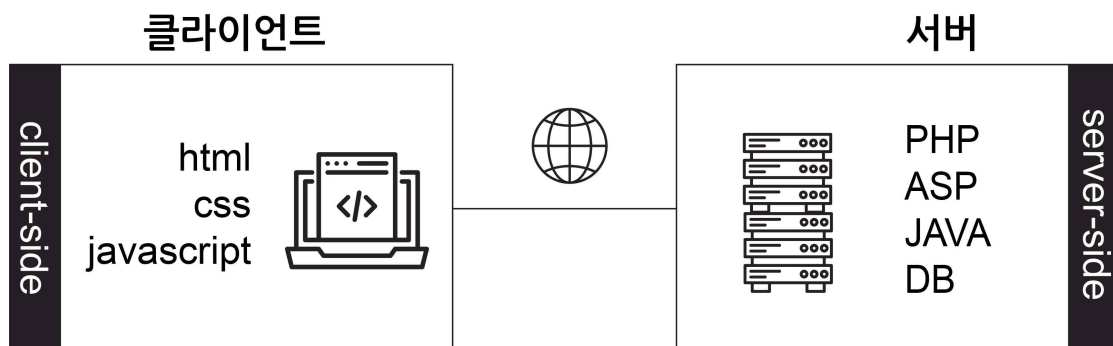


## 1-1 웹 프로그래밍 이란?

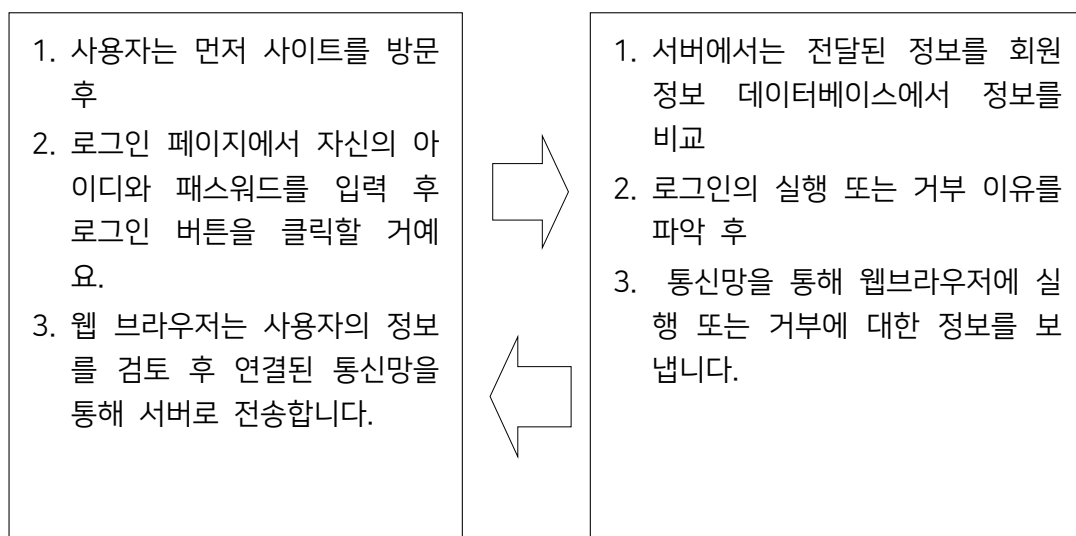
### 웹 프로그래밍의 환경

우리가 현재 사용하고 있는 웹사이트, 또는 웹앱은 두 파트로 나누어 작업이 됩니다. 하나는 사용자의 화면에서 이루어지는 클라이언트사이드와 클라이언트사이드에서 필요한 데이터와 동작을 제어하는 서버사이드입니다. 이 두 분야는 통신으로 연결이 작업을 합니다. 클라이언트 사이드에서는 사용자가 자신이 작업하는 미디어에서 직접 원하는 명령과 조작을 할 수 있으며, 사용하는 미디어에서 그 결과를 직접 확인 할 수 있습니다. 대표적으로 웹 브라우저라 할 수 있습니다. 이 **클라이언트 사이드**에서의 개발을 우리는 **프론트엔드(Front-end)**라고 합니다.

또한 사용자가 요구하는 작업의 데이터 관리나 필요한 작동을 하는 곳을 **서버사이드**로 이 분야의 개발을 **백엔드(Back-end)**라고 합니다.



회원 로그인 시스템을 생각해 봅시다. 사용자가 로그인을 원한다면

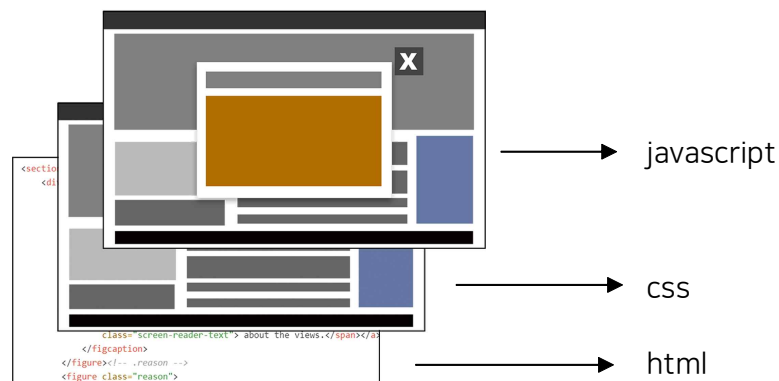


이처럼 사용자의 화면에서 이루어지는 개발을 프론트엔드라 하며 주된 프로그램은 웹브라우저에서 실행되는 HTML, CSS, Javascript을 사용합니다.

또한 프론트엔트에서 필요한 요청을 처리하는 작업은 백엔드에서 하면 개발 환경에 PHP, DB, ASP, JAVA등이 사용됩니다. 여러분은 지금부터 프론트엔드에서 작업 될 Javascript 공부를 이제 시작할 겁니다.

## javascript와 웹 브라우저

먼저 Javascript 이해를 하기 위해서는 우리는 웹 브라우저와 웹 페이지의 관계를 조금 자세히 들여다 볼 필요가 있습니다. 먼저 웹 브라우저에서 렌더링 되어 화면에 나타나는 웹페이지를 보겠습니다.



우리가 흔하게 볼 수 있는 웹 페이지는 3가지의 레이어로 이루어져 있습니다.

1차 html레이어, 2차 CSS레이어, 3차 Javascript 레이어입니다.

### 1. HTML 레이어

```
<section class="main-pitch">
  <div class="three-reasons centered">
    <figure class="reason">
      <a href="#">
        
      </a>
      <figcaption>
        <h3 class="content-title">A Room with a View</h3>
        <p>Every room at Moonwalk Manor, whether you are staying in a discount suite or going Apartment Class&reg;, has a panoramic view.</p>
        <a class="content-button" href="#">Learn More<span class="screen-reader-text"> about the views.</span></a>
      </figcaption>
    </figure></div>
  </section>
```

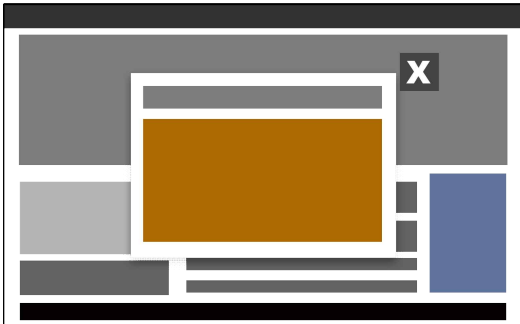
HTML 문서는 콘텐츠가 살아있는 곳으로 브라우저가 쉽게 구문 분석 할 수 있는 마크업 언어로 표시되어 있으며 웹 페이지를 방문한다고 하면 우리는 HTML 문서를 통해 원하는 웹사이트의 페이지는 방문할 수 있습니다. 즉 HTML 문서가 없다면 우리가 알고 있는 웹 페이지는 있을 수 없는 존재라 보면 됩니다.

### 2. CSS 레이어



CSS 영역은 브라우저에 HTML이 어떠한 모양으로 보여야 하는 지를 지시하는 코드작성 되어 있습니다. HTML 문서가 콘텐츠를 처리한다면 CSS는 그저 보여주는 부분을 담당합니다. 만약에 우리가 CSS 레이어를 제외해도 우리는 브라우저 화면에서 필요한 콘텐츠 정보를 수집하고 이해 할 수 있습니다.

### 3. 자바스크립트 레이어



우리는 javascript(자바스크립트) 레이어를 대화형 레이어부르기도 합니다. 자바스크립트는 브라우저 실행되는 작은 프로그램으로 HTML 마크업 및 CSS 규칙과 상호 작용하여 표시되는 내용과 브라우저에서 실행되는 작은 프로그램을 작성하고 현재 문서의 HTML 및 CSS를 변경할 수 있는 스크립팅 언어입니다.

흔히들 우리는 HTML과 CSS를 코딩한다고 하고 Javascript를 프로그래밍한다고 합니다. 이 차이를 한번 점검해 보면 코딩(Coding)는 소스 코드를 조합하는 작업이고 프로그래밍(programming)은 어떤 주어진 작업을 분석하고 논리적으로 해결하는 원하는 작업 진행 할 수 있도록 하는 것을 의미합니다.

사용자가 원하는 웹페이지를 방문하고자 하면 웹브라우저 주소창에 원하는 사이트의 HTML 문서가 있는 주소에서 브라우저를 입력합니다. 웹 브라우저는 인터넷 주소에 있는 HTML 문서를 불러들이고 해당 코드를 조직과 위치를 색인한 다음 참조해야 CSS코드와 자바스크립트 문서를 다운로드 합니다. 준비된 자바스크립트 코드를 실행하여 HTML 마크업이변경하고 그 후에 변경된 부분을 파악하여 다시 확 CSS를 적용하면 우리가 예상하는 것처럼 웹 사이트가 화면에 보이게 됩니다. 그 후 마지막으로 백그라운드에서 상호 작용이나 이벤트에 필요한 스크립트가 준비를 하고 있으면 추가 이벤트의 발생에 맞추어 추가된 스크립트의 작업을 실행하게 됩니다.

이것이 우리가 웹 페이지가 작동되는 원리라고 볼 수 있습니다

그럼 우리가 이러한 자바스크립트로 웹 개발에서 우리가 할 수 있는 일들은 무엇이 있는지 알아봅시다.

## 자바스크립트의 표준화

자바스크립트는 1995년 인터넷 브라우저 넷스케이프사에 근무하는 브렌드 아이크(Brendan Eich, 1961년~)에 의해 모카(Mocha)라는 이름으로 개발되었으며 그 후 라이브스크립트(LiveScript), 그리고 지금 우리가 알고 있는 자바스크립트(javascript)로 이름이 변경되었습니다. 초창기 인터넷 개발에서는 지금과 같이 자바스크립트가 표준 스크립트는 아니었습니다. 마이크로소프트사는 자신들의 브라우저에서는 자바스크립트가 아닌 Jscript가 기본 script언어 이었던 적도 있었습니다.

이렇게 브라우저의 표준 스크립트가 달라 표준화의 필요성을 느끼게 되었고 넷스케이프사를 중심으로 스크립트 언어의 표준화를 요청하였습니다. 이 표준화를 진행하는 기관은 우리가 흔히 ECMA(이크마)라 불리는 국제 정보통신표준화기구(European Computer Manufacturers Association)입니다. 1996년 11월에 ECMA-262라는 표준명세가 만들어지고 1997년 7월에 ECMA 1 버전이 만들어집니다.

우리가 요즘 자바스크립트는 말할 때 ECMA5, ECMA6, ECMAScript 등 여러 가지로 지칭하는 것은 바로 자바스크립트 표준화 버전을 말하는 것입니다. 즉 모두 자바스크립트를 가리키는 말이고 뒤에 숫자는 버전을 뜻합니다.

자바스크립트는 현재 거의 1년마다 업그레이드된 버전이 나오고 있습니다. 그럼 가장 최신 버전을 사용하면 된다고 생각 할 수 있지만 가장 많이 사용되는 버전은 우리가 흔히 말하는 ECMA5 버전입니다. 왜 2009년에 발표된 버전을 사용하는 이유는 자바스크립트 버전을 업그레이드해도 모든 브라우저에서 그 부분을 사용할 수 있도록 하지는 못한다는 점입니다. 브라우저도 내재한 스크립트의 수정이 필요하기에 시간이 필요한 이유입니다. 그런 이 책에서 주로 설명하는 자바스크립트는 ECMA5 버전을 토대로 현재 많은 브라우저에서 사용이 허용된 ECMA6의 내용이 포함되어 있습니다. ECMA6의 내용은 [ES6]이라는 표식이 들어가 있으니 이 점을 참고하시면 됩니다.

## 자바스크립트 기초 문법

브라우저와 에디터의 설정이 끝났다면 이제 우리는 자바스크립트를 시작할 수 있습니다. HTML 문서에 자바스크립트를 실행하기 위해서는 두 가지 방법이 있습니다.

1. HTML 문서의 내부에 `<script>` 요소로 자바스크립트 소스를 직접 작성하기
2. HTML 문서 외부에 있는 자바스크립트 파일, `***.js`(확장자가 js)인 파일을 `<script>` 요소로 HTML 문서에 불러들여 작성하기

이렇게 자바스크립트를 HTML 문서에 작동되도록 도와주는 태그가 `<script>`이며 우리는 먼저 `<script>` 태그를 사용하는 방법부터 알아보겠습니다.

### `<script>`요소 사용하기

`<script>` 태그는 HTML 문서에서 자바스크립트를 작동하도록 합니다. 이런 `<script>` 태그는 HTML 문서 어디든 상관없이 삽입할 수 있으며 여러 개의 `<script>` 태그를 사용해도 무관합니다. 이런 자바스크립트가 작동되는 시점은 HTML 문서에 삽입된 위치에서 작동이 됩니다. 그렇기 때문에 `<script>` 태그가 너무 여러 곳에서 작동이 되면 브라우저가 혼잡할 수 있습니다.

먼저 `<script>` 태그는 사용방법을 알아보시다.

### `<script>` 태그 안에 소스를 직접 작성하는 방법

HTML 문서에 `<script>` 태그 안에 소스를 입력하면 자바스크립트가 작동됩니다.

형식:

```
<script>
  자바스크립트 소스
</script>
```

9번 줄에 아래와 같이 `script` 태그를 넣고 자바스크립트가 작동되는지 확인해 보시다. ( 지금은 자바스크립트가 작동되는지만 확인을 하면 됩니다.

```

1    <!DOCTYPE html>
2    <html lang="ko">
3    <head>
4        <meta charset="UTF-8">
5        <meta name="viewport" content="width=device-width, initial-scale=1.0">
6        <title>javascript시작하기</title>
7    </head>
8    <body>
9        <script>
10         document.write('<h1>자바스크립트를 배웁시다');</h1>
11     </script>
12 </body>
13 </html>

```

자바스크립트 작성이 끝나면 라이브 서버를 이용하여 [alt + L alt+O] 브라우저에서 확인합니다.



결과물 확인하기

## 외부 스크립트 파일 연결하기

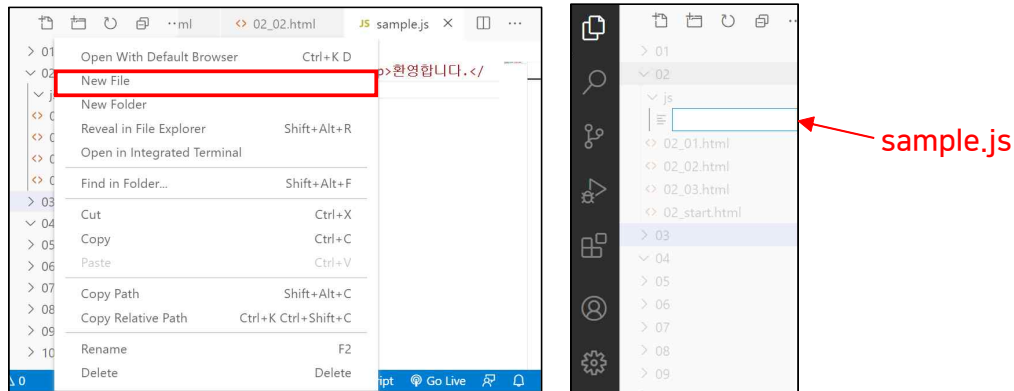
이번에는 HTML 문서와 자바스크립트 분리해서 작업하는 방법을 알아보시다. 외부 자바스크립트를 연결하는 형식은

형식: <script src="자바스크립트 파일 경로"></script>

입니다.

먼저 자바스크립트 외부 파일을 만들어 봅시다.

1. 툴바의 탐색기 아이콘을 선택하면 사이드바에 프로젝트 폴더가 나타납니다.
2. 02 폴더 안의 js 폴더를 찾습니다.
3. js폴더에 **Ctrl** + **N**을 눌러 새 문서를 만들기를 합니다.
4. 또다른 방법은 마우스 오른쪽 버튼을 누르면 서브메뉴가 나타납니다. [new file]를 선택하여 파일을 생성합니다.
5. 파일 이름을 'sample.js'라고 작성 후 엔터키를 누릅니다. 이때 주의사항은 꼭 확장자 **.js**로 저장을 하여야 자바스크립트 파일로 인식이 됩니다.



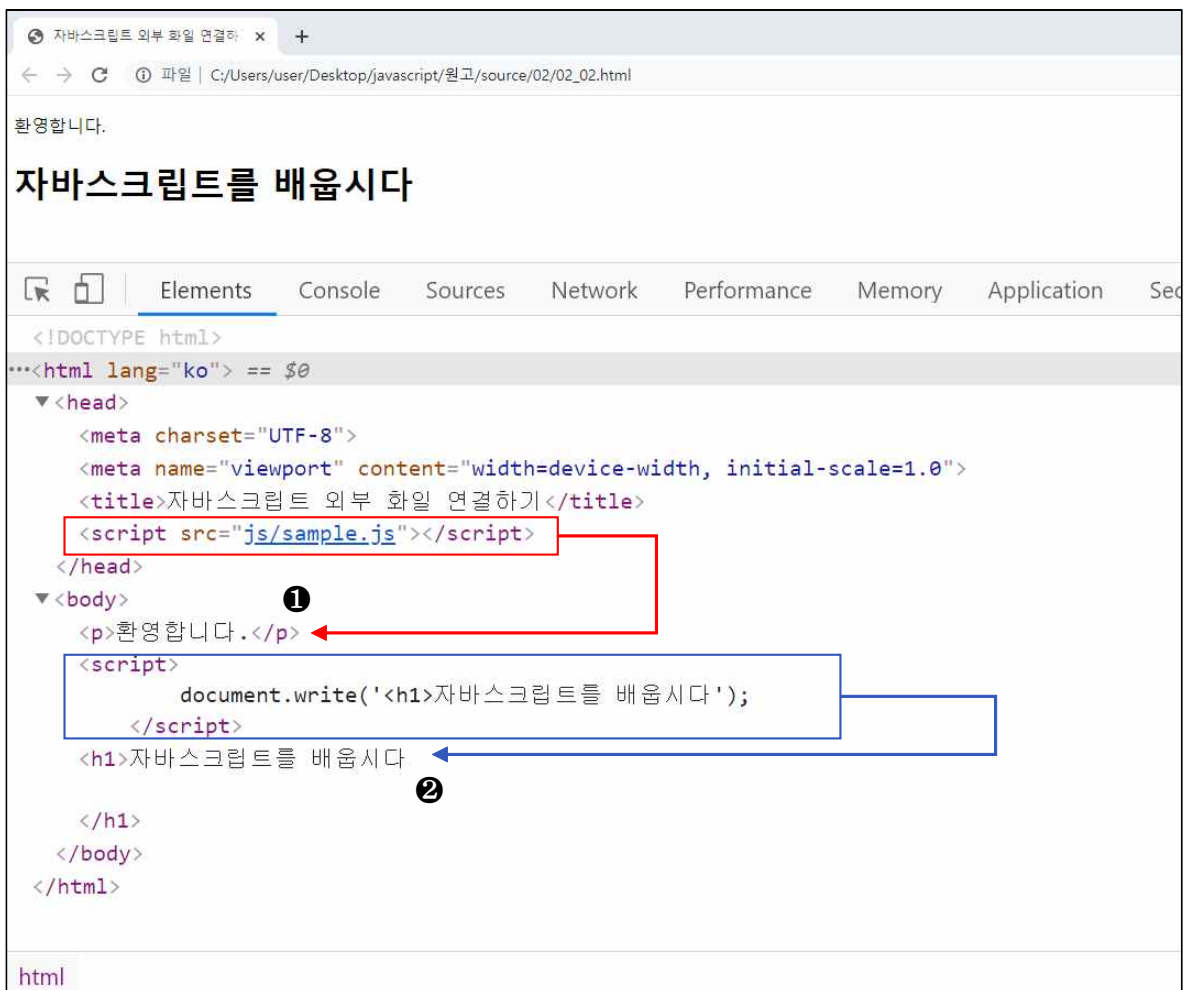
6. 이제 만들어진 HTML 문서에 자바스크립트 소스를 작성해 봅시다.

```
1 1 document.write('<p>환영합니다.</p>') ;
```

5. 그럼 이제 HTML 문서와 자바스크립트 문서를 연결합시다. (실습 파일: 02/02\_02.html. 완성 파일: 02/all/02\_02.html)

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width,
initial-scale=1.0">
6     <title>자바스크립트 외부 화일 연결하기</title>
7     <script src="js/sample.js"></script>
8 </head>
9 <body>
10     <script>
11         document.write('<h1>자바스크립트를 배웁시다');
12     </script>
13 </body>
14 </html>
```

6. 작성이 완료되면 라이브서버[alt+L alt+O]를 통해 브라우저에 확인합니다.



자바스크립트가 작동되는 시점은 HTML 문서에 삽입된 위치에서 작동이 됩니다. 그래서 <head>안에서 외부로 연결된 자바스크립트는 ❶ 위치에서 작동이 되었고, 10번 줄에 있는 자바스크립트 코드는 ❷에서 작동된 점을 주의해서 봅시다. 자바스크립트의 소스가 HTML 문서 어느 부분에서 작동시키는지는 무척 중요한 사항입니다.

또한 <script> 태그가 너무 여러 곳에서 작동이 되면 웹 브라우저가 혼란스러울 수도 있으면 우리가 자바스크립트를 관리할 때도 문제가 생길 수 있습니다.

## 요소의 속성으로 사용되는 경우

```

```

자바스크립트를 요소의 속성으로 사용하는 방법도 있습니다. 이 경우는 이벤트와 자바스크립트를 연결하여 작업하는 경우가 대부분이면 간결한 방법을 사용할 때 많이 사용하는 방법입니다. 하지만 이 방법은 그다지 배우는 과정에서 많이 사용되면 실무에서는 그다지 많이 사용하지는 않습니다.



## 자바스크립트 기본 지식

자바스크립트로 원하는 프로그램을 만들기 위해서는 몇 가지 용어 정리가 필요합니다. 그래서 본격적으로 자바스크립트 공부를 시작하기 전에 우리에게 필요한 용어와 자바스크립트의 서식을 정리해 봅시다.

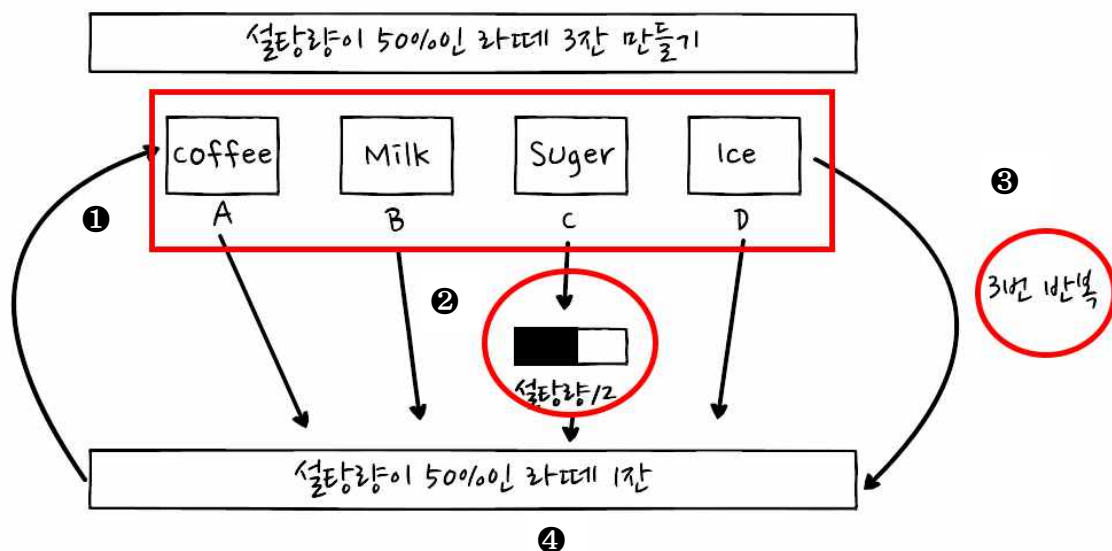
예를 들어 생각해 봅시다. 만약에 우리가 카페 라테를 만들기 위한 프로그램을 만듭시다. 그럼 사용자는 우리가 만든 프로그램을 사용하여 자신이 원하는 카페 라테를 주문합니다.

이 사용자의 요구 내용을 보면 일반적인 카페 라테보다 설탕량이 50% 적은 라테 3잔을 주문한다면 우리가 만든 프로그램에는 이러한 여러 가지 카페 라테 커피를 만들기 위한 여러 준비작업과 그리고 옵션 설정을 준비해 놓아야 합니다.

정리해보면

- 먼저 카페 라테에 필요한 여러 재료를 설정
- 카페 라테를 만들기 위한 기준이 되는 기본 제작 방법
- 사용자의 옵션을 처리하기 위한 계산식
- 작업 제작 명령
- 같은 작업을 여러 번 반복적으로 작업할 수 있는 문장

이러한 부분이 필요합니다.



위에 그림에서 보듯이

- ① 라떼 커피를 만들기 위한 재료방 -> 변수
- ② 옵션을 계산하기 위한 계산식 -> 연산식
- ③ 조건에 맞는 반복적인 작업문 -> 제어문
- ④ 주문버튼 -> 실행문

이러한 여러 파트가 모여서 우리는 프로그래밍을 만들 수 있습니다.

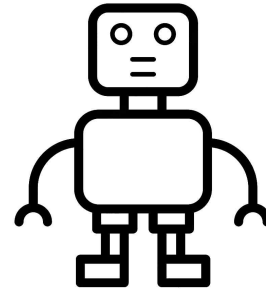
## 자바스크립트 문장

일반적으로 자바스크립트로 명령을 하고자 하면 문장을 만들어야 합니다. 이 부분은 뒤에 객체 파트에서 다시 살펴보겠지만 우리가 필요한 부분을 간단한 정리를 하고 넘어가면 좋을 거 같아요. 자바스크립트 문장이라고 하면 자바스크립트 프로그램에서 한 번의 명령을 말합니다. 가장 기본적인 단위라고 볼 수 있습니다.

예를 들어 살펴봅시다.

우리가 창문을 열려고 합니다. 만약에 사람에게 창문을 열어달라고 해야 한다면 '저 창문 좀 열어 주실래요?'하고 간단하게 말하면 됩니다. 하지만 똑같은 명령을 로봇에 한다면

1. 창문으로 가기	->	1문장
2. 창문에 너의 팔을 올려서	->	1문장
3. 창문 옆으로 밀기	->	1문장
4. 창문에서 팔 내리기	->	1문장
5. 뒤를 돌기	->	1문장
6. 다시 제 자리로 오기	->	1문장



이렇게 여러 단계로 나누어서 명령해야 합니다. 이렇게 단계를 나누어서 해야 하는 명령을 우리는 문장이라고 보시면 됩니다.

우리가 보통 많이 사용하는 문장의 형식은

- 선언식 : `var main = 'text';`
- `오브젝트.메서드( )`;
- `오브젝트.속성` ;

선언식은 우리가 필요한 소스를 미리 설정하고 하는 식이며

오브젝트(객체)는 명령을 하고자 하는 주체 또는 시작점이라고 생각하면 됩니다.

메서드는 활동적인 명령으로 뒤에 '()'가 꼭 붙게 되어 있어요.

속성은 메서드와 같이 필요 오브젝트 뒤에 있지만 '()'가 없습니다.

이러한 형식은 쉽게 생각하면 영어의 문장과 비슷하다고 보면 됩니다.

S (주어)+ V (동사) .	->	객체(오브젝트) . 메서드 ( ) ;
S(주어) + C(형용사) .	->	객체(오브젝트) . 속성 ;

이러한 오브젝트와 메서드 그리고 오브젝트와 속성은 '.'로 연결하는데 체인이 연결되는 모양과 비슷하다고 하여 **체인연결 방식**이라고 얘기합니다. 필요에 따라 여러 체인을 연결하여 사용할

수 있습니다.

우리가 사용하는 일반적으로 사용하는 문장은 ‘.’를 사용하여 문장의 끝을 알려주는 것 같이 자바스크립트에서는 세미콜론(;)을 붙여서 문장의 끝을 냅니다. 만약 문장의 끝은 설정하지 않으면 문장의 끝이 확인이 안되면 자신이 원하는 문장으로 브라우저가 해석하지 않고 엉뚱한 해석을 하거나 오류가 생길 수 있기 때문입니다.

```
document.write("자바스크립트 출력 구문");  
alert('안녕하세요 자바스크립트의 세상 입니다!');
```

\*\* 간혹 문장에서 오브젝트가 생략하는 경우도 있는데 위에 보는 예제처럼 ‘alert( )’라는 메서드는 window 객체에 속하는 메서드 입니다. window 객체는 가장 상위객체이기에 생략하여 사용합니다.

## 자바스크립트 주석

프로그램을 사용시 설명글을 넣고 싶다면 우리는 주석을 사용하면 됩니다. 주석을 모든 프로그램마다 존재하지만 각기 다른 모양을 하고 있습니다.

HTML 문서에서는 ‘<!-- 주석 -->’ 이런 모양이었고, CSS에서는 ‘ /\* 주석 \*/ ’ 이런 모양이었어요. 자바스크립트에서는 두 가지 종류의 주석을 사용할 수 있습니다.

간단한 **한 줄 형식의 주석**을 ‘ // ’을 주석 앞에 붙이면 되고, **여러 줄의 설명을 한 번에 주석**으로 사용하려면 ‘ /\* \*/ ’로 설명글을 감싸면 됩니다. 여러줄의 포인트는 ‘enter’ 키를 사용하여 줄 내림을 한 경우를 말합니다. 설명글이 길어서 에디터 프로그램에서 줄이 내려서 보이는 경우는 ‘enter’를 사용한 경우가 아니므로 한 줄 형식으로 인식됩니다.

형식1	형식2
<pre>/*     설명1     설명2 */</pre>	<pre>//주석문1 //주석문2 //주석문3</pre>

## 결과물 출력하기

우리가 자바스크립트를 사용하여 여러 예제를 풀기 위해서는 필요한 데이터를 입력하고 그 결과물을 확인하는 과정을 거쳐야 합니다. 이 과정이 없다면 우리가 프로그램을 만들기 위해 어떠한 부분에 문제가 있는지 알 수 없고 또 오류가 나오는 게 아니면 결과가 잘 나오는지 알 수 없기 때문입니다. 그래서 본격적으로 자바스크립트를 알기 전에 간단한 출력을 확실한 방법과 입력을 넣는 방법을 미리 점검하고자 합니다. 그럼 출력 방법을 먼저 확인해 봅시다.

### 1. 콘솔창 사용하기

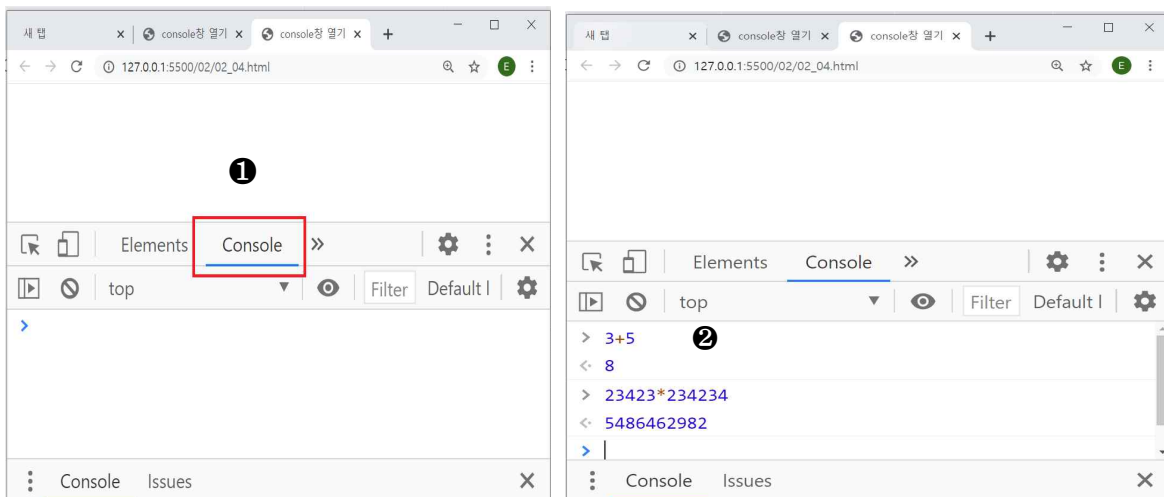
콘솔은 브라우저 안에서 간단한 프로그램을 확인하거나 필요한 결과를 출력할 수 있는 창입니다.

## 콘솔 열기

예제파일(02/02\_04.html)을 열어 크롬 브라우저에서 확인합니다. 아마도 아무 데이터도 없는 빈 화면이 보일 거예요. 이때 [ctrl + shift + J] 를 누르면 화면의 아랫부분에 ❶ 콘솔이 보입니다.

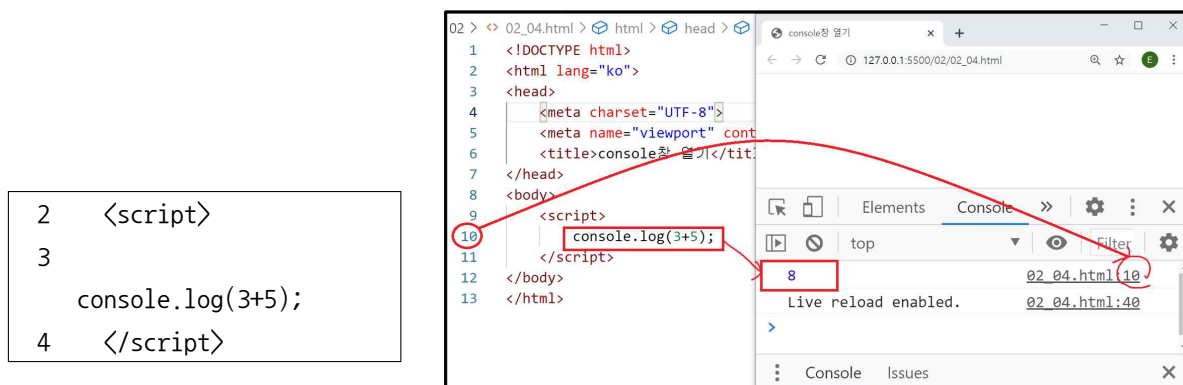
## 콘솔 사용하기

콘솔에 > 에 3+5를 입력하고 엔터키를 누르면 ' < ' 커서 다음에 결과값이 보입니다. 다시 ' > ' 커서에 원하는 계산식을 입력하고 엔터키를 두르면 아래 칸에 ' > ' 커서 다음에 결과값이 보이게 됩니다. ❷



## 콘솔에 출력하기: console.log( ) 사용하기

자바스크립트 내에서 결과값을 콘솔에 보내기 위해서는 console.log( )라는 명령문이 필요합니다. 다시 에디터 프로그램으로 돌아와서 <script> 요소에 다음과 같이 입력하고 다시 브라우저에서 콘솔을 열어 확인해 봅니다.

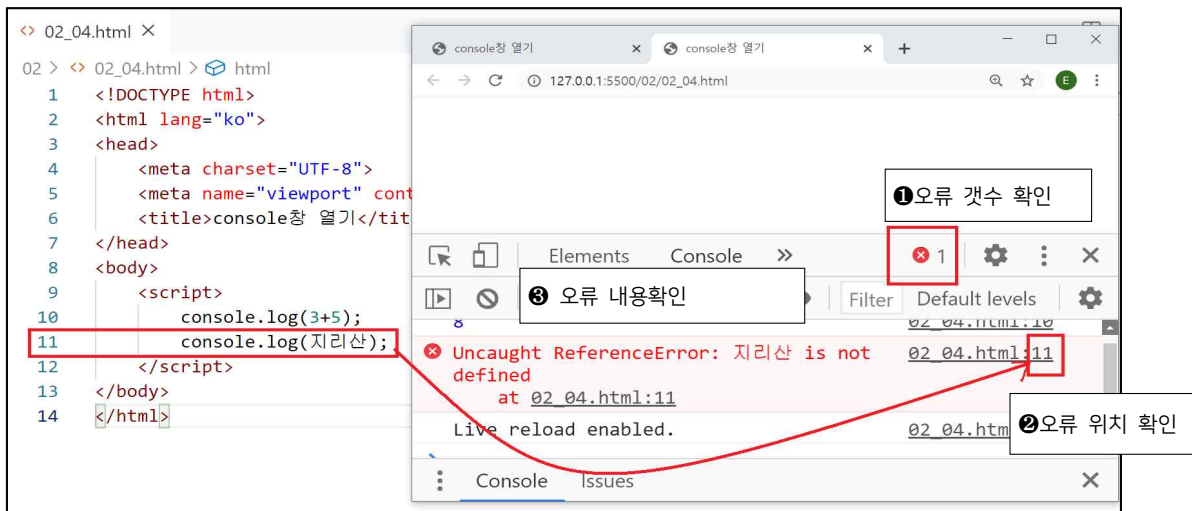


## 콘솔에서 오류 찾아내기

만약에 프로그램을 만들 때 오류가 있다면 프로그램은 작동되지 않습니다. 이때 오류를 쉽게 알 수 있는 곳이 콘솔입니다. 오류로 인해서 프로그램의 작동이 되지 않는다면 우리는 콘솔을 불러

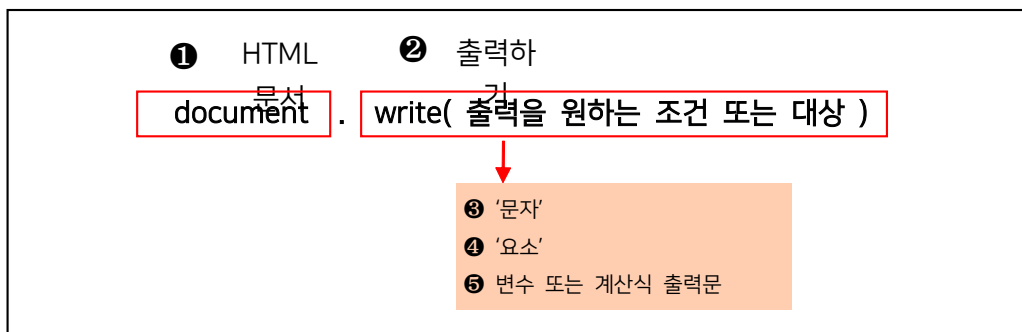
들이면 다음과 같은 상황을 볼 수 있습니다. 콘솔에서는

- ❶ 현재 프로그램에서 가지고 있는 오류의 현황을 개수로 알려줍니다.
- ❷ 오류가 있는 에디터 페이지의 번호를 알려주어 수정할 부분을 쉽게 찾을 수 있도록 합니다.
- ❸ 어떠한 오류가 있는지 그 종류를 알려주어 오류를 쉽게 할 수 있도록 방향을 제시합니다.



## 2. HTML 문서 안에서 확인하기 : `document.write()` 메서드 사용하기

예제를 사용하여 출력값을 확인할 때에 콘솔에 확인하는 방법도 있지만 HTML 문서에 직접 결과값을 출력하는 방법을 알아봅시다. 문장에 직접 결과값을 확인할 때는 '`document.write()`;' 문장을 많이 사용합니다.



- ❶ document : HTML 문서를 말한
- ❷ write()메서드는 결과값은 body요소 안에 심어놓는다는 뜻입니다.  
이때는 출력을 원하는 조건에 따라 ( ) 안에 설정 방법을 달리합니다.
- ❸ 문자 ④ 요소를 출력할 때는 ' ', 또는 " "를 사용합니다.
- ❹ 계산식이나 변수등을 사용할 때는 ' ' 또는 " "를 사용하지 않습니다.

이 부분은 문자형의 데이터형에서 데이터형에 따른 서식 방법과 따옴표의 법칙에서 다시 자세히 배우게 됩니다. 일단 지금은 이렇게 사용한다고 생각하면 됩니다.

3. **alert( )메서드** 사용하기(실습 파일: 02/02\_06.html. 완성 파일: 02/all/02\_06.html)  
alert( )메서드는 우리가 흔히 알고 있는 창을 열어서 원하는 값을 출력하는 역할을 합니다.

## 입력하기

confirm( ) 과 prompt( )를 이용하여 값 전달하기

앞에서 배운 alert( )와 confirm( ), prompt( )메서드는 window 객체 속하는 메서드로 window.alert( )로 사용할 수 있지만 window 객체는 가장 상위객체이기에 우리는 window를 생략하고 사용합니다. alert( )가 브라우저가 사용자에게 값만 전달한다면 confirm( ) 메서드와 prompt( ) 메서드는 브라우저가 사용자에게 답을 요구할 수 있습니다.

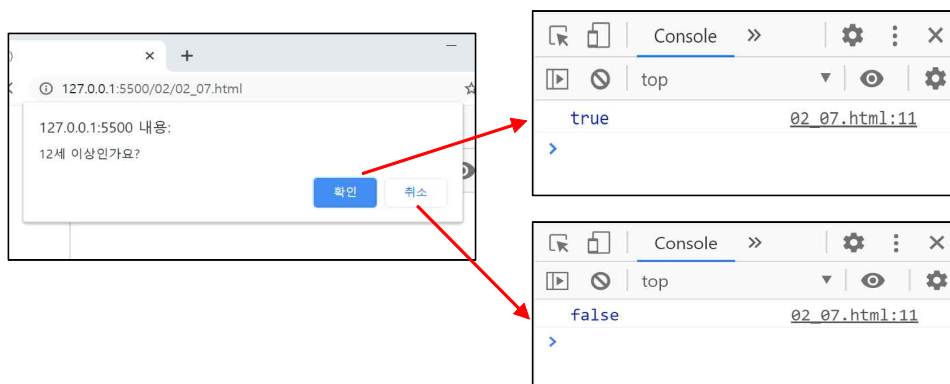
### 1. confirm( ) 메서드사용하기

confirm( ) 메서드는 브라우저가 사용자에게 질문하고 사용자의 답을 기다립니다. 이때 사용자라 할 수 있는 답은 **yes, no** 두 개의 답을 할 수 있습니다. '확인' 버튼을 누르면 'true'의 값을, 취소 버튼을 누르면 'false'의 값을 내보냅니다. 이렇게 'true' 또는 'false'의 값을 내놓는 데이터를 우리는 boolean데이터라고 하는데 데이터형에서 다시 확인을 할 수 있습니다.



```
5 <script>
6     var answer = confirm('12세 이상인가요?');
7     console.log(answer);
8 </script>
```

- ① answer란 그릇에 confirm() 질문의 대답 버튼의 값을 넣으시오.
- ② 콘솔창에 answer의 값을 출력하시오.



## 7. `prompt()` 메서드 사용하기

`prompt()` 메서드는 브라우저가 사용자에게 질문을 하면 사용자는 자신의 답을 보낼 수 있습니다.



```
9    <script>
10      var answer = prompt('나이를 입력하세요?', '숫자로 입력하세요'); ❶
11      console.log(answer);                                             ❷
12    </script>
```

❶ answer란 곳에 `prompt`의 결과값은 넣으시오.

❷ 콘솔창에 `answer`의 값을 출력하시오.

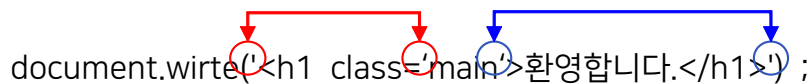
## 2-5 그 밖에 알아두어야 할 자바스크립트 서식

프로그램 서식이라고 하면 프로그래밍언어로 프로그램을 작성할 때 지켜야 할 규칙입니다. 여기에 서는 우리가 자바스크립트 소스를 작성할 때 지켜야 할 규칙을 알아보시다.

### 1. 소스 작성시 따옴표의 법칙을 지킨다.

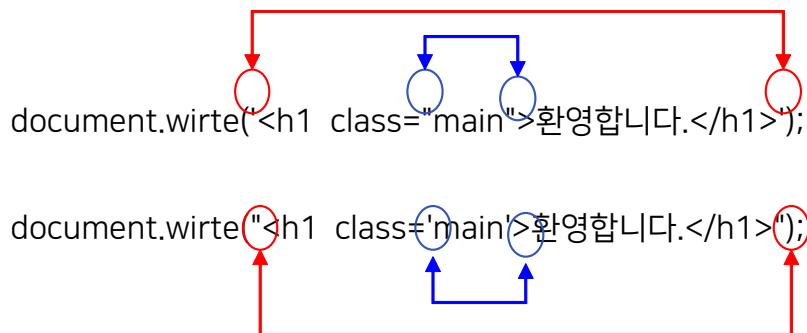
우리가 자바스크립트 소스를 작성하다 보면 데이터 형이나 아니면 요소를 삽입하는 과정에서 홀따옴표(') 또는 쌍따옴표(")를 사용해야 하는 경우를 만나게 됩니다. 따옴표를 사용할 때는 홀따옴표(') 쌍따옴표(") 어느것을 사용하던 문제가 없지만 여러 따옴표를 사용 할때는 따옴표의 법칙을 따라야 합니다. 따옴표를 열어 놓으면 무조건 다음에 만나는 따옴표가 닫는 따옴표로 인식되므로 따옴표 안에 다시 따옴표를 사용해야 한다면 홀따옴표 안에는 쌍따옴표를, 쌍따옴표 안에는 홀따옴표를 사용해야 합니다.

나쁜예



```
document.wirte('<h1 class='main'>환영합니다.</h1>');
```

좋은예



```
document.wirte('<h1 class="main">환영합니다.</h1>');
document.wirte("<h1 class='main'>환영합니다.</h1>");
```

### 2. 대소문자를 다른 문자로 인식하고 사용한다.

자바스크립트는 대소문자를 다른 문자로 인식합니다. 그러므로 같은 명령문이나 사용자 이름을 사용 할 때도 주의를 해야 합니다.

예를 들면 main과 Main은 다른 이름이며, var과 Var는 다른 명령어입니다.

### 3. 사용자의 이름이 필요할 때는 식별자의 법칙에 따라 만들어야 한다.

식별자(identifier)는 우리가 자바스크립트 코드를 만들 때 사용자 이름이 필요한 경우가 있습니다. 변수, 사용자 함수, 사용자 객체 등 사용자가 만들어서 사용할 이름을 지칭합니다. 이럴 때는



식별자 법칙을 지켜서 이름을 만들어야 합니다.

- 한글이나 특별한 나라 언어는 사용할 수 없습니다.
- 알파벳과 숫자, 그리고 특수 문자 중 ‘\_’, '\$’가 가능합니다.
- 첫글자는 숫자를 사용하여서는 안 되며,
- 공백 문자, 예약어 는 사용 할 수 없습니다.

사용 가능한 이름	사용 불가능한 이름
num01	❶ 01num -> 숫자 첫 글자
str01	❷ var -> 예약어
room	❸ num str -> 공간 문자 사용
numRoom	❹ num Room -> 공간 문자 사용
str_01	❺ getElementById -> 예약어
_str	❻ 메인01 -> 한글사용
\$str	❼ '01 -> 사용할 수 없는 특수글자

**\*\*예약어란 이미 자바스크립트에서 먼저 등록한 명령어를 말합니다. 이러한 예약어는 식별자로 사용할 수 없으면 가급적으로 HTML에서 사용하는 class, id 등의 속성명도 사용하지 않는 것이 좋습니다.**

만약에 더 많은 예약어를 알고 싶다면 모질라 개발자 사이트인

'[https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Lexical\\_grammar](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Lexical_grammar)'에서 자바스크립트 버전에 따라 추가된 예약어를 확인할 수 있으며 인터넷 검색을 통해 알아보는 방법도 있습니다.

abstract	arguments	await	boolean
break	byte	case	catch
char	class	const	continue
debugger	default	delete	do
double	else	enum	eval
export	extends	false	final
finally	float	for	function
goto	if	implements	import
in	instanceof	int	interface
let	long	native	new
null	package	private	protected
public	return	short	static
super	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

## 3장. 변수

### 3-1 변수

#### 선언

`const` : 대입이 한번 밖에 되지 않음 = 대입으로 값 변경을 할 수 없음

`let` : 데이터 선언 영역 작업

`var` : 데이터 선언 이름 작업

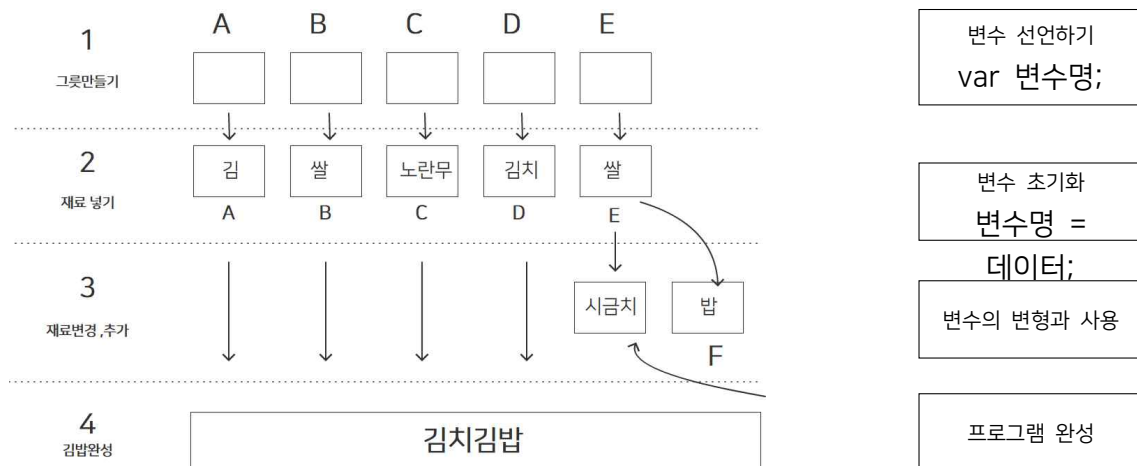
변수를 간단하게 설명한다면 프로그램을 만들 때 필요한 데이터 또는 데이터를 관리하는 시스템 또는 관리체계라 합니다. 예를 들어 생각해 봅시다.



을 거예요.

우리가 김치김밥을 만들어야 한다고 가정해 봅시다.

김치김밥을 만들기 위해서는 많은 재료가 필요하고, 그 재료들을 무작정 사용하는 것이 아니라 재료를 정리하는 그릇에 잘 정리한 다음 김밥을 만들어 나갈 거예요. 우리가 자주 만나는 김밥전문점에서 김밥을 만드는 과정을 생각해 보면 쉽게 이해를 할 수 있



#### 김밥 만들기

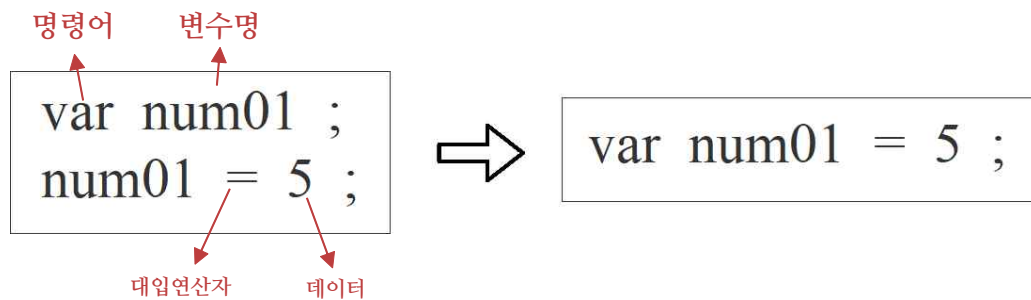
1. 재료를 담은 그릇을 준비한 후
2. 계획에 맞추어 재료를 담는다.
3. 그릇을 사용하는 도중 재료를 변경하거나 필요한 그릇을 추가 할 수 있습니다.
4. 그 재료를 사용하여 김밥을 만듭니다.

#### 변수 만들기

1. 데이터를 담은 변수를 만들어 이름을 붙입니다. (변수의 선언)
2. 만들어진 변수에 데이터를 담습니다. (변수의 초기화)
3. 데이터를 변경을 위해 데이터 바꾸기 또는 추가하기 (변수의 사용)
4. 프로그램을 완성합니다. (프로그램 완성)

그럼 변수 시스템이 이해되었다면 이제 본격적으로 변수를 만들어 봅시다.

## 변수를 만들기



### 변수 선언

변수는 'var'이라는 명령어로 만들어집니다.

먼저 var 명령어를 입력하고 식별자 법칙에 따라 원하는 이름을 넣고 ;(세미콜론)으로 정리하면 변수가 만들어집니다.

### 변수 초기화하기

만들어진 변수에 '='연산자(대입 연산자)를 이용하여 데이터를 넣어 놓으면 초기화가 끝이 납니다. 변수의 초기화는 꼭 변수 선언과 함께 이루어지는 것은 아닙니다. 가끔은 먼저 변수를 만들어 놓고 필요할 때 데이터를 넣을 때도 있습니다.

없는 변수를 불러들이면 에러가 되지만 초기화가 안 된 변수(데이터가 없는 변수)를 불러들이면 'undefined'라는 답이 들어오자 에러는 아닙니다.

### 다수의 변수 생성 방법

여러 변수를 한 번에 생성할 때는 ' , ' (연결연산자)를 이용하여 생성할 수 있습니다.

그럼 이제 실습해 봅시다.

```
10  var sum01, sum02, sum03;    // ' , '를 이용하여 다 수의 변수를 생성한다.
11      sum01 = 3;
12      sum02 = 5;
10      sum03 = '설악산';
11
12  console.log(sum01 , sum02 , sum03);    //console창에서 확인하기
```

```

17   var num01 = 3, num02 = 5, num03 = "라벤더";
18
19   console.log(num01+num02 , num03);           //console창에서  확인하기

```



결과물 확인하기

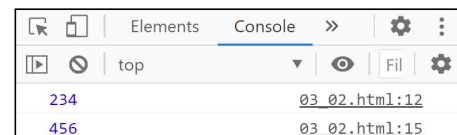
## 변수의 성격

변수에 있는 데이터는 항상 변경 가능합니다. 사전적 의미로 변수는 변화하는 수, 변하지 않는 수를 상수라 합니다. 이처럼 우리가 사용하는 변수도 한번 생성된 후에는 작업에 따라 항상 변경 가능합니다.

```

10   var sum01 = 123;      // 1차 데이터
11   sum01 = 234;
12   console.log(sum01);
13
14   sum01 = 456;          // 2차 변경된 변수값
15   console.log(sum01);  // 출력값이 변경 확인

```



결과물 확인하기

2. 변수를 만들면서 데이터를 즉시 넣을 필요는 없습니다. 만약에 데이터가 없는 변수의 데이터를 불러들이면 에러가 나오는 것이 아니라 'undefined' 데이터형으로 알려줍니다.

하지만 프로그램에 있어서 데이터의 정리는 중요한 부분이기 때문에 필요 없는 변수를 너무 많이 만들거나 변수를 선언해 놓고 잃어버리는 일은 좋지 않기에 항상 계획적인 변수 설계가 필요합니다.

```

17   17 var sum02;           // 값이 초기화 되지 않는 변수
18   18 console.log(sum02);  // 출력값 확인

```

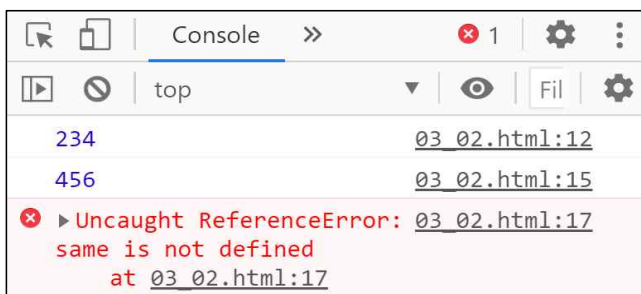


3. 프로그램 시 생성되지 않는 변수의 데이터를 찾으려면 에러가 난다.

작업하던 HTML 문서에 16번 줄에 우리는 선언되지 않는 변수를 불러 봅니다. 출력해 결과를 확인하면 **"same is not defined"** 에러가 나옵니다. 에러가 중간에 나오면 그 이후에 있는 모든

결과물이 출력되지 않는 점도 꼭 확인합니다.

```
14  sum01 = 456; // 2차 변경된 변수값
15  console.log(sum01); // 출력값이 변경 확인
16
17  console.log(same); //선언되지 않는 변수
18
19  var sum02; // 초기화 되지 않는 변수
20  console.log(sum02); // 출력값 확인
```



이름을 설정하여 데이터를 넣어서 작업하는 변수는 자신의 작업하는 영역을 설정하여 작업할 수 있습니다. 대부분이 함수 또는 리터널로 영역이 설정되면 자신의 활동 영역을 제한 또는 확대할 수 있습니다. 변수는 함수 안에서 생성되면 지역함수, 프로그래밍 전체에서 사용할 수 있는 함수는 전역변수라 합니다. 또 중괄호{ } 안에서 작동이 되는 변수를 let, 저장된 데이터값이 변하지 않는 상수를 const로 지정할 수 있습니다.

자세한 내용은 함수를 다루는 장에서 살펴봅시다.

## 3-2 데이터형이란?

우리가 일상생활에서 여러 상품군이 있듯이 프로그램 제작에서 여러 데이터의 종류가 존재합니다.

해조류	김	"텍스트"	string
곡류	쌀	2	number
채소류	시금치	yes	Boolean
육류	소고기	없어요	undefined

계산되는 데이터, 콘텐츠를 나타내는 text 데이터, 참과 거짓 값으로 논리를 펼칠 수는 있는 데이터 등 여러 데이터가 존재합니다.

이제 우리는 변수 안에서 사용되는 여러 데이터형에 간단히 정리하면 아래의 표와 같습니다.

number(숫자)	숫자형으로 사칙연산이 가능한 데이터형
string(문자)	작은 혹은 큰 따옴표로 묶어서 사용되는 문자데이터형
boolean(논리)	true(참), false(거짓)로 값을 가지는 데이터
undefined	자료형을 알 수 없을 때의 데이터 유형(변수의 선언만 이루어졌을 때 나타남)
null	값이 유효하지 않을 때의 데이터형
Infinity	무한 데이터

그럼 각각의 데이터형을 자세히 알아보기 전에 우리는 먼저 데이터형을 알려주는 연산자 'typeof'에 대해 먼저 알아보시다.

### 데이터형을 알려주는 연산자 typeof

자바스크립트를 사용 중 변수에 저장된 데이터형을 알 수 없거나 확인이 필요한 경우에는 typeof라는 연산자를 사용하면 데이터형을 알려줍니다. 그럴 경우가 있냐고요? 물론 단조롭게 변수를 선언하고 초기화를 하는 과정만 있으면 별로 사용할 경우가 없겠지만 복잡하게 변수를 변환하고 여러 메서드에서 사용한 결과값을 가져오는 과정 진행하다 보면 변수의 데이터형이 생각과 다르게 변화될 때가 있습니다. 그럴 때 유용하게 사용할 수 있는 연산자가 'typeof'입니다.

### String형 (문자형)

문자형 데이터는 따옴표에 둘러싸여 있는 문자나 숫자를 의미합니다. 또한 우리가 HTML 요소를 포함하여 출력하고자 할 때도 사용됩니다. 주의할 점은 감싸고 있는 따옴표가 작은따옴표(' ') 또는 큰따옴표(" ") 모두 상관없지만, 따옴표의 법칙에 따라 사용되어야 한다는 점입니다.

```
형식 : var 변수명 = '사용할 문자';  
      var 변수명 = "사용할 문자";  
      var 변수명 = '<h1 class="main">welcome</h1>';
```

## Number형(숫자형)

자바스크립트에서의 숫자형은 단어 그대로 계산 작업이 가능한 정수와 실수를 가리킵니다. 만약에 따옴표는 이용한 숫자(예를 들면 '100' 혹은 "100")는 일반적으로 숫자형에 속하지는 않고 문자형에 속합니다. 하지만 문자형 데이터에 계산이 필요하다면 우리는 이 문자형 데이터를 숫자형 데이터로 바꿀 수 있습니다.

먼저 기본형을 익히고 문자형을 숫자형으로 바꾸는 방법을 알아보시다.

그럼 이번에는 문자형 데이터를 숫자 형 데이터로 바꾸어 봅시다.

```
형식 1 : Number('문자형숫자');  
형식 2 : parseInt('문자형숫자');
```

## Boolean형(논리형)

논리형 데이터는 값이 이제까지의 데이터형과는 달리 어떠한 값이 주어지는 것이 아니라 true (참) 또는 false(거짓) 키워드값만 존재합니다. 주로 데이터들을 비교하거나 논리를 펼쳐서 나오는 참 또는 거짓의 상황을 판단하고자 할 때 사용됩니다.

Boolean 데이터의 결괏값은 문자형 데이터가 아니므로 따옴표를 사용하여서는 안 됩니다. 자바스크립트에서는 프로그램에서는 데이터가 존재하면 true로, 데이터가 존재하지 않으면 false로 인식됩니다. 또 Boolean()메서드를 사용하여 true, false를 출력할 수 있습니다. 메서드 안에 숫자 '0', null, undefined 또는 빈 문자(' ')를 사용하여 false를, 나머지 어떤 문자든 상관없이 데이터가 존재하면 true를 결괏값으로 반환할 수 있습니다.

```
형식: var 변수명 = true, false;  
      var 변수명 = Boolean데이터;  
      var 변수명 = Boolean();
```

## null 과 underfined 데이터

undefined는 데이터형에 속하지만, 데이터형이기보다는 현재 변수의 데이터 상태를 알려주는 역할을 한다. 즉 변수 선언은 했지만, 현재 가지고 있는 데이터가 없는 상태를 알려준다. null은 변수에 저장된 값에 유효하지 않은 상태를 말한다. 우리는 null의 값을 이용하여 현재 변수가 가지



패키지  
1. parcel \*\*  
2. webpack -> react.js

고 있는 값을 비울 때도 사용을 할 수 있다.

```
10 var num ; // 데이터 초기화를 하지 않음
11 console.log(num); // console.log()로 결과값 확인하기
12 var some = 234;
13 console.log(some); // console.log()로 결과값 확인하기
14 some = null; //값 비우기
15 console.log(some); // console.log()로 결과값 확인하기
```

### 3-3 연산자

#### 연산자란?

연산자란 의미와 규칙이 규정된 기호를 말합니다. 예를 들어 봅시다.

아래의 식을 봅시다.

❶ 13 + 5

❷ 10 - 3

❶ 의 답을 물어보면 여러분들은 당연히 18이라고 답을 말할 거예요. 또 ❷의 답을 물어보면 7이라고 답을 할 겁니다. 왜요?

우리는 초등학교 시절부터 '+'란 기호는 기호의 오른쪽과 왼쪽을 더하는 약속을 알고 있기 때문입니다. 이처럼 자바스크립트에서는 여러 기호에 의미와 규칙을 약속하였습니다. 우리는 이러한 약속된 기호를 연산자라고 하고 이 연산자의 대상을 피연산자라 합니다.

var abc = main + 4 ;  
예약어 변수 연산자 피연산자 연산자 피연산자

#### 연산자의 종류

우리가 이제부터 알아볼 연산자는

- |  |  |
|--|--|
| <ul style="list-style-type: none"><li>산술연산자</li><li>증감연산자(단항연산자)</li><li>비교연산자</li><li>조건(삼항)연산자</li></ul> | <ul style="list-style-type: none"><li>대입연산자, 복합대입연산자</li><li>결합연산자</li><li>논리연산자</li></ul> |
|--|--|

입니다.

str+num (X)  
str + num (O) : 공백문자 포함하는 게 맞음!

## 산술연산자

산술연산자는 number형 데이터의 산술 즉 숫자를 계산하도록 약속된 기호들입니다. 우리가 흔히 알고 있는 더하기, 빼기, 곱하기, 나누기, 나머지가 있습니다.

종류	기본형	설명
+	a + b	더하기
-	a - b	빼기
*	a * b	곱하기
/	a / b	나누기
%	a % b	나머지

나누기와 나머지 차이를 알아보시다.

예를 들면 '10 / 3' 과 '10 % 3'의 차이를 알아보시다.

10 / 3 -> 10에서 3을 나누어 나오는 결과값이므로 3.33333이고  
10 % 3 -> 10에서 3을 나누면 값을 3이고 나머지는 1입니다.

## 대입 연산자(=)

대입 연산자(=)는 오른쪽의 작업의 결과물을 왼쪽의 방에 넣는다는 의미로 주로 변수에 값이나 결과값을 넣을 때 사용합니다.

결과 값을 넣는다.

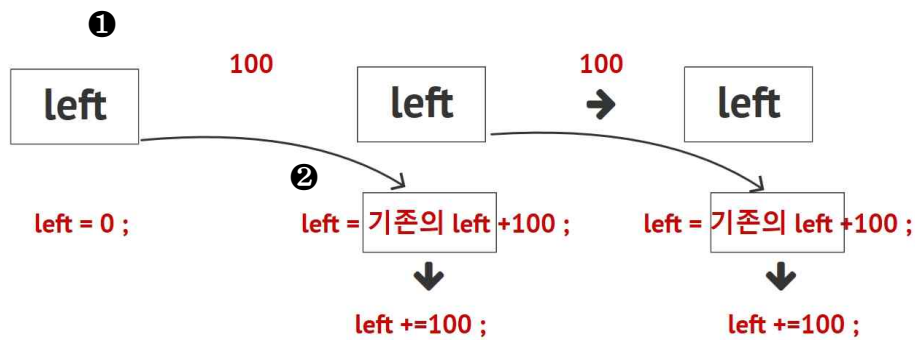
var abc = sub + 3 ;

이때 주의할 점은 우리가 흔히 알고 있는 '='가 오른쪽과 왼쪽이 같다(equal)는 의미가 아니다.

## 복합 대입 연산자 언제 사용 : 기존 메모리 (변수) 변경 -> 메모리 새로운 값 넣기

복합 대입 연산자는 대입 연산자와 함께 산술을 같이 할 때 사용됩니다. 우리가 이 연산자에서 생각할 점을 단순히 같이 진행된다는 간단한 의미보다 언제 우리가 이 연산자를 만나게 되는지를 생각해야 합니다.

예를 들어 생각 해 봅시다.



❶ 변수 left가 있습니다.

❷우리는 이 변수 left에 100을 증가하여 새로운 변수 left 값으로 설정하고 싶습니다.

이것을 종합하면 `left = left + 100`으로 사용 할 수 있습니다.

이를 단축으로 사용하면 `left += 100` 으로 사용합니다.

`left = left + 100 ;` -> `left += 100 ;`

우리는 이를 복합대입 연산자 또는 단축 대입 연산자라고 합니다.

즉 복합대입 연산자를 거의 기존의 변수의 값을 변경하여 새로운 변수의 값으로 다시 설정할 때 많이 사용합니다.

종류	예	설명
=	<code>var x = 5;</code>	변수 x에 5를 대입한다.
+=	<code>x += 3;</code>	변수 x의 값을 가져와 3을 더한 후 다시 변수x의 값으로 설정한다.
-=	<code>x -= 2;</code>	변수 x의 값을 가져와 2을 뺀 후 다시 변수x의 값으로 설정한다.
*=	<code>x *= 4;</code>	변수 x의 값을 가져와 4을 곱한 후 다시 변수x의 값으로 설정한다.
/=	<code>x /= 2;</code>	변수 x의 값을 가져와 2로 나누고 난 후 다시 변수x의 값으로 설정한다.
%=	<code>x %= 2;</code>	변수 x의 값을 가져와 2로 나눴 나머지를 다시 변수x의 값으로 설정한다.

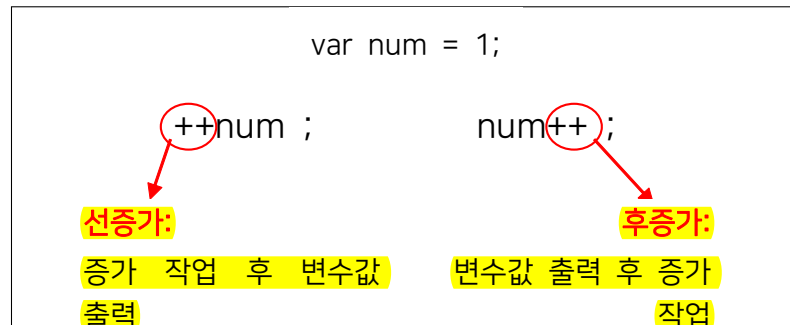
## 증감연산자

증감 연산자는 피연산자가 하나만 필요하여서 단항연산자라고 불리기도 합니다. 숫자 형 데이터를 1씩 증가 또는 감소를 시키는 작업을 합니다.

- 증가 연산자: 데이터를 1씩 증가시킴.
- 감소 연산자: 데이터를 1씩 감소시킴.

이러한 증감 연산자는 데이터의 앞 또는 뒤에 붙여서 작업하는데 이를 선 증가, 후 증가 또는 선 감소, 후 감소라 부릅니다.

증가 연산자로 선 증가와 후 증가의 차이를 알아보시다.



선 증가와 후 증가의 차이는 출력을 하는 시점과 증가를 하는 시점의 차이입니다. 이때 우리가 알아둘 것은 변수명을 만나면 출력을 하고 연산자를 만나면 증가 작업을 합니다.

선 증가

```
var num = 1;
++num;
```

①      ②

① 증가를 먼저하여 num의 값이 2가 됨  
② num의 값을 출력함  
출력값과 변수값이 같음

후 증가

```
var num = 1;
num++;
```

①      ②

① num의 값을 출력함-> 출력값 : 1  
② num의 값을 증가하여 num값 변경  
num값이 2가 됨  
출력값과 변수값이 다름

종류	예	설명
=	var num = 5;	변수 num에 5를 대입한다.
++	++num	현재 num변수값:6 , num변수 출력값:6
	num++	현재 num변수값:6 , num변수 출력값:5
--	--num	현재 num변수값:4 , num변수 출력값:4
	num--	현재 num변수값:4 , num변수 출력값:5

### 결합연산자(+)

우리가 여러 데이터형으로 작업을 하면 다른 데이터형은 같이 선상에서 작업을 할 경우가 많습니다. 예를 들어봅시다.

```
console.log('지리산' 3+4 a) ;
```

문자형      숫자형      변수

이러한 경우에는 자바스크립트는 어떤 데이터형인지 인식하지 못합니다. 이렇게 여러 데이터형을 같이 작업을 할 때 필요한 연산자가 결합연산자입니다.

종류	예
+	문자형 결합: 문자형 데이터와 문자형 데이터를 연결한다.
	좁은 의미 : 문자형과 숫자형을 같이 작업을 할 수 있게 한다.
	넓은 의미 : 여러 다른 데이터형을 연결하여 같이 작업할 수 있게 한다.

먼저 살펴볼 것은 '+' 기호는 우리가 앞에 산술형 연산자로 '더하는' 의미가 있습니다. 먼저 결합 연산자와 산술연산자를 구분하는 방법부터 확인합시다.

## 1. 산술연산자와 결합연산자 구분하기

예를 들면

3 + 5	// 8
산술	
3 + '지리산'	//3지리산
결합	
'설악산' + '지리산'	//설악산지리산
결합	
3 + 5 + '지리산' + 3 + 5	// 8지리산35
① 산술    결합                  결합    결합    ②	

산술연산자를 위한 조건  
1. 양옆이 Number 형  
2. 이전에 결합이 없어야 함

즉 '+' 산술연산자가 되는 조건은에 ①과와 같이 연산자 양옆에 숫자 형 데이터가 오면 산술연산자이지만 만약에 ②과와 같이 연산자 양옆에 숫자 형 데이터 데이터가 오더라도 앞에 결합(또는 문자형)을 만나게 되면 결합연산자가 된다.

## 2. 연결연산자 ' , ' 사용하기

결합연산자인 '+'는 너무 복잡한 수식 안에 사용하다보면 연산자 우선순위에 문제가 일어날 수 있습니다. 이때 우리는 연결연산자인 ' , '를 사용하여 결합연산자 역할을 대신할 경우가 있다.

```

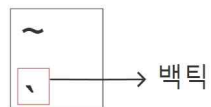
10  var a = 5;
11  var b = 6;
12  console.log('1문항의 답은', a + b, '입니다.');
```

### 3. 백틱사용법(BackTick)

ES6에서는 템플릿 문자열이란 방식이 새로 만들어졌다. 템플릿 문자열은 앞의 여러 데이터 형을 결합이나 연결 연산자로 이용하는 대신 문자열에 수식을 끼어 사용하는 방법이다. 이때의 결합값은 모두 문자형으로 출력되며 이전의 숫자형 또는 다른 데이터형이라도 문자형으로 출력하게 되는 점을 조심해야 한다.

문자형 `${수식}` 문자형

이때 사용되는 백틱은 'quote'가 아닌 '역쿼트:back quote'를 사용합니다. 우리가 사용하는 일반 키보드에서 자판 숫자 '1'자 옆에서 찾을 수 있다.



```

10  var num1 = 5;
11  var num2 = 6;
12  var d = `1문항의 답은 ${num1+num2} 입니다.`;
13  console.log(d);
```

### 비교연산자

연산자 양옆에 있는 두 데이터를 비교하는 연산자이다. 이 연산자는 어떠한 데이터 값이 아닌 Boolean 데이터로 키워드값 true, false가 나오게 됩니다.



먼저 연산자 종류를 알아보자.

종류	설명	설명
$a > b$	a가 b보다 크다	
$a < b$	a가 b보다 작다	
$a \geq b$	a가 b보다 크거나 같다.	
$a \leq b$	a가 b보다 작거나 같다.	
$a == b$	a가 b가 같다.	이때의 같다는 출력 모양이 같다. 그러므로 자료형이 다를지라도 같은 출력 모양이면 true가 나온다. $10 == '10' \rightarrow true$
$a != b$	a가 b가 다르다.	이때의 다르다는 출력 모양이 다르면 true이 나온다. $10 != '10' \rightarrow$ 같은 모양이므로 false $10 != '설악산' \rightarrow$ 다른 모양이므로 true
$a === b$	a가 b가 완전히 일치한다.	이때의 일치는 출력 모양뿐 아니라 데이터형도 같아야 true가 나온다. $10 === '10' \rightarrow$ 같은 출력 모양이지만 데이터형이 다르므로 false
$a !== b$	a가 완전히 일치하지 않는다.	이때의 일치는 출력 모양뿐 아니라 데이터형도 같아야 true가 나온다. $10 !== '10' \rightarrow$ 같은 출력 모양이지만 데이터형이 다르므로 true값이 나온다.

## 논리연산자

논리연산자는 피연산자의 결과 값이 true 아니면 false 상황 이 두 상황을 논리적으로 따지는 연산자이다. 먼저 논리연산자의 종류를 알아보자.

종류	예	설명
&&	$20 < 10 \parallel 10 == '10'$	and 연산자는 연산자 양옆의 두 상황이 모두 true일 때 true 값을 내어 통과시키고자 할 때 사용된다.
$\parallel$	$20 < 10 \&\& 10 == '10'$	or 연산자는 연산자 양옆의 두 상황 모두 false 인 경우에 false 상황을 만듭니다.
!	!num	not 연산자는 true와 false 상황을 반대로 만든다.

## 1. and( && ) 연산자

예를 들어봅시다. 만약에 로그인이라는 시스템을 만들고자 한다면 로그인이 되는 조건을 먼저 설정할 겁니다. 우리가 아는 로그인되는 조건은 먼저 회원 아이디 값이 일치하고 패스워드 값이 일치해야만 로그인 됩니다.

그럼 로그인을 진행하다 보면 나올 수 있는 경우 수를 생각한다면 아래의 표와 같습니다.

값을 일치하는 경우를 true로 일치하지 않는 경우를 false라고 하면 4가지의 상황이 나올 수 있습니다. 이 경우 로그인이 될 수 있는 상황을 아이디의 값도 일치(true), 패스워드값도 일치(true)입니다. 이렇게 모든 상황이 일치(tire)여야만 true을 내는 연산자가 'and' 연산자입니다.

### 로그인하기

통과조건 : id 와 패스워드가 모두 일치해야 된다.

id :

pw :

ID	PW	and (&&)
ture	true	true
true	false	false
false	true	false
false	false	false

로그인할 때 나 올 수 있는 상황

## 2. or ( || ) 연산자

or 연산자는 연산자 양쪽 상황이 모두 false 값이 아니면 true 값이 나오는 상황입니다.

예를 들어 시험을 시험 통과 프로그램을 만들어 봅시다. 먼저 통과 조건을 설정합니다. 이번에는 두 과목의 시험 점수 중 한 과목이라도 70점 이상이면 'true' 값을 내보내 시험을 통과되도록 합니다.

### 시험통과

통과조건 : 두 과목 점수중 한 과목점수라도 70점 이상이면 통과

A과목 :

B과목 :

ID	PW	or (  )
ture	true	true
true	false	true
false	true	true
false	false	false

시험통과 할 때 나올 수 있는 경우 수

## 3. not( ! ) 연산자

not 연산자는 현재 피연산자가 가지고 있는 boolom 값을 반대로 변경합니다. 만약에 'true' 값을 가지고 있다면 'false' 값으로 'false' 값을 가지고 있다면 'true' 값으로 바꾸어 줍니다.

예제로 확인해 봅시다.

### 조건연산자(삼항연산자)

조건연산자는 3개의 피연산자가 필요하여 3항 연산자라고도 불립니다. 조건연산자는 조건을 먼



저 만들고 그 조건에 결괏값 (true, false)의 결과에 따라 실행 값을 출력하도록 하는 연산자입니다.

조건식 ? 조건이 참일 때 출력 실행 값 : 거짓일 때 출력 실행값 ;

## 연산자 우선순위

연산자 우선순위를 설명하기 전에 산술문제를 풀어봅시다. 만약에  $3+4 \times 5$  식을 풀어야 한다면 답은 '35'가 아니라 23입니다. 왜일까요? 산술계산에서는 먼저 'x' 연산자를 한 후에 +연산자를 풀게 되어 있습니다.

$$3 + \boxed{4 * 5} \rightarrow \boxed{3 + 20}$$

1
2

자바스크립트도 여러 연산자가 같은 식에 존재한다면 연산자 우선순위에 따라 해결해야 합니다. 우선 우선순위를 보면 다음과 같습니다.

기능	연산자						
1	괄호	()					
2	증감/논리 연산자 not	++	--	!			
3	산술 연산자 곱셈	*	/	%			
4	산술 연산자 덧셈	+	-				
5	비교 연산자 대소	<	<=	>	>=		
6	비교 연산자 같음	==	===	!=	!==		
7	논리 연산자 and	&&					
8	논리 연산자 or						
9	대입 연산자	=	+=	-=	*=	/=	%=

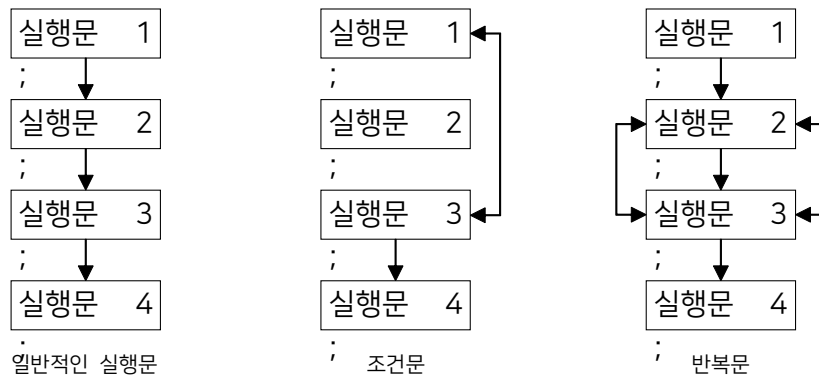
## 4 제어문

조건문 : if  
 선택문 : switch  
 반복문 : for (for in / for of) / while  
 기타문 : break / continue / return

### 4-1 제어문이란

보통 프로그램의 실행 순서는 굉장히 중요합니다. 어떠한 순서로 진행되느냐에 따라 결과값이 달라 나올 수도 있으면 원하는 작업과 다른 방향으로 진행이 필요할 수도 있습니다. 일반적으로 프로그램의 실행 순서는 작성된 순서에 따라 위에서 아래 방향으로 차례대로 실행이 됩니다.

하지만 우리가 프로그램을 제작할 때에는 조건에 따라 실행 순서를 바꾸거나 반복적인 실행을 하도록 프로그램의 흐름을 제어하는 문장을 제어문이라 합니다.

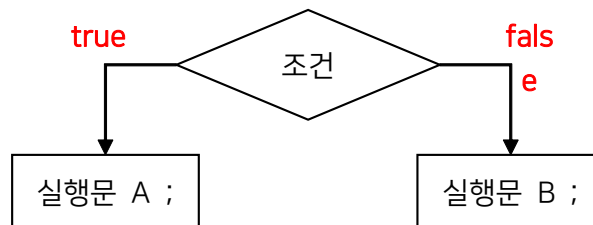


제어문의 종류를 크게 나누어 정리하면 아래의 표와 같습니다.

	설명	종류
조건문	조건에 따라 실행문을 선택할 경우	if문 / if else문 / if else if 문
선택문	선택된 값에 따라 실행문이 결정되는 경우	switch문
반복문	지정된 조건에 따라 실행문을 반복적으로 실행하는 경우	for문 while문 / do while문
기타문	반복을 건너뛰거나 실행을 멈추는 경우	break 문 continue 문

## 4-2 조건문 (if문 / else 문 / if else문)

조건문은 조건식에 나오는 결과값이 true, false에 따라 실행문을 제어하는 경우이다.



### if 문:

조건식의 값이 참(true)값일 경우 실행문을 실행할 때 사용된다.

형식 :

```
if (조건식) {조건이 true 값일 경우 실행문 ;}
```

### if~else 문:

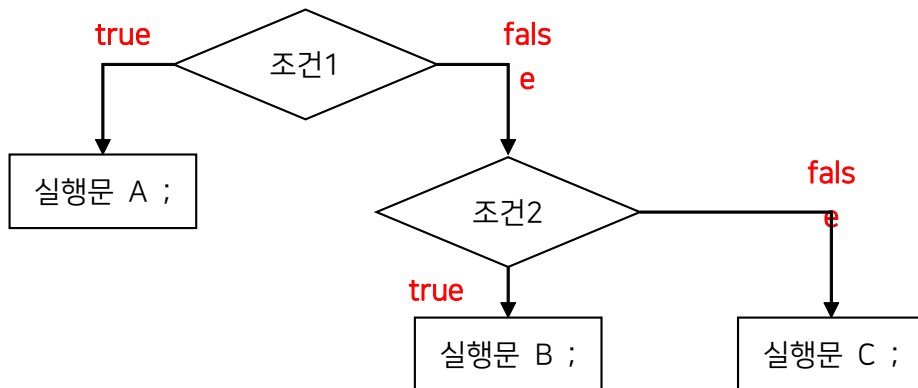
조건식의 값이 참(true)값일 경우 또는 거짓(false) 값일 경우에 실행되는 실행문이 다를 경우

형식 :

```
if (조건식) {
    조건이 true 값일 경우 실행문 ;
} else{
    조건이 거짓일 경우 실행문 ;
}
```

### if~else if 문

처음 제시한 조건 1식의 값이 참(true)값인 경우의 실행문을 정한 후 거짓 값의 경우 수를 다시 모아서 재차 조건을 걸어 참일 경우 거짓일 경우를 나누어 다시 실행문을 실행할 때 사용됩니다. 이때 1차 참인 경우를 제외한 모든 경우 수는 여러 번의 조건을 걸쳐 실행문을 찾을 수 있게 할 수 있습니다.



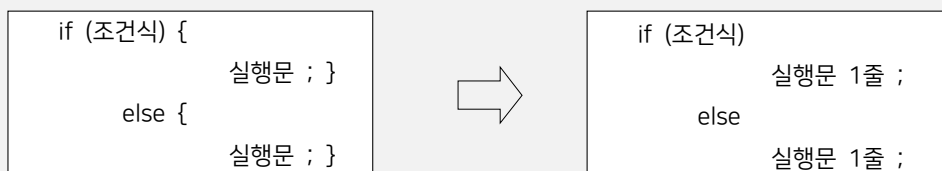
형식 :

```

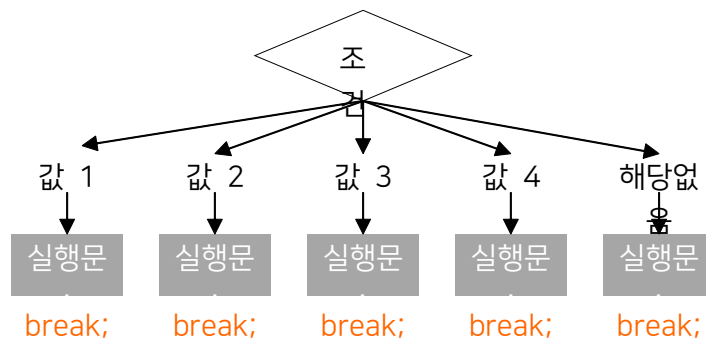
if (조건식 1) {조건1이 true 값일 경우 실행문 ;}
  else if(조건 2){조건2가 true 값일 경우 실행문 ;}
  else if(조건 3) { 나머지 false경우의 실행문 ;}
  else if(조건 4) { 나머지 false경우의 실행문 ;}
  .....
  else{나머지 조건에서 해당되지 않는 경우의 실행문}
  
```

!! 알고 갑시다

자바스크립트의 프로그래밍을 하다보면 괄호 종류가 많이 나옵니다. 대괄호[ ] , 중괄호{ } , 소괄호 또는 괄호 ( )들입니다. 각 각의 괄호들은 자신의 역할이 어느 정도 정해져 움직이게 됩니다. 이중의 중괄호{ }는 실행문들을 그룹형으로 묶는 역할을 많이 합니다. 그래서 제어문에서 나오는 중괄호{ }는 사용해야 하는 실행문을 묶는 역할이기 때문에 생략 가능합니다. 만약에 실행문이 한 줄일 때 중괄호{ }는 생략해도 됩니다.



## 4-3 선택문(switch문)



선택문인 switch문은 다양한 조건이 제시되었을 때 선택된 값에 따라 실행문을 선택하여 실행하도록 하는 제어문입니다. if~else if문과 비슷한 성격이 있지만 여러 조건이 많은 경우에는 if~else if문 보다는 switch문이 더 효과적입니다.

먼저 형식을 살펴봅시다.

형식 :

```
①
switch ( 조건 값 ) {
    ②
    case 값1 : {실행문 그룹 ; }
        break; ④
    case 값2 : {실행문 그룹 ; }
        break;
    case 값3 : {실행문 그룹 ; }
        break;
    default : {실행문 그룹 ; }
    ⑤
}
```

❶ 조건 값에는 대부분 변수의 이름이 옵니다. 이 변수의 값을 가져와서 case 문에 설정된 값에 가져온 값을 대조하여 같은 값일 경우 다음에 오는 실행문을 실행하게 됩니다.

❷ 대조할 값을 case 문에 설정합니다. 이때는 설정값의 데이터형을 문자형과 숫자 형에 맞추어 작업해야 합니다. (문자형은 따옴표로 묶고 숫자 형은 따옴표 없이 설정합니다.)

❸ 실행문의 그룹으로 중괄호{ }을 묶어 사용합니다. 만약에 실행문이 한 줄이 경우는 중괄호{ }를 생략할 수 있습니다.

❹ switch 문에서는 선택된 실행문 실행 후 switch 문에서 빠져나오기 위해서는 각 각의 실행문 뒤에는 break 문을 넣어 주어야 한다. break 문이 실행이 되면 지금 작업하는 switch문의

위치에서 switch문의 중괄호{}에서 빠져 나와 다음 작업으로 넘어갈 수 있습니다.

⑤ case문의 설정된 값에 해당하는 값이 없는 경우에도 원하는 작업을 지시하고 싶다면 default 문으로 실행문을 설정 할 수 있습니다. 이처럼 예외 사항이 필요 없다면 default 문을 생략해도 됩니다. 또 default 문에는 break 문을 사용할 필요 없습니다. 이 실행문이 실행되면 자동으로 switch 문에서 나오기 때문입니다.

## 4-4 반복문 : for문

반복문은 같은 원리를 가진 실행문을 조건이 성립 될 때 까지 계속 반복적으로 실행하도록 하는 제어문입니다. 반복문에는 작업 환경에 따라 for문과 while문으로 나눌 수 있는데 for문을 먼저 알아보도록 합시다.

for문의 특징은 실행하고자 하는 실행문이 반복적으로 진행할 수 있는 조건을 외부에서가 아닌 for문에서 설정할 수 있습니다.

먼저 형식과 작동 원리를 알아보시다.

### 형식 완성하기

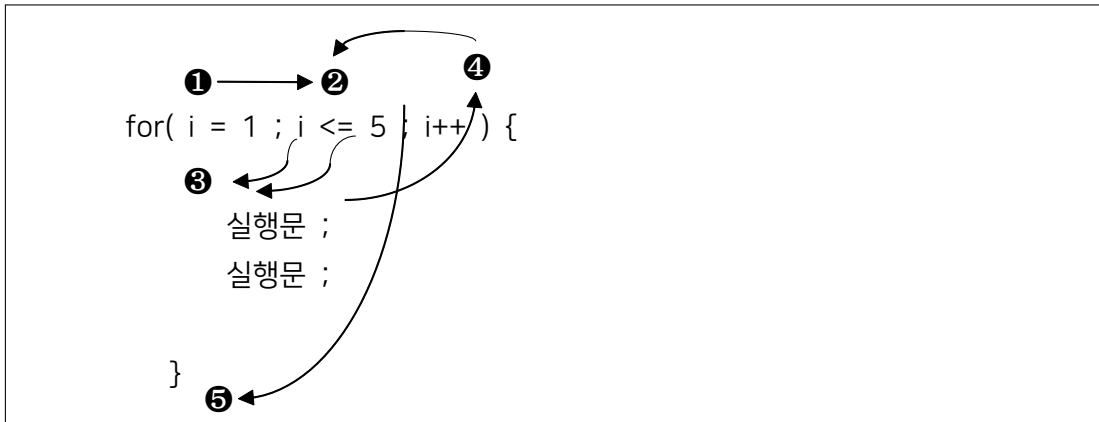
```
for(❶ 시작값설정 ; ❷목적값 설정 ; ❸증감식설정 ) {  
    실행문 ;  
    실행문 ;  
}
```

1. 반복할 준비는 하는 지점으로 반복 카운트의 시작 값을 설정합니다. 대부분 변수를 사용하여 지정합니다.  
초기카운트 파트라고도 불리우며 대부분 변수 i , j , k 순으로 결정합니다.  
원칙은 var 명령어를 사용하지만 대부분 생략하고 변수명만 사용합니다.
2. 반복할 횟수를 점검할 식을 설정합니다.  
대부분 true, false값을 출력하여실행문을 실행할 조건인지를 검토할 수 있게 합니다.
3. 시작 카운트에서 점검식까지 갈수 있도록 증감연산자를 이용해 카운트를 생성합니다.

예를 들어 우리가 원하는 실행문을 5번 실행하고자 합니다. 반복 카운트의 시작은 1에서 시작하여 5일 수도 있고 0에서 시작하여 4까지 알 수도 있습니다. 그런 작업자가 결정하면 됩니다.

- ❶ 우리는 1에서 5까지 카운트하기로 합시다. 그럼 'i = 1'로 설정하고
- ❷ i의 값이 5 이상이 되면 안 되기 때문에 'i <= 5'로 설정합니다.
- ❸ 실행문이 반복하면서 i의 값이 증가하도록 만들어야 하므로 증감연산자를 사용하여 'i++'로 설정합니다.

그럼 다시 정리하면



❶ i에 i를 대입하고 ❷로 넘어가 조건에 맞는지 확인합니다.

❷의 조건에서 true 값이 나오면 ❸의 실행문을 실행합니다.

❸의 실행문이 끝나면 ❹증가 식으로 가서 i의 값을 증가합니다. 이제 i의 값은 2가 된다.

다시 ❷의 조건으로 가서 조건식에서 true 값이 나오면 ❸의 실행문을 실행하고 ❹증가 식으로 갑니다.

i의 값이 ->3

i의 값이 ->4

i의 값이 ->5

가 될 때 까지 실행문을 실행하고 ❹의 증가식으로 가서 i의 값을 증가합니다.

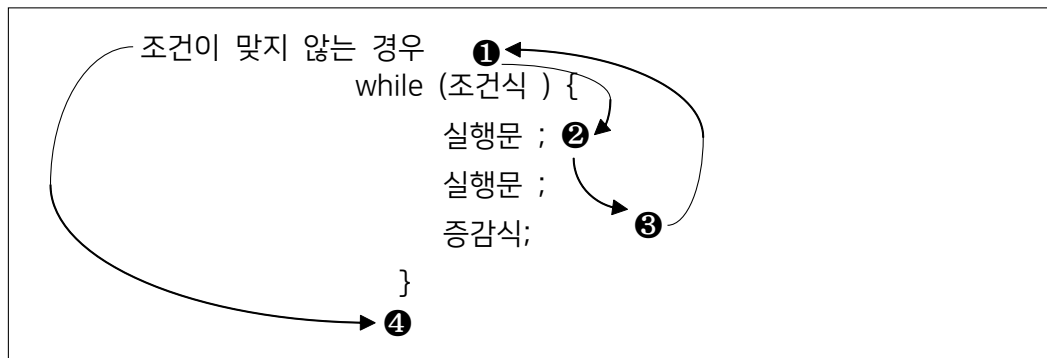
i의 값이 ->6이 되면 다시 ❷의 조건식으로 갑니다. 조건식에서 false 값이 나오면 이제 ❺ for 문에서 빠져나옵니다.

## 4-5 반복문 : while문

자바스크립트에서 반복문은 for문 이외에 while문과 do while문이 있습니다. for문은 for문 내에서 작동되는 카운터에 의해 실행문이 반복된다면 while문과 do while문은 반복문 외부에서 기준의 값을 가져와 특정한 조건이 달성 될 때까지 실행문이 반복된다는 점이 다릅니다.

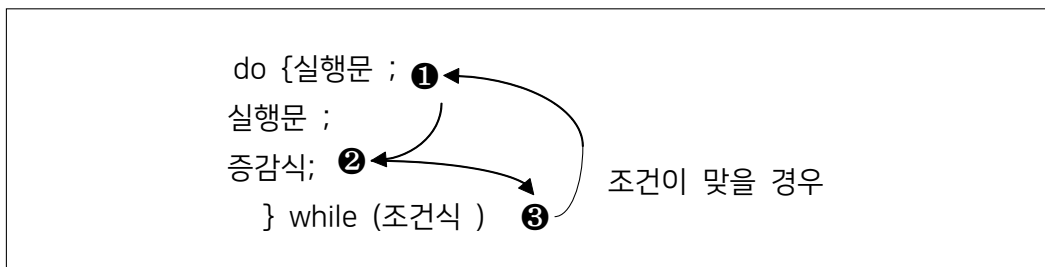
먼저 while문 형식과 작동 원리를 알아보시다.





1. while()의 조건식❶ 에서 반복문이 실행될 조건을 설정합니다. 이때 조건식에는 true 또는 false 가 나올 수 있는 비교연산자가 많이 사용됩니다.
2. while()의 조건식에서 true 값이 나오면 반복문의 실행문 파트로 가서 ❷실행문을 실행 후
3. ❸ 증감 식으로 가서 조건을 증가 또는 감소합니다.
4. 다시 ❶ while문의 조건식으로 가서 조건이 성립되는지 확인 후 다시 ❷실행문 파트로 가서 실행 후 다시 ❸증감 식으로 갑니다. 계속 조건이 성립될 때까지 반복 후 조건식에서 false 값이 나오면 while 문에서 나와 ❹다음의 작업을 실행합니다.

## do-while문



1. ❶ do{ } 문에서 먼저 실행문을 실행합니다.
2. ❷ 실행문 마지막에 증감 식으로 가서 조건을 증가 또는 감소합니다.
3. ❸ while 문의 조건식으로 가서 조건이 성립되는지 확인 후 조건이 true이면 다시 ❶ do문을 실행합니다. 만약에 조건이 false이면 do-while 문에서 빠져나옵니다.
7. do-while 문이 while 문과 다른 점은 조건이 참이 아닌 경우라도 무조건 한번 실행문이 실행된다는 점입니다.

## 5장. 함수

### 5-1 함수란?

이번 장에서 공부할 내용을 함수입니다. 그럼 함수가 무엇인지부터 알아보도록 하죠. 함수는 우리가 사용하는 스마트폰의 단축번호라고 생각하면 됩니다.

그럼 함수를 알아보기 전에 우리가 핸드폰 단축 번호 사용법을 알아보시다.

핸드폰에서 단축번호 사용을 하고자 하면

1. 단축번호 등록하기
2. 단축번호 사용하기

로 나누어서 생각할 수 있습니다.

그럼 단축번호 등록 생성부터 생각해 봅시다.

단축번호를 등록하려면

1 사용 할 번호를 선택 후-> 2 이름을 입력 -> 3 단축번호로 사용할 번호를 입력하고 설정을 완성하면 됩니다.



이렇게 설정이 끝나면 우리는 단축번호를 사용할 준비가 되었습니다.

그럼 이번에는 단축번호를 사용해 봅시다. 단축번호 사용방법은

1. 사용할 단축번호를 선택한다. -> 2. 통화버튼 누르기-> 3. 등록된 번호로 발신한다.

이처럼 단축번호 시스템은 한번 설정하면 우리는 그 단축번호를 선택만 하면 언제든지 등록된 번호로 발신을 할 수 있기에 매번 발신할 번호를 누르지 않아도 쉽고 정확하게 원하는 연락처로 발신을 할 수 있습니다.

프로그램을 만들 때도 마찬가지입니다. 우리가 자주 사용하는 기능 있다면 그 기능을 등록해 놓으면 매번 그 기능을 작성하지 않아도 사용할 수 있습니다. 이러한 기능을 우리는 함수 (function)이라 합니다.

1. 프로그램상에서 자주 사용하는 기능을 함수로 만들어 사용하면 같은 코드 내용을 여러 번 작성하지 않아도 사용할 수 있어 효율성이 높습니다.
2. 사용하는 기능을 함수로 만들어 정리하여 사용하면 프로그램의 정리가 단결하고 기능별로 시작과 끝점을 구분하여 정리할 수 있어 충돌의 위험이 없어집니다.

```
graph LR; A((A상품)) --> D[제품의 원가 + (제품의 원가 x 0.1)]; B((B상품)) --> D; C((C상품)) --> D; D --> E[제품의 가격];
```

1. 사용자가 필요해서 만드는 함수 즉 사용자 정의 함수를 사용하거나
2. 브라우저에 내장된 자바스크립트 내장함수입니다.

## 43

## 5-2 기본 사용자 정의 함수 만들기

사용자 함수는 함수는 만드는 방법에 따라, 그리고 사용하는 방법에 만드는 방법이 달라집니다.

1. 이름 있는 함수
2. 이름 없는 함수
3. 함수식

이번 파트에서는 함수의 기본형인 이름있는 함수를 이용하여 함수를 만드는 방법과 사용하는 방법을 알아보시다. 그리고 4-5장에서 여러 사용자 함수를 만들어 사용하는 방법을 다시 정리해 보도록 합시다.

우리가 변수 만들 때를 생각해 봅시다. 변수는 먼저 변수를 선언하여 초기화를 하면 변수를 사용할 수 있었습니다.

함수도 마찬가지입니다. 우리가 함수를 사용하려면 먼저 함수를 만들어서 선언을 먼저 하고, 그 다음 실행문을 등록하면 하면 됩니다. 먼저 형식을 살펴봅시다.

### 1. 함수 선언하기

```
    ②  
    ①function 함수의 이름( ③ ){  
        기능 실행문 ; ④  
    }
```

- ① 함수는 function이란 명령어로 시작합니다.
- ② 함수의 이름은 식별자의 규칙에 맞추어 작성하면 됩니다.
- ③ 매개변수는 함수에 필요한 변수값을 받는 공간으로 필요가 없어도 꼭 ( )는 생략할 수 없습니다.
- ④ 함수에 필요한 실행문은 중괄호 { }로 묶어서 그룹을 만듭니다.

### 2. 함수 사용하기

함수를 사용하려면 만들어진 함수의 이름을 불러들여 사용합니다.

```
    ① 함수 이름 ( 인수② );
```

- ① 앞에서 선언한 함수의 명을 사용하면 됩니다.
- ② 함수 안에서 작동할 변수들의 필요 값(인수값)을 넣으면 함수에서 알아서 변수의 값을 끌어다 사용할 수 있습니다. 우리는 이 값을 인수(argument)라 합니다.

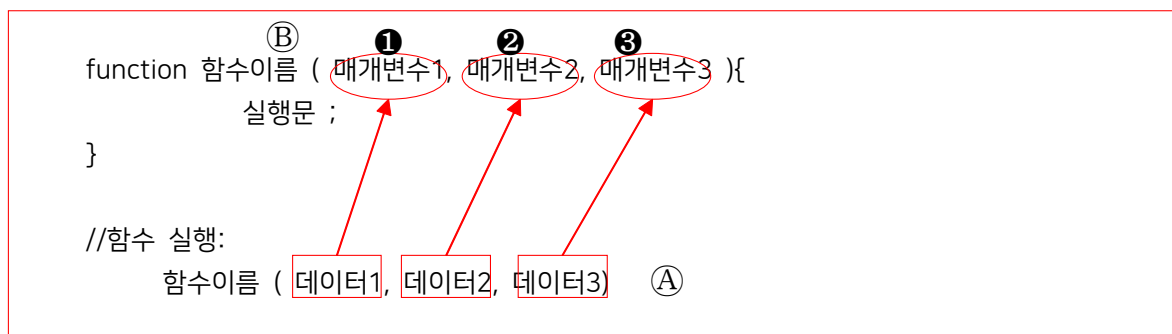
## 5-3 매개변수와 return문

우리가 사용자 정의 함수를 사용할 때에 함수의 이름만 불러 필요한 기능을 실행시키는 것뿐 아니라 기능에 실행하기 위해서는 데이터가 필요할 수가 있습니다.

일반적인 명령어로 예를 들어보면, 우리가 write( )란 명령어를 사용할 경우 '( )'에 무엇을 출력할지를 설정해야 합니다. 안 그러면 자바스크립트는 무엇을 출력할지 모릅니다. 또 우리가 계산기 프로그램을 만들고자 한다면 우리는 계산에 필요한 수를 입력받아야 계산기 프로그램을 실행할 수 있습니다. 이처럼 함수를 만들어 사용하고자 할 때 데이터가 필요한 경우 데이터를 같이 전달하여 함수를 호출 하는 통로를 우리는 매개변수와 인수라 합니다.

### 매개변수와 인수

먼저 매개변수와 인수의 기본 형식과 실행 원리를 살펴봅시다.



④ 함수를 호출할 때 '( )'안에 데이터를 입력한다. 이 경우 여러 개의 데이터가 필요하면 ', '를 이용하며, 이 때 우리가 보내는 데이터를 인수라 합니다.

⑤ 함수 선언 시 필요한 데이터를 받을 변수의 이름을 만들어 놓습니다. 이 경우에는 함수 호출 시 보내는 인수의 순서에 맞추어서 받는 변수를 ', '로 연결하여 생성하며, 이를 매개변수라 합니다. 매개변수는 데이터값의 연결통로로만 사용하며 가인수 함수 호출 할 때 보내는 인수를 실인수라 합니다.

```

12  <script>
13  //함수 만들기
14  function add(num01, num02){           ❶
15      ❷ var hap = num01 + num02;         ❸
16      document.write(hap + '<br>');      ❹
17  }
18  //함수 호출하기
19      add(2,3);                         ❺
20      add(45,3463);
21      add(123685,45690);
22  </script>

```



결괏값 확인

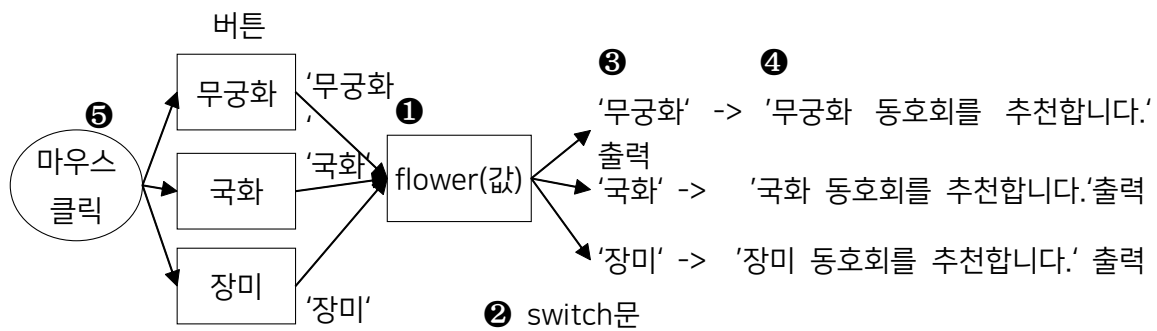
실습을 하나 더 해보죠.

이번의 실습은 버튼을 클릭하여 함수를 불러들여들이는 방법을 살펴봅시다. 이벤트는 이번 장에서 다루는 내용은 아니지만 우리는 간단한 이벤트 핸들러를 사용하여 함수 불러들이기와 매개변수를 사용하는 법을 알아보죠. 우리가 '흔히 버튼을 클릭하면'이란 표현을 한다면, 자바스크립트는 버튼에 '마우스 이벤트 클릭을 발생'하면 이라고 해석하여 받아들입니다.

마우스를 클릭한다는 것은 이벤트를 발생하면 실행하고자 하는 작업이 있다는 말과 같습니다. 아무 이유 없이 그냥 클릭 이벤트를 사용하지는 않을 거예요. 이러한 이벤트로 원하는 작업을 연결하기 위해서는 실행문의 그룹인 함수가 필요합니다. 즉 마우스를 클릭하면 마우스 이벤트가 발생되고 요소에 on+'이벤트이름'을 설정하여 함수를 연결하면 됩니다.

이번에 만드는 함수는 버튼을 클릭하여 오는 매개변수의 값에 따라 동호회를 추천하는 시스템입니다.

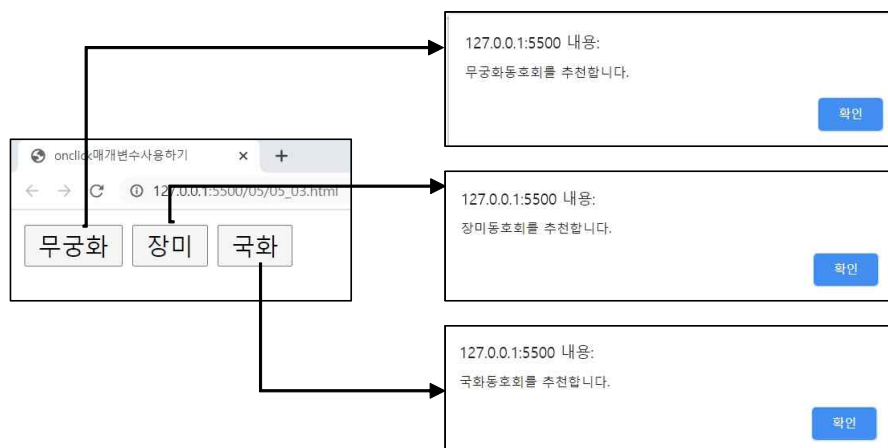
- ❶ 먼저 함수의 이름을 flower라고 하고 값을 받을 매개 변수의 이름을 flowerName이라고 합니다.
- ❷ flowerName의 값에 따라 출력문이 달라지므로 우리는 switch문을 사용해 봅시다.
- ❸ case에 매개변수에 오는 값을 설정하고
- ❹ 각각의 실행문에 alert( )명령어로 출력값을 설정합니다.
- ❺ 각 버튼요소에 onclick핸들러를 만들어 flower함수와 함수로 보낼 매개변수 값을 설정합니다.
- ❻ 값이 잘 나오는지 확인해 보세요.



```

9  <script>
10  function flower(flowerName){①
11      switch(flowerName){②
12          ③ case '무궁화':alert('무궁화동호회를 추천합니다.');

```

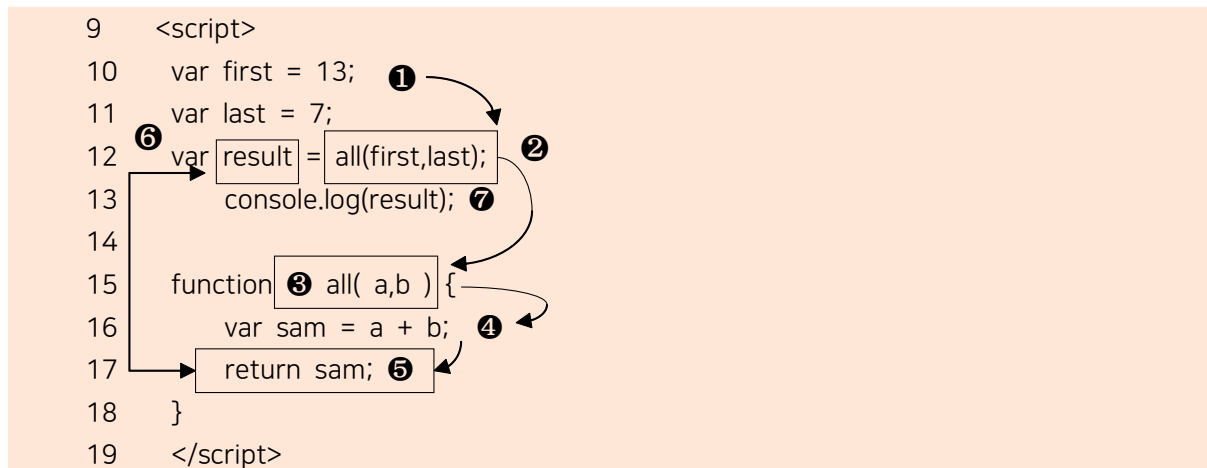


return문

return 문은 함수에서 결과값을 함수 밖으로 넘기는 역할을 합니다. return 문이 실행되는 위치를 항상 함수 실행문의 가장 마지막에 사용하여 함수의 결과값을 외부로 반환한 후 프로그램의 포커스를 함수 밖으로 나오게 합니다.

만약에 return 문이 함수 실행문 중간에 나오게 되면 우리는 return 문 다음에 오는 실행문은 실행이 되지 않는 상태에서 함수를 끝나게 됩니다.

(실습 파일: 04/04.html. 완성 파일: 04/all/04.html)



- ❶ 인수로 사용할 변수를 설정합니다.
- ❷ 변수값을 가지고 함수 all을 불러들이기를 합니다.
- ❸ 함수 all의 실행문으로 넘어가 매개변수 a,b에 인수 값을 대입한 후
- ❹ 변수 sam에 실행문의 값을 대입합니다.
- ❺ 함수 all의 결과값 sam을 함수 외부로 반환합니다.
- ❻ 반환한 값을 변수 result에 저장한 후
- ❼ result의 결과값을 console에 출력합니다.



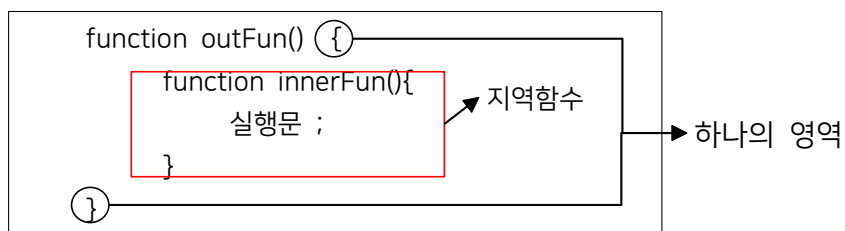
## 5-4 스코프 알아보기

### 함수 스코프(scope)

함수 스코프란 영역에 관한 이야기입니다. 영역이라고 하면 우리는 전역과 지역으로 나누어서 생각할 수 있습니다. 하나의 함수가 생성되면 전체 프로그램 안에서 자신만의 지역이 만들어진다고 보면 됩니다.

아래의 그림을 보고 생각해 봅시다. 우리나라를 전체 프로그램이라고 하면 경기도는 경기도 함수, 경상도라는 지역은 경상도 함수로 생각하면 됩니다.

또한, 지역 안에서 자식 지역을 만들어서 기능을 묶어서 사용할 수도 있는데 이러한 함수를 지역 함수라 합니다.

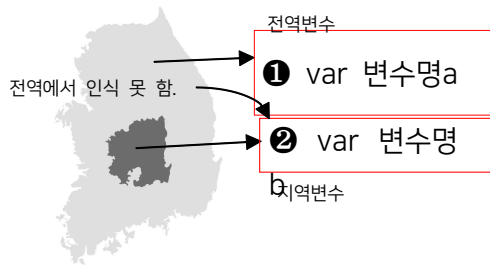


이러한 영역의 문제에서 가장 많이 신경을 써야 할 부분이 바로 변수들입니다. 우리는 지금까지는 변수를 생성할 때 'var'라는 명령어로 변수를 선언하였지만, 함수가 생성되어 영역이 만들어지면서부터 우리가 사용할 수 있는 변수 종류와 체계가 더욱 넓고 복잡해집니다.

그럼 먼저 전역변수와 지역변수를 알아보시다.

### 전역변수와 지역변수

전역변수는 프로그램 전역에서 모두 사용 가능한 변수 말하며 이제까지 우리가 사용한 변수는 전역변수입니다. 지역변수는 변수가 만들어진 지역 안에서만 사용 가능한 변수를 말하며,



```
var 변수명a; → ❶ 전역변수
function 함수명(){
    var 변수명b; → ❷ 지역변수
}
```

❶ 은 전역변수를 프로그램 내의 어느 지역이든 변수 사용 가능합니다. 하지만 지역 내에서만 들어서 변수❷는 만들어진 지역 이외에는 인식이 안 되어 사용하지 못합니다. 이러한 변수를 지역변수 부릅니다.

## 함수 안에서 변수 만들기

함수안에서 변수를 만들 경우 우리는 var라는 명령어를 빼고 변수를 선언하면 됩니다.

```
function 함수명 ( ) {
    var a = '설악산';    -> ❶ 지역변수
    b = '지리산';        -> ❷ 전역으로 사용할 수 있는 변수
}
```

❷의 경우처럼 변수를 선언할 때 var라는 명령을 빼고 변수를 선언하면 우리는 그 변수를 전역 변수처럼 사용할 수는 있습니다. 하지만 이때는 변수의 존재를 먼저 인식되어야 하므로 선언된 함수를 꼭 불러들여서 사용한 후에 가능합니다.

## const 와 let 사용하기 (ES6)

변수란 필요한 데이터를 저장하고 상황에 맞추어 변화하며 사용합니다. 하지만 우리가 고정된 값을 지정하여 사용하고자 하면 const를 이용해 상수를 선언하는 방법을 알아봅시다.

### 1. const 만들기

```
9  <script>
10  ❶ const my = 5;
11      console.log(my);
12  ❷ my = 7;           //에러가 난다.
13  </script>
```

- ❶ const로 상수를 넣을 그릇을 초기화를 하였습니다.
- ❷ 변수 my의 값을 변경하고 console에서 확인하면



12번째의 줄에 type 에러가 나오는 것을 확인할 수 있습니다.

## 2 블록변수 let 사용하기

ES6 버전 부터는 '블록 변수'라는 새로운 변수 체계가 추가되었습니다. 먼저 블록이라고 하면 우리가 사용하고 있는 중괄호{ }로 묶여 있는 영역입니다. 우리는 이미 이러한 실행문 그룹의 영역을 이미 알고 있습니다. if문, for문 등 여러 제어문에서도 사용해 보았고 함수에서도 사용해 보았습니다.

블록 변수는 let이라는 명령어로 변수를 선언하여 만듭니다. 선언된 let 변수는 자신이 속한 중괄호{ } 안에서만 사용 가능한 변수를 만들어 사용할 수 있습니다.

## 5-5 그 밖의 함수 사용법

지금까지는 함수를 사용하는 방법과 함수사용에 필요한 조건에 대해 알아보았다면 이제는 다양한 함수 사용법에 대해 알아봅시다. 하지만 함수가 사용되는 방법에 따라 세 가지 형태로 변경하여 사용됩니다. 사용방법을 보면

1. 이름있는 함수
2. 이름 없는 함수(익명의 함수)
3. 즉시 실행 함수식 입니다.

이름이 있는 함수는 사용자 함수의 기본 방법으로 함수를 선언하여 만들어진 함수의 이름으로 불러서 작업을 진행할 때 사용합니다.

이름 없는 함수는 이름이 없이 함수를 선언하여 하나의 식처럼 변수에 해당하는 함수의 값을 제공하여 사용하는 방법입니다.

즉시 실행 함수식은 함수식이라고도 불리며 정의된 함수와 함께 실행하여 즉시 그 결과값을 받아 사용하는 방법입니다.

### 이름있는 함수만들기

이름있는 함수는 function이라는 명령어를 사용하여 함수 이름을 선언하고, 인수에서 필요한 데이터를 보내 함수 선언에 있는 매개 변수에서 그 데이터를 받아 사용할 수 있는 함수 있다.

```
형식 :  
    함수 선언:    function 함수이름(매개변수) {  
                    필요한 기능의 실행문;  
                }  
    함수 실행:    함수이름(인수) ;
```

### 이름없는 함수(익명의 함수)

```
형식 :  
    var 변수의 이름 = function (매개변수){  
                    필요한 기능의 실행문;  
                }  
    함수 실행:    변수의 값으로 받아서 사용
```

### 함수식

형식1 :

```
var 변수의 이름 = ( function(매개변수){  
    필요한 기능의 실행문;  
})(인수) )
```

형식2 :

```
var 변수의 이름 = ( function(매개변수){  
    필요한 기능의 실행문;  
})(인수)
```

함수 실행:            변수의 값으로 받아서 사용

## 함수의 화살표 표기법[ES6]

ES6 버전에서는 이름 없는 함수일 경우 화살표 표기법이 추가되었습니다. 이 방법은 function 이라는 명령어를 사용하지 않고 화살표(=>)로 함수를 만드는 방법입니다.

매개변수의 따라 표기법이 달라집니다.

### 1. 매개변수가 없으면

```
var all = function(){  
    var result;  
    result=console.log('javascript');  
    return result;  
}  
  
all();
```

```
let all = ( ) =>  
{console.log('javascript');}  
all();
```

### 2. 매개변수가 1개인 경우

```
var all = function(str){  
    var result;  
    result=console.log(str);  
    return result;  
}  
  
all('자바스크립트');
```

매개변수가 1개인 경우:

```
let all = str => {console.log(str);}  
all('자바스크립트');
```

### 3.매개변수가 2개 이상일 경우

```
var all = function(a,b){  
    var result;  
    result=console.log(a+b);  
    return result;  
}  
all(5,10);
```

```
let all = (a, b) => {console.log(a+b);}  
all(5,10);
```

## 6장 객체

### 6-1 객체란?

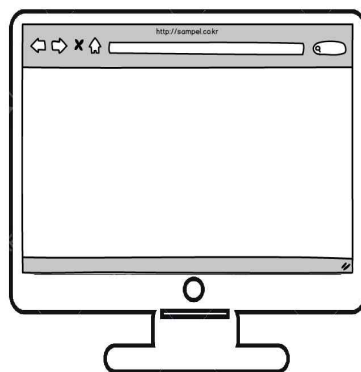
우리가 자바스크립트로 프로그램을 만들 때 필요한 시스템들은 어떤 것들이 있을까요? 브라우저를 제어해야 한다면 브라우저 제어 시스템이 필요하고 HTML을 제어하기 위해서는 HTML을 제어하는 시스템이 필요합니다. 또 시간을 이용한 프로그램을 만들고 싶다면 시간을 제어하는 시스템이 필요합니다. 이렇게 자바스크립트 필요한 시스템을 우리는 객체라 말합니다.

#### 객체란?

예를 들어 생각해 봅시다. 우리가 만약에 브라우저를 제어하고 싶다면 브라우저 객체가 필요합니다. 이 객체 안에는 브라우저의 주소, 브라우저 창의 가로 크기, 브라우저 창의 세로의 크기, 브라우저 창의 이름, 브라우저 위치, 스크롤의 위치 등 브라우저 정보를 알려주는 속성과 브라우저 창 열기, 브라우저 창 닫기, 모달 윈도우 열기 등 실제 작동을 실현하는 메서드가 존재합니다. 우리가 흔히 명령어 또는 내장함수가 여기에 속합니다.

정보 -> 속성

- 브라우저 주소창
- 브라우저 가로 크기
- 브라우저 세로 크기
- 브라우저 위치
- 스크롤의 위치



작동 -> 메서드( )

- 브라우저 창 열기
- 브라우저 창 닫기
- 모달 윈도우 열기
- 스크롤 움직이기
- 주소창의 주소 바꾸기

자바스크립트에서 객체를 사용하기 위한 기본 형식부터 알아보시다.

형식:

```
객체.메서드( );          -> 브라우저.창열기( );
❶객체.속성; 또는 ❷객체.속성 = 값;  -> 브라우저.가로크기;
                                   -> 브라우저.가로크기 = 600px;
```

객체와 메서드( ) 그리고 속성의 연결은 '.'로 연결을 합니다. 우리는 이러한 연결을 체인 사용법이라고 부르며 필요에 따라 여러 번 연결이 가능합니다.

```
ex ) 객체.객체.속성 ;      -> window.location.href ;
      객체.속성.속성 ;      -> document.style.color ;
```

또한 속성의 사용법을 보면 ❶속성은 속성의 값을 가져올 때 사용하며, ❷속성의 사용법은 속성에 값을 지정할 때 사용합니다.

이번에는 객체의 종류에 대해 알아보시다.

## 객체의 종류

객체의 종류에는

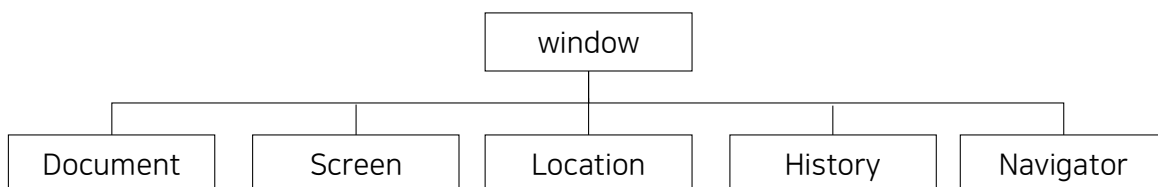
1. 내장객체(기본객체)
2. 브라우저 객체
3. 사용자지정 객체로 크게 나눌 수 있습니다.

### 내장객체

내장객체는 자바스크립트 엔진에 내장이 되어 있는 기본객체들로 문자(string), 날짜(Date), 배열(Array), 수학(Math) 객체 등이 여기에 속합니다.

### 브라우저 객체

브라우저 객체는 브라우저에 내장된 객체로 우리가 흔히 BOM(Browser Object Model)로 얘기 말합니다. 브라우저의 계층 구조를 보면 다음과 같습니다.



브라우저의 가장 상급객체는 window객체로 어느 객체보다 위에 존재하기 때문에 window 객체는 생략하여 사용합니다.

ex: `window.document.getElementById('main')` ; -> `document.getElementById('main')` ;  
윈도우 객체 안에는 여러 객체가 존재하는데 우리가 DOM(Document Object Model)이라 부르는 문서객체도 여기에 포함됩니다.

### 사용자 객체

사용자 객체 또는 사용자 정의 객체는 프로그램 작성 시 사용자가 필요에 따라 시스템을 설계하여 만드는 객체입니다.

예를 들면 미술관에서 미술품 관리 시스템을 만들고자 하면 먼저 미술품 번호, 미술품 제목, 제작자, 카테고리, 제작일과 같은 정보가 있는 속성들과 정보 출력과 같은 메서드를 만드는 것입니다.

### 객체의 프로토타입이란



앞에서 우리는 객체를 시스템이라고 정의를 했습니다. 자바스크립트에 내장된 객체 또는 사용자가 필요에 따라 만든 객체든 모든 객체 안에는 객체의 속성과 메서드()가 설계되어 있습니다. 이러한 내부 구조도를 우리는 프로토타입이라 합니다.

날짜객체를 살펴봅시다. 만약에 우리가 날짜 시스템 필요하다면 이 시스템에 필요한 속성이 있을 거예요.

날짜 시스템 ( 날짜 시스템 프로토타입 )

정보	작동하기
<ul style="list-style-type: none"> <li>• year</li> <li>• month</li> <li>• day</li> <li>• date</li> <li>• week</li> <li>• time</li> </ul>	<ul style="list-style-type: none"> <li>• Year값 가져오지</li> <li>• Month값 가져오지</li> <li>• Day값 가져오지</li> <li>• Date값 가져오지</li> <li>• Week값 가져오지</li> <li>• Time값 가져오지</li> </ul>

## 객체의 인스턴스 만들기

객체의 인스턴스란 구조가 만들어져 있는 시스템에서 기본 시스템을 이용하여 자신이 사용할 복제품을 말합니다. 이 복제품은 기본 객체 시스템의 속성과 메서드를 모두 갖추어 있으면 사용할 수 있습니다. 거의 모든 내장객체는 객체 자체를 사용하지 않고 개체의 시스템을 이용하여 새로운 사용자가 이름을 정한 복제품이 만들어집니다. 인스턴스를 만드는 방법은 var로 자신이 사용할 이름을 선언하고 new라는 연산자를 이용하여 만들고자 하는 객체의 이름을 사용하면 된다.

형식: 내재 객체 인스턴스 만들기

```
var 사용할 객체 이름 = new 객체( ); -> ex) 날짜 객체를 사용하려면
var today = new Date( );
```

형식: 사용자 객체 인스턴스만들기

```
var 사용할 객체 이름 = new Object( ); -> ex) 나의 객체를 만든다면
var My = new Object( );
```

## 6-2 내장 객체 사용하기

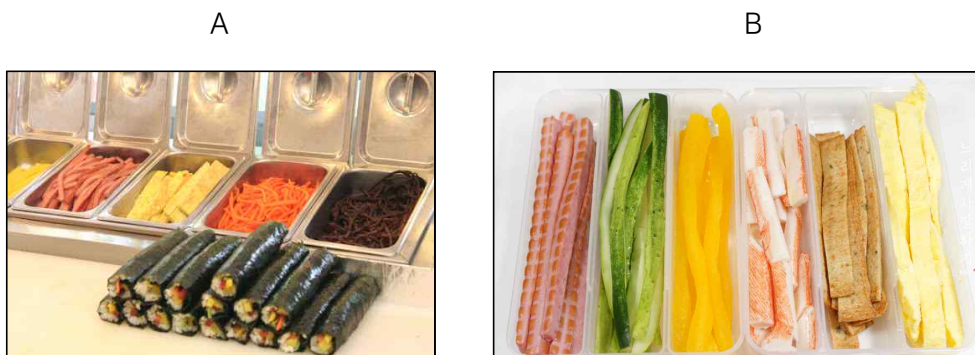
지금까지 우리는 객체에 대해서 알아보았습니다. 브라우저의 자바스크립트 엔진에 내장된 객체중

데이터를 관리하는 Array 객체에 대해서 알아보도록 합시다.

## Array 객체 사용하기

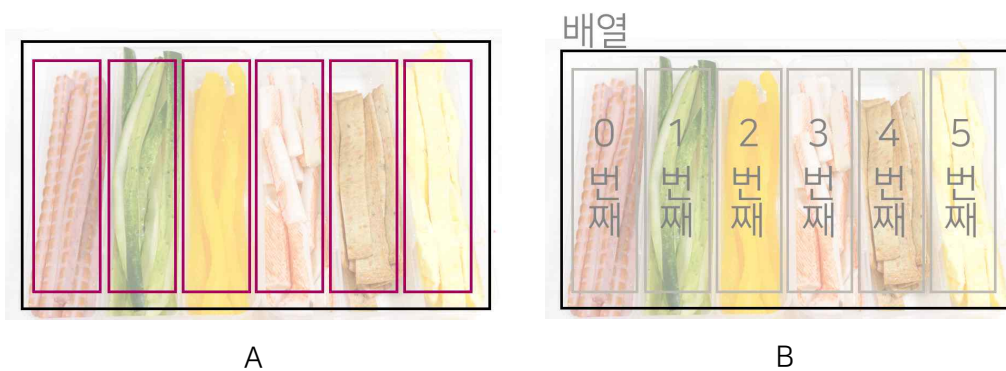
Array(배열) 객체는 데이터를 관리하는 객체입니다. 예를 들어보면 우리가 변수를 알아볼 때 변수는 데이터 관리하는 그릇이라고 표현을 하였습니다. 그럼 변수와 배열객체는 무엇이 틀리는지 부터 알아보시다.

김밥의 재료를 데이터로 정리한다면 변수는 ㉠와 같이 각각의 재료를 따로 그릇을 만들어 사용하는 방법이고 배열은 ㉡와 같이 하나의 그릇에 각각의 칸을 만들어 데이터를 정리하는 방법이 배열입니다.



그럼 배열이 형식을 먼저 봅시다.

배열은 A와 같이 먼저 그릇을 하나 만들고 그 다음 필요한 칸을 만들어서 0번부터 차례로 필요한 데이터를 넣을 만 됩니다.



## 배열 형식

배열 이름



기본형 :

```
var 배열의 이름 = new Array(배열갯수)
배열이름[0] = 값 ;
배열이름[1] = 값 ;
배열이름[2] = 값 ;
```

## 배열 속성과 메서드

속성	기능 설명
length	객체의 갯수를 수치로 나타낸다.
constructor	객체의 생성자를 참조한다.
prototype	속성과 메소드를 추가하여 배열 선언을 확장한다.

메소드	기능
concat()	하나의 배열에 다른 배열의 요소를 결합한다.
join()	문자열로 배열의 요소를 분리하여 1개의 데이터로 결합한다.
pop()	배열의 마지막 요소를 삭제한다.
push()	마지막 인덱스에 다른 요소를 추가한다.
shift()	배열의 첫 요소를 제거한다.
unshift()	배열의 처음에 요소를 추가한다.
splice()	이전 배열요소를 삭제하고 새로운 내용물을 추가하는 형태로 배열 내용을 변경하고, 삭제된 요소들은 반환한다.
slice()	배열 값의 일부분을 선택하여 새로운 배열을 만들어 준다.
sort()	숫자 또는 문자열 순서대로 정렬한다.
reverse()	배열의 요소를 역순으로 나타낸다.

## 6-3 사용자 객체 만들기

사용자가 자신이 만드는 프로그램 안에 필요한 정보를 정의하고 사용하는 시스템을 말합니다. 예를 들어 어떤 미술관에 진열된 작품 관리 프로그램을 만들어야 한다면 그 미술관에 소장되어 있는 작품의 정보와 관리 프로그램에서 필요한 기능을 객체의 형식으로 만들어서 사용하면 됩니다. 먼저 객체를 만드는 방법부터 살펴봅시다.

## 사용자 객체 기본형

사용자 객체의 가장 기본이 되는 형식부터 알아보겠습니다.

형식:

- ❶ var 객체이름 = new Object();
- ❷ 객체이름.속성 = 값;
- 객체이름.속성 = 값;
- ❸ 객체이름.메서드 = function( ){ 실행문 ; }

사용자 객체는

- ❶ var 명령을 이용하여 객체의 이름 선언 후 new 연산자를 새로운 객체를 생성한다.  
주로 객체의 이름은 대부분 대문자로 시작한다.
- ❷ 속성만들기 : 만들어진 객체의 이름에 속성명을 '.'로 연결한다.
- ❸ 메서드 만들기 : 만들어진 객체의 이름에 메서드명을 '.'로 연결한 후 주로 익명의 함수 형태로 실행문을 선언한다.

## 리터널(Literal) 방식으로 만들기

리터널(literal) 방식은 어떠한 이름과 함께 데이터를 저장하는 방식입니다. 객체를 리터널방식으로 정의할 때는 선언 할 객체의 이름에 {}를 사용하여 속성과 값, 그리고 메서드를 정의합니다. 이때 속성, 메서드의 이름에 값을 넣을 때는 ':'으로 사용하고 연결 연산자 ','로 연결합니다. 그리고 그룹을 닫히는 '}' 다음에는 ';'으로 마무리합니다.

형식:

- ❶ var 객체이름 = {
- ❷
- ❸ 속성명 : 값 ,
- ❹ 속성명 : 값 ,
- 메서드명 : function(){
- 실행문 ;
- ❺
- };

## 생성자 함수를 사용해 객체 만들기

기본 형식 또는 리터널 방식으로 객체를 만들면 하나의 객체가 생성됩니다. 그런데 같은 형식으로 된 여러 데이터시스템이 필요한 경우에는 같은 형식의 객체를 여러번 만들어야 하는 단점이 있습니다. 이런 경우에는 함수의 성격을 이용하여 객체를 시스템을 구축하면 됩니다. 우리는 이 방법을 생성자 함수를 이용한 객체라고 합니다.

객체 시스템 만들기:

```
❶function 객체이름(❷매개변수,매개변수){  
    ❸this.속성 = ❷변수명;  
    this.속성 = 변수명;  
    ❹this.매개변수 = function(){실행문}  
}
```

인스턴스만들기:

만들기 1

```
❶var 인스턴스명 = new 객체(❷인수,인수)
```

만들기 2

```
❶배열명 = ❷[객체(인수,인수),[객체(인수,인수)]
```

먼저 객체의 기본을 시스템을 만듭니다.

❶ function으로 객체이름 선언 후 속성에 사용할 ❷ 매개변수를 선언합니다.

❸ this예약어를 사용하여 객체의 속성을 만든 후 사용할 매개변수를 할당합니다.

이때의 this는 지금 선언 된 객체를 가리킵니다.

❹ this예약어를 사용하여 객체의 메서드를 만든 후 익명의 함수(또는 이름있는 함수)를 이용하여 실행문 그룹을 만듭니다.

이렇게 기본 시스템이 만들어 지면 사용할 데이터가 있는 인스턴스를 만듭니다. 인스턴스 여러방  
법으로 만들 수 있습니다.

첫 번째 ❶var 명령어(또는 var가 생략되어도 됨)를 이용하여 인스턴스명을 선언하고 new 연산  
자로 생성될 객체와 ❷객체의 매개변수로 보낼 인수를 설정합니다.

두 번째 방법은 배열로 만드는 방법입니다.

먼저 ❶var 명령어(또는 var가 생략되어도 됨)를 이용하여 인스턴스명을 선언하고 ❷배열을 만  
듭니다. 배열 안에 객체와 인수를 설정하면 됩니다.

## 프로토타입을 이용한 메서드 만들기

앞에서 정의한 바와 같이 프로토타입은 객체 시스템이 가지고 있는 원본이라고 볼 수 있습니다.  
객체를 만들 때 단위 객체에 필요한 메서드를 모두 한 번에 생성이 되면 그만큼 메모리의 손실  
이 필요하며 추가되는 메서드의 관리에 문제가 생길 수 있습니다. 이러한 경우는 대비하여 우리  
는 prototype의 메서드를 만들어서 사용할 수 있습니다.

객체원본을 생성할 때는 객체에 필요한 데이터의 속성만 선언하고 따로 객체의 메서드 앞에  
.prototype을 붙여서 메서드를 만드는 방법이다. 이 경우에는 원본시스템에 모든 시스템이 갖추  
어 있지 않아도 되기 때문에 메모리의 손실을 막을 수 있고, 필요 인스턴스 생성하며 필요한 기  
능을 추가 할 수 있는 점이 편리하다.

객체 시스템 만들기:

```
❶function 객체함수이름(❷매개변수,매개변수){  
    ❸this.속성 = ❷변수명;  
    this.속성 = 변수명;  
}
```

```
❹함수명.prototype.메서드명 = function(){  
    실행문  
}
```

❺인스턴스 생성하기(변수형 또는 배열형)

