230529_실습

JavaScript

배열

배열 생성

1. 배열 리터럴 대괄호([ ])를 사용하여 만드는 방법

```
// 배열 생성 (빈 배열)
var arr = [];

arr[0] = 'zero';
arr[1] = 'one';
arr[2] = 'tow';

for (var i = 0; i < arr.length; i++) {
    console.log(arr[i]);
}
```

```
// 배열 생성 (초기 값 할당)
var arr = ['zero', 'one', 'tow'];

for (var i = 0; i < arr.length; i++) {
    console.log(arr[i]);
}
```

```
// 배열 생성 (배열 크기 지정)
// 쉼표 개수만큼 크기가 지정됨
var arr = [,,,];

for (var i = 0; i < arr.length; i++) {
    console.log(arr[i]);
}

// 값이 할당되지 않아서 undefined 3번 출력
```

2. Array() 생성자 함수로 배열을 생성하는 방법

```
// 배열 생성 (빈 배열)
var arr = new Array();
```

```
arr[0] = 'zero';
arr[1] = 'one';
arr[2] = 'tow';

for (var i = 0; i < arr.length; i++) {
    console.log(arr[i]);
}
```

```
// 배열 생성 (초기 값 할당)
var arr = new Array('zero', 'one', 'tow');

for (var i = 0; i < arr.length; i++) {
    console.log(arr[i]);
}
```

```
// 배열 생성 (배열 크기 지정)
// 원소가 1개이고 숫자인 경우 배열 크기로 사용됨
var arr = new Array(3);

for (var i = 0; i < arr.length; i++) {
    console.log(arr[i]);
}

// 값이 할당되지 않아서 undefined 3번 출력
```

JavaScript 배열의 특징

- 배열 내부의 데이터 타입이 서로 다를 수 있다

```
// 서로 다른 데이터 타입을 담을 수 있다
var arr = [1234, 'test', true];
```

- 배열의 크기는 동적으로 변경될 수 있다

```
var arr = [1234, 'test', true];

// 배열의 크기를 임의로 변경( 3 -> 5 )
// arr[3], arr[4]는 값이 할당 되지 않았기 때문에 undefined
arr.length = 5;

// 새로운 배열을 추가하면 크기는 자동으로 변경 ( 5 -> 6 )
arr[5] = 'apple';

// 새로운 배열 추가로 크기 변경 ( 6 -> 7 )
arr.push('banana');

for (var i = 0; i < arr.length; i++) {
```

```
    console.log(arr[i]);
}

/*
// 출력 결과
1234
test
true
undefined
undefined
apple
banana
*/
```

반복문

for 문

- for 반복문은 어떤 특정한 조건이 거짓으로 판별될 때까지 반복합니다. 자바스크립트의 반복문은 C의 반복문과 비슷합니다. for 반복문은 다음과 같습니다.

```
for ([초기문]; [조건문]; [증감문]) {
      문장
}
```

- for문이 실행될 때, 다음과 같이 실행됩니다.
  1. 초기화 구문인 초기문이 존재한다면 초기문이 실행됩니다. 이 표현은 보통 1이나 반복문 카운터로 초기 설정이 됩니다. 그러나 복잡한 구문으로 표현 될 때도 있습니다. 또한 변수로 선언 되기도 합니다
  2. 조건문은 조건을 검사합니다. 만약 조건문이 참이라면, 그 반복문은 실행됩니다. 만약 조건문이 거짓이라면, 그 for문은 종결됩니다. 만약 그 조건문이 생략된다면, 그 조건문은 참으로 추정됩니다.
  3. 문장이 실행됩니다. 많은 문장을 실행할 경우엔, { } 를 써서 문장들을 묶어 줍니다.
  4. 갱신 구문인 증감문이 존재한다면 실행되고 2번째 단계로 돌아갑니다.
- 예시

```
<form name="selectForm">
  <p>
    <label for="musicTypes">Choose some music types, then click the button below:
</label>
    <select id="musicTypes" name="musicTypes" multiple="multiple">
      <option selected="selected">R&B</option>
```

```
      <option>Jazz</option>
      <option>Blues</option>
      <option>New Age</option>
      <option>Classical</option>
      <option>Opera</option>
    </select>
  </p>
  <p><input id="btn" type="button" value="How many are selected?" /></p>
</form>

<script>
function howMany(selectObject) {
  var numberSelected = 0;
  for (var i = 0; i < selectObject.options.length; i++) {
    if (selectObject.options[i].selected) {
      numberSelected++;
    }
  }
  return numberSelected;
}

var btn = document.getElementById("btn");
btn.addEventListener("click", function(){
  alert('Number of options selected: ' + howMany(document.selectForm.musicTypes))
});
</script>
```

```
for (var i = 0; i < selectObject.options.length; i++) {
  if (selectObject.options[i].selected) {
    numberSelected++;
  }
}
```

do… while 문

- do...while 문은 특정한 조건이 거짓으로 판별될 때까지 반복합니다. do...while 문은 다음과 같습니다.

```
do {
      문장
} while (조건문);
```

- **조건문을 확인하기 전에 문장은 한번 실행됩니다.** 많은 문장을 실행하기 위해선 { }를 써서 문장들을 묶어줍니다. 만약 조건이 참이라면, 그 문장은 다시 실행됩니다. 매 실행 마지막마다 조건문이 확인됩니다. 만약 조건문이 거짓일 경우, 실행을 멈추고 do...while 문 바로 아래에 있는 문장으로 넘어가게 합니다.
- 예시

```
do {
  i += 1;
  console.log(i);
} while (i < 5);
```

while 문

- while 문은 어떤 조건문이 참이기만 하면 문장을 계속해서 수행합니다. while 문은 다음과 같습니다.

```
while (조건문) {
      문장
}
```

- 만약 조건문이 거짓이 된다면, 그 반복문 안의 문장은 실행을 멈추고 반복문 바로 다음의 문장으로 넘어갑니다.
- 조건문은 반복문 안의 문장이 실행되기 전에 확인 됩니다. 만약 조건문이 참으로 리턴된다면, 문장은 실행되고 그 조건문은 다시 판별됩니다. 만약 조건문이 거짓으로 리턴된다면, 실행을 멈추고 while문 바로 다음의 문장으로 넘어가게 됩니다.
- 많은 문장들을 실행하기 위해선, { }를 써서 문장들을 묶어줍니다.
- 예시 1

```
n = 0;
x = 0;
while (n < 3) {
  n++;
  x += n;
}
```

  - 매 반복과 함께, n이 증가하고 x에 더해집니다. 그러므로, x와 n은 다음과 같은 값을 갖습니다.
    - 첫번째 경과 후: `n` = 1 and `x` = 1
    - 두번째 경과 후: `n` = 2 and `x` = 3
    - 세번째 경과 후: `n` = 3 and `x` = 6
  - 세번째 경과 후에, n < 3 은 더이상 참이 아니므로, 반복문은 종결됩니다.
- 예시 2

```
// 다음과 같은 코드는 피하세요.
while (true) {
  console.log("Hello, world");
}
```

Label 문

- 여러분이 프로그램에서 다른 곳으로 참조할 수 있도록 식별자로 문을 제공합니다. 예를 들어, 여러분은 루프를 식별하기 위해 레이블을 사용하고, 프로그램이 루프를 방해하거나 실행을 계속할지 여부를 나타내기 위해 break나 continue 문을 사용할 수 있습니다.

```
label :
       statement
```

- 레이블 값은 예약어가 아닌 임의의 JavaScript 식별자일 수 있습니다. 여러분이 레이블을 가지고 식별하는 문은 어떠한 문이 될 수 있습니다.
- 예시

```
markLoop:
while (theMark == true) {
   doSomething();
}
//레이블 markLoop는 while 루프를 식별합니다.
```

break 문

- break문은 반복문, switch문, 레이블 문과 결합한 문장을 **빠져나올 때 사용**합니다.
  - 레이블 없이 break문을 쓸 때: 가장 가까운 `while`, `do-while`, `for`, 또는 `switch`문을 종료하고 다음 명령어로 넘어갑니다.
  - 레이블 문을 쓸 때: 특정 레이블 문에서 끝납니다.
- break문의 문법은 다음과 같습니다.
  1. `break;`
  2. `break [레이블];`
- break문의 첫번째 형식은 가장 안쪽의 반복문이나 switch문을 빠져나옵니다. 두번째 형식는 특정한 레이블 문을 빠져나옵니다.
- 예시1

```
for (i = 0; i < a.length; i++) {
  if (a[i] == theValue) {
    break;
  }
}
```

- 예시 2 : Breaking to a label

```
var x = 0;
var z = 0
labelCancelLoops: while (true) {
  console.log("Outer loops: " + x);
  x += 1;
  z = 1;
  while (true) {
    console.log("Inner loops: " + z);
    z += 1;
    if (z === 10 && x === 10) {
      break labelCancelLoops;
    } else if (z === 10) {
      break;
    }
  }
}
```

Continue 문

- `continue`문은 while, do-while, for, 레이블 문을 다시 시작하기 위해 사용될 수 있습니다.
  - 레이블없이 continue를 사용하는 경우, 그것은 가장 안쪽의 while, do-while, for 문을 둘러싼 현재 반복을 종료하고, 다음 반복으로 루프의 실행을 계속합니다. break 문과 달리, continue 문은 전체 루프의 실행을 종료하지 않습니다. while 루프에서 그것은 다시 조건으로 이동합니다. for 루프에서 그것은 증가 표현으로 이동합니다.
  - 레이블과 함께 continue를 사용하는 경우, continue는 그 레이블로 식별되는 루프문에 적용됩니다.
- continue 문의 구문은 다음과 같습니다:
  1. `continue;`
  2. `continue label;`
- 예시 1

```
i = 0;
n = 0;
while (i < 5) {
  i++;
  if (i == 3) {
    continue;
  }
  n += i;
}
//i 값이 3일 때 실행하는 continue 문과 함께 while 루프를 보여줍니다. 따라서, n은 값 1, 3, 7, 1
2를 취합니다.
```

- 예시 2

```
checkiandj:
  while (i < 4) {
    console.log(i);
    i += 1;
    checkj:
      while (j > 4) {
        console.log(j);
        j -= 1;
        if ((j % 2) == 0) {
          continue checkj;
        }
        console.log(j + " is odd.");
      }
      console.log("i = " + i);
      console.log("j = " + j);
  }
```

  - checkiandj 레이블 문은 checkj 레이블 문을 포함합니다. continue가 발생하는 경우, 프로그램은 checkj의 현재 반복을 종료하고, 다음 반복을 시작합니다. 그 조건이 false를 반환 할 때까지 continue가 발생할 때마다, checkj는 반복합니다. false가 반환될 때, checkiandj 문의 나머지 부분은 완료되고, 그 조건이 false를 반환 할 때까지 checkiandj는 반복합니다. false가 반환될 때, 이 프로그램은 다음 checkiandj 문에서 계속됩니다.
  - continue가 checkiandj의 레이블을 가지고 있다면, 프로그램은 checkiandj 문 상단에서 계속될 것입니다.

for… in 문

- for… in 문은 객체의 열거 속성을 통해 지정된 변수를 반복합니다. 각각의 고유한 속성에 대해, JavaScript는 지정된 문을 실행합니다. for...in 문은 다음과 같습니다:

```
for (variable in object) {
  statements
}
```

- 예시

```
//다음 함수는 객체와 객체의 이름을 함수의 인수로 취합니다. 그런 다음 모든 객체의 속성을 반복하고 속성
 이름과 값을 나열하는 문자열을 반환합니다.
function dump_props(obj, obj_name) {
  var result = "";
  for (var i in obj) {
    result += obj_name + "." + i + " = " + obj[i] + "<br>";
  }
  result += "<hr>";
  return result;
}
```

  - 속성 make와 model을 가진 객체 car의 경우, 결과는 다음과 같습니다:

```
car.make = Ford
car.model = Mustang
```

- 배열

  - `배열`요소를 반복하는 방법으로 이를 사용하도록 유도될 수 있지만, **for...in** 문은 숫자 인덱스에 추가하여 사용자 정의 속성의 이름을 반환합니다. 따라서 만약 여러분이 사용자 정의 속성 또는 메서드를 추가하는 등 Array 객체를 수정한다면, 배열 요소 이외에도 사용자 정의 속성을 통해 **for...in** 문을 반복하기 때문에, 배열을 통해 반복할 때 숫자 인덱스와 전통적인 `for` 루프를 사용하는 것이 좋습니다.

for… of 문

- `for...of` 문은 각각의 고유한 특성의 값을 실행할 명령과 함께 사용자 지정 반복 후크를 호출하여, 반복 가능한 객체(`배열`, `Map` (en-US), `Set`, 인수 객체 등을 포함)를 통해 반복하는 루프를 만듭니다.

```
for (variable of object) {
      statement
    }
```

- 다음 예는 for...of 루프와 `for...in` 루프의 차이를 보여줍니다. 속성 이름을 통해 for...in이 반복하는 동안, for...of은 속성 값을 통해 반복합니다:

```
let arr = [3, 5, 7];
arr.foo = "hello";

for (let i in arr) {
   console.log(i); // logs "0", "1", "2", "foo"
}

for (let i of arr) {
   console.log(i); // logs "3", "5", "7"
}
```

조건문

if … else 문

- 기본 if … else 문법

```
if (조건) {
  만약 조건(condition)이 참일 경우 실행할 코드
} else {
  대신 실행할 다른 코드
}
```

```
if (조건) {
  만약 조건(condition)이 참일 경우 실행할 코드
}

실행할 다른 코드
```

- 예시

```
let shoppingDone = false;
let childsAllowance;

if (shoppingDone === true) {
  childsAllowance = 10;
} else {
  childsAllowance = 5;
}
```

else if

- 두 가지보다 더 많은 것을 원할때 사용

- 예시

```
let year = prompt('ECMAScript-2015 명세는 몇 년도에 출판되었을까요?', '');

if (year < 2015) {
  alert( '숫자를 좀 더 올려보세요.' );
} else if (year > 2015) {
  alert( '숫자를 좀 더 내려보세요.' );
} else {
  alert( '정답입니다!' );
}
```

조건부 연산자 '?'

- 문법

```
let result = condition ? value1 : value2;
```

- 예시

```
let accessAllowed;
let age = prompt('나이를 입력해 주세요.', '');

if (age > 18) {
  accessAllowed = true;
} else {
  accessAllowed = false;
}

alert(accessAllowed);
```

  - '물음표(question mark) 연산자'라고도 불리는 '조건부(conditional) 연산자'를 사용하면 위 예시를 더 짧고 간결하게 변형할 수 있습니다.
  - 조건부 연산자는 물음표 `?` 로 표시합니다. 피연산자가 세 개이기 때문에 조건부 연산자를 '삼항(ternary) 연산자'라고 부르는 사람도 있습니다. 참고로, 자바스크립트에서 피연산자를 3개나 받는 연산자는 조건부 연산자가 유일합니다.

```
let accessAllowed = (age > 18) ? true : false;
```

  - `age > 18` 주위의 괄호는 생략 가능합니다. 물음표 연산자는 우선순위가 낮으므로 비교 연산자 `>` 가 실행되고 난 뒤에 실행됩니다.

- 아래 예시는 위 예시와 동일하게 동작합니다.

```
// 연산자 우선순위 규칙에 따라, 비교 연산 'age > 18'이 먼저 실행됩니다.
// (조건문을 괄호로 감쌀 필요가 없습니다.)
let accessAllowed = age > 18 ? true : false;
```

- 괄호가 있으나 없으나 차이는 없지만, 코드의 가독성 향상을 위해 괄호를 사용할 것을 권유합니다.

주의:

비교 연산자 자체가 `true`나 `false`를 반환하기 때문에 위 예시에서 물음표 연산자를 사용하지 않아도 됩니다.

```
// 동일하게 동작함
let accessAllowed = age > 18;
```

Event

- 프로그래밍하고 있는 시스템에서 일어나는 사건(action) 혹은 발생(occurrence)인데, 이는 여러분이 원한다면 그것들에 어떠한 방식으로 응답할 수 있도록 시스템이 말해주는 것
- 웹페이지에서 마우스를 클릭했을 때, 키를 입력했을 때, 특정요소에 포커스가 이동되었을 때 어떤 사건을 발생시키는 것
- 이벤트의 종류
  - 포커스 이벤트(focus, blur)
  - 폼 이벤트(reset, submit)
  - 뷰 이벤트(scroll, resize)
  - 키보드 이벤트(keydown, keyup)
  - 마우스 이벤트(mouseenter, mouseover, click, dbclick, mouseleave)
  - 드래그 앤 드롭 이벤트 (dragstart, drag, dragleave, drop)

1. 마우스 이벤트

이벤트	설명

이벤트	설명
click	요소에 마우스를 클릭했을 때 이벤트가 발생
dbclick	요소에 마우스를 더블클릭했을 때 이벤트가 발생
mouseover	요소에 마우스를 오버했을 때 이벤트가 발생
mouseout	요소에 마우스를 아웃했을 때 이벤트가 발생
mousedown	요소에 마우스를 눌렀을 때 이벤트가 발생
mouseup	요소에 마우스를 떼었을 때 이벤트가 발생
mousemove	요소에 마우스를 움직였을 때 이벤트가 발생
contextmenu	context menu(마우스 오른쪽 버튼을 눌렀을 때 나오는 메뉴)가 나오기 전에 이벤트 발생
mouseenter	마우스포인터 요소안 이동

2. 키 이벤트

이벤트	설명
keydown	키를 눌렀을 때 이벤트가 발생
keyup	키를 떼었을 때 이벤트가 발생
keypress	키를 누른 상태에서 이벤트가 발생

3. 폼 이벤트

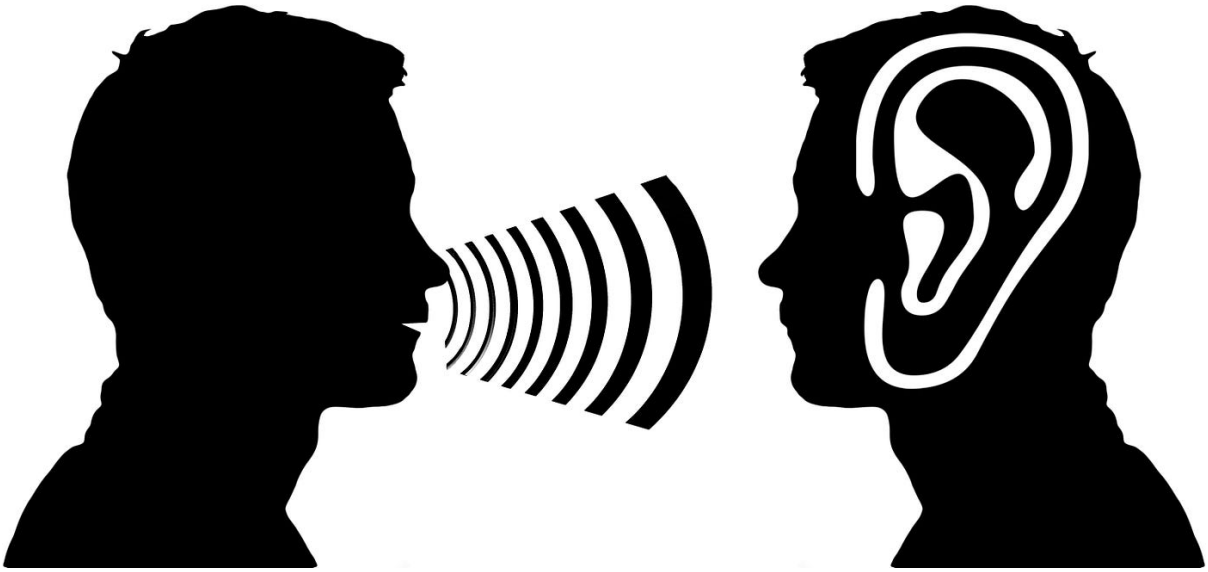
이벤트	설명
focus	요소에 포커스가 이동되었을 때 이벤트 발생
blur	요소에 포커스가 벗어났을 때 이벤트 발생
change	요소에 값이 변경 되었을 때 이벤트 발생
submit	submit 버튼을 눌렀을 때 이벤트 발생
reset	reset 버튼을 눌렀을 때 이벤트 발생
select	input이나 textarea 요소 안의 텍스트를 드래그하여 선택했을 때 이벤트 발생

4. 로드 및 기타 이벤트

이벤트	설명
load	페이지의 로딩이 완료되었을 때 이벤트 발생
abort	이미지의 로딩이 중단되었을 때 이벤트 발생
unload	페이지가 다른 곳으로 이동될 때 이벤트 발생

이벤트	설명
resize	요소에 사이즈가 변경되었을 때 이벤트 발생
scroll	스크롤바를 움직였을 때 이벤트 발생

event listener 👂🎙️



- 이벤트 리스너는 DOM 객체에서 이벤트가 발생할 경우 해당 이벤트 처리 핸들러를 추가할 수 있는 오브젝트이다.
- 이벤트 리스너를 이용하면 특정 DOM에 위에 말한 Javascirpt 이벤트가 발생할 때 특정 함수를 호출한다.
- 이벤트 리스너 등록하기
  - addEventListener

```
DOM객체. addEventListener(이벤트명, 실행할 함수명, 옵션)
```

  - **이벤트명** : Javascript에서 발생할 수 있는 이벤트 명을 입력한다.
  - **함수명** : 해당 변수는 생략 가능하며, 해당 이벤트가 발생할 때 실행할 함수 명을 입력한다.

- **옵션**: 옵션은 생략이 가능하며, 자식과 부모 요소에서 발생하는 버블링을 제어하기 위한 옵션이다.

```
addEventListener(type, listener);
addEventListener(type, listener, options);
addEventListener(type, listener, useCapture);
```

- `type`
  - 수신할 이벤트 유형을 나타내는 대소문자 구분 문자열입니다.
- `listener`
  - 지정한 이벤트(`Event` 인터페이스를 구현한 객체)를 수신할 객체입니다. `handleEvent()` 메서드를 포함하는 객체 또는 JavaScript 함수여야 합니다. 이벤트 수신기 콜백에서 콜백 자체에 대한 정보를 더 알아보세요.
- `options` *Optional*
  - 이벤트 수신기의 특징을 지정할 수 있는 객체입니다. 가능한 옵션은 다음과 같습니다.`capture`이벤트 대상의 DOM 트리 하위에 위치한 자손 `EventTarget`으로 이벤트가 전달되기 전에, 이 수신기가 먼저 발동돼야 함을 나타내는 불리언 값입니다. 명시하지 않을 경우 기본 값은 `false`입니다.`once`수신기가 최대 한 번만 동작해야 함을 나타내는 불리언 값입니다. `true`를 지정할 경우, 수신기가 발동한 후에 스스로를 대상에서 제거합니다. 명시하지 않을 경우 기본 값은 `false`입니다.`passivetrue`일 경우, 이 수신기 내에서 `preventDefault()`를 절대 호출하지 않을 것임을 나타내는 불리언 값입니다. 이 값이 `true`인데 수신기가 `preventDefault()`를 호출하는 경우, 사용자 에이전트는 콘솔에 경고를 출력하는 것 외에 아무런 동작도 하지 않습니다. 명시하지 않을 경우의 기본 값은 `false`지만, Safari와 Internet Explorer를 제외한 브라우저에서 `wheel` (en-US), `mousewheel` (en-US), `touchstart`, `touchmove` (en-US) 이벤트에서의 기본 값은 `true`입니다. 패시브 수신기로 스크롤 성능 향상에서 이 값에 대해 더 알아보세요.`signal` `AbortSignal`입니다. 지정한 `AbortSignal` 객체의 `abort()` 메서드를 호출하면 이 수신기가 제거됩니다. 명시하지 않을 경우 이벤트 수신기가 아무 `AbortSignal`에도 연결되지 않습니다.
- `useCapture` *Optional*
  - 이벤트 대상의 DOM 트리 하위에 위치한 자손 `EventTarget`으로 이벤트가 전달되기 전에, 이 수신기가 먼저 발동돼야 함을 나타내는 불리언 값입니다. 캡처 모드인 수신기는 DOM 트리의 위쪽으로 버블링 중인 이벤트에 의해선 발동하지 않습니다. 이벤트 버블링과 캡처링은 조상-자손 관계를 가진 두 개의 요소가 동일한 이벤트 유형에 대한 수신기를 가지고 있을 때, 두 요소에 이벤트가 전파되

는 방법을 말합니다. 이벤트 전파 모드에 따라 두 요소 중 이벤트를 먼저 수신하는 쪽이 달라집니다. DOM Level 3 Events와 JavaScript Event 순서에서 자세한 설명을 확인하세요. 기본 값은 `false`입니다.

- 예제

```
<html>
 <a>클릭</a>
</html>
<script>
 const a = document.querySelector('a');
  a.addEventListener('click', showConsole);
  function showConsole() {
    console.log("콘솔로그 실행");
  }
</script>
```

```
var t = document.getElementById('target');
if(t.addEventListener){
    t.addEventListener('click', function(event){
        alert('Hello world, '+event.target.value);
    });
} else if(t.attachEvent){
    t.attachEvent('onclick', function(event){
        alert('Hello world, '+event.target.value);
    })
}
```

```
<input type="button" id="target" value="button" />
<script>
    var t = document.getElementById('target');
    t.addEventListener('click', function(event){
        alert(1);
    });
    t.addEventListener('click', function(event){
        alert(2);
    });
</script>
```

```
<input type="button" id="target1" value="button1" />
<input type="button" id="target2" value="button2" />
<script>
    var t1 = document.getElementById('target1');
    var t2 = document.getElementById('target2');
    function btn_listener(event){
        switch(event.target.id){
            case 'target1':
                alert(1);
```

```
                break;
            case 'target2':
                alert(2);
                break;
        }
    }
    t1.addEventListener('click', btn_listener);
    t2.addEventListener('click', btn_listener);
</script>
```

- removeEventListener
  - 이벤트 리스너의 경우 웹 애플리케이션 메모리 누수의 원인이 될 수 있다.
  - 더 이상 해당 이벤트 리스너가 필요 없다고 하면 반드시 추가된 이벤트 리스너는 반드시 삭제해주어야 한다.
  - 특정 페이지에서만 사용하는 이벤트 리스너라면 해당 페이지를 떠날 때 이벤트 리스너를 삭제해준다.

```
DOM객체. removeEventListener(이벤트명, 실행했던 함수명);
```

  - 예제

```
//VUE Project EventListener 삭제 예시 코드
<script>
export default {
  ...
        mounted() {
            a.addEventListener('click', this.showConsole);
        },
        beforeDestroy() {
            a.removeEventListener('click', this.showConsole)
        },

  ...
 }
 </script>
```

    - 해당 예제의 경우 Vue 코드에서 eventListener를 이용했다.
    - 특정 vue페이지에서 addEventListener를 추가했다면 해당 페이지를 떠날 때
    - 반드시 beforeDestroy() 메서드에서 추가한 이벤트 리스너를 removeEventListener로 삭제했다.
    - Vue의 mounted는 해당 페이지가 렌더링 되었을 때 실행되고,

- beforeDestroy는 페이지가 떠나기 전 해당 코드가 실행된다.
- 즉, 더 이상 이벤트 리스너가 필요 없다면 꼭 제거해주어야 한다.

event handler 🖐🕹️🎮🖱️ 트

- 특정 요소에서 발생하는 이벤트를 처리하기 위해 사용하는 함수
- Event(특정 대상에 가하는 어떠한 행동)에 대한 다음 처리과정을 Event handler라고합니다.
  - 예를 들면, "버튼을 눌렀다"라는 Event가 발생하면, "이름을 알아내서 환영 메시지를 띄운다"
  - 여기서 "이름을 알아내서 환영 메시지를 띄운다"라는 일련의 다음 과정을 정의해 놓은게 Event handler 입니다

```
- Ex) 항아리를 깬다. 깰때는 망치로 깬다. 라고 할 때

      - 항아리                : 대상(객체)

      - 깬다                  : Event

      - 망치로 항아리를 깬다 : Event Hadler
```

항아리
(객체)

깬다
(Event)

망치로 항아리를 깬다
(Event Handler)

Element 추가

.createElement()

- 요소를 만듭니다.
- 예제

```
document.body.createElement( 'h1' )
```

.createTextNode()

- 선택한 요소에 텍스트를 추가

```
document.createTextNode( 'My Text' )
```

.appendChild()

- 선택한 요소 안에 자식 요소를 추가

```
var jbBtn = document.createElement( 'button' );
//button 요소를 만들고 jbBtn에 저장
var jbBtnText = document.createTextNode( 'Click' );
//Click이라는 텍스트를 만들고 jbBtnText에 저장
jbBtn.appendChild( jbBtnText );
//jbBtn에 jbBtnText를 삽입
document.body.appendChild( jbBtn );
//jbBtn을 body의 자식 요소로 삽입
```

Event Listener 등록

inline

- 인라인 방식은 이벤트를 HTML 요소의 속성으로 직접 지정하는 방식이다.
- HTML 코드에 자바스크립트를 추가함으로써 결국 HTML코드와 스크립트가 섞여 사용 되기 때문에 관점에 따라서는 유지보수에 안좋을 것이다.

```
<!DOCTYPE html>
<html>
<body>
    <button onclick="testHandler()">Test</button>
    <script>
    function testHandler() {
        alert('Hello world');
    }
    </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
    <button onclick="alert('Hello world');">Test</button>
</body>
</html>
```

property

- 자바스크립트 코드에서 프로퍼티로 등록하여 사용하는 방식이다.
- HTML 코드와 자바스크립트가 섞여 사용되지 않는다.
- 하나의 이벤트 핸들러 프로퍼티엔 하나의 이벤트 핸들러만 바인딩 가능하다.

```
<!DOCTYPE html>
<html>
<body>
  <button id="testBtn">Test</button>
</body>
<script>
    let testBtn = document.querySelector('#testBtn');
    testBtn.onclick = function () {
        alert('Hello world1');
    };

    // 두번째 바인딩된 이벤트 핸들러 (하나의 이벤트 핸들러만 바인딩 가능하기때문에 "Hello world2"가 노출
된다.
    testBtn.onclick = function () {
        alert('Hello world2');
```

```
    };
}
</script>
</html>
```

addEventListener(), attachEvent()

- 의 프로퍼티 방식에서는 1개의 이벤트 핸들러만 바인딩 가능했지만, 하나 이상의 이벤트 핸들러를 바인딩할 수 있다.
- 캡처링과 버블링을 지원한다.
- 문법 : 객체.addEventListener('이벤트타입', 함수명[, 이벤트전파방식]);
  - ＊이벤트타입(바인딩될 이벤트의 문자열)
  - * 함수명(이벤트리스너)
  - * 이벤트전파방식(캡쳐링 사용 여부)

⚠️ ※ 사용시 유의점

- IE 9 이상에서 동작 한다. (IE 8이하에는 attachEvent() 방식을 사용해야 한다.)

- 프로퍼티로 방식은 "on"이 붙은 이벤트 타입을 사용하지
만, addEventListener() 방식은 "on"이 붙지 않은 이벤트 타입을 사용한다.

```
<!DOCTYPE html>
<html>
<body>
    <button id="testBtn2">Test</button>
</body>
<script>
    let testBtn2 = document.querySelector('#testBtn2');
    function testFunc(){
        alert('Hello world1');
    }
    testBtn2.addEventListener('click', testFunc);

    testBtn2.addEventListener('click', function () {
        alert('Hello world2');
    });
}
</script>
</html>
```

preventDefault

- 어떤 이벤트를 명시적으로 처리하지 않은 경우, 해당 이벤트에 대한 브라우저의 기본 동작을 실행하지 않도록 지정.