입문자를 위한 자바스크립트 기초 #0 수업 소개

#1 자바스크립트 소개

자바스크립트란
자바스크립트는 프로그래밍 언어이다.

자바스크립트는 서버 개발, 어플리케이션 개발 등 다양한 목적을 위해 사용할수 있는 팔방미인(?)언어이지만, 주된 활동 분야는 역시 '웹 개발'이다!

웹 개발이란?
인터넷을 통해 서비스되는 웹사이트를 개발하는 일!

웹사이트 개발 삼형제

HTML 웹 콘텐츠를 정의한다
CSS 웹 콘텐츠를 스타일링 한다.
JavaScript 웹사이트의 동작이나 상호작용을 정의한다.

결국 자바스크립트의 역할은?
웹사이트 개발에 있어, 자바스크립트의 역할은

・웹브라우저가 가진 기능을 실행 시키거나
・HTML/CSS 를 통해 렌더링된 화면을 조작할 수 있다.

즉, 자바스크립트는 웹브라우저를 조작함으로써 역할을 수행한다!

내용정리
・자바스크립트는 프로그래밍 언어이다.
・자바스크립트는 주로 웹(웹사이트)개발을 위해 사용한다.
・자바스크립트는 브라우저가 가진 기능을 실행 시키거나, 웹페이지의 내용을 조작할 수 있다.

#2 자바스크립트 코드 사용법

자바스크립트 코드를 더하는 방법은?

두 가지 방법이 있다.

- HTML 문서 내부에 작성하기
- 자바스크립트 파일을 만들고, 그 안에 작성한 코드를 HTML 문서에 연결하기

=>자바스크립트 파일 확장자는. *.js 이다.

필요한 태그는 단 하나
<script></script>

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE-edge">
        <meta name="viewport" content="width=device-width, initial-s">
        <title>HTML 문서</title>
        <style>
            *{ color: red;}
        </style>
    </head>
    <body>
        <h1>반가워요</h1>
        <p>아무 내용이나 채워보고 있어요</p>
        <ul>
            <li>호랑이</li><li>고양이</li><li>살쾡이</li>
        </ul>
        <script src="script.js"></script>
    </body>
</html>
```

#3 객체 그리고 window

객체를 쉽게 이해하자.
객체의 사전적 의미?
실세계에 존재하는 대상 또는 생각할 수 있는 어떤 개념.

자바스크립트에게 객체란?
어떤 사물이나 개념을 소프트웨어적으로 표현하기 위해 사용하는 문법적 수단.
자바스크립트 코드 내에서 객체란 '값 또는 기능을 가지고 있는 데이터'이다.

웹브라우저도 객체입니다.
웹브라우저는 소프트웨어 세계에 존재하는 사물, 즉 객체이다. 그리고 자바스크립트는 웹브라우저라는 객체에게 명령을 내리기 위해 사용하는 언어이다.

사용자가 웹브라우저에게 명령을 내릴 수 있도록 돕기 위해, 자바스크립트는 웹 브라우저와 관련된 다양한 객체들을 제공한다.

대표 선수
웹브라우저 객체의 대표 선수는? window

객체 사용법
객체란 '값 또는 기능을 가지고 있는 데이터'이다.
뒤에 점을 찍으면 값 또는 기능을 사용할 수 있다!

객체.데이터 <-- 객체가 가지고 있는 숫자, 문자 등의 다양한 데이터를 사용할 수 있다.
객체.기능() <-- 객체가 가지고 있는 다양한 기능을 수행할 수 있다(괄호 필수)

자바스크립트에도 주석이 있다.
자바스크립트 코드에서도 주석을 사용할 수 있다.
두 가지 방법을 사용할 수 있다.

// 주석이다 <--- 주석을 한 줄밖에 쓸 수 없다.
/* 주석이다 */ <--- 주석을 여러 줄에 걸쳐 쓸 수 있다.

index.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>자바스크립트를 실습해보자</title>
    </head>
    <body>
        <script src="script.js"></script>
    </body>
</html>
```

---

script.js

```
window.alert(`hello!`);
//window.alert(`hello!`);
window.alert(`hello!`);
```

내용 정리
- 객체는 자바스크립트의 핵심 개념으로, 값 또는 기능을 가지고 있는 데이터!
- 사용자는 자바스크립트 코드를 이용해 웹브라우저 객체에 접근할 수 있다.
- 객체 뒤에 점을 찍으면, 객체가 가진 데이터나 기능을 사용할 수 있다.
- 객체가 가진 기능을 '메소드'라 한다.
- 자바스크립 문법에도 주석이 있다. 두 가지 방법으로 주석을 만들 수 있다.

#4 콘솔(console)소개

콘솔을 소개합니다.
콘솔(console)은 브라우저의 디버깅 콘솔을 의미한다. 콘솔은 브라우저 안에 내장된 브라우저의 하위 객체이므로 브라우저 객체를 통해 접근할 수 있다.

window.console (객체 뒤에 .을 찍으면 객체가 가지고 있는 데이터에 접근할 수 있다.)

디버깅 콘솔이 뭔데
웹브라우저의 '개발자 도구'를 열면 메뉴 중 '콘솔(console)'이라는 항목이 존재하는데, 이것을 누르면 나타나는 화면이 바로 디버깅 콘솔이다.

=>디버깅 콘솔을 이용하면 자바스크립트 코드를 테스트(점검)할 수 있다!

충격적인(?)사실이지만 꿀팁
우리가 웹브라우저에 접근할 수 있게 해주는 객체 window 는 console 을 비롯한 다양한 하위 객체 및 데이터, 메소드 등을 포함하고 있는데, 사용자는 편의를 위해 window.을 생략한 채로 코드를 작성할 수 있다.

window.console <--이렇게 쓰나

console <--이렇게 쓰나 똑같음.

자바스크립트 코드 작성 규칙
왠지 이쯤부터 본격적인 코드 작성이 시작된다는 느낌이 들어서...

・대문자와 소문자를 잘 구분하여 작성한다.
・구문의 끝에는 세미콜론을 입력해 '구문의 끝'을 알릴 수 있다.
이는 선택 사항이다.

・가능하면 한 줄에는 두개 이상의 구문을 쓰지 않는다.

-script.js-
```
window.console.log(`고양이`);
console.log(`고양이`);
console.clear();
```

내용 정리
・console 객체는 브라우저의 디버깅 콘솔에 접근할 수 있게 해 준다.
・console 객체는 window 의 하위 객체이다.
・자바스크립트 코드에서, window.은 생략이 가능하다.
・console.log(x) 는 'x 를 콘솔에 기록해두자'라는 기능의 메소드 호출이다.
・코드에서 세미콜론(;)은 '구문이 끝'을 의미하는 기호이다. 세미콜론의 사용은 선택사항이다.

#5 변수와 숫자, 그리고 문자열

변수를 소개합니다.
변수란 "데이터에 붙이는 이름표'이다. 변수를 이용하면 이름표를 붙여 둔 데이터를 기억해두었다가 필요할 때마다 재사용 할 수 있게 된다.

변수 만드는 방법

let 변수이름 = 데이터;

이게 포인트!
변수가 생성되고 나면, 변수(변수이름)를 지정된 데이터 대신 사용할 수 있다.

변수의 선언과 초기화
변수를 만드는 작업을 '변수의 선언'이라 부르고, 만들어진 변수에 첫 데이터를 지정하는 작업을 '변수의 초기화'라 부른다. 두 과정은 동시에 진행할 수도, 따로 진행할 수도 있다.

변수의 선언과 초기화 따로 하기

```
let 변수이름;  //변수 선언
변수이름 = 데이터;  // 변수 초기화
```

변수는 변한다
변수는 데이터를 기억하기 위해 사용한다. 변수는 한 번에 하나의 데이터만 기억할 수 있으며, 기억하고 있는 값을 바꿔가며 사용할 수 있다.

변수가 기억하고 있는 데이터를 바꿀 수 있다.

```
let 변수이름 = 데이터 1;  //변수 선언과 초기화 동시 진행
변수이름 = 데이터 2;       //새로운 데이터 대입
변수이름 = 데이터 3;       //새로운 데이터 또 대입
```

변수 이름 정할 때 규칙
변수는 '데이터에 붙이는 이름표'이다(라고 했죠?). 변수의 이름은 기본적으로 '짓는 사람 마음대로'지어줄 수 있지만, 몇 가지 제약사항도 존재한다.

제약사항
- 변수명에는 오직 문자와 숫자, 그리고 기호 $과 _만이 포함될 수 있다.
- 변수명의 첫 번째 글자로 숫자가 올 수 없다.
- 이미 다른 뜻을 가지고 있는 단어(키워드)는 변수명으로 사용할 수 없다.

근데, 변수에 뭘 기억시켜?
변수를 이용하면 자바스크립트로 표현할 수 있는 모든 데이터를 기억할 수 있다. 자바스크립트는 다양한 유형의 데이터를 표현 및 사용할 수 있는데, 여기에서는 다음 두 가지 데이터 유형을 사용해 보자.

・숫자
・문자열

=>데이터 유형을 흔히 '자료형'이라 합니다.

숫자
숫자는 우리가 알고 있는 그 숫자가 맞다. 숫자는 정수와 실수로 구분할 수 있다.

정수를 기억하는 변수의 예
let number1 = 10; //양의 정수
let number2 = 0; //0 도 정수다
let number3 = -10; //음수는 앞에 - 기호 사용

실수를 기억하는 변수의 예
let number = 3.14; //양의 실수
let number = -3.14; //음의 실수

문자열
자바스크립트에서 문자열이란, '기호의 순차 수열'을 뜻한다. 문자,숫자,특수
문자 등 다양한 기호를 조합해 만들 수 있는 '기호의 집합'이며, 따옴표로 시작
해 따옴표로 끝나는 데이터이다.

작은 따옴표, 큰 따옴표 모두 사용 가능

let string1= `문자열에는 기호가 들어간다`;
let string2= "12345...!?";

단 시작하는 따옴표와 끝나는 따옴표가 같아야 한다!

let greet = "hello";
console.log(greet);
console.log(greet);
console.log(greet);

```
console.log(greet);


greet = "안녕";
console.log(greet);
console.log(greet);
console.log(greet);
console.log(greet);
```

내용 정리
• 변수는 데이터를 기억하기 위해 사용하는 수단이다.
• 변수는 한 번에 하나의 데이터를 기억하며, 변수에 데이터를 대입(=)하면,
그 때부터 해당 데이터 대신 사용할 수 있게 된다.
• 키워드 let 을 사용해 변수를 선언하며, 데이터는 얼마든지 변경할 수 있다.
• 변수의 이름은 기본적으로 자유롭게 지어줄 수 있으나, 제약사항이 있다.
• 자바스크립트 코드로 표현할 수 있는 모든 데이터는 변수에 대입될 수 있다.

#6 상수 그리고 window.prompt

상수를 소개합니다
상수란 수식에서 '변하지 않는 값'을 뜻한다. 변수와 반대되는 것으로, 상수는
'단 하나의 데이터만을 위해 사용하는 이름표'로써, 값의 변경이 불가하다.

상수 만드는 방법

```
const 상수이름 = 데이터;
```

상수도 변수처럼 지정된 데이터 대신 사용할 수 있다.
상수 이름 짓기 규치근 변수 이름 짓기와 동일하다.

상수의 선언과 초기화
상수를 만들 때는 선언과 동시에 초기화를 해주어야 한다. 그러지 않으면 데이터를 지정할 수 없다.

상수의 선언과 초기화 따로 하면?
const 상수이름;
상수이름 = 데이터; //에러 발생!

상수는 자신이 선언될 때 지정된 데이터로 고정된다. 선언 이후 데이터를 대입하려고 하면 에러가 발생한다.

window.prompt(x)
window 객체의 prompt(x) 메소드는 사용자로부터 문자열을 입력받을 수 있는 다이얼로그 박스를 열어주는 메소드이다. 괄호 안에 x 로는 다이얼로그 박스에 띄울 메시지(문자열)를 전달할 수 있다.

prompt("다이얼로그 박스에 이렇게 쓸 거야");

prompt(x)의 문자열 반환
prompt(x)메소드는 실행 시에 사용자로부터 문자열을 입력받고, 입력받은 문자열을 자신이 사용된 곳에 그대로 '반환'한다. 즉, 메소드가 문자열로 바뀌게 된다.

const data = prompt("아무거나 써 봐요");

주의! 실제로 코드가 바뀌는 게 아닌 마치 바뀐 것처럼 실행된다는 의미이다!

const data = "아무거나 썻다 어쩔래!";

let aaa;
aaa = 5;
console.log(aaa);
aaa = 8;
console.log(aaa);

```
const bb = 5;
console.log(bb);
bb = 7;
console.log(bb);    //에러가 출력됨.


window.prompt();

window.prompt("프롬프트 다이얼로그에 표시될 문자열");


const promptMessage = "프롬프트에 표시할 문자열";
prompt(promptMessage);


const promptMessage = "프롬포트에 표시할 메시지";
const result = prompt(promptMessage);
console.log(result);
```

내용정리

• 상수는 '변하지 않는 값'이라는 뜻이다.
• 자바스크립트 상수는 const 키워드를 이용해 선언할 수 있으며, 선언과 동시에 데이터를 초기화 해주어야 한다.
• 상수는 데이터를 바꿀 수 없다.
• window 객체의 prompt 메소드는 사용자로부터 문자열 입력을 받는 다이얼로그를 생성해 준다.
• prompt 는 입력받은 문자열을 반환해주어 코드에서 활용할 수 있게 해 준다.

#7 템플릿 리터럴

기존의 문자열은 따옴표를 이용해 표현했지만, 템플릿 리터럴은 백틱을 이용해 표현한다.

기존의 문자열은 따옴표를 이용해 표현했지만, 템플릿 리터럴은 백틱을 이용해 표현한다.

```
//따옴표를 이용한 기존의 문자열
const str1 = '작은 따옴표';
const str2 = "큰 따옴표";

//백틱을 이용한 템플릿 리터럴
const str3 = `이게 백팁입니다`;
```

=>백틱은 물결(~)키를 쉬프트(shift)없이 누르면 입력할 수 있습니다.

표현식 삽입 방법
템플릿 리터럴은 표현식을 내장할 수 있는 문자열 표현법이다. 이는 문자열의 내용에 데이터를 삽입한다는 것을 의미한다. 템플릿 리터럴로 표현한 문자열 내부에 플레이스홀더($[])를 기입하고, 그 안에 데이터를 기입하면 데이터는 문자열의 멤버가 된다.

```
const data1 = "데이터";
const str1 = `문자열 중간에 ${data1} 삽입하기`;
console.log(str1);

const data2 = 100;
const str2 = `숫자도 문자로 녹아든다 : ${data2} !`
console.log(str2);
```

```
const data1 = "데이터";
const str1 = `문자열 중간에 ${data1} 삽입하기`;
```

```
console.log(str1);

const data2 = 100;
const str2 = `숫자도 문자로 녹아든다 : ${data2} !`
console.log(str2);
VM663:3 문자열 중간에 데이터 삽입하기
VM663:7 숫자도 문자로 녹아든다 : 100 !
undefined
console.log(str1);
VM736:1 문자열 중간에 데이터 삽입하기
undefined
console.log(str2);
VM765:1 숫자도 문자로 녹아든다 : 100 !
undefined
```

표현식 삽입 방법

데이터를 반환하는 메소드 또한 데이터와 마찬가지로 취급할 수 있다.

```
const str = `그렇다면 이건 될까? ${prompt('그렇다면 이건 될까?')}`
console.log(str);

->그렇다면 이건 될까? Out Of Memory!
undefined


let str = "따옴표 문자열";
console.log(str);
str = `백틱 문자열!!??`;
console.log(str);


->따옴표 문자열
script.js:4 백틱 문자열!!??
```

```
let number = 21;
let player = "안드레아 피를로";

console.log(
    `제가 좋아하는 축구선수는 ${player} , 그의 등번호는 ${number}`
);
```

```
let number = 21;
let player = "안드레아 피를로";

console.log(
    `제가 좋아하는 축구선수는 ${player} , 그의 등번호는 ${21}` //직접 데이터를 입력가능
);
```

```
let number = prompt("등번호를 입력 하세요.");
let player = prompt("좋아하는 축구 선수는?");

console.log(
    `제가 좋아하는 축구선수는 ${player}, 그의 등번호는 ${number}`
);
```

내용 정리
・템플릿 리터럴은 표현식을 내장할 수 있는 문자열 표현법이다.
・백틱을 이용해 표현하며, 문자열에 데이터를 삽입할 수 있다.
・데이터 삽입 시에는 플레이스홀더(${})를 사용하며, 삽입된 데이터는 문자열의 멤버가 된다.
・데이터를 대신해 사용될 수 있는 것이라면 무엇이든 템플릿 리터럴에 삽입될 수 있다. 변수, 메소드 모두 가능.

${} = 플레이스 홀더
` = 백틱

#8 연산 그리고 연산자

연산을 하자
연산이란 '식이 나타낸 일정한 규칙에 따라 계산함'을 뜻한다. 사용자는 자바스크립트 코드를 통해 연산을 처리하는 식을 만들 수 있고, 여기에 사용하는 기호를 '연산자'라 한다!

여기에는 사용해 볼 연산(과 연산자)
・산술 연산
・대입 연산

산술 연산
산술 연산은 더하기, 빼기 등 수를 이용한 계산이다. 두 개의 숫자 데이터를 피연산자로 받아서 하나의 숫자 데이터를 결과로 반환한다.

연산 결과가 반환된다

산술 연산을 처리하는 식을 사용하면, 해당 식은 연산 결과를 반환한다.

즉, 식이 데이터로 대체된다!

let number = 3 + 3;

let number = 6;

=>실제 코드가 바뀌는 게 아닌, 실행 시에 결과가 반영된다는 뜻!

대입 연산
대입 연산은 오른쪽 피연산자의 데이터를 왼쪽 피연산자에 대입한다. 왼쪽 피연산자로는 주로 변수나 상수가 자리하며, 오른쪽 피연산자로는 데이터 또는 데이터를 반환하는 식이 자리한다.

연산자 우선순위
하나의구문에 여러 개의 연산자가 함께 사용 되는 경우가 있다. 이 경우 연산자 우선순위가 반영되어 우선순위가 높은 것부터 계산된다. 아래는 주요한 예시들.

곱하기, 나누기가 더하기, 빼기보다 우선순위가 높다.
1 + 3 * 2 - 2;

대입 연산은 대입 연산자 오른쪽에 있는 식보다 무조건 나중에 처리된다.
변수 = 1 + 2 + 3;

연산자 우선순위
연산자 우선순위에서 밀리지만 먼저 처리하고자 하는 연산이 있는 경우 사용자는 해당 연산 부분을 소괄호로 감싸주어 연산 순서를 바꿀 수 있다.

곱하기(*)가 가장 우선순위가 높으나, 실제로는 2-2 가 먼저 계산된다.

1 + 3 * (2 - 2);

1 + 3 * 0;

```
console.log(3 +  4);
console.log(3 - 4);
console.log(3 * 4);
console.log(3 / 4);
console.log(3 % 4);
```

```
let result = 1+ 1;
console.log(result);
result = 1 - 1;
console.log(result);
result = 1 * 1;
console.log(result);
result = 1 / 2;
console.log(result);
```

내용 정리
• 연산이란 '식이 나타낸 일정한 규칙에 따라 계산함'을 뜻한다.
• 산술 연산은 두 항을 계산한 결과를 변환한다.
• 대입 연산은 연산자 오른쪽 항을 왼쪽 항에 대입한다. 주로 변수 초기화에 사용하며, 복합 대입 연산자를 이용해 연산 처리를 추가하기도 한다.
• 연산자가 여러 개 사용된 구문을 만들 수 있다. 이 경우 우선순위가 적용된다.

#9 null undefined boolean

널(null)데이터는 '없다'를 의미하는 데이터이다. 의도적으로 데이터가 없음을 나타내기 위해 사용하는 일종의 표현 수단이다.

```
let number;
```

```
// 이 변수는 숫자 0 이다!
number = 0;

// 이 변수에는 아무 것도 없다!
number = null;
```

undefined
언디파인드(undefined)는 아직 데이터가 정의되지 않았음을 나타낸다.

```
let number;

// 여기에는 아직 값이 정의되지 않았어! undefined 야.
console.log(number);   //undefined

// number 는 3 이라고 정의한 다음 확인하자.
number = 3;
console.log(number); //3
```

boolean
불리언(불린이라고 해도 돼요)은 숫자, 문자열과 같은 데이터 타입 중 하나이다.
true 와 false,단 두 가지 값만 존재한다. 참과 거짓 여부를 나타내기 위해
사용하는 데이터이다.

```
let value;
value = true; // 소문자로 써야 합니다.
value = false; // 참, 거짓 모두 마찬가지.
value = False; // 이거 안되고
value = True;   // 이것도 안 돼요.
```

```
let data;   //선언은 했지만 정의는 아직...
```

```
console.log(data);   // ???

data="hello";   // 문자열로 초기화!
console.log(data);   // ???

data = null;    // 여기엔 아무것도 없다!
console.log(data);
```

```
undefined
hello
null
```

null 과 undefined 와의 차이는 선언만 하고 정의를 안했다라는 것.

```
let data = true;
console.log(data);

data=false;
console.log(data);

data=True;
console.log(data);
```

```
true
false
ReferenceError
```

```
let data = true;
console.log(typeof data);

data=false;
console.log(typeof data);

data=True;
console.log(typeof data);
```

boolean
boolean
ReferenceError

내용정리
• null 은 데이터가 없음을 나타내기 위해 사용하는 데이터이다.
• undefined 는 정의되지 않은 상태임을 나타내기 위해 사용하는 데이터이다.
• boolean 은 자료형으로, 참 또는 거짓 둘 중 한 가지 상태를 나타내기 위해 사용하는 논리적인 데이터 타입이다.

#10 DOM (Document Object Model 문서 객체 모델)

DOM 을 알아보기 전에

웹브라우저는 HTML 문서를 해석하고, 화면을 통해 해석된 결과를 보여준다.
해석한 HTML 코드를 화면을 통해 보여주는 과정을 '렌더링'이라 한다.

렌더링의 세부 과정
브라우저는 HTML 코드를 해석해서 요소들을 트리 형태로 구조화해 표현하는 문서(객체)를 생성한다. 이를 DOM 이라 하며, 브라우저는 DOM 을 통해 화면에 웹 콘텐츠들을 렌더링한다.

DOM 의 존재 목적
DOM 은 자바스크립트를 사용해서 웹 콘텐츠를 추가, 수정, 삭제하거나 마우스 클릭, 키보드 타이핑 등 이벤트에 대한 처리를 정의할 수 있도록 제공되는 프로그래밍 인터페이스(interface)이다.

내용 정리
• HTML 코드를 해석해서 요소들을 트리 형태로 구조화해 표현하는 형식이다.
• DOM 은 자바스크립트를 사용해서 웹 화면의 콘텐츠를 추가, 수정, 삭제하거나 이벤트를 처리할 수 있도록 프로그래밍 인터페이스를 제공한다.

#11 문서에 접근하자 그리고 조작하자, document!

window.document
브라우저 객체 window 의 document 속성은 창이 포함한 문서를 참조한다.
즉,window.document 은 현재 브라우저에 렌더링되고 있는 문서를 의미하며,
이 속성을 이용하면 해당 문서에 접근할 수 있다.

좀 있어 보이게 표현하면?
window.document 는 페이지 콘텐츠, 즉 DOM 에 대한 진입점 역할을 하는 프로그래밍 인터페이스이다! 이를 이용하면 페이지의 정보를 얻거나 웹 요소를 생성 및 조작할 수 있다.

속성과 메소드를 제공한다
document 는 문서(HTML,XML,SVG 등)에 대한 공통의 속성과 메소드를 제공

한다.즉,다양한 API(Application Programming Interface)를 제공한다.

너무 많아서 다 해 볼 수는 없으므로, 아래 관련문서 링크.
=> https://developer.mozilla.org/ko/docs/Web/API/Document

다음 두 메소드는 요소를 선택하기 위해 사용할 수 있는 대표적인 메소드이다.

document.querySelector
document.getElementById

=>두 메소드는 모두 요소(Element) 객체를 반환한다.

document.querySelector

document 의 querySelector 메소드는 선택자를 인자로 전달받아, 전달받은 선택자와 일치하는 문서 내 첫 번째 요소(Element)를 반환한다.일치하는 요소가 없으면 '없다'라는 의미의 null 데이터를 반환한다.
인자로 전달되는 선택자는 문자열 타입의 '유효한 CSS 선택자'를 의미한다.

```
// P 태그를 선택하자!
document.querySelector("p");

// id 가 text 인 요소를 선택하자!
document.querySelector("#text");

// class 가 text 인 요소를 선택하자!
document.querySelector(".text");
```

document.getElementById
document 의 getElementById 메소드는 id 를 인자로 전달받아, 전달받은 선택자와 일치하는 문서 내 요소(Element)를 반환한다.일치하는 요소가 없으면'없다'라는 의미의 null 데이터를 반환한다.

인자로 전달되는 선택자는 문자열 타입의 'id'를 의미한다.

```
// id 가 text 인 요소를 선택하자!
document.getElementById("text");

// id 가 image 인 요소를 선택하자!
document.getElementById("image");
```

textContent
textContent 속성은 해당 노드가 포함하고 있는 텍스트 콘텐츠를 표현하는 속성이다.textContent 를 통해 요소가 포함한 텍스트를 읽을 수도, 변경할 수도 있다.

```
// p 요소를 반환받아 상수로 이름을 붙인다!
const p = document.querySelector("p");

// p 요소의 textContent 속성을 콘솔에 출력해보자!
console.log(p.textContent)

// p 요소의 textContent 값을 변경해보자!
p.textContent = "텍스트를 이걸로 바꿔!"
```

-index.html-

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>자바스크립트 실습해보자</title>
        <style>

        </style>
    </head>
    <body>
```

```
        <h1>큰 제목입니다</h1>
        <p>헬로우 이건 문단입니다.</p>
        <p id="text">아이디가 있는 문단입니다</p>
        <p class="paragraph">클레스가 있는 문단입니다</p>

        <script src="script.js"></script>
    </body>
</html>
```

-script.js-

```
console.log(document.querySelector("h1"));
console.log(document.querySelector("p"));
console.log(document.querySelector("#text"));
console.log(document.querySelector(".paragraph"));


console.log(document.getElementById("text"));
console.log(document.getElementById("p"));
```

```
const h1 = document.querySelector("h1");
const p = document.getElementById("text");

console.log(h1.textContent);
h1.textContent = "헬로우 여러분!!";

p.textContent = "겟엘리먼트바이아이디!!";
console.log(p.textContent);
```

내용 정리

- window 의 document 속성은 창이 포함한 문서를 참조한다.
- document 는 모든 종류의 문서에 대한 공통의 속성과 메소드를 제공한다.
- document 의 querySelector 메소드는 선택자를 인자로 전달받아, 전달받은

선택자와 일치하는 문서 내 첫 번째 요소(Element)를 반환한다.
・document 의 getElementById 메소드는 id 를 인자로 전달받아, 전달받은 선택자와 일치하는 문서 내 요소(Element)를 반환한다.
・textContent 속성은 노트가 포함하고 있는 텍스트 콘텐츠를 표현한다.

#12 비교 연산자
자바스크립트에서 비교란?
자바스크립트는 주어진 두 항을 비교할 수 있는 '비교 연산자'를 제공한다.
자바스크립트에서는 다음 두 가지 유형이 비교를 할 수 있다.

・크냐 작냐(대소 비교)
・같냐 다르냐(등가 비교)

=>무언갈 비교하는 작업 또한 '연산'의 일종이라 할 수 있습니다!

비교 연산의 특징
비교 연산식은 언제나 boolean 데이터를 반환한다.
크냐 작냐를 비교하는 대소 비교, 같냐 다르냐를 비교하는 등가 비교는 모두 하나의 질문이며, 질문에 대한 답이 참(true) 또는 거짓(false)인 것이다.

예를 들어)
3 이 5 보다 작냐? true!
6 과 6 이 다르냐? false!

대소 비교
대소 비교는 네가지 연산자를 사용해 처리할 수 있다.
(연산자 의미는 앞에 있는 피연산자 기준)
크다 > , 작다 < , 크거나 같다 >= , 작거나 같다 <=

등가 비교
등가 비교는 다음 네 가지 연산자를 사용해 처리할 수 있다.
등호(이퀄 사인,=) 하나는 대입연산자이므로, 기호 개수를 잘 봐가며 사용하자.

같다 === , 같지 않다 != , 완전히 같다 === , 완전히 같지 않다 !==

같으면 같은 거지 완전한 건 뭐야.
등가 비교를 할 때, 등호(=)의 개수에 따라 비교 규칙에 차이를 보인다.
== 는 '추상적(abstract) 같음 비교'로써,자료형이 서로 다르더라도 같다고 판단할 수 있는 비교이다.
=== 는 '엄격한(strict) 같음 비교'로써, 자료형과 데이터가 모두 일치해야만 같다고 판단한다.

```
console.log('1' == 1)  // true
console.log('1' === 1)  //false
```

-script.js-

```
let num1 = 3;
let num2 = 3;

console.log(num1 > num2);
console.log(num1 < num2);

let num3 = '3';
let num4 = 3;

console.log(num1 == num2);
console.log(num3 === num4);


console.log(num1 != num2);
console.log(num3 !== num4);
```

내용 정리
• 비교 연산자를 이용하면 대소 및 등가 비교를 수행할 수 있다.
• 비교 연산식은 언제나 boolean 데이터를 반환한다.

• 이퀄(=)하나는 대입 연산자를, 두 개부터는 비교 연산자 역할을 담당한다.
• 등가 비교에는 추상적 비교와 엄격한 비교가 각각 존재한다.

#13 조건문 if~else
조건문
조건문이란 주어진 조건의 참/거짓 여부에 따라 프로그램의 흐름을 결정할 수 있는 구문을 뜻한다.
여기에는 조건이란,불리언 데이터를 반환하거나 불리언 데이터로 해석할 수 있는 표현식을 의미한다.

조건의 예)
게임 캐릭터의 HP 가 0 입니까? => true 일 경우 캐릭터는 죽는다
지하철 요금을 낼 만큼의 돈이 있습니까? => true 일 경우 지하철을 탈 수 있다.

if 문
키워드 if 를 이용해 만드는 구문 'if 문'은 가장 일반적인 형태의 조건문이다.
if 문의 기본 형태는 다음과 같다.

```
if(조건){
  // 조건이 true 일 때 실행할 코드
}
```

if 문 사용 예

```
let number = 3;
if(number === 3){
  console.log("It is true!");
```

```
}
```

코드해석)
변수 number 의 값이 3 과 같다면 콘솔에 "It is true!"를 출력해라
(3 과 같지 않을 경우 아무 일도 일어나지 않음)

if 문에 else 를 추가하자
if 는 조건이 참일 때 할 일을 정의하는 구문을 만든다. 원한다면 사용자는 else 를 추가하여 조건이 거짓일 때의 할일을 추가 정의할 수 있다.

```
if(조건){
  //조건이 true 일 때 실행할 코드
}else{
  //조건이 false 일 때 실행할 코드
}
```

=>else 를 추가한 if 문을 if-else 문이라 부르기도 한다!

if-else 문 사용 예

```
let number = 5;
if(number === 3){
  console.log("It it true!");
}else{
  console.log("It it false!");
}
```

변수 number 의 값이 3 과 같다면 콘솔에 "It is true!"를 출력해라.
그렇지 않을 경우에는 콘솔에 "It is false!"를 출력해라.

이 점을 잊지말자
조건문은 '주어진 조건에 따라 서낵적으로 구문을 실행한다'는 특징을 가진 하나의 구문일 뿐이다. 조건문이 실행을 마치고 나면 코드 진행 흐름은 이어지는 다음 구문으로 자연스럽게 이동한다.

조건문은 하나의 구문일 뿐

```
let number = 5;              //<--구문 하나

if(number == 3){             //<--구문 하나
 console.log("It it true!");
 console.log("It it true!");
 console.log("It it true!");
}else{
 console.log("It it false!");
 console.log("It it false!");
 console.log("It it false!");
}

console.log("Hello~~");    //<--구문 하나
```

내용 정리
• 조건문은 조건의 참/거짓 여부에 따라 프로그램의 흐름을 결정하는 구문이다.
• 키워드 if 를 이용해 만드는 if 문이 대표적인 조건문이다.
• if 문은 조건이 참일 경우에만 실행될 코드를 정의할 수 있는 구문이다.
• 키워드 else 를 추가하여 조건이 거짓일 경우에 실행될 코드를 추가 정의할 수 있다.

#14 반복문 while
반복문
반복문은 비슷하거나 동일한 구문을 반복해서 수행할 수 있는 구문이다.
대표적인 반복문은 다음 두 가지이며, 두 반복문은 구조 및 동작방식에서 차이를

보인다.

・while (이번 시간에는 이거 함)
・for 문

while 문
키워드 while 을 이용해 만드는 구문 while 문은 '주어진 조건이 참일 동안에 구문을 반복하는' 반복문이다. while 문의 기본 형태는 다음과 같다.

```
while(조건){
  //조건이 true 인 동안에 반복 수행할 코드
}
```

=>조건이란 불리언을 반환하거나 불리언으로 해석될 수 있는 표현식!
=>반복 구문은 '루프 loop'라고도 한다.

```
let number = 1;

// number 가 3 보다 작은 동안에는 반복해라
while(number < 3){
  console.log(number) // 출력한 다음
  number += 1 // number 를 1 씩 증가
}
```

-script.js-

```
let number = 1;

while(number < 10){
    console.log(number);
    number += 1;
}
```

-script.js-

```
while(window.confirm("메시지!!")){
    console.log("확인 버튼을 누르고 있군요.");
}
console.log("취소 버튼을 눌렀군요!");
```

내용 정리

- 반복문은 비슷하거나 동일한 구문을 반복해서 수행할 수 있는 구문이다.
- while 문은 주어진 조건이 true '동안에' 구문을 반복하는 반복문이다.
- 다르게 말하면, 조건이 false 가 될 때까지 구문을 반복한다.
- 조건 확인 -> 구문 실행 -> 조건 확인 -> ??
- 반복 구문은 '루프(loop)'라고도 한다.

#15 반복문 for

반복문

반복문은 비슷하거나 동일한 구문을 반복해서 수행할 수 있는 구문이다.

- while 문
- for 문 (이번 시간에는 이거 함)

for 문

for 문은 구문 작성시 반복을 위해 필요한 세 가지 요소를 한 곳에 모아
작성함으로써 보다 명시적으로 반복 횟수를 표현할 수 있는 직관적인 구문이다.

```
for(초기식; 조건식; 반복식){
  // 조건이 true 인 경우 반복 수행할 코드
}
```

=>초기식:반복 조건의 초기화 작업

=>반복식:반복이 한 번 끝날 때마다 실행될 작업

```
for(let i = 1; i <= 3; i++){
  console.log(i);
}
```

이 과정은 조건식이 false 를 반환할 때까지 반복된다!

-script.js-

```
for(let i = 1; i <= 8; i += 1){
    console.log("반복 진행 중입니다.");
    console.log(i);
}
```

```
for(let i = 1; i <= 8; i += 1){
    if(i % 2 == 0){
        console.log(i);
    }
}
//짝수만 출력
```

```
for(let i = 8; i >= 1; i -= 2){
  console.log(i);
}
```

내용 정리

- 반복문은 비슷하거나 동일한 구문을 반복해서 수행할 수 있는 구문이다.
- for 문은 초기식,조건식,반복식으로 반복 횟수를 명시적으로 표현할 수 있다.
- for 문의 초기식에서 let 키워드를 사용해 선언한 변수는 for 문의 실행이 끝나

면 사용할 수 없다.

#16 함수 만들기

함수
함수는 '호출될 수 있는 코드 조각'이다. 변수를 선언하고 데이터를 대입하면 변수의 이름을 데이터 대신 사용할 수 있는 것처럼, 함수를 선언하면 함수의 이름을 코드 조각 대신 사용할 수 있다.

```
let value = "문자열";

// value 는 "문자열"대신 사용된다!
console.log(value);
```

```
let work = console.log("!");

// work 는 콘솔에 !를 출력하는 기능 대신 사용된다?
work
```

함수를 만드는 두 가지 방법
함수를 만드는 방법 중 대표적인 두 가지를 알아보자.

- 함수 선언식
- 함수 표현식

함수 선언식
함수 선언식은 다음과 같은 형태를 가진다.

```
function 함수명(){
  // 함수의 기능을 표현한 구문
}
```

=> 위와 같은 선언 이후, 함수명은 중괄호 안의 기능 대신 사용될 수 있다!

함수 표현식
함수 표현식은 다음과 같은 형태를 가진다.

```
const 함수명 = function(){
  // 함수의 기능을 표혀한 구문
}
```

=>마찬가지로 선언 이후, 함수명은 중괄호 안의 기능 대신 사용될 수 있다!

만들었으면 사용을 해야지
함수가 만들어지고 나면, 함수명은 스스로 보관하고 있는 구문 대신 사용될 수 있다. 그리고 함수가 사용되기 위해서는, 함수를 '호출'해야만 한다!

```
const sayHello = function(){
  let number = 3 +  3;
  console.log(number);
}

sayHello();    //함수 호출
```

=>함수를 호출할 때는 함수의 이름 뒤에 반드시 소괄호를 붙여줘야 합니다!

함수의 이름을 지을 때는
함수의 기능을 호출하기 위해서는 함수의 이름을 알아야 한다. 함수 이름을 정할 때의 규칙은 변수 이름 정할 때의 규칙과 유사하나, 다음과 같은 사항들을

추가적으로 고려하는 것이 좋다.

- 함수의 기능을 적절하게 표현할 수 있는 이름을 사용하자
- 명사보다는 동사로 된 이름을 사용하자(기능이므로)
- 소문자로 시작하되, 여러 단어가 섞인 경우 카멜표기법을 사용하자

```
let number = 3;
let myNumber = 5;

console.log(muNumber);
```

카멜 표기법으로 구별하는 방법

```
//함수 선언식-
function sayHello(){
   console.log("Hello");
   console.log("Hi");
   console.log("안녕");
}


sayHello()    //함수 호출문

sayHello()    //이와같이 재사용이 용의함.
sayHello()    //함수 선언식은 호이스팅이 가능함.
sayHello()
sayHello()


//함수 표현식-
```

```
const sayBye = function(){
   console.log("good bye~~");
}

sayBye();

//함수 안에서 선언된 변수는 함수 밖에서 사용할수 없음.
//이런것을 지역변수 라고도 함.
```

내용 정리

• 함수는 '호출될 수 있는 코드 조각'이다.
• 함수는 선언식, 표현ㅅ힉 같은 방법을 이용해 만들 수 있고, 사용을 위해서는 함수를 호출해야 한다.
• 함수 호출시에는 반드시 함수명 뒤에 소괄호를 붙인다.
• 함수 안에서 선언한 변수는 함수 안에서만 쓸 수 있다.
• 선언식은 함수 정의보다 호출문을 앞서 작성할 수 있지만, 표현식은 그럴수 없다.

#17 함수와 return

데이터 반환
함수를 만들 때, 함수가 데이터를 반화하도록 할 수 있다.
함수가 데이터를 반환한다는 것은, 함수 호출문이 데이터로 대체됨을 뜻한다!

```
// 그냥 3 을 출력
console.log(3);
```

// 3 을 반환하는 함수 호출문을 출력
console.log(getThree());

저절로 되는 건 아니에요
함수가 데이터를 반환하려면, '이 데이터를 반환한다'라는 구문을 함수 내부에 추가해주어야 한다. 이때 키워드 return 이 사용된다.

```
function getThree(){
  // 이 함수를 호출하면, 호출문이 3 을 반환할 것!
  return 3;
}
```

=> 함수 내부에 return 문 외 다른 코드를 포함해도 된다.

return 너는 누구니
return 은 함수 내부에서 함수의 부가 기능을 위해 사용하는 키워드이다.
return 은 두 가지 기능을 가지고 있다.

- 함수로부터 데이터를 반환한다.
- 함수를 끝낸다.

데이터를 반화한다
return 은 뒤에 붙은 데이터를 반환한다. 이때 데이터는 딱 하나씩만 반환할 수 있다.

```
function getData(){
  let result = 3 + 2 + 1;
  return result;
}
console.log(getData());
```

```
function getData(){
  let result1 = 3 + 2 + 1;
  let result2 = 4 + 3 + 2 + 1;
  return result1;
  return result2;  // 소용 없음.
}
console.log(getData());
```

함수를 끝낸다.
return 은 데이터 반환 뿐 아니라 함수를 강제로 종료하는 역할도 수행할 수 있다.
종료 목적만 있는 경우에는 return 뒤에 데이터를 기입하지 않아도 된다.

```
function sayHello(){
  console.log("Hi");
  console.log("Hello");
  return;
  console.log("안녕!");
}
sayHello();
```

결과:
Hi!
Hello!

---

confirm 은 다이얼로그를 띄운 다음에 다이얼로그 에서 받은 값을 반환한다.

```
const result = window.confirm("확인해봐요!");
console.log(result);
```

```
function noReturn(){
    console.log("반환하지 않는다, 아무것도!");
}

const result = noReturn();
console.log(result); //이때 리절트는 undefined 가 나온다.
```

```
function thereIsReturn(){
    console.log("반환한다, 무언가를!");
    return 10;   //함수를 종료시킨다 라는 리턴의 기능으로
}                //리턴을 여러번 쓸수 없으며
                 //return 10,20,30,40,50 이 경우 50 만 적용됨.

const result = thereIsReturn();
console.log(result);
```

```
function thereIsReturn(){
    console.log("반환한다, 무언가를!");
    let num = 5; //지역 함수도 사용가능. 함수 밖에서는 사용불가!
    return num;
}
```

```
const result = thereIsReturn();
console.log(result);
```

내용 정리
• 함수를 만들 때, 함수가 데이터를 반환하도록 할 수 있다.
• 함수가 데이터를 반환하면, 함수 호출문이 데이터로 대체된다.
• return 은 함수로부터 데이터를 반환하는 역할을 한다.
• return 은 함수를 강제로 종료시키는 역할도 한다.

#18 함수와 매개변수

함수에 재료를 전달하세요
함수 호출문은 '함수명+소괄호'이다. 소괄호의 역할은, 함수가 실행될 때 사용할 재료를 전달받는 것이다. 재료란 데이터를 뜻한다.

```
yourFunc(여기);
```

재료를 뭐라고 부를 건데?
재료를 전달받아 기능을 수행하는 함수를 만들 때는, 재료의 이름을 정해 주어야 한다. 이를 '매개변수(parameter)'라 한다.

```
/*
함수 호출 시, sayVegetable 함수에는
데이터 하나를 전달할 수 있다.
전달된 데이터를 vagetable 이라 부르고,
```

이것을 사용해 구문을 수행하도록 하겠다!
*/

```
function sayVegetable(vegetable){
  console.log("함수에 전달된 채소는?");
  console.log(vegetable);
}


sayVegetable("당근")  //vegetable 은 "당근"
sayVegetable("오이")  //vegetable 은 "오이"
```

결과:
함수에 전달된 채소는?
당근
함수에 전달된 채소는?
오이

재료를 몇 개 사용할 건데?
매개변수는 원하는 만큼 추가할 수 있다. 구분자 쉼표(,)를 이용해 추가하고,
호출 시에도 그에 맞게 재료를 전달한다.

```
function sayFood(food1,food2){
  console.log(food1);
  console.log(food2);
  console.log("먹고싶다!");
}

sayFood("치킨","돈까스");
sayFood("피자","햄버거");
```

--결과--
치킨
돈까스
먹고싶다!
피자
햄버거
먹고싶다!

'재료'라고 부르지는 않고요
재료를 전달받기 위해 만들어 둔 변수를 매개변수(parameter),
실제 함수 호출 시에 전달하는 데이터를 인자(argument)라 부른다!

```
function sayFood(food1,food2){
  console.log(food1);
  console.log(food2);
  console.log("먹고싶다!");
}

sayFood("치킨","돈까스");
sayFood("피자","햄버거");
```

재료를 이용해 반환값을 만든다
인자로 데이터를 전달받고, 이를 이용해 구문을 수행한 다음 그에 따른 결과값을 반환하는 함수를 만들 수 있다.

```
function whatIsBigger(n1,n2){
  if(n1 > n2){
    return n1;
  }else{
    return n2;
```

```
  }
}
console.log(whatIsBigger(3,5));
console.log(whatIsBigger(10,6));

--결과--
5
10

--------
```

우리가 만들 수 있는 함수는
함수를 만들 때 매개변수는 있어도 되고 없어도 된다. 키워드 return 을 통한 데이터 반환 또한 있어도 되고 없어도 된다. 이로써 우리가 만들 수 있는 함수의 유형은 다음의 네 가지로 정리된다.

- 매개변수도, 반환도 없는 함수
- 매개변수는 있지만, 반환은 없는 함수
- 매개변수는 없지만, 반환은 있는 함수
- 매개변수와 반환이 모두 있는 함수

```
//매개변수도, 반환도 없는 함수

function sayAnything(){
    console.log("아무 말이나 하자!");
}
sayAnything();
sayAnything();


//함수를 호출할때 전달되는 값을 콘솔에 출력

function sayAnything2(ant){
```

```
    console.log(ant);
}

sayAnything2("바빠서 영상을 못 만들고 있어요");
sayAnything2("만드는 법을 까먹을 지경ㅠㅠ");


//함수를 호출할때 전달되는 값을 콘솔에 출력 다수
//숫자 만큼 구문을 반복해서 출력함.

function sayAnything3(ant,number){
    for(let i = 0; i < number; i++){
        console.log(ant);
    }
    console.log(number);
}
sayAnything3("바쁘다",6);
sayAnything3("만들다",2);


----------------------

/*숫자 하나를 전달받아 그 숫자가 홀수면 홀수 짝수면 짝수라 문자열을
반환 해주는 함수.*/
function oddEven(number){
    if(number % 2 == 1){
        return "홀수";
    }else{
        return "짝수";
    }
}
console.log(oddEven(10));
console.log(oddEven(7));


내용 정리
```

・함수 호출 시 데이터를 재료로 전달할 수 있다.
・함수 정의시 재료를 전달받기 위해 소괄호 안에 만드는 변수를 매개변수라 한다.
・함수 호출시 전달하는 재료를 인자라 한다.
・매개변수의 사용 여부는 자유롭게 결정할 수 있으며, 개수 또한 자유롭게 결정할 수 있다.

#19 이벤트 그리고 이벤트 핸들러

이벤트(event)가 뭔데요
'사용 중이거나 프로그래밍 중인 시스템 내에서 일어나는 사건'을 뜻한다.
이 강의에서 우리는 웹을 다루고 있고,(당연히)웹에서도 이벤트가 발생한다.

웹에서 발생하는 이벤트의 예
・웹페이지 사용자가 버튼을 클릭했다,클릭 이벤트!
・웹페이지 사용자가 키보드를 눌렀다, 키다운 이벤트!
・웹페이지 사용자가 입력 폼의 내용을 제출했다,제출 이벤트!
・외 다수

발생하면 어떡하죠?
각각의 이벤트들은 이벤트 핸들러(handler)를 가질 수 있다. 이벤트 핸들러란
이벤트가 발생되면 실행될 코드 블록을 뜻하며, 주로 함수가 이 역할을 담당한다.
이벤트 핸들러 역할을 수행할 함수를 정의하는 것을 이벤트 핸들러 등록이라 한다.

=> event handler register:이벤트가 발생하면,이 함수를 호출해라!

예를 들어봅시다
웹 사용자가 버튼(button)요소를 클릭 했을 때,경고 다이얼로그 박스를 띄워

환영의 메시지를 보여주고 싶다면?

```
const handleClick = function(){
   window.alert("환영합니다^^");
}

const button = document.querySelector("button");

button.onclick = handleClick //여기가 포인트!
```

구문 기본 형태
이벤트가 발생할 수 있는(혹은 발생할 예정인)타겟을 선택하고, 이벤트 핸들러 속성에 이벤트 핸들러를 대입한다.

```
타겟.on 이벤트명 = 이벤트핸들러함수;

button.onclick = handleClick;
```

이거 헷갈리면 안 돼요
이벤트 핸들러를 등록하기 위해 이벤트 속성에 함수를 대입하는 것과 함수 호출문을 대입하는 것은 엄연히(!)다르다.

```
// handleClick 함수를 대입한다 (이벤트 핸들러 등록)
button.onclick = handleClick;

// handleClick 호출 후 반환값을 대입한다 (굳이?) <--이렇게 하면 안됨.
button.onclick = handleClick();
```

-index.html-

```
<!DOCTYPE html>
<html lang="en">
      <head>
```

```
        <meta charset="UTF-8">
        <title>자바스크립트 실습해보자</title>
        <style>

        </style>
    </head>
    <body>
        <input type="text" id="typing" />
        <input type="button" id="push" value="PUSH">
        <script src="script.js"></script>
    </body>
</html>
```

-script.js-

```
const inputType = document.querySelector("#typing");
const inputClick = document.querySelector("#push");

const handleTyping = function(){
    console.log("타이핑 되고 있어요!");
}

const handleClick = function(){
    console.log("클릭되고 있어요!");
}

inputType.onkeydown = handleTyping;
inputClick.onclick = handleClick;


// 텍스트 창에 타이핑을 하면 "타이핑 되고 있어요!"가 출력
// 버튼 클릭하면 "클릭되고 있어요!"가 출력
```

==========

```
const inputType = document.querySelector("#typing");
const inputClick = document.getElementById("push");

typingHandle = function(){
    console.log("키보드 입력중 입니다!");
}

clickHandle = function(){
    console.log("마우스 클릭중 입니다!");
}

inputType.onkeydown = typingHandle;
inputClick.onclick = clickHandle;
```

==========

내용 정리

• 이벤트란'시스템 내에서 일어나는 사건'을 뜻한다.
(마우스 클릭,키보드 입력,마우스 휠 스크롤 등 다양해요)
• 각각의 이벤트들은 이벤트 발생 시 대응으로 이벤트 핸들러를 가질 수 있다.
• 이벤트 핸들러 정의하는 작업을 이벤트 핸들러 등록이라 한다.
• 이벤트 속성에 함수를 대입하는 것과 함수 호출문을 대입하는 것은 다르다.

#20 addEventListener

onclick, onkeydown 과 같은 이벤트 속성을 통해 이벤트 핸들러를 등록하는 것보다 현대적인 방법은, addEventListener 메소드를 활용하는 것이다.

클릭 이벤트 핸들러를 등록하는 경우의 예

```
// 이전 영상에서 학습한 방식
```

```
target.onclick = function(){}

// addEventListener 의 방식
target.addEventListener('click', function(){})
```

똑같잖아 뭐가 현대적이야
addEventListener 함수를 사용하는 것은 이벤트 핸들러 속성을 사용하는 것에 비해 몇 가지 이점을 제공한다.

- 이전에 추가한 이벤트 핸들러를 제거할 수 있는 대응 메소드가 존재한다.
- 같은 리스너(타겟)에 대해 다수의 핸들러를 등록할 수 있다.
- 추가적인 옵션 사항들이 제공된다.

이벤트 객체
이벤트 객체는 추가적인 기능과 정보를 제공하기 위해 이벤트 핸들러에 자동으로 전달되는 데이터이다. 이를 활용하기 위해서는 이벤트 핸들러 함수에 매개변수를 추가하여 이벤트 발생 시마다 전달받을 수 있도록 해야 한다.

```
// click 이벤트가 발생하면 함수를 호출하겠다!
target.addEventListener('click',function(){})

// click 이벤트가 발생하면 함수를 호출하겠다!
// + 이때 자동으로 전달되는 이벤트 객체를 매개변수 event 에 대입하겠다!
target.addEventListener('click',function(event){})


//--script.js--
const btn1 = document.getElementById("one");
const btn2 = document.getElementById("two");
const btn3 = document.getElementById("three");

const handleClick = function(){
```

```
    console.log("저를 클릭 하셨나요?");
}

btn2.addEventListener('click',handleClick); //핸들클릭 등록
btn2.addEventListener('click',function(){    //핸들클릭 추가 등록
    console.log("또 다른 핸들러가 추가 등록되었다!");
});
```

```
//--script.js--
const btn1 = document.getElementById("one");
const btn2 = document.getElementById("two");
const btn3 = document.getElementById("three");

const handleClick = function(event){
    console.log(event.target);
}

btn1.addEventListener('click',handleClick);
btn2.addEventListener('click',handleClick);
btn3.addEventListener('click',handleClick);
```

내용정리

• addEventListener 메소드를 활용해 이벤트 핸들러 등록을 할 수 있다.
• 이는 이벤트 핸들러 속성을 사용하는 것보다 현대적인(좋은)방법이다.
• 이벤트 객체는 추가적인 기능과 정보를 제공하기 위해 이벤트 핸들러에 자동 전달되는 데이터이다.
• 이벤트 발생 시에 이벤트 핸들러가 호출될 때 이벤트 객체가 전달되는데, 이때 이벤트 핸들러 함수의 매개변수를 통해 이벤트 객체를 받을 수 있다.

#21 createElement 그리고 appendChild

document.createElement
document 의 createElement 메소드는 지정된 이름의 HTML 요소를 만들어 반환해 준다.

```
document.createElement('div');

document.createElement('p');

document.createElement('a');
```

만들었으면 다야?
HTML 요소가 만들어지고 또 반환 되었다고 해서, 해당 요소가 곧장 웹 브라우저 화면에 추가되는 것은 아니다.

```
document.createElement('div');

document.createElement('p');

document.createElement('a');
```

=>브라우저 화면에 위 요소(div,p,a)들이 곧장 추가될 것 같음? 아님.

그럼 어떻게 해야 해?
appendChild 메소드는 DOM 내 개별 요소('노드'라고도 함)에 자식 요소를 추가할 때 사용하는 메소드이다.

기본 사용법
target.appendChild(자식으로_추가할_요소) -하위 추가-

예제

```
const p = document.createElement("p");
document.body.appendChild(p);
```

appendChild vs append
appendChild 메소드와 비슷한 역할을 하는 append 메소드도 있다.
타겟 요소에 자식 요소를 추가한다는 점에서 같으나, 차이점도 존재한다.

주요한 차이
• appendChild 의 경우 추가한 자식 노드를 반환하지만,append 는 반환 데이터가 없다.
• append 를 이용하면 요소에 노드 객체 또는 문자열을 자식 요소로 추가할 수 있지만, appendChild 는 노드 객체만을 추가할 수 있다.

```
<!DOCTYPE html>
<html lang = "en">
    <head>
        <meta charset="utf-8">
        <title>자바스크립트 실습해보자</title>
        <style>

        </style>
    </head>
    <body>
        <button id="push">눌러</button>
        <div id="area"></div>

        <script src="script.js"></script>
    </body>
</html>
```

--

```
//script.js
const button = document.getElementById("push");
const div = window.document.getElementById("area");

button.addEventListener('click', function(){
    console.log("div 생성 중!");
    const newDiv = document.createElement("div");

    newDiv.style.backgroundColor = "red";
    newDiv.style.width = "200px";
    newDiv.style.height = "200px";

    div.appendChild(newDiv);    //어펜드차일드는 반환값이 있음.
    //div.append("하하");  어펜드는 문자열도 출력됨.
});
```

내용 정리
・document 의 createElement 메소드는 지정된 이름의 HTML 요소를 만들어 반환해 준다.
・appendChild 메소드는 노드에 자식 요소를 추가할 때 사용하는 메소드이다.
・append 메소드 또한 자식 요소를 추가할 때 사용할 수 있는 메소드인데, appendChild 와 기능적으로 다른 면이 있다.

<!-- #22 value 속성 그리고 preventDefault
<inpt>,<select>처럼 사용자로부터 입력을 받는데 사용 되는 요소들이 있다.
여기에서 사용자가 입력한 값을 읽어들일 때는 요소의 value 속성에 접근하자.

차이를 기억하자
-요소의 텍스트에 접근하고 싶다:textContent 또는 innerText
-사용자가 요소에 입력한 값에 접근하고 싶다:value

// 대상 요소의 사용자 입력값을 읽어 콘솔에 출력하자!
console.log(target.value)

// 대상 요소의 사용자 입력값을 "변경값"으로 바꾸자!
target.value = "변경값"

<입력 폼을 다루어 보자>
여러 입력 요소를 포함할 수 있는 폼 요소로부터 여러 입력 밧을 읽을 수 있다.

```
<form>
    <input name="nickname" placeholder="닉네임" />
    <input name="character" placeholder="특징" />
    <input type="submit" value="입력 완료" />
</form>
```

=>form 에서 submit 이벤트 발생하면 입력 값들을 읽자! 앗, 그런데 문제가...

<form 은 원래 action 이 있잖아>
form 은 action 에 지정한 url 로 입력값을 제출한다.
입력값이 없을때는 새로고침이 됨.

```
<form action="없는데요">
    <input name="nickname" placeholder="닉네임" />
    <input name="character" placeholder="특징" />
    <input type="submit" value="입력 완료" />
</form>

-->

<!DOCTYPE html>
<html lang="ja">
    <head>
        <meta charset="UTF-8">
        <title>자바스크립트 실습해보자</title>
        <style>
```

```
        </style>
    </head>
    <body>
        <input type="text" id="text" placeholder="아무거나">
        <input type="button" id="button" value="PUSH">

        <script src="script.js"></script>
    </body>
</html>
```

```
//script.js

const textInput = document.getElementById("text");
const button = document.getElementById("button");

button.addEventListener('click',function(){
    console.log(textInput.value);   //사용자 입력값을 출력해줌.

    textInput.value = "클릭하면 이렇게 바뀝니다"; //직접 쓰지 않고도 입력값이 바뀜.
});
```

```
//script.js

const form = document.querySelector("form");
//쿼리 셀렉터로 폼을 선택함.
```

```
form.addEventListener("submit", function(){ //서브밋 이벤트 발생시 아래의 콘솔을 읽어들임.
    console.log(form.name.value);  //폼에 네임이 네임 안에 벨루를 읽어들이겠다.
    console.log(form.town.value);  //폼에 네임이 타운 안에 벨루를 읽어들이겠다.
});  // (중요!!)폼 안에서는 네임이 식별자의 역할을 함!!!
```

```
//script.js

const form = document.querySelector("form");
//쿼리 셀렉터로 폼을 선택함.

form.addEventListener("submit", function(event){
    event.preventDefault();  //프리벤트 디폴트는 기존의 기능을 차단한다
    //리다이렉션 기능을 차단후 내가 원하는 기능을 실행하겠다는 의미.
    console.log(form.name.value);
    console.log(form.town.value);
});
```

내용 정리
- 사용자가 입력한 값을 읽어들일 때는 요소의 value 속성에 접근하면 된다.
- value 에 접근함으로써 입력 값 읽기 또는 쓰기를 수행할 수 있다.
- form 에 포함된 입력 요소의 name 을 통해 각 입력 요소에 접근할 수 있다.
- form 에서 이벤트가 제출되는 submit 이벤트가 발생하면 action 속성의 url 로 리다이렉트되지만, 이벤트 객체를 통해 기본 기능을 차단할 수 있다.

```
<!--
```

#23 삼항 연산
삼항 연산은 이름 그대로 세 개의 항을 이용해 결과를 반환하는 연산이다.
보통 if 문의 단축 형태로 사용되기 때문에, 삼항 조건 연산식 이라고도 부른다.

//구문
조건식? 참일_경우의_결과:거짓일_경우의_결과

- 조건식:조건 역할을 하는 표현식
- 참일_경우의_결과:조건식의 결과가 참일 경우 반환될 결과
- 거짓일_경우의결과:조건식의 결과가 거짓일 경우 반환될 결과

구체적인 예를 보며 생각해보자

3 > 2 ? "참" : "거짓"

3 > 2
조건식

"참"
조건이 참이면 반환할 결과

"거짓"
조건식이 거짓이면 반환할 결과

-->

```
let result;

result = 3 > 2 ? "true" : "false";

console.log(result);
```

```
<!DOCTYPE html>
<html lang="ja">
    <head>
        <meta charset="UTF-8">
        <title>자바스크립트 실습해보쟈</title>
        <style>

        </style>
    </head>
    <body>
        <slect>
            <option value="foot" selected>축구</option> //기본선택이 축구로 되있음.
            <option value="base">야구</option>
            <option value="basket">농구</option>
        </slect>
        <button>PICK</button>

        <script src="script.js"></script>
    </body>
</html>
```

```
const select = document.querySelector("select");
const button = document.querySelector("button");

button.addEventListener('click',function(){
    console.log(select.value); //사용자의 입력값을 확인.
    let result;
    result = select.value == "foot" ?
    "축구를 선택했네요" :
    "축구를 선택하지 않았네요!"
```

```
    alert(result);
});



/*
내용 정리
- 삼항 연산은 세 개의 항을 이용해 결과를 반환하는 연산이다.
- 삼항 연산식 또는 삼항 조건 연산식이라고도 부른다.
- 첫 번째 항인 조건식의 결과에 따라 남은 두 항 중 하나를 반환한다.
*/



------------

<!--
#23 타이머 만들기 - setTimeout setInterval clearInterval
setTimeout
정해진 시간이 지나고 나면 주어진 함수를 실행 해주는 타이머 메소드!

사용방법 setTimeout(실행할_함수, ms_단위의_시간)

사용 예 // 1000ms 가 지나고 나면 함수를 실행한다!
setTimeout(function(){
    console.log("재미있다!")
},1000)

setInterval
일정한 시간 간격에 따라 함수를 반복 실행할 수 있도록 해주는 타이머 메소드!

사용 방법 setInterval(반복_시행할_함수, ms_단위의_시간)

사용 예  // 500ms 마다 함수를 반복 실행한다!
        setInterval(function(){
            console.log("안녕하세요^^")
        },500);
```

clearInterval
setInterval 메소드가 호출되어 반복 실행할 함수 타이머를 등록하면, 타이머는 0 이 아닌 숫자를 반환한다. 숫자는 타이머의 ID 를 의미하며, 이를 clearinterval 메소드에 전달하면 해당 타이머의 반복 실행이 취소된다.

```
// 셋팅된 타이머의 반환값(ID)을 변수에 저장하자
let timer;
timer = setInterval(function()){
    console.log("안녕하세요^^")
}, 500)

//셋팅된 타이머를 멈춰주세요(취소해주세요)!
clearInterval(timer)

-->
```

```
setTimeout(function(){
    console.log(1234);
    window.alert("1234");
},3000);
//3 초뒤 한번만 실행함.
```

```
setInterval(function(){
    console.log("hello");
},2000)
//2 초마다 실행됨.
```

```
// setTimeout(function(){
//         console.log(1234);
//         window.alert("1234");
// },3000);

const button = document.querySelector("button");

let interId;
interId = setInterval(function(){
      console.log("hello");
},1000)
console.log(interId);

button.addEventListener('click',function(){
      clearInterval(interId);
})


/* 내용 정리
- setTimeout 메소드는 정해진 시간이 지나고 나면 주어진 함수를 실행한다.
- setInterval 메소드는 시간 간격에 따라 주어진 함수를 반복 실행한다.
- setInterval 메소드는 0 이 아닌 수자를 반환하는데, 이는 타이머의 ID 이다.
- clearInterval 메소드는 주어진 ID 에 해당하는 타이머를 제거한다(멈춘다);
*/
```

Element.classList
웹 요소(Element)로부터 클래스 코렉션을 반환하는 읽기 전용 속성이다.

```
<!--
```

#25 클래스 리스트(classList)

```
<p class="hello greet good">
  안녕하세요
</p>

const p = document.querySelector('p');
console.log(p.classList);

//DOMTokenList { 0: 'hello',
1: 'greet', 2: 'good'}
```

메소드도 몇개 있어요
반환된 클래스 콜렉션은 유용한 메소드를 다수 포함하고 있다.

메소드 - add
기능 - 지정한 클래스 값 추가
사용 예 - add("new_class")

메소드 - remove
기능 - 지정한 클래스 값 제거
사용 예 - remove("old_class")

메소드 - toggle
기능 - 클래스 값 토글링
사용 예 - toggle("some_value")

```
-->

<!DOCTYPE html>
<html lang="ja">
    <head>
        <meta charset="UTF-8">
        <title>자바스크립트 실습해보쟈</title>
        <style>
```

```
        </style>
    </head>
    <body>
        <h1 class="text heading">실험용 텍스트</h1>
        <button id="add">add</button>
        <button id="remove">remove</button>
        <button id="toggle">toggle</button>

        <script src="script.js"></script>
    </body>
</html>
```

```
const h1 = document.querySelector("h1");
const addBtn = document.querySelector("#add");
const removeBtn = document.querySelector("#remove");
const toggleBtn = document.querySelector("#toggle");

console.log(h1.classList);
//콘솔에서 클레스 리스트를 표시해 준다.
```

```
<!DOCTYPE html>
<html lang="ja">
    <head>
        <meta charset="UTF-8">
        <title>자바스크립트 실습해보쟈</title>
        <style>
            .text{
                color:red;
```

```
            }
        </style>
    </head>
    <body>
        <h1>실험용 텍스트</h1>
        <button id="add">add</button>
        <button id="remove">remove</button>
        <button id="toggle">toggle</button>

        <script src="script.js"></script>
    </body>
</html>
```

```
const h1 = document.querySelector("h1");
const addBtn = document.querySelector("#add");
const removeBtn = document.querySelector("#remove");
const toggleBtn = document.querySelector("#toggle");

addBtn.addEventListener('click',function(){
    h1.classList.add('text'); //h1 폰트 빨강색 적용
});
removeBtn.addEventListener('click',function(){
    h1.classList.remove('text'); //h1 폰트 빨강색 해제
});
toggleBtn.addEventListener('click',function(){
    h1.classList.toggle('text');  //h1 폰트 지정시는 생성, 생성시는 해제함.
})
```

내용 정리

- classList 는 요소로부터 클래스 콜렉션을 반환하는 읽기 전용 속성이다.

- 반환된 클래스 콜렉션은 유용한 메소드를 다수 포함하고 있다.
- 일반적으로 toggle 은 있으면 제거하고, 없으면 생성한다는 의미이다.

#26 로컬스토리지 (localStorage)

window.localStorage
localStorage 속성은 현재 도메인의 로컬 저장소에 접근할 수 있게 해 준다!
로컬 저장소는 웹브라우저에서 각 도메인에 대해 할당해주는 저장 공간으로, 여기에는 데이터를 영구적으로 보관할 수가 있다.
데이터 보관 시에는 데이터의 이름과 데이터의 실제 값을 각각 지정하며, 이때 데이터 타입은 문자열 형태만 허용된다.

영구적으로 보관한다는 말은?
브라우저를 껏다가 켜거나 페이지를 새로고침해도 해당 페이지에 데이터가 남아있도록 할 수 있다는 뜻!

로컬스토리지 사용법
로컬스토리지로부터 데이터를 읽거나 쓸 때에는 메소드를 이용해 접근한다.

메소드명 - setItem
기능 - 키와 밸류를 전달받아 저장
사용 예 - setItem("key","value")

메소드명 - getItem
기능 - 전달받은 키에 해당하는 밸류를 반환
사용 예 - getItem("key")

메소드명 - removeItem
기능 - 전달받은 키에 해당하는 데이터 삭제
사용 예 - removeItem("key")

메소드명 - clear

기능 - 모든 데이터 삭제
사용 예 - clear()

=>로컬스토리지의 데이터 이름은 중복될 수 없다!

const myName = "윤호" //처음에 한번

console.log(myName);

//웹브라우저에 저장하는 방식
localStorage.setItem("myName",myName); //setItem 은 로컬스토리지 데이터를 저장해줌.

----

const myName = localStorage.getItem("myName");

alert(myName);

----

localStorage.setItem("cat","고양이");
localStorage.setItem("dog","강아지");
localStorage.setItem("myName","강윤호");

//문자열만 저장 가능. 영구적으로 브라우저에 저장해줌.

----

localStorage.clear();

//클리어는 인자가 없으며, 다 지워줌.

내용 정리
・window 객체의 localStorage 속성은 현재 도메인의 로컬 저장소에
접근할 수 있게 해 준다.
・데이터를 영구적으로 보관할 수 있으며, 데이터는 키와 밸류를 각각 가진다.
・키는 중복될 수 없다. 그러나 밸류는 같은 것을 쓸 수 있다.
・문자열 형태의 데이터만 보관할 수 있다.
・로컬스토리지로부터 데이터를 읽거나 쓸 때에도 메소드를 이용해 접근한다.
