStudyCh07/javascript07_03.html

```
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="UTF-8">
<title>window 객체의 메소드</title>
<style>
  div {
    width: 600px;
  }
  .div1 {
    height: 300px;
    background-color: red;
    border: 1px solid black;
  }
  .div2 {
    height: 300px;
    background: green;
    border: 1px solid black;
  }
  .div3 {
    height: 300px;
    background: blue;
  }
  #scrollBtn {
    position: relative;
  }
</style>
<script>
  function winScrollBy(xnum, ynum, elem) {
    // 문서 상단에서 윈도우 스크롤(상대위치) 버튼까지의 거리를 구한다.
    let btnTop = elem.style.top;

    /* 윈도우 스크롤(상대위치) 버튼의 Y축 위치는 단위를 포함하여(예 : 100px)
     * 반환되기 때문에 String 객체의 substring() 메소드와 indexOf() 메소드를
     * 사용해 단위를 제외한 숫자 형태의 데이터만 추출하고 매개변수로 넘어온
     * ynum과 덧셈하여 다시 px 단위를 연결한 데이터를 만들었다.
     **/
    elem.style.top = (Number(btnTop.substring(0, btnTop.indexOf("p")))
        + Number(ynum)) + "px";

    // 문서를 현재 위치에서 xnum, ynum 만큼 스크롤 시킨다.
    scrollBy(xnum, ynum);
  }
</script>
</head>
<body>
  <!-- window.print() 메서드로 운영체제의 인쇄 대화상자를 띄운다. -->
  <input type="button" onclick="window.print();" value="프린트하기" /><br/>
```

```
  <!-- window.scrollTo() 메서드로 문서를 x=0px, y=200px 위치로 스크롤 시킨다. -->
  <input type="button" onclick="window.scrollTo(0, 200);"
    value="윈도우 스크롤(절대위치)" /><br/>
  <input type="button" onclick="winScrollBy(0, 100, this);"
    value="윈도우 스크롤(상대위치)" id="scrollBtn"/><br/>
  <div class="div1"> </div>
  <div class="div2"></div>
  <div class="div3"></div>
  <div class="div1"></div>
  <div class="div2"></div>
  <div class="div3"></div>
</body>
</html>
```

▶ 예제 7-4 타이머로 시계 만들기

- JavaScriptStudyCh07/javascript07_04.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>타이머로 시계 만들기</title>
<script>
  let timer;
  let clockStart = function() {

    /* window는 브라우저 객체 모델에서 최상위 객체로 생략이 가능하다.
     * setInterval(함수, 밀리초) 메서드는 첫 번째 인수로 지정한 함수를 두 번째
     * 인수로 지정한 시간 간격으로 반복해서 호출하는 메서드 이다. 다시 말해 지정한
     * 시간 간격으로 함수를 반복해서 실행해 주는 타이머라고 할 수 있다.
     * setInterval() 메서드를 취소하려면 clearInterval(참조변수)로 호출하면 된다.
     * 아래의 setInterval() 메서드를 취소하려면 clearInterval(timer)로 호출하면 된다.
     **/
    timer = window.setInterval(function() {

          /* 현재 날짜와 시간 정보를 출력하기 위해 1초 단위로
           * Date 객체를 생성해서 id가 result인 요소에 출력한다.
           **/
          let date = new Date();
          document.getElementById('result').innerHTML =
                  date.toLocaleTimeString();
        }, 1000);
  }
```

```
  // html 문서의 구성요소가 모두 로딩 되면 이벤트를 처리할 이벤트 핸들러 등록
  window.onload = function() {
    document.getElementById('result').innerText =
      "3초 후에 시계가 자동 시작됩니다.";

    /* setTimeout(함수, 밀리초) 메서드는 첫 번째 인수로 지정한 함수를 두 번째 인수로
     * 지정한 시간 후에 한 번 호출해 주는 메서드 이다. 다시 말해 지정한 시간이 지나면
     * 지정한 함수를 단 한 번 실행해 주는 타이머라 할 수 있다.
     * setTimeout() 메서드를 취소하려면 clearTimeout(참조변수)로 호출하면 된다.
     * 아래의 setTimeout() 메서드를 취소하려면 clearTimeout(timeout)으로 호출하면 된다.
     **/
    let timeout = window.setTimeout(function() {
      clockStart();
    }, 3000);
  };
</script>
</head>
<body>
  <!-- 타이머를 정지하려면 clearInterval(참조변수)로 호출하면 된다. -->
  <input type="button" value="타이머 정지" onclick="clearInterval(timer)" />
  <div id="result"></div>
</body>
</html>
```

▶ 예제 7-5 screen 객체를 이용해 모니터 정보 다루기

screen 객체는 사용자 모니터 정보를 제공하는 객체로 모니터의 너비나 높이 또는 모니터의 표현 가능한 컬러 bit 정보를 반환하는 속성을 제공한다.

- JavaScriptStudyCh07/javascript07_05.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>screen 객체를 이용해 모니터 정보 다루기</title>
<script>
  /* width : 화면의 전체 너비를 반환
   * height: 화면의 전체 높이를 반환
   * availWidth : 작업 표시줄을 제외한 화면의 너비를 반환
   * availHeight : 작업 표시줄을 제외한 화면의 높이를 반환
   * colorDepth : 이미지를 표시하기 위한 색상 팔레트의 컬러 bit를 반환
   **/
  document.write("전체 화면 너비 : " + screen.width +
      ", 높이 : " + screen.height + "<br/>");
```

```
  document.write("작업 표시줄 제외 화면 너비 : " + screen.availWidth
      + ", 높이 : " + screen.availHeight + "<br/>");
  document.write("컬러 해상도 : " + screen.colorDepth + "bit");
</script>
</head>
<body>
</body>
</html>
```

▶ 예제 7-6 location 객체를 이용해 페이지 주소(URL) 다루기

location 객체는 웹 브라우저의 현재 주소(URL)를 저장하고 있는 객체로 웹 브라우저의 주소를 지정하거나 새로 고침 할 수 있는 속성과 메서드를 제공하고 있다.

- JavaScriptStudyCh07/javascript07_06.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>location 객체를 이용해 페이지 주소(URL) 다루기</title>
<script>
  /* location 객체는 웹 브라우저의 주소 표시줄과 관련된 속성과 메서드를 제공하는 객체이다.
   * 웹 브라우저의 현재 주소 즉 URL에 대한 정보와 새로 고침(reload) 메서드를 제공한다.
   **/
  function locationMove() {
    /* location 객체의 href 속성으로 브라우저의 URL을 읽어오거나 설정할 수 있다.
     * 아래와 같이 href 속성을 설정하면 브라우저의 주소 표시줄(URL)이 변경되기
     * 때문에 지정한 URL로 이동하게 된다.
     **/
    location.href="http://www.naver.com";
    //console.log(location.href);
  }

  function locationReload() {
    // reload() 메서드는 현재 페이지를 다시 로드(새로 고침, F5) 한다.
    location.reload()
    alert('현재 페이지를 새로 고침 했습니다.');
  }

  // html 문서의 구성 요소가 모두 로딩 되면 이벤트를 처리할 이벤트 핸들러 등록
  window.onload = function() {

    // 현재 URL를 읽어온다.
    let href = location.href;
```

```
    // 현재 URL에서 프로토콜, 호스트네임, 포트 번호를 읽어온다.
    let origin = location.origin;

    // 현재 URL에서 경로 이름을 읽어온다.
    let pathname = location.pathname;

    // 현재 URL에서 프로토콜 정보를 읽어온다.
    let protocol = location.protocol;

    // 현재 URL에서 호스트 이름과 포트 번호를 읽어온다.
    let host = location.host;

    // 현재 URL에서 호스트 이름을 읽어온다.
    let hostname = location.hostname;

    // 현재 URL에서 포트 번호를 읽어온다.
    let port = location.port;

    // 현재 URL에서 쿼리 스트링을 읽어온다.
    let search = location.search;

    let result = "<h2>location 객체 정보</h2>";
    result += "<ul>";
    result += "    <li>href : " + href + "</li>";
    result += "    <li>origin : " + origin + "</li>";
    result += "    <li>pathname " + pathname + "</li>";
    result += "    <li>protocol : " + protocol + "</li>";
    result += "    <li>host : " + host + "</li>";
    result += "    <li>hostname : " + hostname + "</li>";
    result += "    <li>port  :" + port + "</li>";
    result += "    <li>search : " + search + "</li>";
    result += "    <li>" + (search.length == 0 ? "쿼리 스트링 없음" : "") + "</li>";
    result += "</ul>";

    /* 문서에서 id가 locationStatus인 문서객체를 선택하고
     * 이 요소의 내용을 result의 내용으로 변경한다.
     **/
    let div = document.querySelector("#locationStatus");
    div.innerHTML = result;
  }
</script>
</head>
<body>
  <div id="locationStatus"></div>
```

```
  <input type="button" value="네이버 가기" onclick="locationMove();" />
  <input type="button" value="reload" onclick="locationReload();"/>
</body>
</html>
```

▶ 예제 7-7 navigator 객체를 이용해 브라우저 정보 접근하기

navigator 객체는 사용자의 웹 브라우저 정보와 운영체제 정보를 저장하고 있는 객체이다.

- JavaScriptStudyCh07/javascript07_07.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>navigator 객체를 이용해 사용자 브라우저 정보 접근하기</title>
<script>
  //문서 객체가 모두 로딩 되면 이벤트를 처리할 이벤트 핸들러 등록
  window.onload = function() {
    // 사용자 웹 브라우저와 운영체제의 종합 정보를 반환한다.
    let userAgent = navigator.userAgent;

    // 웹 브라우저의 코드명을 반환한다. 현재 모든 브라우저는 Mozilla를 반환한다.
    let appCodeName = navigator.appCodeName;

    // 웹 브라우저의 이름을 반환한다. 현재 모든 브라우저는 Netscape를 반환한다.
    let appName = navigator.appName;

    // 웹 브라우저의 버전 정보를 반환한다. 현재 모든 브라우저는 5.0(Windows)를 반환한다.
    let appVersion = navigator.appVersion;

    // 웹 브라우저가 사용하는 언어 정보를 반환한다. 한국어인 경우 ko-KR을 반환한다.
    let language = navigator.language;

    // 웹 브라우저 엔진 이름을 반환 한다. 크롬 Gecko
    let product = navigator.product;

    /* 사용자 컴퓨터의 운영체제의 bit 정보를 반환한다. 윈도우 64bit를
     * 사용하더라도 브라우저가 32bit를 사용한다면 Win32가 반환된다.
     */
    let platform = navigator.platform;

    // 온라인 상태 여부를 반환한다. 인터넷이 연결되어 있는 상태는 true를 반환한다.
    let onLine = navigator.onLine;
```

```
    let result = ""
    result += "<ul>";
    result += "    <li>userAgent : " + userAgent + "</li>";
    result += "    <li>appCodeName : " + appCodeName + "</li>";
    result += "    <li>appName : " + appName + "</li>";
    result += "    <li>appVersion : " + appVersion + "</li>";
    result += "    <li>language : " + language + "</li>";
    result += "    <li>product : " + product + "</li>";
    result += "    <li>platform : " + platform + "</li>";
    result += "    <li>onLine : " + onLine + "</li>";
    result += "</ul>";

    /* 문서에서 id가 result인 문서객체를 선택하고
     * 이 요소의 내용을 result의 내용으로 변경한다.
     **/
    let div = document.querySelector("#result");
    div.innerHTML = result;
  }
</script>
</head>
<body>
  <h2>사용자 웹 브라우저 정보</h2>
  <div id="result"></div>
</body>
</html>
```

▶ 예제 7-8 history 객체를 이용해 방문한 사이트 정보 다루기

history 객체는 사용자가 방문했던 웹 페이지 목록을 저장하고 있는 객체이다. 목록의 수를 반환하는 length 속성을 제공하고 방문했던 페이지 목록을 앞뒤로 이동해가면서 접근할 수 있는 메서드를 제공하고 있다.

- JavaScriptStudyCh07/javascript07_08.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>history 객체를 이용해 방문한 사이트 정보 다루기</title>
<script>
  function goBack() {

    /* history 객체의 go() 메서드의 인수에 음수를 지정하면 이전 방문
     * 사이트로 이동하고 양수를 지정하면 다음 방문 사이트로 이동한다.
     * history 객체의 length 속성은 방문 사이트 목록에 저장된 개수를 반환한다.
```

```
     */
    history.go(-(history.length - 1));

    /* 문서에서 id가 cnt인 문서객체를 선택하고 이 요소의 내용에
     * 현재 history 객체에 저장된 사이트 목록의 개수를 출력한다.
     **/
    let cnt = document.querySelector("#historyCnt");
    cnt.innerHTML = history.length;
  }

  function anchorMove(elem, page) {
    elem.href = page;
    elem.target = "subFrame"

    /* 문서에서 id가 cnt인 문서객체를 선택하고 이 요소의 내용에
     * 현재 history 객체에 저장된 사이트 목록의 개수를 출력한다.
     **/
    let cnt = document.querySelector("#historyCnt");
    cnt.innerHTML = history.length;

    // 다음과 같이 iframe의 내용을 변경할 수 있다.
    //document.getElementById("content").contentWindow
    //  .document.body.innerHTML = "main : " + history.length;
  }
</script>
</head>
<body>
  <a href="#" onclick="anchorMove(this, 'javascript07_01.html');">1페이지</a><br />
  <a href="#" onclick="anchorMove(this, 'javascript07_02.html');">2페이지</a><br />
  <a href="#" onclick="anchorMove(this, 'javascript07_03.html');">3페이지</a><br />
  <!--
    iframe 요소는 별도의 웹 문서를 이미지나 테이블과 같은 개체처럼 현재 문서의
    특정 위치에 삽입할 수 있는 기능을 제공한다. 아래와 같이 a 요소에 링크를 설정하고
    target 속성에 iframe 요소의 name 속성에 지정한 이름을 지정하면 된다.
    iframe 요소를 아이프레임(iframe) 또는 인라인 프레임(Inline frame)이라고 부른다.

    <a href="iframe에 출력할 html문서" target="subFrame">이동하기</a>
  -->
  <iframe name="subFrame" src="about:blank" width="400" height="200"
    id="content"></iframe><br/>
  <button type="button" onclick="goBack();">처음으로</button>   
  <!--
    history 객체의 back() 메서드는 방문 사이트 목록에서 현재 페이지의
    바로 이전 페이지로 이동하며 forward() 메서드는 현재 페이지의 바로
    이후에 방문한 페이지로 이동한다.
```

```
    -->
    <button type="button" onclick="history.back();">이전으로</button>   
    <button type="button" onclick="history.forward();">다음으로</button><br/>
    <div id="historyCnt"></div>
</body>
</html>
```

8. 문서 객체 모델(Document Object Model)

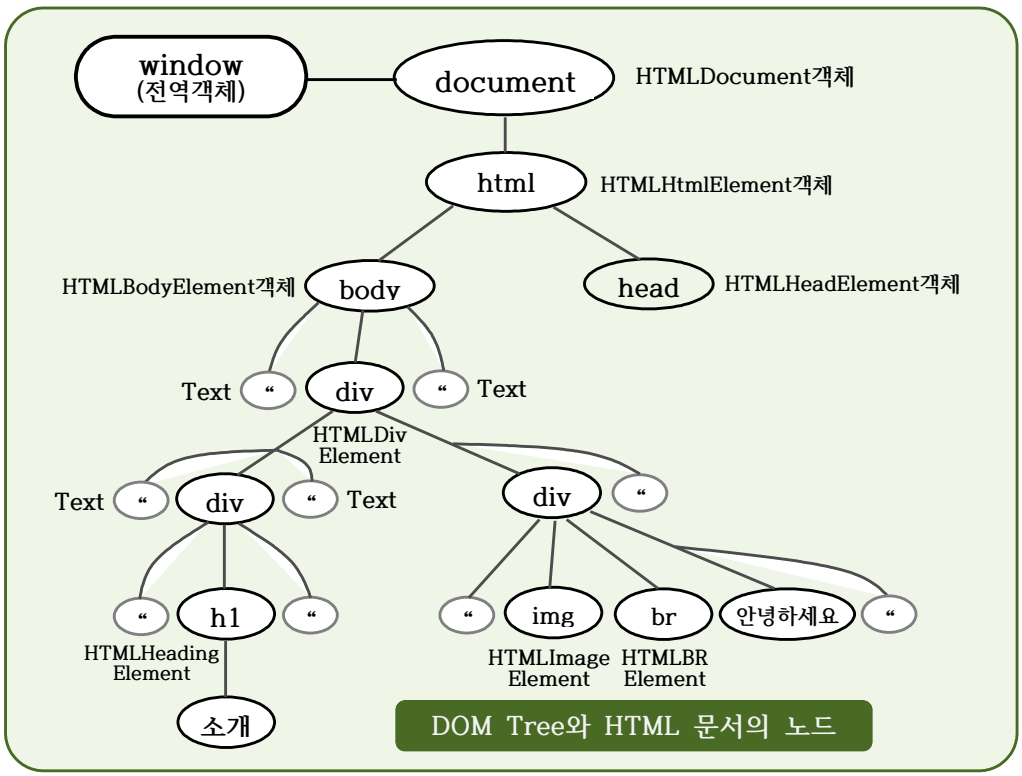
브라우저 객체 모델(BOM)은 브라우저 제공 업체들이 독자적으로 구현한 객체 모델이라 각 브라우저마다 지원하는 기능이 서로 다를 수 있다. 하지만 문서 객체 모델(DOM, Document Object Model)은 HTML이나 XML과 같은 마크업 언어로 작성된 문서에 접근해 동적으로 문서를 조작하기 위한 표준적인 프로그래밍 인터페이스라고 할 수 있다. DOM은 문서의 구조화된 표현을 제공하며 프로그래밍 언어가 DOM 구조에 접근할 수 있는 방법을 제공하여 문서 구조, 문서의 내용, 스타일 등을 추가, 수정, 삭제할 수 있도록 지원한다.

DOM이라는 이름과 같이 HTML 문서에서 요소(태그), 속성, 텍스트 등의 모든 구성 요소를 객체로 다루고 있으며 HTML 문서의 계층 구조를 그대로 객체화했기 때문에 DOM의 구조 또한 document 객체를 최상위로 해서 똑같이 계층 구조를 이루고 있으며 이런 DOM의 계층 구조를 나무에 비유해서 DOM Tree라고 부른다.

DOM에서 사용하는 객체를 노드(Node)라고 하고 상 하위 노드 간의 관계를 부모노드와 자식노드라고 하며 같은 위치의 노드는 형제노드라고 부른다. 노드에는 12가지 노드 타입(Node Type)이 있지만 잘 쓰이지 않는 것도 있으며 요소노드, 속성노드, 텍스트노드 정도만 기억하면 된다.

요소노드(element node) : <a>, , 등과 같은 태그를 의미
속성노드(attribute node) : id, name 등과 같은 태그 내의 속성을 의미
텍스트노드(text node) : 요소 안의 내용을 의미

DOM 참고 : https://developer.mozilla.org/ko/docs/Web/API/Document_Object_Model
DOM Tree 참고 : https://dom.spec.whatwg.org/



* Document Object Model의 상속구조 및 주요기능

	Node	Document	HTMLDocument	Element	HTMLElement
상속 구조	Node	Node -> Document	Node -> Document -> HTMLDocument	Node -> Element	Node -> Element -> HTMLElement
기능	DOM의 최상위 객체 노드를 탐색하고 조작하기 위한 기본적인 프로퍼티와 메서드 제공 xml, html 파일 DOM 컨트롤 가능 - 노드타입을 알 수 있는 속성 - 부모노드에 접근기능 - 형제노드에 접근기능 - 자식노드에 접근기능 - 자식노드를 추가, 삭제, 교체하는 기능	노드의 한 종류로 HTML과 XML 문서의 루트 객체로서 엘리먼트 노드와 이벤트, 속성 노드, 텍스트 노드, 주석노드까지 생성하는 팩토리 메서드 제공 xml, html 파일 DOM 컨트롤 가능 - 노드 생성 기능 - 이벤트 생성 기능 - id, tagName으로 특정 노드 찾는 기능 - 이벤트 모델 기능	HTML 태그 전용 속성과 메서드 제공 - body, image 속성 - write(), open() - getElementById()	노드의 한 종류로 HTML과 XML 문서의 태그를 조작하기 위한 기본적인 속성과 메서드를 제공 속성을 다루는 기능 이벤트 모델 구현 xml, html 파일 DOM 컨트롤 가능	HTML 태그 전용 속성과 메서드 제공
주요 속성	Attr[] attributes NodeList childNodes Node firstChild Node lastChild Node nextSibling Node previousSibling Node parentNode String textContent String nodeValue String nodeName unsignedshort nodeType	Element documentElement Element firstElementChild Element lastElementChild long childElementCount	HTMLElement body String cookie HTMLCollention images HTMLCollection links	String className String id DOMTokenList classList HTMLCollection children Element firstElementChild Element lastElementChild Element parentElement Element previousElementSibling Element nextElementSibling	String innerText String innerHTML CSS2Properties style int offsetWidth int offsetHeight int offsetLeft int offsetTop
주요 메소드	boolean hasAttributes() boolean hasChildNodes() Node cloneNode() Node appendChild() Node insertBefore() Node removeChild() Node replaceChild()	Element getElementById() Element querySelector() NodeList querySelectorAll() Element createElement() prepend() append() replaceChildren()	close() open() write()	HTMLCollection getElementsByTagName() HTMLCollection getElementsByClassName() before() after() remove() String getAttribute() setAttribute() removeAttribute() boolean toggleAttribute() boolean hasAttribute()	

DOM을 활용하면 자바스크립트를 이용해 DOM에서 요소를 선택하여 읽어오거나 DOM에서 요소를 추가, 수정 삭제할 수도 있다. 이렇게 DOM에서 문서 객체를 자유롭게 제어하려면 원하는 요소를 잘 선택할 수 있어야 한다. DOM에서 원하는 요소를 선택하는 방법은 매우 다양하며 원하는 요소를 읽어와 새로운 요소를 추가하고 수정, 삭제하고 DOM을 제어하는 방법에 대해서 알아보자.

▶ 예제 8-1) DOM에서 다양한 방법으로 문서 객체 선택하기

- JavaScriptStudyCh08/javascript08_01.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>DOM에서 다양한 방법으로 문서 객체 선택하기</title>
<script>
  // id, children, TagName, ClassName, CSS Selector를 이용한 요소 선택
  // HTML 문서의 모든 요소가 메모리에 준비되면
  window.onload = function() {

    // DOM에서 id가 plan인 요소를 찾아 HTMLElement 객체로 읽어온다.
    const plan = document.getElementById("plan");

    // 다음은 plan요소의 자식 요소들을 HTMLCollection 객체로 읽어온다.
    console.log(plan.children);

    // DOM에서 모든 li 요소를 찾아 HTMLCollection 객체로 읽어한다.
    const liTags = document.getElementsByTagName('li');
    console.log(liTags);

    /* HTMLCollection 객체는 유사 배열(Array-Like Object)로 배열과 유사한
     * 자료구조를 가진 객체이다. 배열이 아니라서 배열이 제공하는 메서드를
     * 사용할 순 없지만 숫자 형식의 인덱싱(indexing)이 가능하고 length
     * 프로퍼티를 사용할 수 있다.
     **/
    console.log(liTags[3], " - ",  liTags.length);
    for(li of liTags) {
      console.log(li);
    }

    // DOM에서 class가 done-list인 요소를 찾아 HTMLCollection 객체로 읽어온다.
    const doneList = document.getElementsByClassName('done-list');
    console.log(doneList);

    // DOM에서 CSS 선택자로 class가 plan-list인 첫 번째 요소를 Element 객체로 읽어온다.
    const content = document.querySelector('.plan-list');
```

```
    console.log(content);

    // DOM에서 CSS 선택자로 class가 plan-list인 모든 요소를 NodeList 객체로 읽어온다.
    const contentAll = document.querySelectorAll('.plan-list');
    console.log(contentAll);

    // NodeList 객체도 유사 배열이지만 배열처럼 forEach 함수를 사용할 수 있다.
    contentAll.forEach((value, i) => {
      console.log(i, value);
    });
  }
</script>
</head>
<body>
  <div id="container">
    <div class="header">
      <h1>DOM Tree</h1>
    </div>
    <div class="content">
      <!-- 앞으로 스터디 할 목록 -->
      <h2>프로그래밍 스터디 계획</h2>
      <ul id="plan">
        <li class="plan-list">JavaScript 함수</li>
        <li class="plan-list">JavaScript DOM</li>
        <li class="plan-list">JavaScript Event</li>
        <li class="plan-list">jQuery 함수 선택자</li>
        <li class="plan-list">jQuery Effect</li>
      </ul>
    </div>
    <div class="content">
      <!-- 완료된 스터디 목록 -->
      <h2>프로그래밍 스터디 완료</h2>
      <ul id="done">
        <li class="done-list">HTML 기본 태그</li>
        <li class="done-list">HTML5 시멘틱 태그</li>
        <li class="done-list">CSS3 선택자</li>
        <li class="done-list">CSS3 박스모델</li>
        <li class="done-list">JavaScript 제어문</li>
      </ul>
    </div>
  </div>
</body>
</html>
```

▶ 예제 8-2) DOM 트리 탐색하기

- JavaScriptStudyCh08/javascript08_02.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>DOM 트리 탐색하기</title>
<script>
  /* DOM 트리에서 부모, 자식, 형제간의 관계를 이용한 요소 노드 탐색
   * Element 객체의 children, parentEelement, previousElementSibling,
   * nextElementSibling, firstElementChild, lastElementChild 프로퍼티는
   * DOM 트리에서 요소 노드를 대상으로 탐색할 수 있는 프로퍼티들 이다.
   **/

  // HTML 문서의 모든 요소가 메모리에 준비되면
  window.onload = function() {

    // DOM에서 id가 container인 요소를 찾아 HTMLElement 객체로 읽어온다.
    const container = document.getElementById("container");

    /* container의 자식 요소 중 index 1번째 자식을 HTMLElement 객체로 읽어온다.
     * HTMLElement는 12가지 노드의 종류 중에서 요소 노드의 정보를 담은 객체이다.
     * container의 index 1번째 자식은 첫 번째 class="content" 요소이다.
     **/
    const content1 = container.children[1];
    console.log(content1);
    //console.dir(content1)

    /* content1의 바로 다음 형제 요소의 index 1번째 자식 요소의 모든 자식 요소를
     * 읽어온다. 즉 id='done'의 모든 자식 요소를 HTMLCollection 객체로 읽어온다.
     **/
    const doneList = content1.nextElementSibling.children[1].children;
    console.log(doneList);

    /* doneList의 index 0번 요소의 부모 요소의 첫 번째 자식 요소 선택
     * 'HTML 기본 태그'가 내용인 요소를 선택하여 HTMLElement 객체로 가져옴
     **/
    console.log(doneList[0].parentElement.firstElementChild)

    /* doneList의 index 0번 요소의 부모 요소의 마지막 자식 요소 선택
     * 'JavaScript 제어문'이 내용인 요소를 선택하여 HTMLElement 객체로 가져옴
     **/
    console.log(doneList[0].parentElement.lastElementChild)
```

```
    /* 다음에서 doneList[0]은 "HTML 기본 태그"가 내용으로 있는 li 요소이며 이 요소의
     * 부모 요소(id='done')의 부모 요소(class='content')를 선택하고 바로 앞의 형제
     * 요소(class='content')를 선택한다. 그리고 그 앞의 형제 요소(class='header')를
     * 선택하여 HTMLElement 객체로 읽어온다.
     **/
    const header = doneList[0].parentElement.parentElement
                          .previousElementSibling.previousElementSibling;
    console.log(header);

    /* childNodes 프로퍼티는 여러 유형의 Node 객체가 담기 NodeList가 반환된다.
     * 위에서 사용한 요소 노드만 담긴 HTMLCollection 객체와는 다르게 Node 객체가
     * 담긴 NodeList 객체는 DOM에 존재하는 모든 Node를 가져오기 때문에 가져온
     * 리스트의 목록에 접근하면 ElementNode(요소 노드)만 존재하는 것이 아니라
     * ElementNode를 비롯한 TextNode, CommentNode 등등 다양한 노드가 있다.
     * 실제로 대부분의 작업은 요소 노드를 대상으로 이루어지므로 이 예제 앞쪽의
     * 요소 노드에 접근하는 방법을 잘 숙지하면 된다. 많지는 않겠지만 특별한 경우
     * 요소 노드가 아니라 그 외의 노드 타입에 접근해 할 경우가 있을 수도 있으므로
     * 요소 노드외의 다양한 유형의 노드 타입이 있다는 것을 잘 기억해 두자.
     * 노드 타입 참고 : https://dom.spec.whatwg.org/#interface-node
     **/
    let nodeList = container.childNodes[3].childNodes
    console.log(nodeList);
    for(node of nodeList) {
      console.log(node);
    }
  }
</script>
</head>
<body>
  <div id="container">
    <div class="header">
      <h1>DOM Tree</h1>
    </div>
    <div class="content">
      <!-- 앞으로 스터디 할 목록 -->
      <h2>프로그래밍 스터디 계획</h2>
      <ul id="plan">
        <li class="plan-list">JavaScript 함수</li>
        <li class="plan-list">JavaScript DOM</li>
        <li class="plan-list">JavaScript Event</li>
        <li class="plan-list">jQuery 함수 선택자</li>
        <li class="plan-list">jQuery Effect</li>
      </ul>
    </div>
    <div class="content">
```

```
        <!-- 완료된 스터디 목록 -->
        <h2>프로그래밍 스터디 완료</h2>
        <ul id="done">
          <li class="done-list">HTML 기본 태그</li>
          <li class="done-list">HTML5 시멘틱 태그</li>
          <li class="done-list">CSS3 선택자</li>
          <li class="done-list">CSS3 박스모델</li>
          <li class="done-list">JavaScript 제어문</li>
        </ul>
      </div>
    </div>
</body>
</html>
```

▶ 예제 8-3) 요소 노드의 프로퍼티

- JavaScriptStudyCh08/javascript08_03.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>요소 노드의 프로퍼티</title>
<script>
  /* 요소 노드의 프로퍼티를 이용해 내용을 읽어오거나 변경하기
   * innerHTML, outerHTML, textContent, innterText 프로퍼티는
   * 요소의 내용을 읽어오거나 설정할 수 있는 프로퍼티들 이다.
   **/

  // HTML 문서의 모든 구성 요소가 메모리에 준비되면
  window.onload = function() {

    // class='content'인 요소 중에서 첫 번째 요소를 Element 객체로 읽어온다.
    let content = document.querySelector(".content");

    /* innerHTML 프로퍼티는 요소 안에 있는 모든 HTML을 문자열로 읽어오거나 요소
     * 안의 내용을 다른 HTML로 변경할 수 있는 프로퍼티이다. 이 프로퍼티는 요소
     * 안의 내용을 읽어올 때 공백과 들여쓰기를 포함해서 있는 그대로 읽어온다.
     **/
    console.log(content.innerHTML);
    content.innerHTML = `<h2>과일 리스트</h2>
                         <ul><li>수박</li><li>망고</li></ul>`;
    content = document.querySelector(".content");
    console.log(content.innerHTML);
```

```
    /* outerHTML 프로퍼티는 innerHTML과는 다르게 선택된 해당 요소를 포함해서
     * 전체 HTML을 문자열로 읽어오거나 해당 요소를 다른 HTML로 변경할 수 있다.
     **/
    console.log(content.outerHTML);
    content.outerHTML = `<div class='test'>
                            <h2>새로운 스터디 계획</h2>
                            <ul id='plan'>
                              <li>오라클 SQL 활용 </li>
                              <li>자바 기초 문법</li>
                            </ul>
                          </div>`;

    // innerHTML 프로퍼티를 이용해 기존의 내용 뒤에 새로운 내용도 추가할 수 있다.
    document.querySelector("#plan").innerHTML += "<li>자바 제어문 활용</li>";

    // 문서에서 id='done'인 요소를 선택
    let planList = document.querySelector("#done");

    /* innerText와 textContent 프로퍼티는 요소 안의 내용 중에서 HTML 코드를 제외한
     * 텍스트를 문자열로 읽어오거나 설정할 수 있다. 요소의 내용을 설정할 때 HTML
     * 코드가 포함되어도 문서에 적용되지 않고 일반 텍스트로 취급된다.
     **/
    console.log(planList.innerText);
    console.log(planList.textContent);

    // 태그를 포함해 요소의 내용을 변경했지만 태그 또한 일반 텍스트로 취급된다.
    planList.children[1].innerText = "<p>HTML5 Form 태그</p>";
    planList.children[3].textContent = "<p>CSS3 레이아웃을 위한 속성</p>";
  }
</script>
</head>
<body>
  <div id="container">
    <div class="header">
      <h1>DOM Tree</h1>
    </div>
    <div class="content">
      <!-- 앞으로 스터디 할 목록 -->
      <h2>프로그래밍 스터디 계획</h2>
      <ul id="plan">
        <li class="plan-list">JavaScript 함수</li>
        <li class="plan-list">JavaScript DOM</li>
        <li class="plan-list">JavaScript Event</li>
        <li class="plan-list">jQuery 함수 선택자</li>
        <li class="plan-list">jQuery Effect</li>
```

```
      </ul>
    </div>
    <div class="content">
      <!-- 완료된 스터디 목록 -->
      <h2>프로그래밍 스터디 완료</h2>
      <ul id="done">
        <li class="done-list">HTML 기본 태그</li>
        <li class="done-list">HTML5 시멘틱 태그</li>
        <li class="done-list">CSS3 선택자</li>
        <li class="done-list">CSS3 박스모델</li>
        <li class="done-list">JavaScript 제어문</li>
      </ul>
    </div>
  </div>
</body>
</html>
```

▶ 예제 8-4) 요소 노드를 추가, 수정, 삭제하기

- JavaScriptStudyCh08/javascript08_04.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>요소 노드를 추가, 수정, 삭제하기</title>
<script>
  /* 요소 노드를 추가, 수정, 삭제하기
   * createElement(), cloneNode(), prepend(), append(), before(),
   * after(), remove() 함수를 활용해 요소를 추가, 수정 삭제하기
   **/

  // HTML 문서의 모든 구성 요소가 메모리에 준비되면
  window.onload = function() {

    // id='plan'인 요소를 선택하고 Element 객체로 읽어온다.
    const plan = document.querySelector("#plan");

    // 새로운 li 태그를 생성하고 내용을 추가한 후 plan의 마지막 자식으로 추가
    const li1 = document.createElement("li");
    li1.textContent = "오라클 SQL 스터디 준비";
    plan.append(li1);

    // li1을 자식을 포함해 복제하고 내용을 수정한 후 plan의 첫 번째 자식으로 추가
    const li2 = li1.cloneNode(true);
```

```
    li2.textContent = "JavaScript 제어문 복습";
    plan.prepend(li2);

    /* li2를 자식을 포함해 복제하고 내용을 수정한 후 plan의 index 1번째
     * 자식 요소를 읽어와서 그 요소의 바로 앞에 새로 복제한 요소를 추가
     **/
     const li3 = li2.cloneNode(true);
    li3.textContent = "함수 만들기 집중 공략";
    const planChild1 = plan.children[1];
    planChild1.before(li3);
    //planChild1.after(li3);

    // plan의 첫 번째 자식을 선택하고 삭제하기
    plan.firstElementChild.remove();

    // plan의 첫 번째 자식을 가져와 id='done'인 요소의 첫 번째 자식으로 이동
    const done = document.querySelector("#done");
    done.prepend(plan.firstElementChild);
    //done.append(plan.lastElementChild);

    // plan의 첫 번째 자식을 가져와 done의 index 1번째 자식의 앞과 뒤로 이동
    done.children[1].before(plan.firstElementChild);
    done.children[1].after(plan.firstElementChild);
  }
</script>
</head>
<body>
  <div id="container">
    <div class="header">
      <h1>DOM Tree</h1>
    </div>
    <div class="content">
      <!-- 앞으로 스터디 할 목록 -->
      <h2>프로그래밍 스터디 계획</h2>
      <ul id="plan">
        <li class="plan-list">JavaScript 함수</li>
        <li class="plan-list">JavaScript DOM</li>
        <li class="plan-list">JavaScript Event</li>
        <li class="plan-list">jQuery 함수 선택자</li>
        <li class="plan-list">jQuery Effect</li>
      </ul>
    </div>
    <div class="content">
      <!-- 완료된 스터디 목록 -->
      <h2>프로그래밍 스터디 완료</h2>
```

```
        <ul id="done">
          <li class="done-list">HTML 기본 태그</li>
          <li class="done-list">HTML5 시멘틱 태그</li>
          <li class="done-list">CSS3 선택자</li>
          <li class="done-list">CSS3 박스모델</li>
          <li class="done-list">JavaScript 제어문</li>
        </ul>
      </div>
    </div>
</body>
</html>
```

▶ 예제 8-5) HTML 태그의 속성 다루기

- JavaScriptStudyCh08/javascript08_05.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>HTML 태그의 속성 다루기</title>
<script>
  /* HTML 태그의 속성 다루기
   * 웹 브라우저가 HTML 문서를 읽어 DOM을 구성할 때 HTML에서 사용한 속성은
   * 요소 노드의 프로퍼티로 만들어 진다. 한 가지 주의해야 할 점은 HTML에서
   * 사용하는 속성 이름이 JavaScript에서 사용하는 프로퍼티의 이름과 동일하지
   * 않을 수 있다. 또한 HTML 태그에서 사용한 속성이 표준 속성이 아닐 경우에
   * 요소 노드의 프로퍼티로 읽어 올 수 없으며 이때는 요소 노드가 지원하는
   * 메서드를 사용해야 한다.
   *
   * HTML 태그의 표준 속성 참고
   * https://html.spec.whatwg.org/#attributes-3
   **/

  // HTML 문서의 모든 구성 요소가 메모리에 준비되면
  window.onload = function() {

    // id='plan'인 요소를 선택하고 Element 객체로 읽어온다.
    const plan = document.querySelector("#plan");

    // HTML에서 class 속성은 요소 노드에서 className 프로퍼티로 접근한다.
    console.log(plan.parentElement.className);

    // 요소 노드의 프로퍼터에 값을 지정하면 이전에 설정된 속성을 덮어쓴다.
    // plan.parentElement.className = "content_plan";
```

```
/* 아래 HTML 문서에서 id='plan'인 ul 태그에 value 속성을 사용했지만 이것은
 * HTML 표준에 정의되어 있는 ul 태그의 표준 속성이 아니다. 그래서 브라우저가
 * HTML 문서를 읽어 DOM을 구성할 때 요소 노드의 프로퍼티로 생성되지 않는다.
 **/
console.log("plan.value : ", plan.value);

// id 속성은 표준 속성이므로 요소 노드의 프로퍼티로 읽어올 수 있다.
console.log(plan.id);

/* 요소 노드의 프로퍼티로는 비표준 속성에 접근할 수 없지만 요소 노드의
 * getAttribute() 메서드를 이용하면 HTML에서 설정한 모든 속성을 읽어올 수 있다.
 * 이 메서드는 실제 HTML 태그에 지정한 표준과 비표준 속성 모두를 읽을 수 있다.
 **/
console.log(plan.getAttribute("value"));
console.log(plan.getAttribute("id"));

/* 요소 노드의 setAttribute() 메서드를 사용하면 HTML 태그에 표준과 비표준
 * 속성 모두를 설정할 수 있으며 기존에 이미 설정된 속성도 수정할 수 있다.
 **/
plan.parentElement.setAttribute("id", "content");
plan.setAttribute("class", "plan");
plan.setAttribute("value", "javascript-study");

// 요소 노드의 removeAttribute() 메서드로 HTML 태그의 속성을 삭제할 수 있다.
plan.parentElement.removeAttribute("id", "content");
plan.removeAttribute("class");
plan.removeAttribute("value");

/* 웹 문서를 작성하다 보면 비표준 속성을 사용해야 할 때도 있는데 이 때는
 * 무분별하게 비표준 속성을 사용하기 보다는 꼭 필요한 속성인지 한 번 더
 * 따져보고 꼭 필요하다면 HTML5에서 비표준 속성을 사용하기 위한 약속된
 * 방식을 사용하는 것을 권장하고 싶다. HTML5에서는 비표준 속성이 필요한
 * 사용자가 "data-*" 형식으로 사용자 속성을 만들어 사용할 수 있는데 이렇게
 * 만들어진 속성은 요소 노드의 dataset 프로퍼티로 접근할 수 있다.
 **/
plan.children[3].setAttribute("data-study", "waiting");
plan.children[4].setAttribute("data-study", "waiting");

// id='plan'의 자식인 li 요소 중에서 data-study 속성이 없는 모든 요소 선택
let waiting = document.querySelectorAll("#plan > li:not([data-study])");
for(li of waiting) {
  /* 요소 노드의 setAttribute() 메서드 또는 dataset 프로퍼티를 이용해
   * HTML 태그에 동적으로 "data-*" 형식의 속성을 추가할 수 있다.
   **/
```

```
        li.setAttribute("data-study", "studying");
        //li.dataset.study = "studying"
      }
    }
  </script>
  <style>
    li[data-study='studying']::after {
      content :  " - studying";
      color: blue;
    }
    li[data-study='waiting']::after {
      content :  " - waiting";
      color: red;
    }
  </style>
</head>
<body>
  <div id="container">
    <div class="header">
      <h1>DOM Tree</h1>
    </div>
    <div class="content">
      <!-- 앞으로 스터디 할 목록 -->
      <h2>프로그래밍 스터디 계획</h2>
      <ul id="plan" value="study-plan">
        <li class="plan-list">JavaScript 함수</li>
        <li class="plan-list">JavaScript DOM</li>
        <li class="plan-list">JavaScript Event</li>
        <li class="plan-list">jQuery 함수 선택자</li>
        <li class="plan-list">jQuery Effect</li>
      </ul>
    </div>
    <div class="content">
      <!-- 완료된 스터디 목록 -->
      <h2>프로그래밍 스터디 완료</h2>
      <ul id="done">
        <li class="done-list">HTML 기본 태그</li>
        <li class="done-list">HTML5 시멘틱 태그</li>
        <li class="done-list">CSS3 선택자</li>
        <li class="done-list">CSS3 박스모델</li>
        <li class="done-list">JavaScript 제어문</li>
      </ul>
    </div>
  </div>
</body>
```

```
</html>
```

▶ 예제 8-6) HTML 문서의 스타일 다루기

- JavaScriptStudyCh08/javascript08_06.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>HTML 문서의 스타일 다루기</title>
<script>
  /* HTML 문서의 스타일 다루기
   * 요소 노드의 style 프로퍼티를 사용해 스타일을 적용하면 인라인(inline)
   * 방식의 스타일이 적용되어 스타일 우선 순위가 높아지는 문제가 있다.
   * 또한 같은 스타일을 여러 태그에 적용하려면 아래와 같이 비슷한 코드를
   * 여러 번 중복해서 작성해야 하는 문제가 발생한다. 중복 코드가 많아지면
   * 오류가 발생될 확률이 높아지고 유지 보수 비용도 많이 올라간다. 그래서
   * 부득이한 경우가 아니라면 자바스크립트에서 직접 스타일을 적용하는 것
   * 보다 class 선택자로 미리 스타일 규칙을 만들어 놓고 자바스크립트에서
   * HTML 태그에 class 속성을 추가했다가 삭제하는 방식이 권장되는 방식이다.
   **/

  // HTML 문서의 모든 구성 요소가 메모리에 준비되면
  window.onload = function() {

    // id='plan'인 요소를 선택하고 Element 객체로 읽어온다.
    const plan = document.querySelector("#plan");

   /* 자바스크립트에서 HTML의 스타일을 다루려면 요소 노드의 style 프로퍼티를
    * 이용해 스타일 속성에 접근할 수 있다. CSS에서 사용하는 스타일 속성이
    * 다음과 같이 여러 단어로 되어 있는 경우 자바스크립트의 style 프로퍼티로
    * 접근할 때는 낙타 표기법(Camel case)을 사용해야 한다.
    *
    * background-color -> 요소노드.style.backgroundColor
    **/
    /*
    plan.children[0].style.backgroundColor = "#FFAAAA";
    plan.children[0].style.textDecoration = "line-through";
    plan.children[3].style.backgroundColor = "#FFAAAA";
    plan.children[3].style.textDecoration = "line-through";
    */

    /* HTML에서 class 속성은 요소 노드에서 className 프로퍼티로 접근한다.
     * 다음과 같이 className 프로퍼터에 값을 지정하면 기존에 설정되어 있는
```

```
     * class 속성을 덮어 쓰기 때문에 기존 class가 사라지므로 기존에 설정된
     * 스타일 등이 적용되지 않는 문제가 발생한다. 아래와 같이 className에
     * 클래스 이름을 지정하면 기존의 plan-list 클래스는 삭제된다.
     **/
    //plan.children[0].className = "done";

    /* 요소 노드의 classList 프로퍼티를 이용하면 기존 클래스 그대로 새로운
     * 클래스를 추가하고, 삭제 할 수 있으며 기존 클래스를 수정할 수도 있다.
     * classList 프로퍼터는 클래스를 추가(add), 삭제(remove), 치환(replace)
     * 토글(toggle) 할 수 이는 메서드를 지원한다.
     **/
    //plan.children[0].classList.add("done");

    // 여러 개의 인수를 지정해 여러 클래스를 지정할 수 있다.
    //plan.children[0].classList.add("test1", "test2");

    // 기존 클래스를 새로운 클래스로 치환할 수 있다.
    //plan.children[0].classList.replace("done", "studying");

    // 기존에 존재하는 클래스를 삭제할 수 있으며 동시에 여러 개를 삭제할 수 있다.
    //plan.children[0].classList.remove("test1", "test2");

    /* toggle() 메서드는 해당 요소에 지정한 클래스가 없으면 추가하고 있으면
     * 삭제하는 기능을 제공한다. 이 메서드의 두 번째 인수로 true를 지정하면
     * 추가하는 기능만 제공하고 false를 지정하면 삭제하는 기능만 제공하게 된다.
     **/
    plan.children[0].classList.toggle("done");
    plan.children[3].classList.toggle("done");
  }
</script>
<style>
  .content {
    width: 400px;
  }
  li.done {
    text-decoration: line-through;
    background-color: #EAEAEA;
  }
  li.done::after {
    content :  " - 1차 완료";
    color: red;
  }
  li.studying {
    background-color: #FFEAEA;
  }
```

```
  li.studying::after {
    content :  " - 진행중";
    color: blue;
  }
</style>
</head>
<body>
  <div id="container">
    <div class="header">
      <h1>DOM Tree</h1>
    </div>
    <div class="content">
      <!-- 앞으로 스터디 할 목록 -->
      <h2>프로그래밍 스터디 계획</h2>
      <ul id="plan">
        <li class="plan-list">JavaScript 함수</li>
        <li class="plan-list">JavaScript DOM</li>
        <li class="plan-list">JavaScript Event</li>
        <li class="plan-list done">jQuery 함수 선택자</li>
        <li class="plan-list">jQuery Effect</li>
      </ul>
    </div>
    <div class="content">
      <!-- 완료된 스터디 목록 -->
      <h2>프로그래밍 스터디 완료</h2>
      <ul id="done">
        <li class="done-list">HTML 기본 태그</li>
        <li class="done-list">HTML5 시멘틱 태그</li>
        <li class="done-list">CSS3 선택자</li>
        <li class="done-list">CSS3 박스모델</li>
        <li class="done-list">JavaScript 제어문</li>
      </ul>
    </div>
  </div>
</body>
</html>
```

9. 이벤트 처리(Event handling)

웹 페이지에서 사용자가 버튼을 클릭 하거나, 마우스 포인터를 특정 요소 위에 올리거나, 입력 상자의 내용을 변경 할 때와 같이 사용자가 취하는 동작으로 발생하는 사건을 이벤트라고 한다. 웹 페이지에서 발생하는 이벤트에는 사용자 동작에 의해 발생하는 이벤트가 있으며 웹 문서가 메모리에 로드되었거나 하는 특정 상황에 따라 자동으로 발생되는 이벤트가 있다.
프로그래밍에서 이벤트가 발생하면 이를 처리하기 위한 함수와 같은 이벤트 처리 코드를 미리 등록하여 이벤트를 처리하는데 이런 프로그래밍 모델을 이벤트 기반 모델(Event Driven Model) 또는 이벤트 모델(Event Model) 이라고 한다. 이 때 이벤트를 처리하기 위해 등록한 함수를 이벤트 핸들러(Event Handler) 또는 이벤트 리스너(Event Listener)라고 부른다.
자바스크립트의 이벤트 모델은 문서 객체에 이벤트 리스너를 등록하는 방법에 따라서 다음과 같은 방법이 있다.

▶ DOM Level 0 이벤트 모델

이 이벤트 모델은 W3C에서 DOM이 표준화되기 전의 모델이며 이벤트 속성에 이벤트 처리 코드를 등록하는 방식이다. 또한 하나의 이벤트에 하나의 리스너만 연결할 수 있는 이벤트 모델이며 아래와 같이 두 가지 방식이 있다.

- **고전 이벤트 모델** : 문서 객체의 이벤트 속성에 이벤트 핸들러를 등록하는 방식
- **인라인 이벤트 모델** : HTML 태그의 이벤트 속성에 이벤트 핸들러를 등록하는 방식

▶ DOM Level 2 이상의 이벤트 모델

문서 객체에 이벤트를 처리할 리스너를 등록해 이벤트를 처리하는 방식으로 DOM Level 2 이벤트 모델을 표준 이벤트 모델이라고 불렀다. 이 이벤트 모델은 이벤트 하나에 여러 개의 리스너를 연결할 수 있는 이벤트 모델이며 현재 대부분의 모던 브라우저에서 지원하는 이벤트 모델이다. 참고로 2023년 7월 현재 DOM Level 4 이벤트 명세가 정의되어 있는 상태이며 대부분의 모던 브라우저들은 DOM Level 3 이벤트 명세를 지원하고 있다. DOM Level 3 명세에서는 DOM Level 2에서 보다 더 많은 요소들과 이벤트들이 추가되었으며 DOM 유효성 검사기가 추가되었다고 한다.
현재에는 다양한 웹 기술의 발전에 따라서 DOM Level에 따른 표준만으로 이벤트 전체를 정의할 수 없는 상황이 되었으며 이렇게 다양한 이벤트를 위해서 DOM Level 명세 이외에 HTML5 관련 이벤트 명세와 모바일 장치를 위한 이벤트 명세 등이 있다.

▶ 이벤트 타입과 이벤트 속성

분류	이벤트타입 (이벤트 이름)	이벤트 속성	발생 상황
윈도우	resize	onresize	윈도우 크기가 변경 될 때
	scroll	onscroll	스크롤 했을 때
	error	onerror	에러가 발생할 때
문서 읽기	load	onload	HTML 문서(이미지, 스크립트, CSS)가 완전히 로딩 되었을 때

분류	이벤트타입 (이벤트 이름)	이벤트 속성	발생 상황
문서 읽기	unload	onunload	문서를 닫거나 다른 문서로 이동할 때 브라우저를 새로 고침 해도 발생함.
	abort	onabort	이미지 로딩이 중단되었을 때
마우스	click	onclick	마우스 왼쪽 버튼이 클릭되었을 때
	dblclick	ondblclick	마우스 왼쪽 버튼이 더블클릭 되었을 때
	mousedown	onmousedown	마우스 버튼을 눌렀을 때
	mouseup	onmouseup	마우스 버튼을 눌렀다 뗄 때
	mouseover	onmouseover	마우스 커서가 요소에 올라올 때
	mouseout	onmouseout	마우스 커서가 요소를 벗어날 때
	mouseenter	onmouseenter	마우스 커서가 요소에 올라올 때 이벤트 버블링이 발생하지 않음
	mouseleave	onmouseleave	마우스 커서가 요소를 벗어날 때 이벤트 버블링이 발생하지 않음
	mousemove	onmousemove	마우스 커서가 움직일 때
	contextmenu	oncontextmenu	마우스 오른쪽 버튼을 클릭할 때 컨텍스트 메뉴가 표시될 때
	dragdrop	ondragdrop	요소를 드래그 앤 드롭 했을 때
키보드	keydown	onkeydown	키보드 키를 누를 때
	keypress	onkeypress	키보드 키를 눌렀다 뗄 때 (a, 1 등과 같이 출력이 되는 키만 동작하고 Shift, Esc 등의 키는 반응하지 않음)
	keyup	onkeyup	키보드 키를 눌렀다 뗄 때 keypress() 메소드 다음으로 실행됨
폼	focusin	onfocusin	요소가 포커스를 받을 때
	focusout	onfocusout	요소에서 포커스가 사라질 때
	focus	onfocus	요소가 포커스를 받을 때 이벤트 버블링이 발생하지 않음
	blur	onblur	요소에서 포커스가 사라질 때 이벤트 버블링이 발생하지 않음
	input	oninput	값이 입력될 때
	select	onselect	입력 양식의 하나가 선택될 때
	change	onchange	입력된 값이나 선택 항목이 변경될 때
	reset	onreset	리셋 버튼을 눌렀을 때
	submit	onsubmit	서브밋 버튼을 눌렀을 때

표 9-1 이벤트 타입과 속성

Event 참고 : https://developer.mozilla.org/ko/docs/Web/API/Event
MouseEvent 참고 : https://developer.mozilla.org/ko/docs/Web/API/MouseEvent
KeyboardEvent 참고 : https://developer.mozilla.org/ko/docs/Web/API/KeyboardEvent

▶ 예제 9-1) 이벤트 핸들러 등록하기

- JavaScriptStudyCh09/javascript09_01.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>이벤트 핸들러 등록하기</title>
<script>
  window.onload = function() {

    /* 이벤트 프로퍼티에 이벤트 핸들러 등록
     * 문서 객체의 이벤트 프로퍼티에 이벤트를 처리할 핸들러를 등록하고 있다.
     * 이벤트 프로퍼티는 이벤트 이름 앞에 on이 포함된다. 아래는 click 이벤트의
     * 이벤트 프로퍼티인 onclick에 이벤트를 처리할 핸들러를 등록하는 예이다.
     **/
    const btn1 = document.querySelector('#btn1');
    btn1.onclick = function(e) {
      console.log("고전 이벤트 핸들러");

      /* onclick 속성에 등록된 이벤트 핸들러를 제거하고 있다.
       * 이 이벤트 핸들러가 한 번 실행 후 이벤트 핸드러를 제거 했기
       * 때문에 두 번째 클릭 부터는 이 핸들러는 실행되지 않는다.
       **/
       //btn1.onclick = null;
    }

    /* 이벤트 프로퍼티에 새로운 함수를 등록하면 이전에 등록한 핸들러를 덮어쓴다.
     * 이벤트 프로퍼티 방식으로 이벤트 핸들러를 등록하는 방식도 하나의 이벤트
     * 핸들러만 등록할 수 있고 다중 이벤트 핸들러를 등록하거나 여러 개의 이벤트
     * 핸들러를 제어하기가 힘들다.
     **/
    /*
    btn1.onclick = function() {
      console.log("btn1 버튼 클릭됨");
    }
    */

    function btnClick() {
      console.log("표준 이벤트 핸들러");
```

```
    }

    function mouseOver() {
      const result = document.getElementById("result");
      result.classList.toggle("effect")

      // 이벤트 리스너 안에서 this는 이벤트가 발생한 객체를 가리킨다.
      result.innerHTML += this.id + "에서 이벤트 발생<br/>";
    }

    /* addEventListener() 메서드를 이용한 이벤트 리스너 등록
     * 문서 객체의 addEventListener() 메서드를 이용해 이벤트를 처리할 리스너를
     * 등록하는 방식은 자바스크립트 이벤트 처리에서 가장 권장하는 방식이다.
     * 이 방식은 하나의 문서 객체에 여러 이벤트 리스너를 등록할 수 있기 때문에
     * 동일한 이벤트 타입에 다중 이벤트 리스너를 등록하거나 여러 개의 이벤트
     * 리스너를 제어할 수 있다.
     *
     * EventType : 이벤트 이름("click", "dblclick", "keydown" 등등)
     * EventListener : 이벤트가 발생했을 때 이벤트를 처리할 함수를 지정
     * useCapture : 이벤트 캡처링 여부로 true면 캡처링, false면 버블링을 의미
     *              기본값은 false
     *
     * addEventListener(EvetType, EventListener, useCapture);
     **/
    const btn2 = document.querySelector('#btn2');
    btn2.addEventListener("click", btnClick);
    btn2.addEventListener("mouseover", mouseOver);

    // 앞에서 등록한 이벤트 리스너 삭제하기
    //btn2.removeEventListener("mouseover", mouseOver);
  }
</script>
<style>
  div {
    width: 500px;
    height: 100px;
    margin-top: 30px;
    overflow: auto;
    border: 2px solid salmon;
  }
  .effect {
    background-color: green;
    color: yellow;
  }
</style>
```

```
</head>
<body>
  <button id="btn1">고전 이벤트</button>
  <button id="btn2">표준 이벤트</button>
  <!--
    HTML 태그에서 직접 이벤트를 다루는 인라인 이벤트 모델은 요즘 잘 사용되지
    않는 방식으로 이 방식은 HTML 코드와 자바스크립트 코드가 뒤섞여서 코드의
    가독성이 떨어지고 협업이나 유지보수에도 좋지 않은 방식이다.
  -->
  <button onclick="alert('인라인 이벤트 핸들러');">인라인 이벤트</button>
  <div id="result"></div>
</body>
</html>
```

▶ 이벤트 객체와 이벤트 프로퍼티

웹 페이지에서 이벤트가 발생하면 이벤트에 대한 상세한 정보를 담고 있는 이벤트 객체가 자동으로 만들어 진다. 이 이벤트 객체를 통해서 어떤 이벤트가 발생했는지 또는 이벤트가 발생할 때 마우스 버튼의 위치 등과 같은 다양한 정보를 알아낼 수 있다. 이벤트 객체는 이벤트를 처리하는 리스너의 첫 번째 매개변수를 통해서 전달 받을 수 있다. 또한 이벤트 객체는 이벤트 종류에 따라서 다양한 프로퍼티가 존재하며 다음은 이벤트 객체가 가지는 프로퍼티에 대해서 정리한 표이다. 분류에서 공통부분은 이벤트 타입과 상관없이 모든 이벤트 객체들이 공통적으로 가지고 있는 프로퍼티이다.

분류	프로퍼티	설 명
공통	target	이벤트가 발생한 요소
	type	이벤트 이름 또는 종류(click, mousedown, keydown 등등)
	currentTarget	이벤트 핸들러가 등록된 요소
	timeStamp	이벤트 발생 시각(페이지가 로드된 후부터 경과된 밀리 초)
	bubbles	이벤트가 버블링 단계 인지를 판단하는 값
마우스 이벤트	button	눌린 마우스 버튼(0:왼쪽, 1:가운데(휠), 2:오른쪽)
	screenX	모니터 영역에서 마우스 커서의 x좌표
	screenY	모니터 영역에서 마우스 커서의 y좌표
	clientX	브라우저 표시 영역에서 마우스 커서의 x좌표
	clientY	브라우저 표시 영역에서 마우스 커서의 y좌표
	pageX	문서 영역에서 마우스 커서의 x좌표
	pageY	문서 영역에서 마우스 커서의 y좌표
	offsetX	이벤트 발생 요소에서 마우스 커서의 x좌표
	offsetY	이벤트 발생 요소에서 마우스 커서의 y좌표

키보드 이벤트	key	눌린 키가 가지고 있는 값
	code	누린 키의 코드 값
마우스, 키보드 이벤트	altKey	이벤트가 발생할 때 alt 키를 눌렀는지
	ctrlKey	이벤트가 발생할 때 ctrl 키를 눌렀는지
	shiftKey	이벤트가 발생할 때 shift 키를 눌렀는지
	metaKey	이벤트가 발생할 때 meta 키를 눌렀는지 (Windows는 window 키, Mac은 cmd 키)
※ 마우스 이벤트의 좌표관련 프로퍼티는 MouseEvent 객체 외에도 HTMLElement객체, Window객체, Document객체, Screen객체에서 제공하는 크기 및 위치관련 메서드와 프로퍼티가 있다.		

표 9-2 이벤트 객체의 주요 프로퍼티

▶ 예제 9-2) 이벤트 객체와 이벤트 객체의 프로퍼티

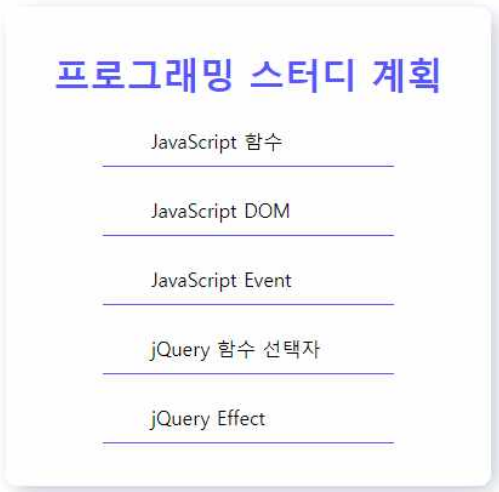
- JavaScriptStudyCh09/javascript09_02.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>이벤트 객체와 이벤트 객체의 프로퍼티</title>
<script>
  /* DOM이 완전히 해석되어 메모리에 준비되면
   * DOMContentLoaded 이벤트는 HTML 문서가 완전히 해석되어 메모리에 준비되고
   * 현재 문서에 연결된 스크립트 파일이 다운로드가 되면 발생하는 이벤트이다.
   *
   * 참고 사이트 :
   * https://developer.mozilla.org/ko/docs/Web/API/Document/DOMContentLoaded_event
   **/
  window.addEventListener('DOMContentLoaded', function() {
    console.log("DOM 로드 완료");

    // divArea 요소를 선택하고 mousedown 이벤트 리스너를 등록
    const divArea = document.querySelector("#divArea");
    divArea.addEventListener("mousedown", mouseDown);
  });

  /* 마우스 버튼이 눌려질 때 이벤트를 처리하는 함수
   * 이벤트 리스너의 첫 번째 매개변수로 이벤트 객체를 받을 수 있다.
   **/
  function mouseDown(e) {
    const eventInfo = document.querySelector('#eventInfo');
```

```
    var result = [];

    // 이벤트 리스너 안에서 this는 이벤트가 발생한 문서 객체를 가리킨다.
    result.push('이벤트 발생객체 this : ' + this.id);
    result.push('이벤트 발생객체 target : ' + e.target.id);
    result.push('이벤트 타입 : ' + e.type);
    result.push('눌린 버튼 : ' + getButtonType(e));
    result.push('뷰포트에서 X 좌표 : ' + e.clientX);
    result.push('뷰포트에서 Y 좌표 : ' + e.clientY);
    result.push('#divArea에서 X 좌표 : ' + e.offsetX);
    result.push('#divArea에서 Y 좌표 : ' + e.offsetY);
    result.push('altKey : ' + e.altKey);
    result.push('ctrlKey : ' + e.ctrlKey);
    result.push('shiftKey : ' + e.shiftKey);
    result.push('metaKey : ' + e.metaKey);

    eventInfo.innerHTML = '<h3>이벤트 발생 정보</h3>';
    eventInfo.innerHTML += result.join('<br/>');
  }

  // 이벤트 객체를 받아서 클릭된 버튼의 종류(좌/우/중앙)를 반환하는 함수
  function getButtonType(e) {
    switch(e.button) {
      case 0 : return 'left';
      case 1 : return 'center';
      case 2 : return 'right';
    }
  }

</script>
<style>
  #divArea {
    width: 600px;
    height: 400px;
    border: 1px solid #9999FF;
  }
</style>
</head>
<body>
  <div id="divArea"></div>
  <div id="eventInfo"></div>
</body>
</html>
```

[연습문제 9-1] 마우스 클릭이벤트 활용

다음과 같이 화면을 구성하고 스터디가 완료된 항목을 마우스로 클릭하면 아래 두 번째 그림과 같이 클릭된 항목은 텍스트를 흐리게 표시하고 텍스트에 취소선과 완료라는 문구가 추가되도록 작성하고 완료 처리된 항목에 마우스를 클릭하면 다시 원래대로 표시 될 수 있도록 프로그램을 작성하시오. 이 기능은 항목이 처음 클릭되면 완료로 표시되고 다시 클릭되면 원래대로 표시되고 또 다시 클릭되면 완료로 표시되고 다시 클릭되면 원래대로 표시되는 토글 동작 되도록 구현하면 됩니다.



▶ 예제 9-3) 이벤트 전파

- JavaScriptStudyCh09/javascript09_03.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>이벤트 전파와 기본 이벤트 제어</title>
<script>
```

```
/* 이벤트 전파(Event Propagation)
 * 이벤트는 보통 하나의 문서 객체에 대해 발생하지만 서로 중첩된
 * 문서 객체에서 상위 또는 하위로 이벤트가 전파되는 특징을 가진다.
 * 이벤트 전파는 다음과 같이 이벤트 버블링과 이벤트 캡처링이 있다.
 *
 * - 이벤트 버블링 :
 *   이벤트가 발생한 객체로부터 순서대로 상위 객체로 이벤트가 전파되는 것을
 *   의미하며 이벤트가 발생한 객체부터 document 객체까지 상위로 전파된다.
 *
 * - 이벤트 캡처링 :
 *   document 객체에서 시작해 이벤트가 발생한 객체까지 하위 객체로 이벤트가
 *   전파되는 것을 의미한다.
 *
 * - 이벤트 전파 막기
 *   이벤트 객체의 stopPropagation() 메서드를 사용하면 이벤트 전파를 막을 수 있다.
 *   e.stopPropagation();
 *
 * 기본 이벤트(Default Event)
 * HTML 문서에서 어떤 요소는 기본적으로 연결된 이벤트 핸들러를 가지고 있어서
 * 별도의 이벤트 핸들러를 등록하지 않아도 특정 이벤트가 발생하면 기본적인
 * 동작을 하는데 이런 기본 동작을 기본 이벤트(Default Event)라고 부른다.
 * 예를 들면 HTML 문서에서 a 요소가 클릭되면 href 속성에 연결된 주소로 이동하는
 * 기본 동작을 하는데 이런 기본 동작을 기본 이벤트라고 한다. 이렇게 a 요소와
 * 같이 기본 동작이 연결되어 있어서 특정 이벤트가 발생하면 무조건 기본 동작을
 * 수행하는 요소들이 있다. 대표적으로 form 요소와 관련있는 입력 컨트롤이 이런
 * 기본 동작을 수행하며 checkbox, radio, text 입력 상자와 같이 input 요소가
 * 기본 동작을 가지고 있다. checkbox를 예를 들면 사용자가 클릭할 때 마다 체크
 * 박스의 항목이 체크 되거나 해제되는 기본 동작을 한다. 이외에도 submit 버튼
 * 있으며 submit 버튼이 클릭되면 폼 안에 있는 입력 컨트롤에 입력된 정보를
 * form 요소의 action 속성에 지정된 url로 전송하는 기본 동작을 수행한다.
 * 만약 폼이 전송될 때 폼 유효성 검사를 수행해서 폼 전송을 취소해야 된다면
 * submit 되는 기본 동작을 멈춰야 하는데 이렇게 요소가 가진 기본 이벤트를
 * 취소해야 할 때 종종 발생한다.
 *
 * - 기본 이벤트 취소
 *  이벤트 객체의 preventDefault() 메서드를 사용하면 기본 이벤트를 멈출 수 있다.
 *  e.preventDefault();
 **/
window.addEventListener("DOMContentLoaded", function() {
  console.log("DOM Loaded");

  const container = document.querySelector("#container");
  const h1 = document.querySelector("h1");
  const content = document.querySelector(".content");
  const plan = document.querySelector("#plan");
```

```
const planList = document.querySelectorAll(".plan-list");

/* 동일한 이벤트 타입의 이벤트 리스너가 등록되어 있으면 이벤트가 발생한
 * 요소의 이벤트 리스너가 실행되고 그 다음 상위 요소로 이벤트가 전달되어
 * 상위 요소에 등록된 이벤트 핸들러가 차례대로 실행된다. 이렇게 이벤트가
 * 상위 요소로 전달되는 것을 이벤트 버블링(Event Bubbling) 이라고 한다.
 *
 * 이벤트 리스너의 첫 번째 파라미터로 이벤트 정보를 담고 있는 이벤트 객체가
 * 전달되는데 이 객체를 통해서 이벤트의 다양한 정보를 알아낼 수 있다.
 *
 * type : 이벤트 종류(이벤트 이름), target : 이벤트가 발생한 요소,
 * currentTarget : 이벤트 핸들러가 등록된 요소, bubbles : 버블링 단계 여부,
 **/
container.addEventListener("click", function(e) {
  console.log("container click");
  console.log(e.target);
  console.log(e.currentTarget);
});
h1.addEventListener("click", function(e) {
  console.log("h1 click");
  console.log(e.target);
});
content.addEventListener("click", function(e) {
  console.log("content click");
  console.log(e.target);
});
plan.addEventListener("click", function(e) {
  console.log("plan click");
  console.log(e.target);
  console.log(e.currentTarget);
});
for(let li of planList) {
  li.addEventListener("click", function(e) {
    console.log(`${li.textContent} click`);
    console.log(e.target);

    /* 이벤트 전파를 막는 메소드
     * 이벤트 버블링을 막는 기능을 무분별하게 사용하기 보다는 보다
     * 신중하게 생각하고 꼭 필요한 것인지 판단하여 적용해야 한다.
     **/
    e.stopPropagation();
  });
}
```

```
    const naverLink = document.querySelector("#naver");
    naverLink.addEventListener("click", function(e) {
      // 기본 이벤트 취소하기
      e.preventDefault();

      // 이벤트 전파 취소하기
      //e.stopPropagation();
    });
  });
</script>
<style>
  * {
    border: 2px solid black;
    margin: 20px;
  }
  ul {
    list-style-type: none;
  }
</style>
</head>
<body>
  <div id="container">
    <div class="header">
      <h1>DOM Tree</h1>
    </div>
    <div class="content">
      <!-- 앞으로 스터디 할 목록 -->
      <h2>프로그래밍 스터디 계획</h2>
      <ul id="plan">
        <li class="plan-list">JavaScript 함수</li>
        <li class="plan-list">JavaScript DOM</li>
        <li class="plan-list">JavaScript Event</li>
        <li class="plan-list">jQuery 함수 선택자</li>
        <li class="plan-list">jQuery Effect</li>
      </ul>
    </div>
    <div id="link">
      <a id="naver" href="http://www.naver.com">네이버 가기</a>
      <a id="daum" href="http://www.daum.net">다음 가기</a>
    </div>
  </div>
</body>
</html>
```

▶ 예제 9-4) 이벤트 위임

- JavaScriptStudyCh09/javascript09_04.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>이벤트 위임</title>
<script>
  /* 이벤트 위임(Event Delegate)
   * 이벤트가 발생한 요소에서 직접 처리하지 않고 이벤트 버블링을 활용해 상위
   * 요소에서 이벤트를 다루는 방식을 이벤트 위임(Event Delegation)이라고 한다.
   **/
  window.addEventListener('DOMContentLoaded', function() {
    console.log("DOM 로드 완료");

    // 이벤트 위임 전에 li 요소에 이벤트 리스너를 등록하는 코드
    const plan = document.querySelector("#plan");
    for(li of plan.children) {
      li.addEventListener("click", liClick);
    }

    // "항목 추가" 버튼이 클릭되면 스터디 항목을 추가하는 이벤트 리스너 등록
    const btn = document.querySelector("#btn");
    btn.addEventListener("click", function() {
      const li = document.createElement("li");
      li.textContent = "오라클를 활용한 SQL 쿼리";

      /* 동적으로 새로 추가한 li 요소는 앞에서 li 요소에 지정한 click 이벤트
       * 리스너가 적용되지 않기 때문에 별도로 이벤트 리스너를 등록해야 한다.
       **/
      //li.addEventListener("click", liClick);
      plan.append(li);
    });

    /* li 요소의 상위 요소인 ul(id='plan') 요소에 click 이벤트 리스너를 등록하고
     * li 요소에서 click 이벤트가 발생하게 되면 이벤트 버블링 단계에서 상위 요소인
     * ul 요소로 이벤트가 전파되어 아래에 등록한 click 이벤트 리스너가 실행된다.
     * 이 때 이벤트 리스너 안에서 이벤트 객체의 target 속성을 사용해 실제 이벤트가
     * 발생한 요소의 클래스를 토글 시키고 있으므로 실제 이벤트가 발생한 객체인
     * li 요소에 done 클래스가 추가되거나 삭제되어 스타일이 적용되거나 해제된다.
     * 이렇게 자식 요소에서 발생한 이벤트를 이벤트 버블링을 활용해 부모 요소에서
     * 다루는 방식을 이벤트 위임(Event Delegation)이라고 한다. 이벤트 위임을 활용해
     * 동적으로 새롭게 추가되는 하위 요소에 별도로 이벤트 리스너를 등록하지 않아도
     * 기존의 이벤트 리스너를 활용해 새로운 요소의 이벤트를 처리할 수 있다.
```

```
    **/
    //plan.addEventListener("click", liClick);
  });

  // 이벤트 위임 전에 항목이 클릭되면 class='done'를 토글하는 함수
  function liClick(e) {
    e.target.classList.toggle("done");
  }

  /* 이벤트 위임에서 항목이 클릭되면 class='done'를 토글하는 함수
   * 이벤트 위임으로 li 요소의 상위 요소인 ul 요소에 이벤트 리스너를 등록했기
   * 때문에 ul 요소를 클릭해도 스타일이 적용되거나 해제되기 때문에 명확하게
   * li 요소만 스타일을 적용하기 위해서 li 요소가 가지는 plan-list 클래스를
   * 가지고 있는지 체크해서 있으면 done 클래스를 토글 되도록 해야 한다.
   **/
  /*
  function liClick(e) {
    if(e.target.classList.contains("plan-list")) {
      e.target.classList.toggle("done");
    }
  }
  */
</script>
<style>
  li {
    padding: 5px;
  }
  .done {
    opacity : 0.5;
    text-decoration: line-through;
  }
</style>
</head>
<body>
  <div id="container">
    <h2>프로그래밍 스터디 계획</h2>
    <ul id="plan">
      <li class="plan-list">JavaScript 함수</li>
      <li class="plan-list">JavaScript DOM</li>
      <li class="plan-list">JavaScript Event</li>
      <li class="plan-list">jQuery 함수 선택자</li>
      <li class="plan-list">jQuery Effect</li>
    </ul>
  </div>
  <button id="btn">항목 추가</button>
```

```
</body>
</html>
```

▶ 예제 9-5) 마우스 이벤트

- JavaScriptStudyCh09/javascript09_05.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>마우스 이벤트</title>
<style type="text/css">
  .mouseover {
    border: 2px dotted blue;
    opacity: 0.5;
  }
  .mouseout {
    border: none;
  }
</style>
<script>
  window.addEventListener('DOMContentLoaded', function() {
    console.log("DOM 로드 완료");

    const imgs = document.querySelectorAll("img");

    for(let img of imgs) {
      img.addEventListener("mouseover", mouseOver);
      img.addEventListener("mouseout", mouseOut);
      img.addEventListener("dblclick", dblClick);
      img.addEventListener("mouseenter", mouseEnter);
      img.addEventListener("mouseleave", mouseLeave);
      img.addEventListener("contextmenu", mouseConetxtMenu);
    }

    const btn = document.querySelector("#btn");
    btn.addEventListener("click", function(e) {
      const img2 = document.querySelector("#img2");

      // 마우스 더블 클릭 이벤트 강제로 발생시키기
      img2.dispatchEvent(new Event("dblclick"));
    });

  });
```

```
  function mouseOver(e) {
    e.target.classList.remove('mouseout');
    e.target.classList.add('mouseover');
  }
  function mouseOut(e) {
    e.target.classList.remove('mouseover');
    e.target.classList.add('mouseout');
  }
  function dblClick(e) {
    if(e.target.id == "img2") {
      alert("2번째 이미지를 더블클릭...")
    }
  }
  function mouseEnter(e) {
    if(e.target.id == "img3") {
      console.log("3번째 이미지로 mouseenter")
    }
  }
  function mouseLeave(e) {
    if(e.target.id == "img1") {
      e.target.style.border = "2px solid red";
    }
  }
  function mouseConetxtMenu(e) {
    // id='img1' 일때 마우스 우클릭 기본 메뉴 취소하기
    if(e.target.id == "img1") {
      e.preventDefault();
    }
  }
</script>
</head>
<body>
  <div id="container1">
    <img id="img1" src="images/modern_javascript.jpg" />
  </div>
  <div id="container2">
    <img id="img2" src="images/make_html5.jpg" />
  </div>
  <div id="container3">
    <img id="img3" src="images/web_recipe.jpg" />
  </div>
  <div id="container4">
    <button id="btn">두 번째 이미지 더블 클릭 발생시키기</button>
  </div>
```

```
</body>
</html>
```

▶ 예제 9-6) 키보드 이벤트

- JavaScriptStudyCh09/javascript09_06.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>키보드 이벤트</title>
<style type="text/css">
  .input_bg {
    background-color: #FFEAEA;
  }
  #result {
    width: 680px;
    height: 300px;
    border: 1px solid #9999FF;
    overflow: auto;
  }
</style>
<script>
  window.addEventListener('DOMContentLoaded', function() {
    console.log("DOM 로드 완료");

    const result = document.querySelector("#result");
    const data = document.querySelector("#data");
    let input = "";

    data.addEventListener("focusin", e => {
      e.target.classList.toggle("input_bg");
    });
    data.addEventListener("focusout", e => {
      e.target.classList.remove("input_bg");
    });
    data.addEventListener("keydown", e => {
      result.innerHTML += `keydown - key : ${e.key}, code : ${e.code},
      altKey : ${e.altKey}, shiftKey : ${e.shiftKey}, ctrlKey : ${e.ctrlKey}<br>`;
    });
    data.addEventListener("keypress", e => {
      result.innerHTML += `keypress - key : ${e.key}, code : ${e.code},
      altKey : ${e.altKey}, shiftKey : ${e.shiftKey}, ctrlKey : ${e.ctrlKey}<br>`;
    });
```

```
  });
</script>
</head>
<body>
  <input type="text" name="data" id="data">
  <input type="button" value="확인">
  <div id="result"></div>
</body>
</html>
```