

특별 부록

PDF 전자책

세상의 속도를
따라잡고 싶다면

Do
it!

자바스크립트 입문

고경희 지음

실전
프로젝트

- 01 · 숫자 맞추기
- 02 · 이미지 슬라이드 쇼 만들기
- 03 · 온도 변환기 만들기
- 04 · 라이트 박스 만들기
- 05 · 로컬 스토리지 만들기
- 06 · 타이머 만들기



QR 코드를
열어 보세요!

PDF 내려받기

이지스퍼블리싱

[실전 1] 숫자 맞추기 게임

숫자 맞추기 게임은 컴퓨터가 만든 숫자를 사용자가 범위를 좁혀가면서 맞추는 게임입니다. 아마 친구들과 모여서 한 사람이 제시한 숫자를 나머지 친구들이 차례로 돌아가며 맞추는 (일명 up & down 게임) 게임을 해 본 적도 있을 것입니다.

프로그램 흐름 미리 그려보기

1. 빈 노트를 꺼내놓고 게임이 어떤 식으로 진행될지를 단계별로 미리 그려보겠습니다. 우선 컴퓨터에서 숫자를 정합니다. 이 숫자는 사용자가 알 수 없지만 여기에서는 '23'이라는 숫자를 예로 들어보겠습니다.

1	컴퓨터가 숫자 정함	예) 23
---	------------	-------

2. 이번에는 사용자가 숫자를 제시합니다. 숫자 범위는 정해주는 게 좋겠죠? 여기에서는 1 에서 100 사이의 숫자만 사용하겠습니다. 예를 들어, '55'라는 숫자를 제시해 보겠습니다.

1	컴퓨터가 숫자 정함	예) 23
2	사용자가 숫자 정함	예) 55

3. 컴퓨터가 정한 숫자와 사용자가 제시한 숫자를 비교해 더 큰지, 더 작은지, 혹은 맞혔는지 알려줍니다. 사용자가 제시한 숫자보다 컴퓨터가 정한 숫자가 더 크다면 사용자에게 좀 더 큰 숫자를 제시하라고 'UP'이라고 알려주고, 컴퓨터가 정한 숫자가 더 작다면 좀더 작은 숫자를 제시하라고 'DOWN'이라고 알려주겠습니다. 맞혔다면 '맞혔습니다'라고 알려주고 게임이 끝납니다. 여기에서는 사용자가 제시한 숫자 '55'를 입력하면 'DOWN'이라고 표시되겠네요.

1	컴퓨터가 숫자 정함		23
2	사용자가 숫자 제시		55
3	컴퓨터 숫자와 사용자 숫자 비교		'55' 입력 결과 : DOWN
	3-1	'컴퓨터 숫자 > 사용자 숫자'라면 'UP' 표시	
	3-2	'컴퓨터 숫자 < 사용자 숫자'라면 'DOWN' 표시	

	3-3	‘컴퓨터 숫자 = 사용자 숫자’라면 ‘맞힘’ 표시. 게임 끝.	
--	-----	------------------------------------	--

4. 숫자를 맞추지 못했다면 사용자가 다시 다른 숫자를 제시합니다. 우리는 2 번 단계에서 ‘55’라는 숫자를 제시했었는데, 컴퓨터가 ‘DOWN’이라고 했으므로 아까 입력했던 값보다 작은 값을 제시합니다. 여기에서는 ‘10’을 제시해 보겠습니다.

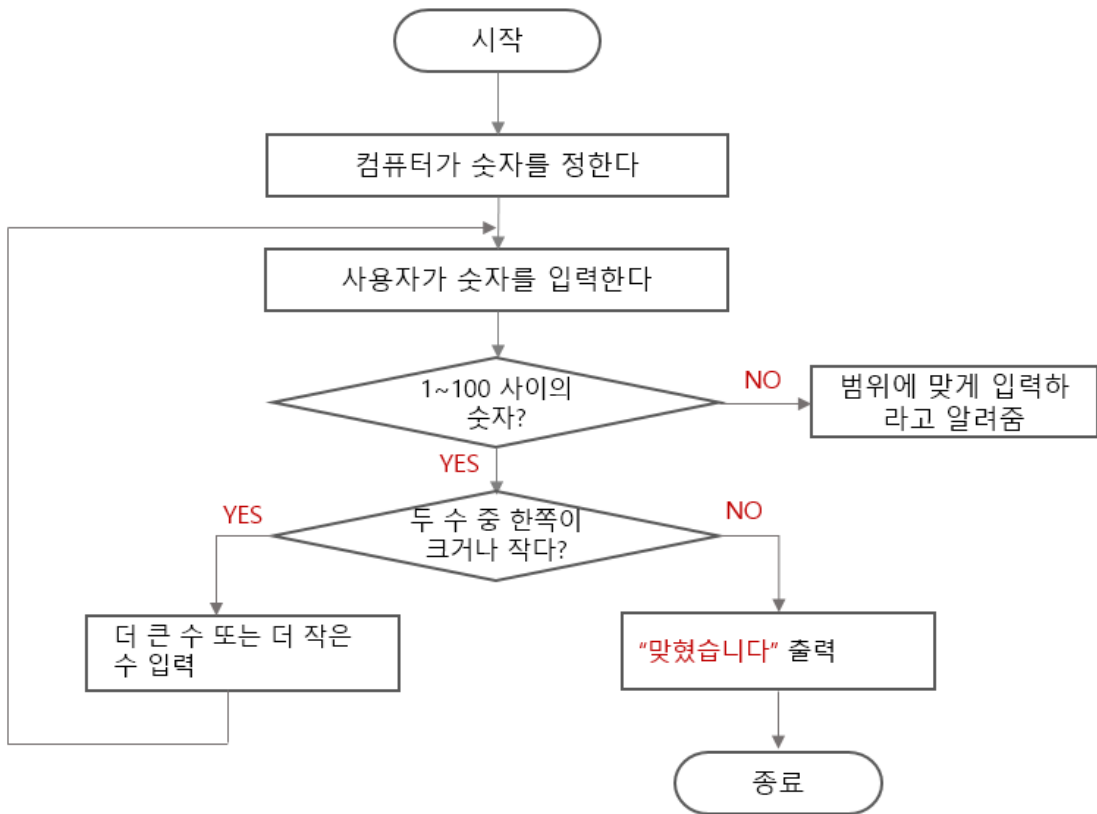
1	컴퓨터가 숫자 정함		23
2	사용자가 숫자 제시		55
3	컴퓨터 숫자와 사용자 숫자 비교		‘55’ 입력 결과 : DOWN
	3-1	‘컴퓨터 숫자 > 사용자 숫자’라면 ‘UP’ 표시	
	3-2	‘컴퓨터 숫자 < 사용자 숫자’라면 ‘DOWN’ 표시	
	3-3	‘컴퓨터 숫자 = 사용자 숫자’라면 ‘맞힘’ 표시. 게임 끝.	
4	사용자가 숫자 제시		10

5. 다시 한번 컴퓨터 숫자와 사용자 숫자를 비교해서 ‘UP’인지, ‘DOWN’인지, 아니면 맞혔는지 확인합니다. 여기에서는 컴퓨터가 정한 숫자 ‘23’보다 사용자 숫자가 작으므로 ‘UP’이 표시되겠네요.

1	컴퓨터가 숫자 정함		23
2	사용자가 숫자 제시		55
3	컴퓨터 숫자와 사용자 숫자 비교		DOWN
	3-1	‘컴퓨터 숫자 > 사용자 숫자’라면 ‘UP’ 표시	
	3-2	‘컴퓨터 숫자 < 사용자 숫자’라면 ‘DOWN’ 표시	
	3-3	‘컴퓨터 숫자 = 사용자 숫자’라면 ‘맞힘’ 표시. 게임 끝.	
4	사용자가 숫자 제시		10
5	컴퓨터 숫자와 사용자 숫자 비교		‘10’ 입력 결과 : UP
	5-1	‘컴퓨터 숫자 > 사용자 숫자’라면 ‘UP’ 표시	

5-2	'컴퓨터 숫자 < 사용자 숫자'라면 'DOWN' 표시	
5-3	'컴퓨터 숫자 = 사용자 숫자'라면 '맞힘' 표시. 게임 끝.	

6. 여기까지 하면 2~3 단계와 4~5 단계가 반복되는 걸 알 수 있습니다. 사용자가 컴퓨터 숫자를 맞추기 전까지는 계속 반복할 것입니다. 이것을 순서도로 그려보면 다음과 같이 그릴 수 있습니다. 이 순서도를 참고해 프로그램을 만들어 보세요.



Up & Down 게임 순서도

[해설]

1. 무작위 수 만들기

1. 비주얼 스튜디오 코드에서 upAndDown#upAndDown.html 문서를 불러와서 간단히 소스를 분석해 보겠습니다. 사용자가 숫자를 입력한 후 #check 버튼을 클릭하면 숫자를 비교하고 계산한 후 #display 영역에 결과를 표시합니다. #reset 버튼을 클릭하면 웹 문서를 다시 불러와 처음부터 다시 시작하게 되죠.

```
<div id="wrapper">
  <h1>숫자 맞추기 게임</h1>
  <p>1에서 100 사이의 수를 입력하세요.</p>
  <form action="">
    <label><input type="text" id="try" autocomplete="off" autofocus></label>
    <input type="button" value="확인" id="check" class="btn btn-1">
    <input type="button" value="다시" id="reset" class="btn btn-2">
  </form>
  <div id="display" class="output"></div>
  <div id="counter" class="footer"></div>
</div>
```



2. #check 버튼을 클릭했을 때 숫자를 비교하고 결과를 표시해야 합니다. 우선 #check 부분에 finding() 함수를 연결하겠습니다. 그리고, [취소] 버튼을 클릭했을 때는 현재 페이지를 다시 로딩 하도록 할 것입니다. 다음과 같이 두 개의 버튼에 함수를 연결합니다.

▶ window.location.reload() 명령은 현재 문서를 다시 불러오는 함수로 자바스크립트 안에 미리 만들어져 있습니다.

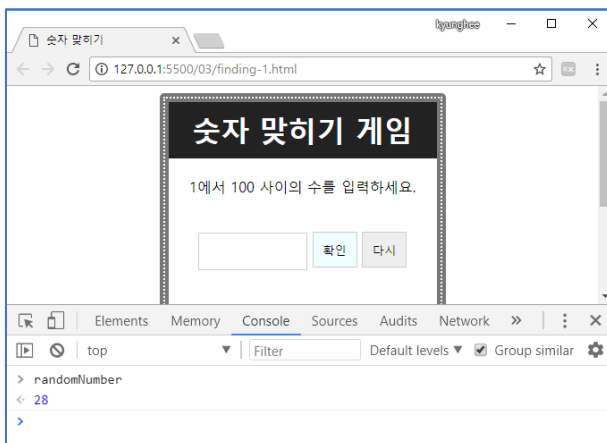
```
<label><input type="text" id="try" autocomplete="off" autofocus></label>
<input type="button" value="확인" id="check" class="btn btn-1" onclick="finding( )">
<input type="button" value="다시" id="reset" class="btn btn-2"
onclick="window.location.reload()">
```

3. js 폴더에 upanddown.js 파일을 만들고 upAndDown.html 문서에 연결합니다. 이제부터 upanddown.js 파일에 소스를 작성해 보겠습니다. 우선 몇 번 만에 숫자를 맞는지 확인하기 위해 count 라는 변수를 선언하고 초깃값을 0으로 지정합니다. 그리고, 무작위 수를 만들기 위해 자바스크립트 안에 이미 만들어져 있는 Math.random() 이라는 함수를 사용하는데, Math.random() 함수 결과값은 0과 1 사이의 숫자이기 때문에 1에서 100 사이의 무작위 수를 구하기 위해 결과값에 100을 곱하고 정수로 바꾼 후 1을 더해 줍니다.

```
var count = 0;

var randomNumber = Math.floor(Math.random()*100) + 1;
```

4. 문서를 저장한 후 웹 브라우저에서 확인해 볼까요? [Ctrl + Shift + J]를 눌러 콘솔 창을 엽니다. 그리고 randomNumber라고 입력해 보세요. 1과 100 사이의 무작위 수가 표시됩니다.



2. 사용자가 입력한 숫자 가져오기

컴퓨터에서 무작위 수를 만들었으니 이제 사용자가 숫자를 입력할 차례군요. 사용자가 텍스트 필드에 값을 입력하고 [확인] 버튼을 클릭하면 텍스트 필드의 값을 가져옵니다. 그리고, 이 프로그램은 1에서 100 사이의 숫자만 사용할 것이기 때문에 사용자가 입력한 값이 1과 100 사이의 숫자인지도 확인해 보겠습니다.

1. 계속해서 upanddown.js 파일에 작성합니다. #try 필드에 사용자가 입력한 값을 가져와 userNumber라는 변수에 저장해 보겠습니다. 텍스트 필드의 값을 가져올 때는 텍스트 필드 요소(#try)의 value 속성을 이용합니다. 사용자 입력 값은 [확인] 버튼을 클릭한 후에야 가져올 수 있으므로 finding() 함수 안에 선언합니다.

```
var count = 0;

var randomNumber = Math.floor(Math.random() * 100) + 1;

function finding() {
  var userNumber = document.querySelector('#try').value;
}
```

2. 사용자가 1과 100 사이의 숫자를 입력했는지 확인하려면 if 문과 else 문을 사용합니다. 조건에 맞는다면 if 다음의 명령을 실행하고 조건에 맞지 않는다면 else 다음의 명령을 실행합니다. 제대로 동작하는지 확인하기 위해 if 문에서 console.log를 사용해 사용자 입력 값을 표시해 보겠습니다.

```

var count = 0;

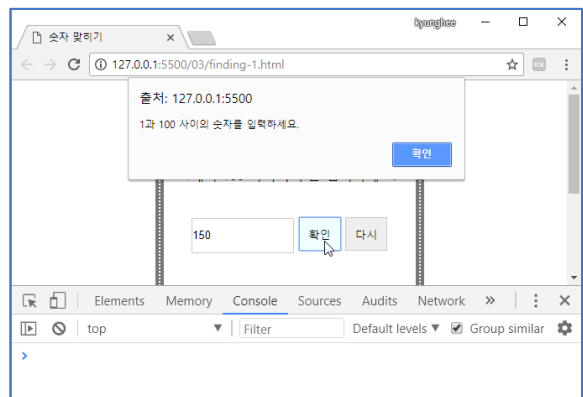
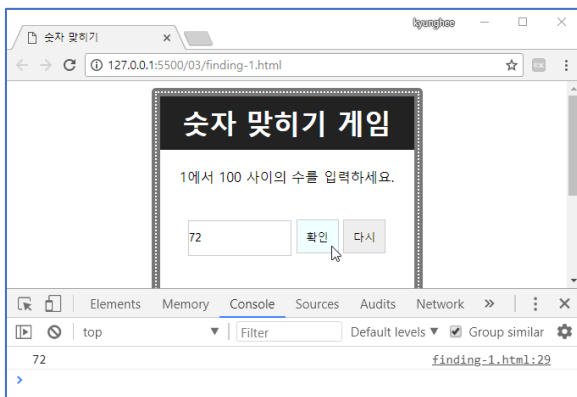
var randomNumber = Math.floor(Math.random()*100) + 1;

function finding() {
  var userNumber = document.querySelector('#try').value;

  if (userNumber >=1 && userNumber <= 100) {
    console.log(userNumber);
  }
  else {
    alert("1과 100 사이의 숫자를 입력하세요.");
  }
}

```

3. 문서를 저장한 후 웹 브라우저로 확인해 보겠습니다. 텍스트 필드에 1과 100 사이의 아무 숫자나 입력한 후 [확인] 버튼을 클릭하면 콘솔 창에 방금 입력한 숫자가 표시되어야 합니다. 100 이상의 숫자를 입력하면 알림 창이 떠야 하고요. 혹시 제대로 동작하지 않는다면 지금까지 입력한 소스 중에 오타는 없는지 확인하세요.



3. 사용자 숫자와 컴퓨터 숫자 비교하기

사용자가 입력한 1과 100 사이의 숫자를 가져왔다면 이제 컴퓨터 숫자와 비교해 봐야겠죠? 컴퓨터 숫자보다 사용자 숫자가 작다면 'UP'이라고 표시하고, 컴퓨터 숫자보다 사용자 숫자가 크다면 더 작은 수를 입력하라고 'DOWN'이라고 표시할 것입니다. 컴퓨터 숫자와 사용자 숫자가 같다면 '맞았습니다'라고

표시할 것입니다.

1. 앞에서 if 문 안에 있는 `console.log()` 문을 삭제합니다. 그리고 그 자리에 다음과 같은 소스를 추가합니다. 조건을 체크할 것이므로 if...else 문을 사용해야겠죠? 그런데 조건이 하나가 아니라 여러 개입니다. 이럴 경우에는 if...else 문을 중첩해서 사용합니다. 그리고 1과 100 사이의 숫자를 가져와서 처리했다면 count 값도 1 증가시켜 줍니다.

```
function finding() {
  var userNumber = document.querySelector('#try').value;

  if (userNumber >=1 && userNumber <= 100) {
    if (randomNumber > userNumber) { // 컴퓨터 숫자가 클 경우 - 'UP' 표시
    }
    else if (randomNumber < userNumber) { // 컴퓨터 숫자가 작을 경우 - 'DOWN' 표시
    }
    else { // 컴퓨터 숫자를 맞혔을 경우

    }
    count++;
  }
  else {
    alert("1과 100 사이의 숫자를 입력하세요.");
  }
}
```

2. 이제 조건에 따라 그 결과를 화면에 표시해 보겠습니다. HTML 소스를 보면 #display인 요소에 결과를 표시해야겠군요. 앞에서 입력한 if 문과 else 문에 다음과 같은 소스를 추가합니다.

[팁] 화면에 표시할 내용이 텍스트뿐이라면 innerHTML 대신 innerText 메서드를 사용해도 됩니다.

```

function finding() {
    var userNumber = document.querySelector('#try').value;

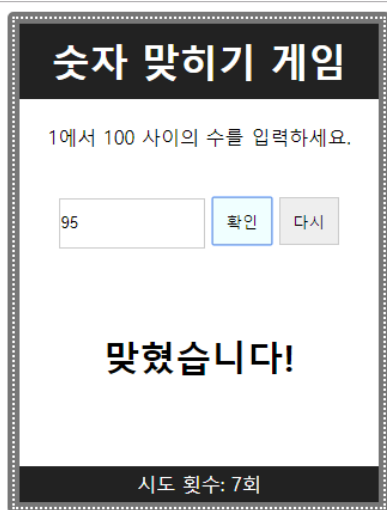
    if (userNumber >=1 && userNumber <= 100) {
        if (randomNumber > userNumber) {
            document.querySelector('#display').innerHTML = "UP!"; // #display 영역에 'UP!' 표시
        }
        else if (randomNumber < userNumber) {
            document.querySelector('#display').innerHTML = "DOWN!"; // #display 영역에 'DOWN!' 표시
        }
        else {
            document.querySelector('#display').innerHTML = "<span style='color:red'>맞혔습니
다!</span>"; // #display 영역에 빨간색으로 '맞혔습니다!' 표시
        }
        count++;

        document.querySelector('#counter').innerHTML = "시도 횟수: " + count + "회"; //
#counter 영역에 시도 횟수(count) 표시

    }
    else {
        alert("1과 100 사이의 숫자를 입력하세요.");
    }
}

```

3. 문서를 저장한 후 웹 브라우저에서 확인해 보세요. 숫자를 맞힐 때까지 다른 숫자를 입력하면서 게임을 진행할 수 있습니다. 검정색으로 표시된 영역에는 시도 횟수가 표시됩니다. 그런데 텍스트 필드에 숫자를 입력하고 [확인]을 클릭해도 입력했던 숫자가 지워지지 않기 때문에 새로운 숫자를 입력하려면 기존에 입력했던 숫자를 직접 삭제한 후 입력해야 합니다. 너무 번거롭지요? 소스를 좀 수정해야겠네요.



숫자 맞추기 게임

1에서 100 사이의 수를 입력하세요.

95

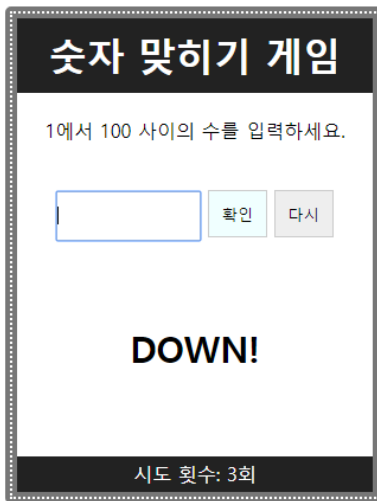
맞혔습니다!

시도 횟수: 7회

4. 사용자 숫자를 가져와 크기를 비교한 후에 그 결과를 화면에 표시했다면 사용자가 다시 숫자를 입력할 수 있도록 텍스트 필드를 지워야 합니다. else 문 다음에 다음과 같은 소스를 추가합니다. 소스 추가가 끝나면 [Ctrl+S]를 눌러 소스를 저장합니다.

```
else {
    document.querySelector('#display').innerHTML = "<span style='color:red'>맞혔습니
다!</span>";
}
document.querySelector('#try').value = "";
count++;
document.querySelector('#counter').innerHTML = "시도 횟수: " + count + "회";
```

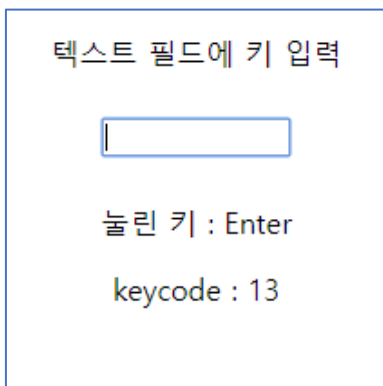
5. 웹 브라우저에서 다시 한번 확인해 볼까요? 사용자가 입력한 숫자와 컴퓨터 숫자를 비교한 결과가 화면에 표시되면서 텍스트 필드는 다음 숫자를 입력할 수 있게 비워질 것입니다. 숫자 맞추기 게임의 기본 기능이 완성되었습니다.



4. [Enter] 키 눌러서 함수 실행하기

지금까지는 [확인] 버튼을 눌러야 `finding()` 함수를 실행할 수 있었습니다. 키보드에서 값을 입력하고 마우스로 손을 옮겨 [확인] 버튼을 눌러야 하기 때문에 약간 번거롭습니다. 지금처럼 [확인] 버튼을 눌러 실행할 수도 있지만, 마우스까지 손을 뻗지 않고 키보드의 [Enter] 키를 눌러 함수를 실행한다면 좀 더 편리할 것입니다. 자바스크립트에서 [Enter] 키를 알아내는 방법을 알아보겠습니다.

1. 웹 브라우저에서 `upAndDown₩keycode.html` 문서를 열고 텍스트 필드에 키보드의 아무 키나 눌러보세요. 어떤 키가 눌렸는지, 그리고 키코드 값은 얼마인지 표시됩니다. 예를 들어, [Enter] 키를 누르면 키코드가 '13'이라고 표시됩니다.



2. 웹 문서에서 마우스나 키보드를 조작하거나 웹 문서를 불러오는 등 웹 문서에서 발생하는 모든 동작을 '이벤트(event)'라고 하는데, 키보드의 키를 눌렀을 때는 `keypress`라는 이벤트가 발생합니다. #try 요

소에 keypress 이벤트가 발생했을 때(즉, 텍스트 필드에서 어떤 키를 눌렀을 때) 함수를 실행하려면 다음과 같이 추가합니다. 이 소스는 finding() 함수를 선언하기 전에 추가합니다.

```
var count = 0;

var randomNumber = Math.floor(Math.random()*100) + 1;

document.querySelector('#try').onkeypress = function(e) {

}
```

3. 눌린 키의 키코드 값이 '13'인지 확인하고 함수를 실행해야겠죠? 키코드 값은 event.keycode 속성을 가져오면 되는데, 여기에서는 이벤트 e로 표기했으므로 e.keycode 값을 가져와 확인하면 되겠네요. 그리고 파이어폭스의 경우 keypress 이벤트에서 e.keycode 대신 e.which 속성을 사용해야 키코드 값을 알 수 있기 때문에 두 가지 속성을 함께 체크합니다. 다음과 같은 소스를 추가하세요. [Ctrl+S]를 눌러 문서를 저장합니다.

```
var count = 0;

var randomNumber = Math.floor(Math.random()*100) + 1;

document.querySelector('#try').onkeypress = function(e) {
  if (e.keycode == 13 || e.which == 3) { // 눌린 키가 [Enter]키라면
    finding(); // finding() 함수 실행
    return false; // keypress 이벤트가 발생했을 때 브라우저가 기본으로 하는 동작 취소
  }
}
```

4. 웹 브라우저에서 다시 확인해 보겠습니다. 텍스트 필드에 숫자를 입력하고 [Enter] 키를 눌러보세요. [확인]을 누르지 않더라도 입력한 숫자를 체크해서 'UP'인지 'DOWN'인지 표시됩니다.

숫자 맞추기 게임

1에서 100 사이의 수를 입력하세요.

확인

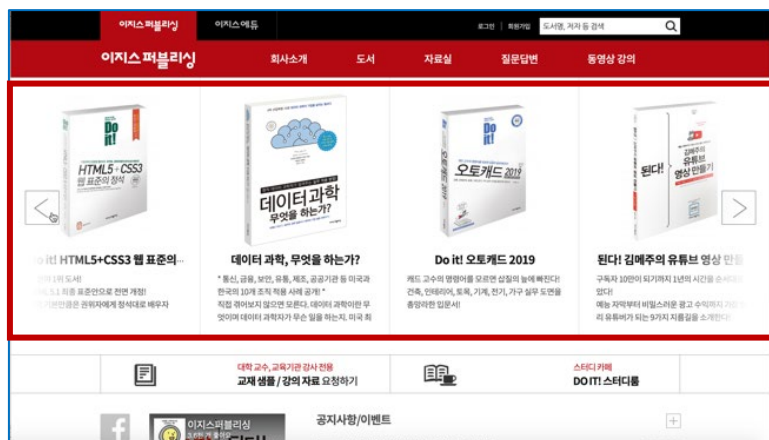
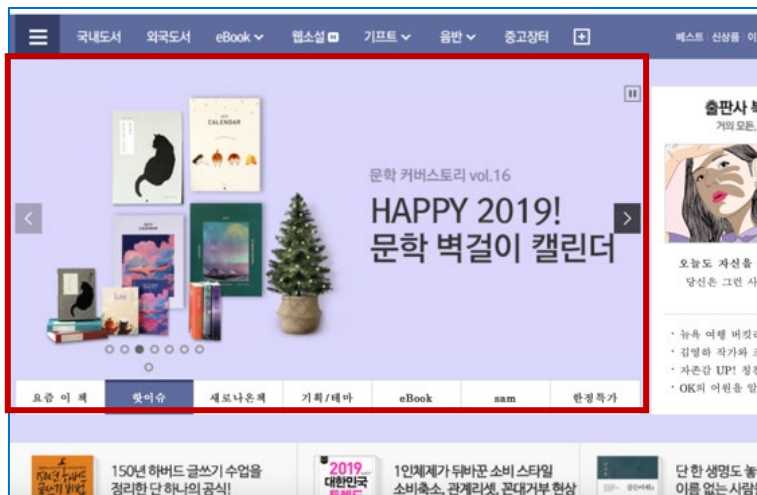
다시

DOWN!

시도 횟수: 1회

[실전 2] 이미지 슬라이드 쇼

이번에는 '이미지 슬라이드 쇼'를 만들어 보면서 DOM을 어떻게 활용할 수 있는지 연습해 보겠습니다. 이미지 슬라이드 쇼는 쇼핑몰 웹 사이트에서 주력 상품 여러 개를 번갈아 가면서 보여주거나 웹 사이트의 공지 사항이나 주요 내용을 하나씩 바꿔가면서 보여줄 때 많이 사용합니다. 클릭해서 다음 슬라이드로 이동할 수 있는 버튼이 좌우에 있고, 슬라이드 아래쪽이나 옆에 슬라이드 간에 이동할 수 있는 내비게이션이 제공되기도 합니다.



프로그램 흐름 미리 그려보기

웹 사이트에서 사용하는 이미지 슬라이드 쇼는 아주 다양합니다. 하지만 기본적인 방식은 여러 개의 이미지 중 한 이미지만 화면에 표시하고 좌우 화살표(또는 같은 기능을 하는 다른 표시)를

클릭해서 이전 이미지나 다음 이미지를 볼 수 있게 하는 것이죠.

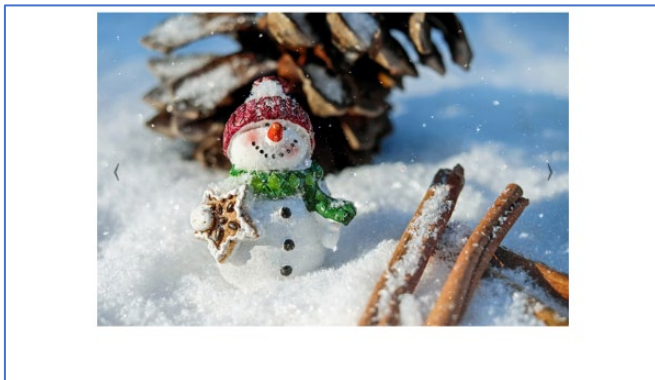
1	슬라이드 쇼에 사용할 이미지들을 배열에 저장하고, 모두 감춥니다.	
2	기본적으로 첫 번째 이미지만 화면에 표시합니다.	
3	[이전] 버튼을 클릭할 경우	
	3-1	'컴퓨터 숫자 > 사용자 숫자'라면 'UP' 표시
	3-2	'컴퓨터 숫자 < 사용자 숫자'라면 'DOWN' 표시
	[다음] 버튼을 클릭할 경우	
	3-3	현재 이미지가 마지막 이미지라면 첫 번째 이미지를 표시합니다.
	3-4	현재 이미지가 마지막 이미지가 아니라면 배열에서 다음 이미지를 가져와 표시합니다.

위 설계 순서를 참고해 프로그램을 직접 만들어 보세요.

[해설]

1. 이미지만 화면에 표시하기

이미지 슬라이드 쇼 기능은 한 번에 하나의 이미지만 표시하고 있다가 [이전] 버튼이나 [다음] 버튼을 클릭하면 다른 이미지를 화면에 표시합니다. 이 예제에서는 CSS의 display 속성을 사용해 사용자 동작(이벤트)에 따라 특정 이미지를 표시하고 다른 이미지는 감추는 방법에 대해 알아보겠습니다.



좌우 클릭해서 이전/다음 슬라이드 이동하기

1. 웹 브라우저에서 `slideshow\slideshow.html` 문서를 불러오면 이미지 슬라이드 쇼에 사용할 이미지 4개가 화면에 표시되어 있습니다. 슬라이드 쇼에 사용할 이미지는 이전 이미지나 다음 이미지에 쉽게 접근할 수 있어야 하므로 여기에서는 배열 형식으로 저장해 보겠습니다. 브라우저 창은 그대로 열어 두세요.



2. 비주얼 스튜디오 코드에서 새 문서를 만든 후 `slideshow\js` 폴더에 `slideshow.js` 파일로 저장합니다. 그리고 `slideshow\slideshow.html` 문서에서 `slideshow.js` 파일을 연결하세요.

3. 이제부터 `slideshow.js` 파일에 자바스크립트 소스를 작성해 보겠습니다. 우선, 이미지 4개를 배열 형식으로 가져오기 위해 그래서 `querySelectorAll` 메서드에서 `img` 태그로 접근하기로 하겠습니다. 웹 문서에서 `#container` 요소의 자식 요소인 `img` 요소들을 모두 가져와 `slides` 배열로 저장합니다. 그리고 슬라이드 쇼의 왼쪽과 오른쪽에 표시될 '이전' 버튼과 '다음' 버튼도 가져와서 변수에 저장합니다.

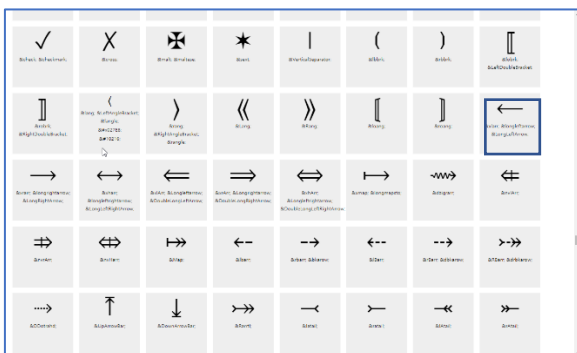
```
var slides = document.querySelectorAll("#container > img"); // 이미지들을 저장한 배열
var prev = document.querySelector("#prev"); // 이전 버튼
var next = document.querySelector("#next"); // 다음 버튼
```

[이전] 버튼이나 [다음] 버튼에 버튼 이미지를 사용할 수도 있지만, 반응형 웹에서는 이미지 크기 조절도 신경 써야 하고 서버에서 이미지를 가져와야 한다는 단점이 있습니다. 그래서 '이전'이나 '다음'이라는 의미를 알 수 있도록 화살표를 많이 사용하는데 키보드 상의 '<' 나 '>'를 사용할 수도 있고 특수 기호 중에서 화살표를 골라 사용할 수도 있습니다.

예제 소스의 `#prev` 요소와 `#next` 요소를 보면 `⟨`과 `⟩` 라는 낯선 코드를 사용하고 있는데, 이것은 웹 문서에서 특수 문자나 기호를 삽입할 때 사용하는 '엔티티 이름(Entity name)'입니다.

W3C(www.w3c.org)에서 제공하는 엔티티 코드 표는 <https://dev.w3.org/html5/html-author/charref>에서 확인할 수 있습니다. 웹 문서에서 사용할 수 있는 특수 문자나 특수 기호에는 간단하게 엔티티 이름이 표시되어 있습니다.

예를 들어, 왼쪽 화살표를 나타내는 특수 기호에는 '`⟨` `⟨` `&lange;`' 이라고 표시되어 있는데, 기호 위로 마우스 포인터를 올리면 '`⟨` 과 `➎`' 같은 엔티티 코드가 함께 표시됩니다. 세 개의 엔티티 이름과 두 개의 엔티티 코드 중 어떤 것을 사용해도 됩니다. 엔티티 이름이나 엔티티 코드 뒤에 오는 세미콜론(`;`)은 꼭 붙여주어야 합니다.



4. 이미지 슬라이드 쇼에서는 문서를 처음 불러올 때 첫 번째 이미지를 표시하고 나머지 이미지는 모두 감춥니다. 그리고 [이전] 버튼이나 [다음] 버튼을 클릭하면 새로 가져온 이미지를 화면에 표시하고 나머지 이미지는 모두 감추죠.

앞에서 입력했던 소스 다음에 아래와 같은 소스를 추가합니다. slides 배열에 있는 여러 이미지 중 어떤 이미지를 화면에 표시할 것인지 프로그램에 알려주어야 하므로 슬라이드 인덱스 값을 저장하는 변수 current를 선언한 것입니다. 초깃값은 0으로 지정해서 첫 번째 이미지를 화면에 표시하도록 합니다.

```
var current = 0;
```

5. n번째 슬라이드를 화면에 표시하는 showSlide(n) 함수를 만들어 보겠습니다. n=1이면 두 번째 이미지를, n=2이면 세 번째 이미지를 표시할 것입니다. 앞에서 입력했던 소스 다음에 다음 소스를 추가합니다.

```
showSlide(current); // showSlide( ) 함수를 실행해 current 위치의 이미지 표시

function showSlide(n) { // 모든 이미지를 감춘 후 인덱스가 n인 이미지만 화면에 표시
    for(var i=0; i<slides.length; i++) { // slides 배열의 처음부터 끝까지 반복
        slides[i].style.display = "none"; // 모든 이미지를 화면에서 감춤
    }
    slides[n].style.display = "block"; // n번째 이미지를 화면에 표시
}
```

6. 수정한 소스를 저장한 후 웹 브라우저에서 [F5]를 수정한 내용을 적용합니다. 첫 번째 이미지만 표시되고 나머지 이미지들은 화면에 보이지 않을 것입니다. 이제 [이전] 버튼과 [다음] 버튼을 클릭했을 때 동작하게 만들면 되겠습니다.



7. [이전] 버튼을 클릭했을 때 어떤 동작을 하게 될까요? 현재 이미지가 첫 번째 이미지인지 체크해서 첫 번째 이미지라면 마지막 이미지를 표시하고, 첫 번째 이미지가 아니라면 현재 위치에서 1을 뺀 위치의 이미지를 표시하면 됩니다. 다음과 같은 소스를 추가합니다.

```
function prevSlide() {
    if (current > 0) current -= 1; // 현재 이미지가 첫 번째 이미지가 아니라면 이전 위치로
    else current = slides.length - 1; // 현재 이미지가 첫 번째라면 마지막으로
    showSlide(current); // 이동한 위치의 이미지 표시
}
```

8. [다음] 버튼을 클릭했을 때 실행할 함수도 비슷하게 만들 수 있습니다. 현재 이미지가 마지막 이미지라면 첫 번째 이미지를 표시하고, 마지막 이미지가 아니라면 현재 위치에서 1을 더한 위치의 이미지를 표시합니다. 다음과 같은 소스를 `</script>` 태그 앞에 추가합니다.

```
function nextSlide() {
    if (current < slides.length - 1) current += 1; // 현재 이미지가 마지막 이미지가 아니라면 다음
    위치로
    else current = 0; // 현재 이미지가 마지막이라면 첫 번째 위치로
    showSlide(current); // 이동한 위치의 이미지 표시
}
```

9. 마지막으로 `prev` 요소와 `next` 요소를 클릭했을 때 방금 정의한 함수를 실행하도록 해야겠죠? `showSlide()` 함수를 실행하는 소스 다음에 다음과 같은 소스를 추가하세요.

```
showSlide(current);
prev.onclick = prevSlide; // prev를 클릭하면 prevSlide 함수 실행
next.onclick = nextSlide; // next를 클릭하면 nextSlide 함수 실행
```

10. 수정한 소스를 저장한 후 웹 브라우저에서 [F5]를 누른 후 결과를 확인해 보세요. 이미지의 왼쪽이나 오른쪽에 있는 화살표 버튼을 클릭하면 이전 이미지나 다음 이미지가 표시될 것입니다.



[실전 3] 온도 변환기

단위 변환기는 '인치 ↔ 센티미터'나 '섭씨 ↔ 화씨', 'm² ↔ 평'처럼 실생활에서 필요한 여러 단위를 바꿔주는 프로그램입니다. 주요 포털 사이트에서도 자체 단위 변환기를 제공하고 있는데, 이는 웹 문서 안에 자바스크립트 소스를 추가하여 작성한 것입니다. 여기에서는 '섭씨'에서 '화씨'로 또는 '화씨'에서 '섭씨'로 온도를 변환해 주는 프로그램을 작성해 보겠습니다.

섭씨 ↔ 화씨 온도 변환기

100 °C

↔

212 °F

섭씨 ↔ 화씨 온도 변환기

212 °F

↔

100 °C

프로그램 흐름 미리 그려보기

기본적으로 '섭씨'에서 '화씨'로 변환하지만, 중간에 있는 화살표(↔)를 클릭하면 '화씨'에서 '섭씨'로 변환할 수 있는 프로그램을 만들어 보겠습니다.

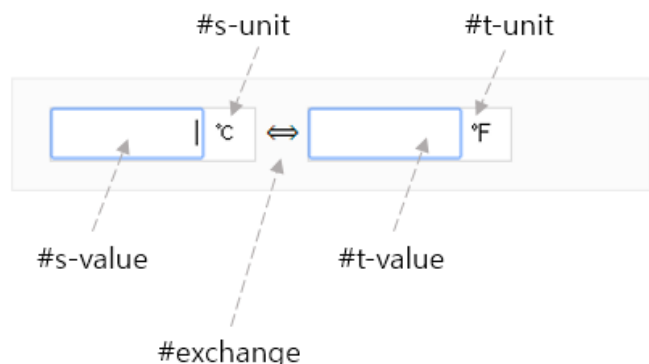
1	'섭씨 → 화씨'인지, '화씨 → 섭씨'인지 저장하는 변수를 만들고, '섭씨 → 화씨'로 변환하는 경우 true로 지정하고, '화씨 → 섭씨'로 변환하는 경우 false로 지정합니다.	
2	현재 상태가 '섭씨 → 화씨'라면	
	A-1	왼쪽 텍스트 필드에는 °C를, 오른쪽 텍스트 필드에는 °F를 붙입니다
	A-2	왼쪽 텍스트 필드 값(섭씨온도)을 가져와 화씨온도로 계산한 후
	A-3	오른쪽 텍스트 필드에 표시합니다.
3	현재 상태가 '화씨 → 섭씨'라면	
	B-1	왼쪽 텍스트 필드에는 °F를, 오른쪽 텍스트 필드에는 °C를 붙입니다.
	B-2	현재 이미지가 마지막 이미지가 아니라면 배열에서 다음 이미지를 가져와 표시합니다.
	B-3	오른쪽 텍스트 필드에 표시합니다.
4	'화살표'를 클릭하면 현재 상태를 바꿉니다.	

화면 요소 살펴보기

여기에서 사용할 converter\converter.html 문서는 미리 HTML 태그와 CSS 소스가 작성되어 있기 때문에 화면의 각 요소가 어떻게 구성되어 있는지 먼저 확인해야 합니다. 직접 HTML 태그와 CSS 소스까지 작성해 보겠다면 각 요소의 id 값을 어떻게 지정할지 결정하고 기억해 두어야 합니다. converter.html 문서의 구성은 다음과 같습니다.

각 요소에 붙인 id 이름에서 s-는 'source'를 의미하고, t- 는 'target'을 의미합니다. 변환할 때 왼쪽에는 주어지는 값(source)을, 오른쪽에는 계산한 값(target)을 표시한다고 기억해 두세요.

위 설계 순서와 화면 요소 그림을 참고해 프로그램을 직접 만들어 보세요.



[해설]

1. 상태에 따라 단위 바꾸기

여기에서 만들어 볼 온도 변환기는 기본적으로 '섭씨'에서 '화씨'로 변환합니다. 그래서 왼쪽 텍스트 필드에는 °C를 붙이고, 오른쪽 텍스트 필드에는 °F를 붙입니다. 우선 가운데 화살표(#exchange)를 클릭하면 두 개 필드의 단위를 바꿔주도록 소스를 작성해 보겠습니다.



1. 이제부터 자바스크립트 소스를 작성해 보겠습니다. converter.html 문서에는 왼쪽과 오른쪽, 두 개의 텍스트 필드가 있고 각 텍스트 필드 옆에는 단위 부분이 있었지요? 왼쪽 필드의 단위 부분과 오른쪽 필드의 단위 부분을 가져와서 각각 sUnit과 tUnit으로 저장합니다. 그리고 '섭씨'에서 '화씨'로 변환하는 것과 '화씨'에서 '섭씨'로 변환하는 두 개의 상태가 있으므로 boolean 유형의 변수 cToF를 선언합니다. cToF 변수는 '섭씨'에서 '화씨'로 변환하는 것을 true로 지정합니다.

```
var sUnit = document.querySelector("#s-unit"); // 왼쪽 필드의 단위.
var tUnit = document.querySelector("#t-unit"); // 오른쪽 필드의 단위.
var cToF = true; // 섭씨에서 화씨로 변환 (Celsius To Fahrenheit)

var source = document.querySelector("#s-value"); // 왼쪽 텍스트 필드
var target = document.querySelector("#t-value"); // 오른쪽 텍스트 필드.
```

2. #exchange 요소(⇔화살표 부분)를 클릭했을 때 실행할 함수를 작성해 보겠습니다. converter.html 파일로 이동한 후 #exchanage 요소의 HTML 태그에 onclick="exUnit()" 소스를 추가합니다. 그리고 저장하세요.

```
<span id="exchange" onclick="exUnit()"> &#xA0; </span>
```

▶ 는 양방향 화살표를 표시하는 엔티티 코드입니다.

3. 이제 단위를 서로 바꾸는 exUnit() 함수를 정의해 보겠습니다. 혹시 텍스트 필드에 값이 입력되어 있다면 단위를 바꾸기 전에 입력값을 먼저 지워야겠죠? 그리고 cToF 변수값이 true라면, cToF 변수를 false로 바꾸고, 왼쪽에 °F를, 오른쪽에 °C를 표시합니다. cToF 변수값이 false라면, cToF 변수를 true로 바꾸고, 왼쪽에 °C를, 오른쪽에 °F를 표시합니다. 이 때 °F와 °C는 엔티티 코드를 사용해 와 로 표기합니다.

▶true 값을 false로 바꾸거나 false 값을 true로 바꿀 때 NOT 연산자를 사용해 cToF = !cToF; 로 지정해도 됩니다.

이전 스크립트 소스 다음에 아래 소스를 추가합니다.


```
function exUnit() {
    source.value = ""; // 왼쪽 텍스트 필드 값 지움
    target.value = ""; // 오른쪽 텍스트 필드 값 지움

    if(cToF) { // 현재 '섭씨'에서 '화씨'로 변환 상태라면
        cToF = false; // '화씨'에서 '섭씨' 변환 상태로 바꿈
        sUnit.innerHTML = "&#xA0;"; // 왼쪽 단위 필드에 화씨 기호
        tUnit.innerHTML = "&#xA0;"; // 오른쪽 단위 필드에 섭씨 기호
    }
    else { // 현재 '화씨'에서 '섭씨' 변환 상태라면
        cToF = true; // '섭씨'에서 '화씨' 변환 상태로 바꿈
        sUnit.innerHTML = "&#xA0;"; // 왼쪽 단위 필드에 섭씨 기호
        tUnit.innerHTML = "&#xA0;"; // 오른쪽 단위 필드에 화씨 기호
    }
}
```


4. 여기까지 작성한 소스를 저장하고 converter.html 파일에 연결하세요. 그리고 웹 브라우저에서 converter.html 파일을 열어 확인해 보겠습니다. 기본적으로 왼쪽에 °C가 표시되고 오른쪽에 °F가

표시되어 있습니다. 가운데 화살표를 클릭해 보세요. 왼쪽에 °F를, 오른쪽에 °C가 표시됩니다. '화씨'에서 '섭씨'로 변환할 수 있겠지요?

섭씨 ↔ 화씨 온도 변환기

°C  °F

섭씨 ↔ 화씨 온도 변환기

°F  °C

2. 상태에 따라 값 변환하기

본격적으로 온도를 변환하는 함수를 만들어 보겠습니다. 왼쪽의 텍스트 필드에 사용자가 값을 입력하면 함수에서 값을 변환한 후 오른쪽 텍스트 필드에 최종값을 표시합니다.

1. 값을 어느 시점에 계산할 것인지 결정합니다. 이 예제에는 [계산] 같은 버튼이 없기 때문에 값을 입력하자마자 바로 계산해야 합니다. 그래서 계산에 필요한 값을 입력하는 왼쪽 텍스트 필드에 다음과 같이 함수를 실행하는 소스를 추가합니다. `keyup`은 키보드의 키에서 손을 떼자마자, 즉 값을 입력하자마자 발생하는 이벤트입니다. `html` 파일로 들어가 다음을 작성합니다.

```
<input type="text" id="s-value" onkeyup="converter( )">
```

2. 이제 `converter()` 함수를 작성해 보겠습니다. `cToF` 변수가 `true`라면 '섭씨'에서 '화씨'로 변환해야 겠죠? 왼쪽에 있는 `#s-value` 요소에 입력하는 값은 섭씨온도가 되고, 그 값을 가져와 계산한 후 오른쪽에 있는 `#t-value` 요소에 화씨온도로 표시하면 됩니다. `cToF` 변수가 `false`라면 `#s-value`에 입

력한 값은 화씨온도가 되고, 그 값을 가져와 계산한 후 #t-value에 섭씨온도로 표시합니다. cToF 값이 true인지 false인지에 따라 계산식을 적용하면 되겠지요? 온도를 변환하는 공식은 다음과 같습니다.

cToF = "true"라면	섭씨온도가 주어짐 → 화씨온도 = (섭씨온도 * 1.8) + 32
cToF = "false"라면	화씨온도가 주어짐 → 섭씨온도 = (화씨온도 - 32) / 1.8

스크립트 소스에 다음과 같이 converter() 함수 소스를 추가한 후 저장합니다.

```
function converter( ) {
  if(cToF) {
    target.value = source.value * 1.8 + 32; // 섭씨 → 화씨 계산
  }
  else {
    target.value = (source.value - 32) / 1.8; // 화씨 → 섭씨 계산
  }
}
```

3. 모두 저장한 뒤 웹 브라우저에서 converter.html 파일을 열어 확인해 보세요. 기본 상태는 섭씨 온도에서 화씨온도로 바꾸는 것이죠? 왼쪽 텍스트 필드에 100이라고 입력하면 즉시 오른쪽 텍스트 필드에 212가 표시됩니다. 즉 100°C 는 212°F인 걸 알 수 있습니다.

섭씨 ↔ 화씨 온도 변환기

100 °C

↔

212 °F

4. 텍스트 필드 사이에 있는 ↔를 클릭한 후 왼쪽 텍스트 필드에 212라고 입력해 보세요. 오른쪽 텍스트 필드에 100이라고 표시된다면 계산이 제대로 되는 것입니다.

섭씨 ↔ 화씨 온도 변환기

212 °F

↔

100 °C

3. 계산 값 반올림하기

온도 변환기 예제에서 온도를 변환할 때 소수점 값을 입력해서 결과값이 소수점이 되거나, 화씨를 섭씨로 변환할 때 나눗셈을 사용하기 때문에 소수점 결과값이 나올 경우가 있는데, 소수점 이하 몇 자리까지 표시할지 지정하지 않으면 소수점 이하 무한한 수로 표시될 수도 있습니다. 예를 들어, 100°F 를 섭씨온도로 변환하면 37.7777...처럼 표시되죠.

섭씨 ↔ 화씨 온도 변환기

100 °F

↔

37.7777777777 °C

이 경우 `toFixed()` 메서드를 사용해서 소수점 이하 몇 번째 자리까지 표시할 것인지 지정할 수 있습니다. 예를 들어, 소수점 이하 2자리로 표시하고 싶다면 '`결과값.toFixed(2)`'라고 사용합니다. 앞에서 살펴본 예제 소스의 경우, `converter()` 함수에서 섭씨온도를 화씨온도로 변환하거나 화씨온도를 섭씨온도로 변환하는 소스를 다음과 같이 수정하면 됩니다.

```
function converter( ) {  
  if(cToF) {  
    target.value = (source.value * 1.8 + 32).toFixed(2);  
  }  
  else {  
    target.value = ((source.value - 32) / 1.8).toFixed(2);  
  }  
}
```

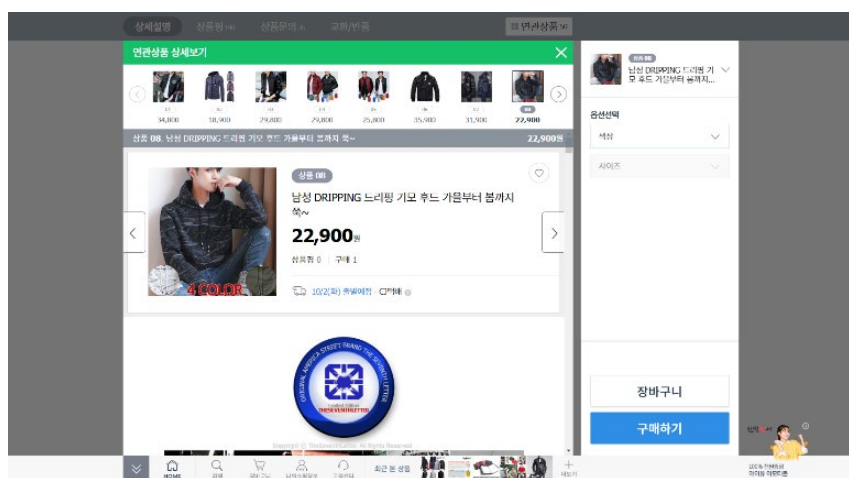
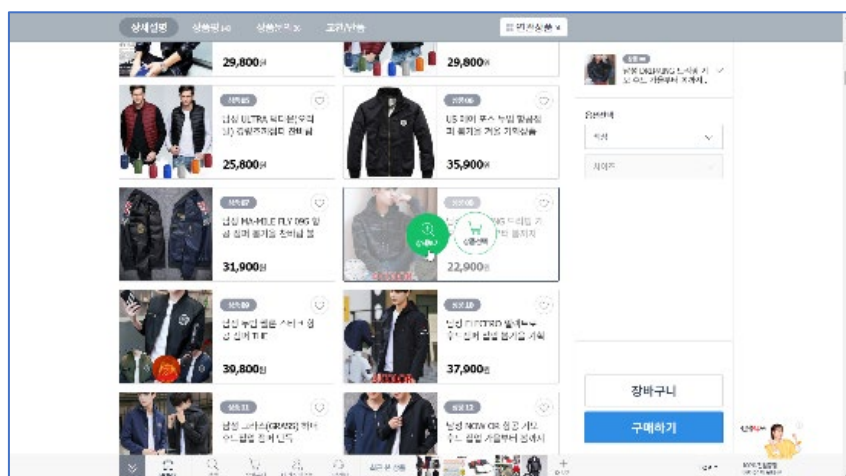
이렇게 수정한 예제를 사용해 100°F를 섭씨온도로 변환하면 37.78로 표시됩니다.

섭씨 ↔ 화씨 온도 변환기

°F ↔ °C

[실전 4] 라이트 박스

최근에는 웹 사이트에서 '라이트 박스(lightbox)' 효과를 자주 볼 수 있습니다. 라이트 박스란 원하는 내용을 전면에 표시한 후 웹 문서의 다른 부분은 어둡게 처리해서 시선을 집중시키는 것을 말합니다. 주로 이미지 갤러리나 사진이 많은 쇼핑몰 사이트, SNS 사이트 등에서 특정 이미지를 강조해 보여주려고 할 때 많이 사용합니다.



간단한 라이트 박스 효과는 지금까지 공부한 메서드와 속성을 사용해서 DOM을 조작해서 만들어 볼 수 있습니다. 도전해 볼까요?

프로그램 흐름 미리 그려보기

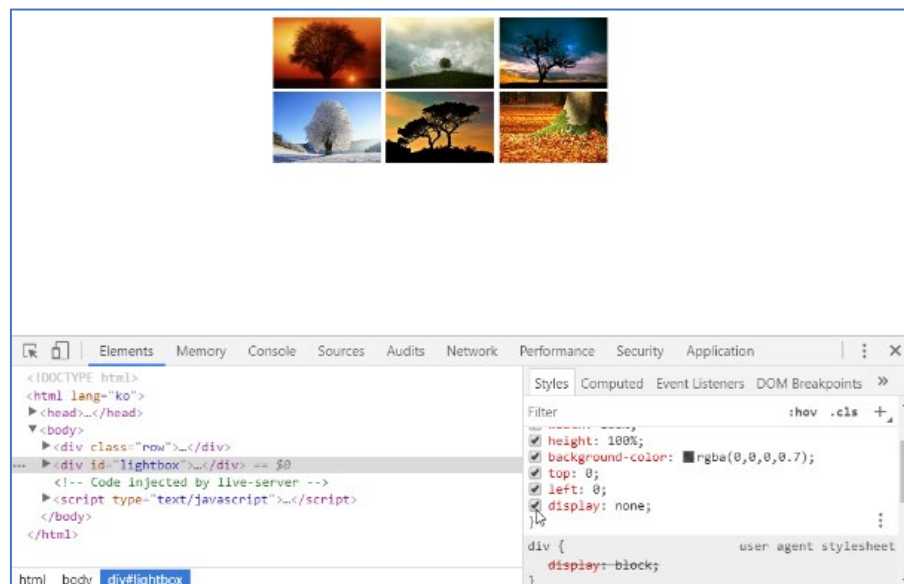
1. 웹 브라우저에서 lightbox\lightbox.html 문서를 불러와 웹 개발자 도구 창을 열면 [Elements] 탭에 웹 문서 소스가 간략히 표시됩니다. 소스 중에서 .row 요소는 썸네일 이미지 6개가 들어 있는 영역이고, #lightbox 요소는 배경을 진하게 바꾸면서 큰 이미지가 표시될 라이트 박스 영역입니다.

```

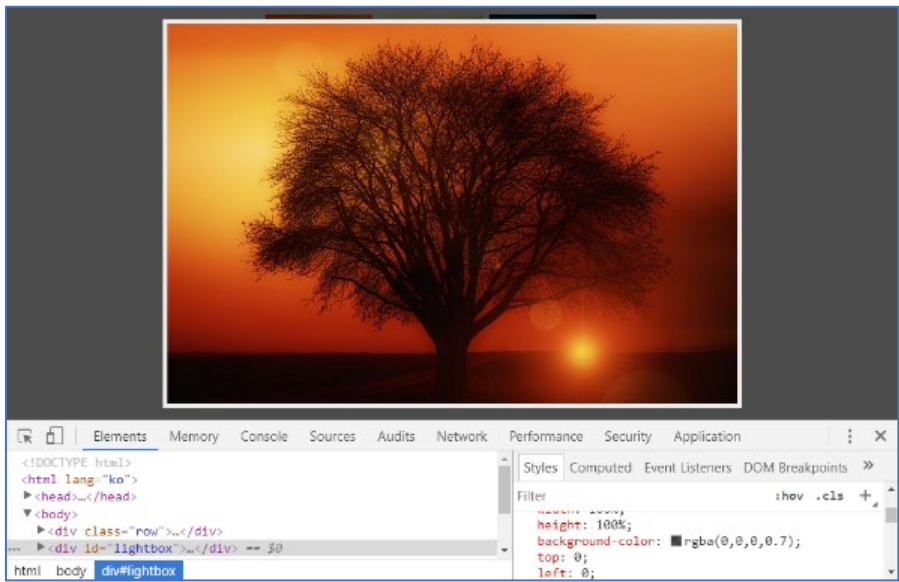
...<!DOCTYPE html> == $0
<html lang="ko">
  <head>...</head>
  <body>
    <div class="row">...</div>
    <div id="lightbox">...</div>
    <!-- Code injected by live-server -->
    <script type="text/javascript">...</script>
  </body>
</html>

```

2. <div id="lightbox">...</div>를 클릭한 후 오른쪽의 [Styles] 탭을 보면 #lightbox에 적용한 CSS 속성들이 표시됩니다. 그 중에서 'display:none' 앞의 체크 박스를 클릭해서 체크 표시를 지웁니다.



3. 적용했던 `display:none` 속성이 해제되면서 `#lightbox` 영역이 표시됩니다. 이미 라이트 박스가 만들어져 있었는데 화면에서 감춰둔 상태였던 거죠.

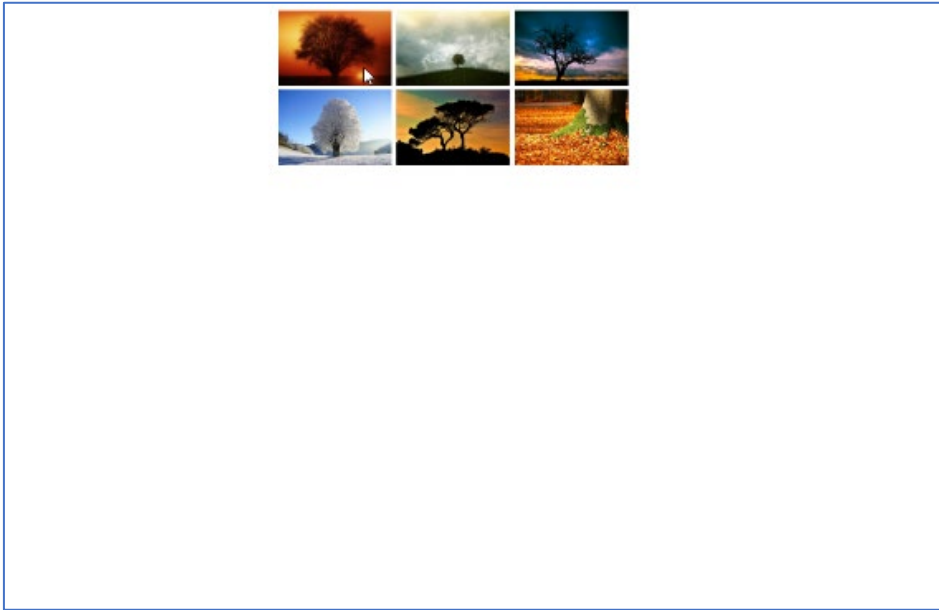


lightboxWlightbox.html 문서에는 썸네일 이미지 6개와 화면에서 감춰진 라이트 박스 영역이 만들어져 있습니다. 이제부터 만들려고 하는 프로그램은 썸네일 이미지 중 하나를 누르면 그 이미지가 라이트 박스에 표시되고, 라이트 박스를 누르면 라이트 박스가 사라지게 하는 것입니다. 과정을 간단히 정리하면 다음과 같습니다.

1	웹 문서에 포함되어 있는 썸네일 이미지 6개를 배열로 저장합니다.
2	라이트 박스 영역을 미리 잡아놓고 화면에서 감춥니다.
3	썸네일 이미지 배열 중 하나를 누르면 그 이미지를 라이트 박스 영역에 표시하고, 라이트 박스 영역을 화면에 표시합니다.
4	라이트 박스 영역을 누르면 라이트 박스를 감춥니다.

[해설]

여기에서 [는](#) 여러 이미지들이 나열된 화면에서 이미지를 누르면 큰 이미지가 라이트 박스에 표시되고, 라이트 박스 부분을 누르면 라이트 박스가 닫히는 프로그램을 만들어 보겠습니다.



1. 비주얼 스튜디오 코드에서 `lightbox\lightbox.html` 문서를 열어 소스를 확인해 보세요. 목록에서 6개의 `` 태그를 사용하고 있고, `` 태그는 `src` 속성에 썸네일 이미지 경로를 지정하고, `data-src` 속성에 큰 이미지 파일 경로를 지정하고 있습니다. 썸네일 이미지와 큰 이미지를 따로 지정해 놓은 거죠.

▶ `data-*` 속성은 사용자가 필요할 때 추가해서 사용할 수 있는 속성입니다.

```
<ul>
  <li></li>
  <li></li>
  <li></li>
  <li></li>
  <li></li>
  <li></li>
</ul>
```

2. 새 문서를 만들고 `lightbox\js` 폴더에 `lightbox.js` 파일로 저장합니다. `lightbox.js` 파일을 `lightbox.html` 파일에 연결해 주어야겠죠?

```
<script src="js/lightbox.js"></script>
```

3. 이제부터 `lightbox\js\lightbox.js` 파일에 자바스크립트 소스를 작성해 보겠습니다. 가장 먼저 6개의 이미지를 모두 가져와 배열로 저장하겠습니다. 그리고 큰 이미지를 표시할 라이트 박스 영역과 그 안의 큰 이미지도 객체로 저장합니다. 라이트 박스 영역의 소스를 보면 `<div>` 태그 안에 `` 태그를 사용했습니다.

```

var pics = document.querySelectorAll(".pic");
// pic인 요소들을 가져와 pics 라는 변수에 저장

var lightbox = document.querySelector("#lightbox");
// 라이트 박스 영역(#lightbox)를 가져와 lightbox 변수에 저장

var lightboxImage = document.querySelector("#lightboxImage");
// 라이트 박스 요소 안에 있는 이미지(#lightboxImage)를 가져와 lightboxImage 변수에 저장

```

4. 썸네일 이미지를 클릭해서 라이트 박스 영역에 표시하는 함수를 정의해 보겠습니다. 우선 썸네일 이미지를 클릭했을 때 `showLightbox()` 함수를 실행하도록 하겠습니다. 여기에서는 `addEventListener`를 이용해 클릭 이벤트와 함수를 연결합니다. `showLightbox()` 함수는 다음 단계에서 작성합니다.

```

for(var i=0; i < pics.length; i++) { // pics 배열에 있는 모든 이미지에서 반복
    pics[i].addEventListener("click", showLightbox); // 이미지 누르면 showLight 함수 실행
}

```

5. 이제 썸네일 이미지를 클릭했을 때 실행할 `showLightbox()` 함수를 정의해 보죠. 썸네일 이미지를 누르면 해당 `` 태그에 있는 큰 이미지 파일 경로를 가져옵니다. `this`(이벤트가 발생한 썸네일 이미지)에서 `data-src` 속성값을 가져와 `bigLocation` 변수에 저장하겠습니다. 그리고, 라이트 박스의 이미지 파일 경로를 `bigLocation` 값으로 수정하면 되겠죠? 마지막으로 감춰져 있던 라이트 박스를 화면에 표시합니다. 아래의 `showLightBox()` 함수 소스를 앞의 소스 다음에 추가합니다.

```

function showLightbox( ) {
    var bigLocation = this.getAttribute("data-src"); // bigLocation = this.data-src; 도 가능
    lightboxImage.setAttribute("src", bigLocation); // lightboxImage.src = bigLocation; 도 가능.
    lightbox.style.display = "block"; // 라이트박스 이미지를 화면에 표시
}

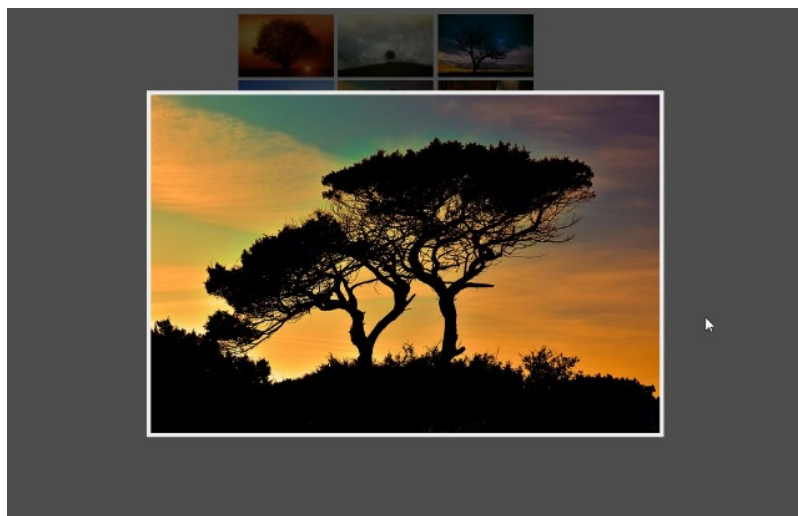
```

6. 남은 것은 화면에 표시했던 라이트 박스를 다시 감추는 것입니다. 여기에서는 사용자가 화면의 아무데나 누르면 라이트 박스를 닫을 것입니다. 이번에는 `addEventListener` 대신 `lightbox` 요소 자체에 연결해 보겠습니다. 다음 소스를 `</script>` 태그 앞에 추가합니다.

- ▶ 만일 [닫기] 버튼을 따로 만든다면 [닫기] 버튼도 가져와서 변수에 저장한 후 그 변수에 라이트 박스를 닫는 함수를 연결합니다.
- ▶ 라이트 박스의 너비와 높이가 각각 100% 이므로 화면의 아무데나 누르면 라이트 박스가 닫히게 됩니다.

```
lightbox.onclick = function( ) { //click 이벤트가 발생했을 때 실행할 함수 선언  
    lightbox.style.display = "none"; // lightbox 요소를 화면에서 감춤  
}
```

7. 지금까지 추가한 소스를 저장한 후 웹 브라우저에서 확인해 볼까요? 썸네일 이미지 중 아무 이미지나 누르면 클릭한 이미지의 큰 이미지가 라이트 박스로 표시됩니다. 그리고 화면의 아무 곳이나 누르면 열렸던 라이트 박스가 사라질 것입니다.



[실전 5] 로컬 스토리지

책에서 완성한 '준비물 목록 확인 프로그램'에는 한 가지 아쉬운 점이 있습니다. 만약 우리가 만든 프로그램으로 여행 점검 목록을 작성하던 중 브라우저 창이 닫혀 다시 index.html 웹 문서를 불러온다면 앞서 작성했던 내용이 모두 사라지고 백지로 나타납니다. 시간적 여유를 두고 여행 점검 목록을 작성한다면 웹 문서를 여닫는 것에 상관없이 작성한 목록을 계속 보여줄 수 있어야겠지요.

HTML5에 새로 도입된 로컬 스토리지를 이용하면 여행 점검 목록 내용을 브라우저에 저장해 둘 수 있습니다. 이제부터 이 로컬 스토리지가 무엇인지 알아본 다음 체크리스트 프로그램을 보완해 보겠습니다.

HTML5의 웹 스토리지 간단 소개

웹 브라우저에는 '쿠키(cookie)'라는 기능이 있습니다. 사용자가 웹 사이트에 접속해서 사이트를 옮겨 다니는 동안 사이트와 관련된 정보들이 사용자 컴퓨터에 쿠키 형태로 저장됩니다. 사용자의 컴퓨터에 정보를 기록한 파일이 생기는 것이죠. 쿠키에는 웹 사이트뿐만 아니라 접속했던 개인의 정보가 저장되어 있는데 사이트 간에 교차 스크립트 같은 기법을 사용해 쿠키를 악용할 수도 있습니다. 또한 크기가 작기 때문에 복잡한 자료는 저장할 수 없죠.

HTML5 표준이 등장하면서 클라이언트(웹 브라우저) 쪽에 자료를 저장하는 방법으로 등장한 것이 '웹 스토리지(web storage)'입니다. 웹 스토리지의 자료들도 웹 사이트를 서핑하거나 브라우저 탭을 닫으면 웹 사이트와 관련된 정보들을 저장합니다. 하지만 사용자가 웹 스토리지의 정보를 서버로 전송하지 않는 이상 서버에서 사용자 PC로 들어와 스토리지 정보를 읽어가지 못합니다. 이것이 쿠키와 가장 큰 차이점입니다. 그리고 웹 스토리지는 도메인당 2~10MB(보통 5MB 정도)의 자료를 저장할 수 있는데, 쿠키보다 용량이 큼니다.

웹 스토리지 자료는 '키(Key)'와 '값(Value)'이 하나의 쌍으로 저장되고 키를 기준으로 검색할 수도 있습니다. 웹 스토리지는 저장하는 자료의 형태에 따라 세션 스토리지(Session Storage)와 로컬 스토리지(Local Storage)로 나누어집니다.

● **세션 스토리지**: 브라우저 창이나 탭을 닫을 때까지를 하나의 세션(session)이라고 하는데, 세션 동안의 자료를 기억하고 있다가 세션이 끝나면, 즉 브라우저 창이 닫히거나 탭이 닫히면 자료를 모두 지워버립니다.

● **로컬 스토리지**: 로컬 스토리지는 세션이 끝나도 계속해서 자료를 저장할 수 있는 스토리지입니다. 로컬 스토리지 역시 객체이기 때문에 로컬 스토리지에 있는 메서드를 사용해 웹 브라우저 화면에 입력한 내용을 웹 브라우저에 저장하거나, 저장해 둔 내용을 가져와 프로그래밍에 사용할 수 있습니다. 우리가 앞서 만들었던 준비물 점검 프로그램의 경우, 사용자가 웹 브라우저 창을 닫았다가 다시 열면 기존 내용이 사라졌었죠. 이 점을 보완하려면 로컬 스토리지를 활용하면 됩니다.

HTML5에서 제공하는 웹 스토리지 API에는 세션 스토리지와 로컬 스토리지를 프로그래밍에 사용할 수 있도록 스토리지 객체를 제공하고 있습니다. 여기에서는 '로컬 스토리지' 중심으로 사용법을 알아볼 텐데, 세션 스토리지를 사용하는 방법도 비슷합니다.

sessionStorage 객체와 localStorage 객체

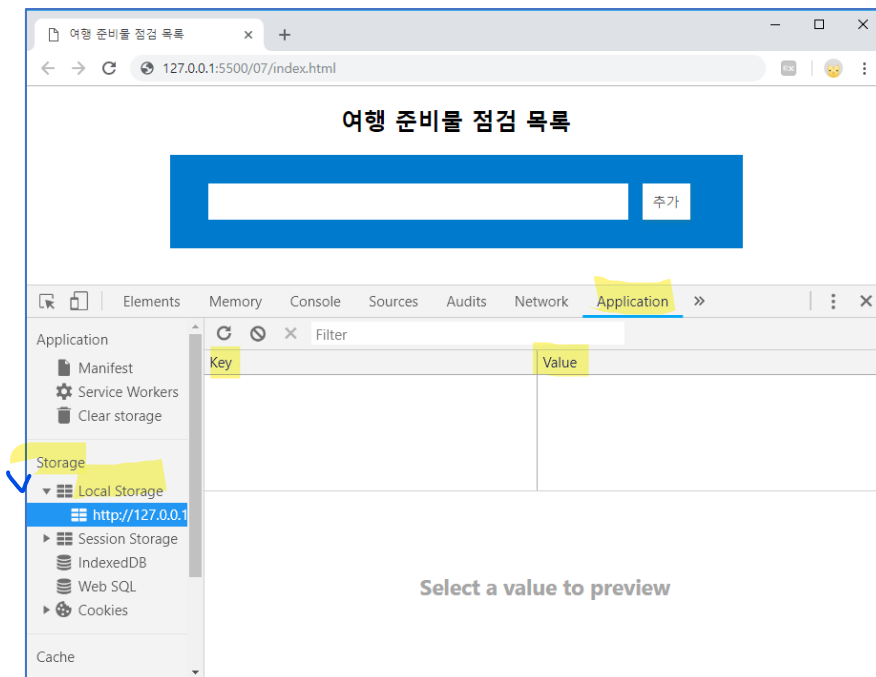
웹 스토리지에 정보를 저장할 때는 **키'와 '값'**이 하나의 쌍으로 저장됩니다. 나중에 이 '키' 값을 사용해 원하는 '값'만 가져올 수도 있죠. 스토리지에 키/값 쌍으로 된 자료를 저장하거나 저장된 값을 가져오기 위해 스토리지 객체를 사용합니다. 세션 스토리지를 사용한다면 sessionStorage 객체를 사용하고, 로컬 스토리지를 사용한다면 localStorage 객체를 사용합니다. 이 두 가지 객체에 모두에서 사용할 수 있는 프로퍼티와 메서드는 다음과 같습니다.

length	객체에 저장되어 있는 키/값 쌍의 개수를 나타냅니다.
key(n)	n번째 키의 이름을 반환합니다.
getItem(key)	키에 해당하는 값을 반환합니다. 만일 주어진 키에 해당하는 항목이 없다면 null을 반환합니다.
setItem(key, value)	주어진 키에 키/값 쌍이 있는지 확인하고 없다면 해당 키의 값(value)을 저장합니다. 해당 키에 값이 있을 때는 기존 값을 새로운 값으로 업데이트합니다.
removeItem(key)	주어진 키에 키/값 쌍이 있는지 확인하여 있다면 해당 키의 값을 삭제합니다. 만일 해당 키에 해당하는 항목이 없다면 아무것도 하지 않습니다.
clear()	모든 키/값 쌍을 삭제합니다. 만일 항목이 없다면 아무것도 하지 않습니다.

로컬 스토리지가 어떤 것인지 감이 잘 안 잡히죠? 콘솔 창에서 간단히 연습해 보겠습니다.

checklist#checklis.html 문서를 웹 브라우저에서 열고 웹 개발자 도구 창의 [Application] 탭을 클릭해 보세요. 여기에는 HTML5에서 기본으로 제공하는 다양한 프로그래밍 기능들이 표시됩니다. [Storage] 항목에는 웹 브라우저의 저장 공간과 관련된 여러 기능이 모여 있는데 이 중에서 Local Storage 앞에 있는 ▶를 클릭한 후 그 아래 표시되는 현재 문서 항목을 클릭합니다. 현재 문서에서 로컬 스토리지에 아직 아무것도 저장되어 있지 않군요.

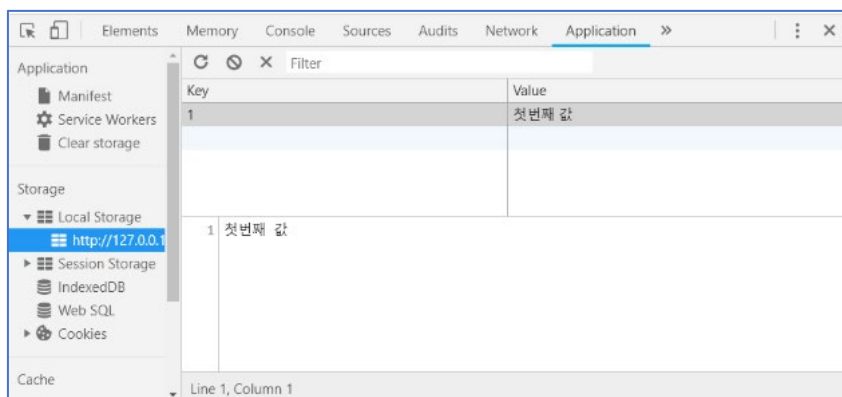
▶ 아직 저장된 내용은 없지만, 로컬 스토리지에 'Key'와 'Value' 칼럼이 나뉘어 있는 것을 볼 수 있습니다.



개발자 도구 창에서 [Console] 탭을 클릭해서 콘솔 창으로 이동합니다. 콘솔 창에 다음 소스를 입력합니다. 로컬 스토리지의 1이라는 key에 '첫 번째 값'이라는 내용을 저장하는 것입니다. 콘솔 창에는 아무 변화도 없을 것입니다.

```
localStorage.setItem(1, "첫 번째 값")
```

로컬 스토리지에 저장되었는지 확인해 보겠습니다. 개발자 도구 창에서 [Application] 탭을 클릭해서 로컬 스토리지를 확인해 보세요. 'Key'에는 '1'이, 'Value'에는 '첫 번째 값'이 저장되어 있을 것입니다.



로컬 스토리지에 저장된 값을 가져오려면 `getItem()` 메서드에서 key를 지정하면 됩니다. 콘솔 창에 다음과 같이 입력하세요.

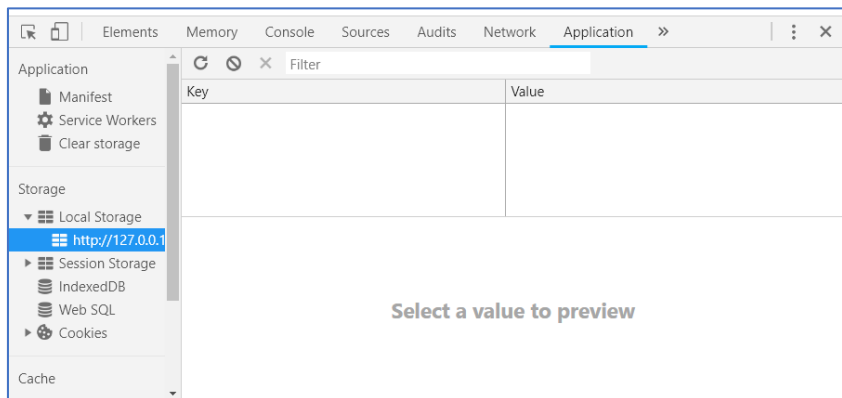
```
localStorage.getItem(1)
```

```
"첫 번째 값"
```

로컬 스토리지에 저장하고 가져오는 방법을 연습해 보았는데, 준비물 점검 목록 프로그램을 마무리하기 위해 연습했던 로컬 스토리지의 내용은 지워야겠죠? 콘솔 창에 다음과 같이 입력해서 로컬 스토리지 내용을 삭제합니다. [Application] 탭을 클릭해 보면 로컬 스토리지가 비어 있는 것을 볼 수 있을 것입니다.

```
localStorage.clear( )
```

[Application] 탭을 클릭해 보면 로컬 스토리지가 비어 있는 것을 볼 수 있을 것입니다.



배열을 로컬 스토리지에 저장하고 가져오기

이제 로컬 스토리지에 하나의 자료를 저장하고 가져오는 것은 할 수 있겠죠? 그런데 우리가 준비물 점검 목록에서 사용할 자료는 복합 자료형인 배열입니다. 배열에는 여러 값이 나열되어 있기 때문에 로컬 스토리지에 저장하거나 로컬 스토리지의 값을 가져올 때 약간 다른 방법을 사용해야 합니다.

배열을 로컬 스토리지에 저장할 때는 `JSON.stringify()` 메서드를 사용하여 배열에 있는 여러 값을 하나의 문자열처럼 변환해서 저장합니다. 그리고 로컬 스토리지에 있는 여러 값으로 된 문자열을 가져와 배열 형태로 변환할 때는 `JSON.parse()` 메서드를 사용합니다.

▶ `JSON(JavaScript Object Notation)`은 자료를 주고받는 방식을 가리키는 말로 '제이슨'이라고 읽습니다. 자바스크립트 객체를 정의하는 것과 같은 방법을 사용합니다.

콘솔 창에서 배열을 로컬 스토리지에 저장하고 가져오는 것을 연습해 보겠습니다. 콘솔 창에서 다음과 같이 `colors` 라는 배열을 만들고 `JSON.stringify()` 메서드를 사용해 `colors` 배열을 문자열로 바꾼 다음 로컬 스토리지의 `"mycolor"`라는 키에 저장합니다.

```
var colors = ["red", "green", "blue", "white", "black"]
localStorage.setItem("mycolor", JSON.stringify(colors))
```

개발자 도구 창에서 [Application] 탭을 클릭해서 확인해 보면 로컬 스토리지에 `colors` 배열의 내용이 저장되어 있을 것입니다.

이번에는 로컬 스토리지에서 `JSON` 문자열을 가져와 `newColors` 변수에 저장해 보세요. `newColors` 변수값을 확인해 보면 배열 형태로 변환된 것을 확인할 수 있을 것입니다.

```
var newColors = JSON.parse(localStorage.getItem("colors"))
newColors
▶ (5) ["red", "gree", "blue", "white", "black"]
```

콘솔 창에 로컬 스토리지를 지우는 소스를 입력해 로컬 스토리지를 지우세요.

```
localStorage.clear( )
```

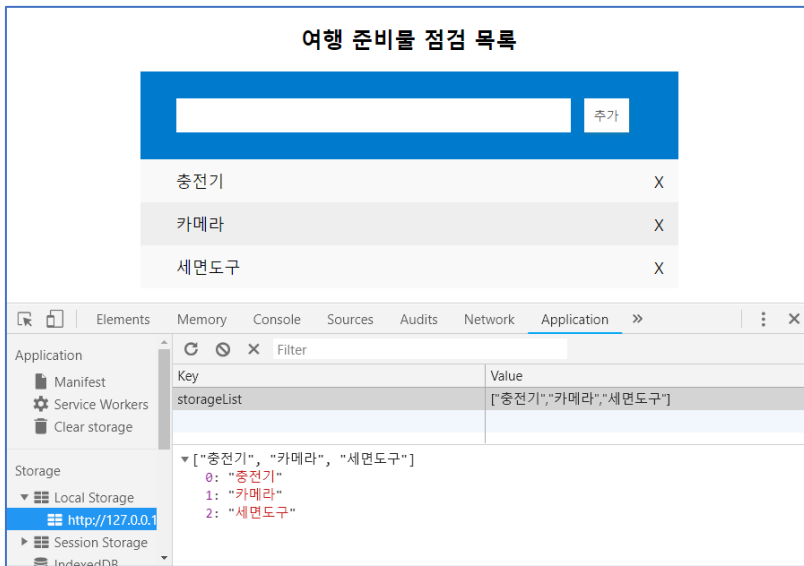
로컬 스토리지를 사용해 준비물 점검 목록 만들기

이제 로컬 스토리지를 이용하여 준비물 점검 목록 내용을 브라우저에 저장해 둘 수 있습니다. 준비물 점검 목록에서 배열 값을 로컬 스토리지에 저장하고, 저장된 로컬 스토리지 값을 가져와서 다시 표시하도록 소스를 수정해 보겠습니다.

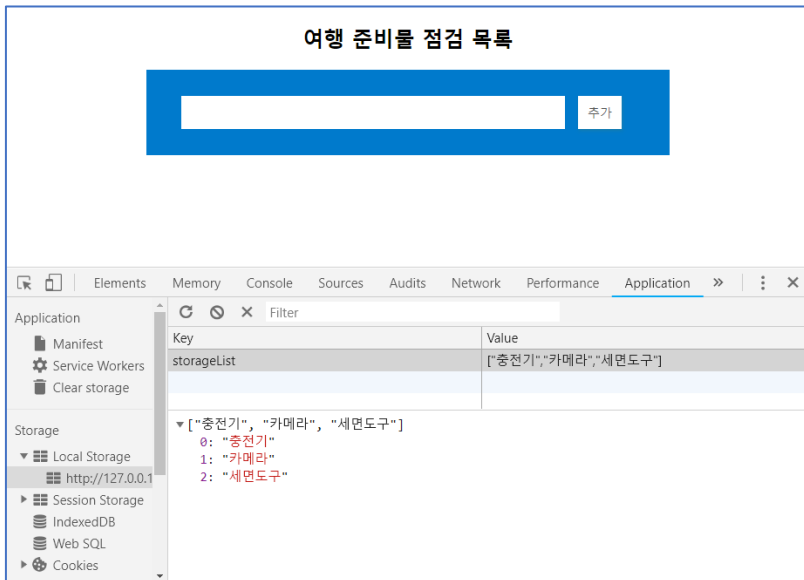
1. checklist.html 문서에 연결된 checklist.js의 소스는 addList() 함수에서 사용자가 텍스트 필드에 입력한 내용을 itemList 배열에 추가했습니다. 이렇게 만든 itemList를 로컬 스토리지에 저장하는 것도 addList() 함수 안에 작성합니다. 다음과 같이 showList() 함수 실행 소스 바로 앞에 로컬 스토리지에 저장하는 소스를 추가하세요. [Ctrl + S]를 눌러 소스를 저장하세요.

```
function addList() {
    var item = document.querySelector('#item').value;
    if(item != null) {
        itemList.push(item);
        document.querySelector('#item').value = "";
        document.querySelector('#item').focus();
    }
    localStorage.setItem("storageList", JSON.stringify(itemList));
    showList();
}
```

2. 웹 브라우저에서 checklistWindex.html 문서를 열고 몇 가지 항목을 추가한 후 웹 개발자 도구 창을 열고 [Application] 탭을 열어 항목들이 제대로 로컬 스토리지에 저장되었는지 확인해 보세요.



3. 웹 브라우저 창을 닫고, 다시 한번 checklistWindex.html을 웹 브라우저 창에 열어보세요. 분명 로컬 스토리지에는 자료가 저장되어 있는데 웹 브라우저 화면에는 표시되지 않는군요. checklistWindex.html 문서를 열었을 때 로컬 스토리지에 값이 있다면 그걸 불러와 화면에 보여주는 걸 제일 먼저 해야 합니다.



4. checklistWjsWchecklist.js 소스가 열려있는 웹 편집기로 돌아와 소스를 수정해 보겠습니다. 로컬 스토리지에서 자료를 가져오는 getItem() 함수를 다음과 같이 선언합니다. 위치는 어디에 넣어도 상관없지만, addList() 함수 앞에 넣도록 하겠습니다.

```
function getItems() {
    var storedData = localStorage.getItem("storageList");
    if(storedData != null) itemList = JSON.parse(storedData);
        showList();
}

function addList() {
    .....
}
```

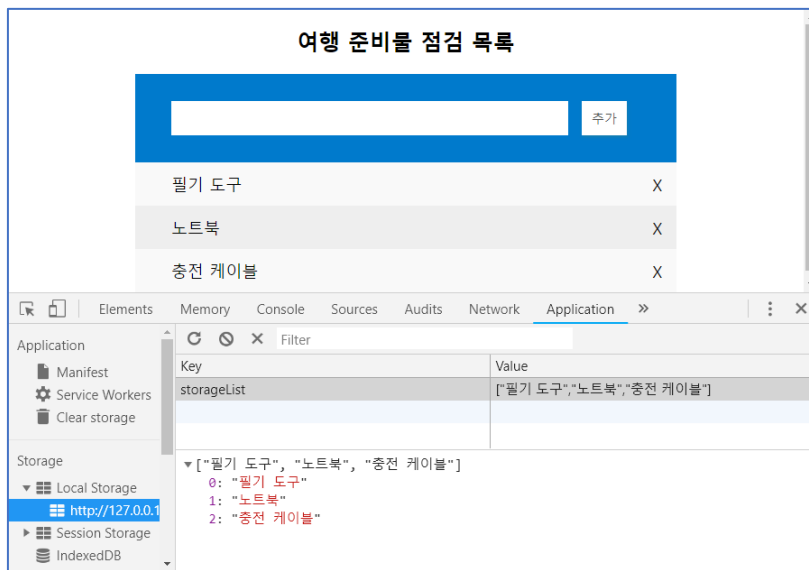
5. `getItems()` 함수를 선언했다면 어딘가에서 함수를 실행하는 소스를 추가해야겠죠? 방금 `getItems()` 함수를 선언한 소스 앞에 `getItems()` 함수를 실행하는 소스를 추가한 후 [Ctrl + S]를 눌러 소스를 저장하세요.

```
getItems( )
```

6. 웹 브라우저에서 `checklistWindex.html` 문서를 불러온 후 [Ctrl + Shift + J]를 눌러 콘솔 창에서 다음과 같이 입력해서 로컬 스토리지를 지웁니다.

```
localStorage.clear( )
```

7. 이제 텍스트 필드에 몇 가지 준비물을 입력하고 [추가]를 눌러보세요. 그리고 웹 개발자 도구 창에서 [Application]을 클릭하고 로컬 스토리지를 확인합니다. 제대로 저장이 되었지요?



8. 웹 브라우저 창을 닫았다가 다시 연 후 checklistWinex.html 문서를 불러오세요. 로컬 스토리지에 저장되어 있던 내용이 먼저 화면에 표시되지요? 이제 언제든지 준비물을 추가하거나 삭제하더라도 그 내용이 웹 브라우저에 저장되기 때문에 걱정 없을 것입니다.



[실전 6] 타이머

Date 객체에는 날짜 정보뿐만 아니라 시간 정보도 포함되어 있어서, 몇 가지 메서드를 이용하면 시간과 관련된 프로그램을 작성할 수 있습니다. 여기에서는 간단한 타이머를 만들어 보겠습니다.

사용자가 '분' 값이나 '초' 값을 입력한 후 [START] 버튼을 클릭하면 1초씩 줄어드다가 시간이 모두 지나면 '타이머 종료'라고 표시하는 프로그램입니다. 이 프로그램을 만들어 보면서 타이머가 어떻게 동작하는지, 자바스크립트에서 '시간'을 어떻게 처리하는지에 대해 배울 수 있습니다.

프로그램 흐름 미리 그려보기

타이머는 '시간을 지정'하고 '시작' 버튼 클릭하면 '시간이 1초씩 줄어드는' 간단한 프로그램입니다. 하지만 이것을 논리적으로 구성하는 일은 생각만큼 만만하지 않습니다. 타이머가 어떻게 동작하는지 단계별로 생각해 보겠습니다.

타이머를 사용하는 사람이 가장 먼저 하는 일은 시간을 지정하는 것입니다. 보통 '몇 분 몇 초'처럼 분 단위 시간과 초 단위를 함께 입력하겠죠? 하지만 '3분'처럼 분 단위 값만 있고 초 단위 값을 입력하지 않을 수도 있고, '15초'처럼 분 단위 없이 초 단위 값만 입력할 수도 있습니다. 프로그램을 작성할 때는 프로그램에 값을 입력하는 여러 경우의 수를 다 따져봐야 합니다. 여기에서는 분 단위 값이 없거나 초 단위 값이 없을 경우 그 값을 '0'으로 설정할 것입니다. 그래야만 '분'과 '초'를 함께 계산에 사용할 수 있습니다.

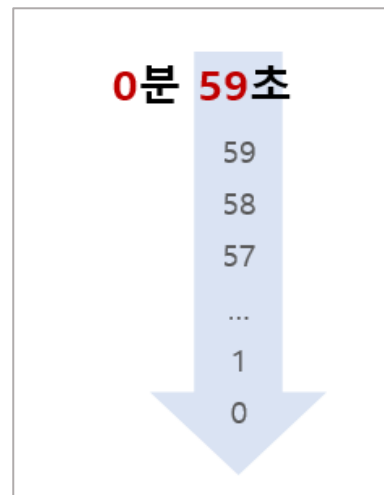
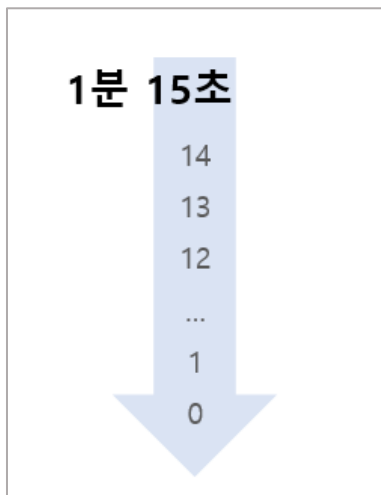
예를 들어, '1분 15초'라는 시간을 지정하고 타이머를 실행한다고 생각해 보겠습니다. '시작' 버튼을 누르면 타이머를 실행

합니다.

과정을 표로 나타내면 다음과 같습니다.

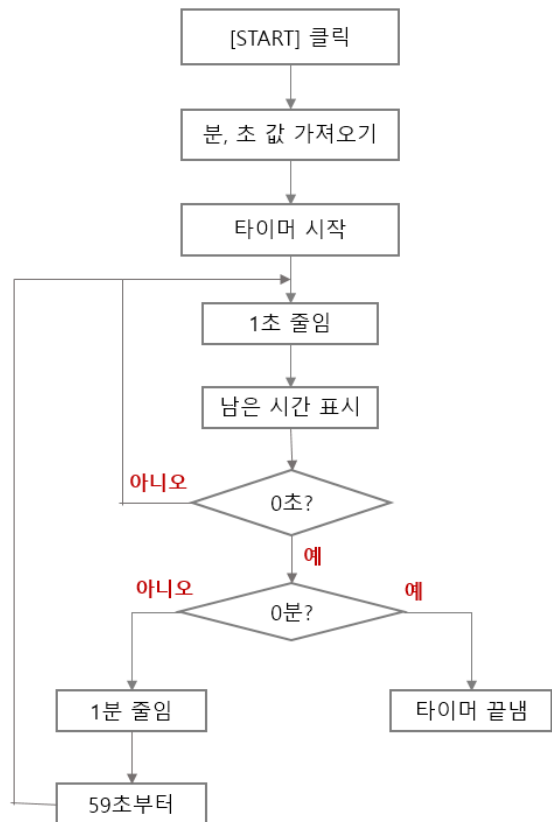
1	시간 지정하기 ('초' 값만 있다면 '분' 값을 0 으로, '분' 값만 있다면 '초' 값을 0 으로 설정합니다.)	
2	'시작' 버튼 눌러 함수를 실행합니다.	
3	'초' 값을 1 초 줄입니다.	
4	지정한 시간에서 1 초 줄인 값을 화면에 표시합니다.	
5	'초' 값이 0 인지 확인	
	5-1	'초' 값이 0 이 아니라면 3 번 단계로 이동
	5-2	'초' 값이 0 이라면 6 번 단계로 이동
6	'분' 값이 0 인지 확인	
	6-1	'분' 값이 0 이 아니라면 값을 1 분 줄이고 '초' 값을 59 로 만듭니다.
	6-2	'분' 값이 0 이라면 타이머를 종료합니다.

'1분 15초'에서는 초 값이 0이 되면 '1분'을 '0분'으로 바꾸고 초 값은 '0초'에서 '59초'로 바꿉니다. 그리고 초 값을 다시 1씩 줄입니다.



이 상태에서 다시 ‘초’ 값이 0이 되면 ‘분’ 값을 확인하는데, ‘분’ 값이 0이지요? ‘분’ 값과 ‘초’ 값이 모두 0이면 0분 0초, 즉 타이머가 종료됩니다.

여기까지의 진행 방법을 순서도로 그리면 다음과 같습니다. 눈으로 보기만 하는 것보다 손으로 익히는 것이 기억에 더 잘 남습니다. 직접 노트에 따라 그려보세요.



함수 반복해서 실행하기 및 종료하기

타이머에서 기본적으로 할 일은 '초' 값을 1씩 줄이는 함수를 만들고 그 함수를 1초마다 반복 실행하는 것입니다. 그래서 함수를 일정 시간마다 반복하고, 반복을 멈추는 방법부터 알아보겠습니다.

1. setInterval() 메서드

일정 시간마다 같은 함수를 반복하려면 `setInterval()` 메서드를 사용합니다.

괄호 안에는 함수 이름과 시간이 들어가는데, 이 때 함수 이름은 괄호 없이 사용하고, 시간은 밀리초를 사용합니다. 예를 들어 `add()` 함수를 함수를 1초마다 반복한다면 다음처럼 사용합니다.

```
setInterval(add, 1000)
```

▶ 0.5초~ 10초까지 밀리초로 변환한 값

초(sec)	밀리초(msec)
0.5	500
1	1000
2	2000
3	3000
4	4000
5	5000
6	6000
7	7000
8	8000
9	9000
10	10000

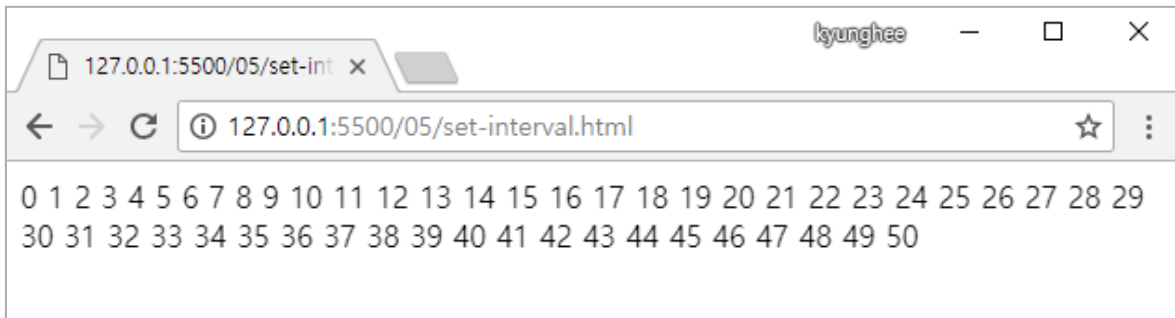
Timer\set-interval.html 문서에 있는 다음 소스는 `setInterval(rpt, 500)`을 사용해서 `rpt()` 함수를 0.5초마다 반복해서 실행하는 것입니다. `rpt()` 함수는 `num` 변수의 값을 화면에 표시하고 1 증가

시키는 함수입니다. 결과적으로 0.5초마다 숫자가 1씩 증가하게 되겠죠? 그런데 어디에서 멈추라고 지정하지 않았기 때문에 브라우저 화면에 숫자가 끝없이 표시됩니다.

```
var num = 0;

setInterval(rpt, 500); // rpt 함수를 500 밀리초(0.5초)마다 실행

function rpt( ) {
    document.write(num + " ");
    num++;
}
```



2. clearInterval() 메서드

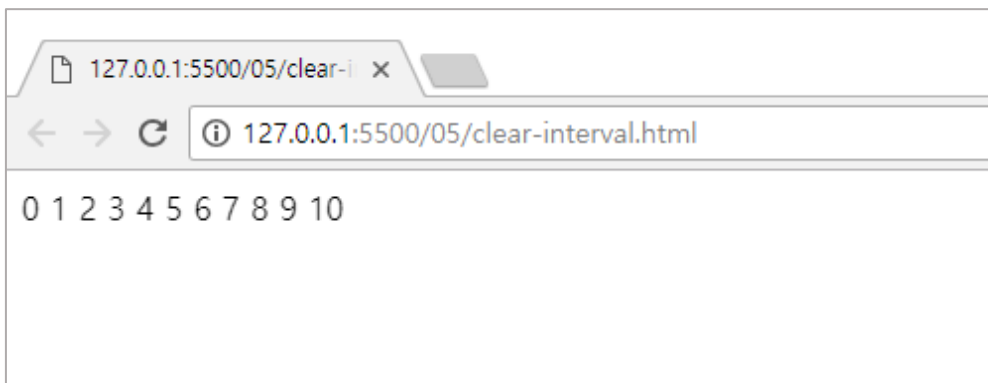
setInterval() 메서드로 반복하던 함수를 종료할 경우에는 clearInterval() 메서드를 사용합니다. clearInterval() 메서드를 사용하려면 멈춰야 할 대상이 무엇인지 알 수 있도록 setInterval() 메서드를 변수에 지정하고, 그 변수 이름을 clearInterval() 메서드에게 알려줍니다.

TimerWclear-interval.html 문서의 예제는 0.5초마다 rpt 함수를 실행하는 setInterval() 메서드를 counter라는 변수로 저장한 후, rpt 함수로 실행되는 숫자가 10보다 커지면 clearInterval() 메서드를 사용해 counter를 멈춥니다. 즉, 화면에는 10까지만 표시되겠죠. timerWinterval.html 파일을 비주얼 스튜디오 코드로 열어 확인해 보세요.

```
var num = 0;

var counter = setInterval(rpt, 500);

function rpt( ) {
    document.write(num + " ");
    num++;
    if (num > 10) clearInterval(counter); // num이 10보다 크면 counter의 반복 실행 멈춤
}
```



setInterval() 메서드와 clearInterval() 메서드의 사용법을 이해했다면 앞에서 그려본 설계도를 참고해서 직접 타이머를 만들어 보세요. ✓

60 ~ 59 --- 1

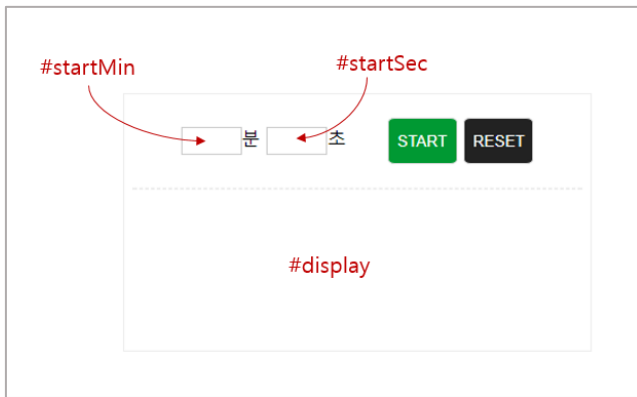
[해설]

1. 시간 값 가져오기

여기에서 만들 타이머는 시간을 입력하고 [Start] 버튼을 누르면 시간이 1초씩 줄어들면서 남은 시간을 표시합니다. 그리고 타이머가 동작하는 중에 [Reset] 버튼을 누르면 타이머가 초기화되는 형태입니다. 우선 입력한 시간 값을 가져오는 방법부터 살펴보겠습니다.

1. 비주얼 스튜디오 코드에서 timer\timer.html 문서를 열어 소스가 어떻게 구성되어 있는지 살펴보세요. #startMin 영역과 #startSec 영역에 시간을 입력하고 [Start] 버튼을 누르면 #display 영역에 줄어드는 시간을 표시할 것입니다.

```
<div id="wrapper">
  <form>
    <label><input type="text" id="startMin">분</label>
    <label><input type="text" id="startSec">초</label>
    <input type="button" value="START" class="start-btn">
    <input type="button" value="RESET" class="reset-btn">
  </form>
  <hr>
  <div id="display" class="remaining"></div>
</div>
```



2. [START] 버튼을 클릭했을 때와 [RESET] 버튼을 클릭했을 때, 각각 `startTimer()` 함수와 `resetTimer()` 함수를 실행하도록 다음과 같은 소스를 추가합니다.

```
<input type="button" value="START" class="start-btn" onclick="startTimer()">
<input type="button" value="RESET" class="reset-btn" onclick="resetTimer()">
```

3. 이제 [START] 버튼을 클릭했을 때 실행할 `startTimer()` 함수를 작성하겠습니다. 우선 빈 문서를 만든 후 `timerWjs` 폴더에 `timer.js`로 저장합니다. 그리고 `timer.html`로 와서 `timer.js` 파일을 연결합니다.

```
<script src="js/timer.js"></script>
```

`timer.js` 파일로 가서 각 텍스트 필드 영역에서 값을 가져오는 소스를 삽입합니다. 영역에 값이 입력되어 있지 않다면 변수에 0을 저장합니다. 이 때 `min`과 `sec` 변수는 `startTimer()` 함수뿐만 아니라 `resetTimer()` 함수에서도 사용할 것이므로 함수 외부에 선언합니다.

▶ 좀 더 완벽한 프로그램이 되려면 가져온 값을 체크해서 0보다 작은 값일 경우에는 잘못 입력했다고 알려주어야 합니다. 즉, 값이 0이거나 0보다 커야 하죠. 하지만 여기에서는 최대한 소스를 간결하게 작성하였고, 타이머 동작 방법을 공부하기 위해 입력한 값의 조건은 체크하지 않았습니다.

```

var min, sec; // 전역 변수 선언

function startTimer( ) {

    min = document.querySelector('#startMin').value; // '분' 값 가져옴

    if (min == "") min = 0; // '분' 값이 없다면 0으로 지정

    sec = document.querySelector('#startSec').value; // '초' 값 가져옴

    if (sec == "") sec = 0; // '초' 값이 없다면 0으로 지정

}

```

▶ 전역 변수란 스크립트 소스 전체에서 사용할 수 있는 변수를 말합니다. 즉, `<script>`와 `</script>` 사이에서 자유롭게 사용할 수 있는 변수입니다. 전역 변수가 무엇인지 기억나지 않는다면 04-2 함수 선언 및 호출하기 ##쪽 내용을 복습하고 오세요.

4. 시간 값을 가져왔으면 이제 시간을 1초씩 줄여야겠지요? `setInterval()` 메서드를 사용해 타이머 함수 `countTimer()` 함수를 반복 실행합니다. 이 때 나중에 타이머를 중지할 수 있도록 `setInterval()`이 실행되는 타이머 이름을 timer라고 지정했습니다.

변수 timer를 선언한 후, `startTimer()` 함수 끝에 `setInterval()` 소스를 추가합니다.

```

var min, sec;

var timer; // timer 변수 선언

function startTimer() {

    min = document.querySelector('#startMin').value;

    if (min == "") min = 0;

    sec = document.querySelector('#startSec').value;

    if (sec == "") sec = 0;

    timer = setInterval(countTimer, 1000); // 1초(1000밀리초)마다 countTimer() 함수 실행

}

```

2. 가져온 시간 값을 1초씩 줄이기

이제부터 타이머를 작동시킬 `countTimer()` 함수를 작성해 보겠습니다. 타이머는 ‘초’ 값을 1초씩 줄이는데 ‘초’ 값이 0이 되면 ‘분’ 값이 있을 때와 ‘분’ 값이 없을 때를 나눠서 생각해야 합니다.

1. 타이머 함수는 `if` 문과 `else` 문을 사용해 ‘초’ 값이 0일 경우와 그렇지 않을 경우를 나누어 생각합니다. ‘초’ 값이 0이 아니라면 1초 줄이고 `#display` 영역에 남은 시간을 표시합니다.

```
function countTimer( ) {
    if(sec != 0) { // ‘초’ 값이 0이 아닐 때 실행할 명령들
        sec--;
        document.querySelector('#display').innerText = min + "분 " + sec + "초 남았습니
다.";
    }
    else { // 초’ 값이 0일 경우 처리할 부분. 일단 비워두세요.

    }
}
```

2. ‘초’ 값이 0이 되거나 처음부터 ‘초’ 값을 입력하지 않았다면 ‘분’ 값을 확인합니다. 이번에도 ‘분’ 값이 있을 때와 없을 경우로 나누어 생각해야 하므로 `if`문과 `else`문을 사용합니다. ‘분’ 값이 있다면 1분 줄이고 ‘초’ 값을 59로 지정합니다. 앞에 비워둔 `else`문에 다음 소스를 추가합니다.


```

else {
  if (min != 0) {    // (sec=0 이면서) '분' 값이 0이 아닐 때 실행할 명령
    min- -;    // 1분 줄이고
    sec = 59;    // '초' 값을 59초부터 시작
  }
  else {    // (sec = 0 이면서) '분' 값이 0일 때, 즉 타이머가 끝났을 때 실행할 명령.
    // 일단 비워두세요.
  }
}
}

```

3. '분'과 '초'가 모두 0이 되면 타이머가 끝난 것이므로, 반복 실행하던 것을 멈춰야 합니다. 앞에서 비워 두었던 else 문에 타이머를 정지시키고 타이머가 끝났다는 내용을 표시하는 소스를 추가합니다.

```

else {
  clearInterval(timer);    // 타이머 종료
  document.querySelector('#display').innerText = "타이머 종료";
}

```

4. 여기까지 작성했다면 [Ctrl+S]를 눌러 추가한 소스를 저장한 후 웹 브라우저에서 확인해 보세요. 여기에서는 간단히 '5'초를 입력하고 [START]를 눌러보겠습니다.

5. 지정한 시간에서 1초씩 줄어들면서 화면에 남은 시간이 표시될 것입니다. 그리고 지정한 시간

이 모두 지나면 '타이머 종료'라고 표시됩니다. 그런데 타이머가 끝난 후에도 입력했던 값인 '5'초가 그대로 남아 있군요. 타이머가 종료되면 입력했던 내용이 모두 지워지도록 설정해 보겠습니다. 아직 브라우저 창을 닫지 마세요.



The screenshot shows a web application with a timer. At the top, there are two input fields: the first is empty and followed by the Korean character '분' (minutes), and the second contains the number '5' and is followed by '초' (seconds). To the right of these fields are two buttons: a green 'START' button and a dark grey 'RESET' button. Below these elements, separated by a dashed horizontal line, is a large text area that displays '0분 3초 남았습니다.' (0 minutes 3 seconds remaining).

6. 소스를 작성하던 편집기 창으로 돌아옵니다. 타이머를 종료하는 소스 다음에 '분'과 '초' 텍스트 필드 영역을 지우는 소스를 추가합니다. [Ctrl+S]를 눌러 소스를 저장합니다.

```
else {  
    clearInterval(timer);  
    document.querySelector('#display').innerText = "타이머 종료";  
    document.querySelector('#startMin').value = "";  
    document.querySelector('#startSec').value = "";  
}
```

7. 열려있던 브라우저 창으로 돌아와 [F5]를 눌러 수정된 소스를 반영합니다. 다시 한번 시간을 입력하고 [START] 버튼을 누르세요. 이번에는 타이머가 종료되면서 텍스트 필드 영역도 모두 지워질 것입니다.



This screenshot shows the same timer interface as before, but after a reset. The '분' (minutes) input field is now empty, and the '초' (seconds) input field is also empty. The 'START' and 'RESET' buttons remain. Below the dashed line, the text area now displays '타이머 종료' (Timer ended).

3. 타이머 리셋하기

우리가 만드는 타이머에는 [RESET] 버튼이 있습니다. [RESET] 버튼을 클릭하면 타이머를 종료하고 모든 값이 지워져야 합니다. 타이머를 리셋하는 함수를 작성해 보겠습니다.

1. 타이머 실행 중에 타이머를 리셋(reset)하려면 실행 중이던 타이머를 종료하고, 남은 시간이 표시되던 부분과 시간이 입력됐던 텍스트 필드를 모두 지워야 합니다. 앞에서 작성했던 함수 뒤에 다음과 같은 소스를 추가한 후 [Ctrl+S]를 눌러 저장하세요.

```
function resetTimer() {
  clearInterval(timer); // 타이머 종료

  document.getElementById("display").innerText = ""; // 표시 영역 지움

  document.getElementById('startMin').value = ""; // '분' 값 지움
  document.getElementById("startSec").value = ""; // '초' 값 지움
}
```

2. 웹 브라우저에서 timerWtimer.html을 불러오세요. 시간을 입력한 후 [START] 버튼을 눌러 타이머를 실행합니다. 여기에서는 2분 30초를 입력했습니다.

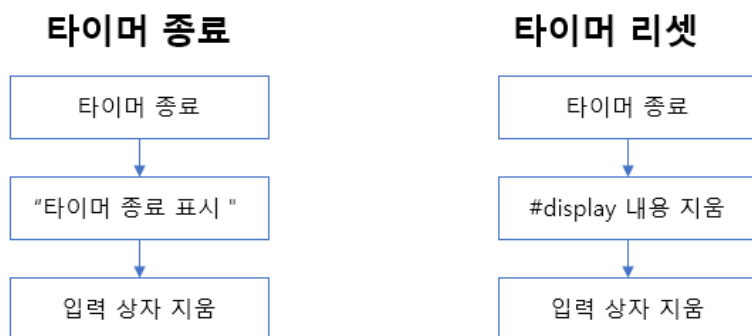
3. 타이머가 실행되는 도중에 [RESET] 버튼을 눌러보세요. 타이머가 취소되면서 텍스트 필드와 시간 표시 부분이 모두 지워질 것입니다.



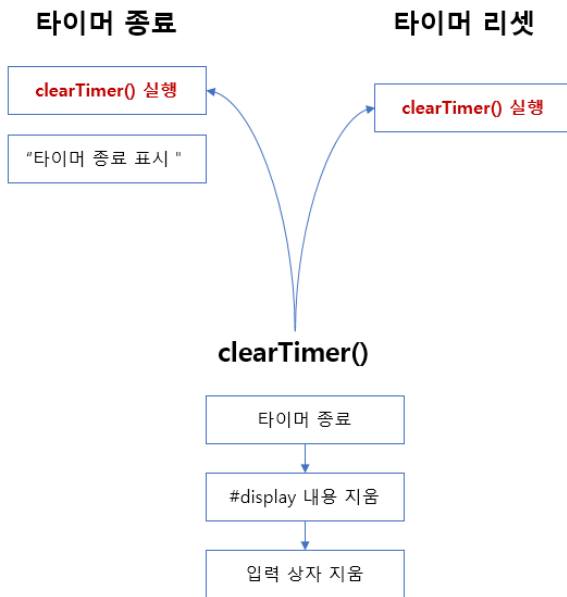
4. 코드 재사용하기

여기까지만 실행해도 우리가 예상했던 대로 타이머가 움직입니다. 하지만 자바스크립트 소스를 보면 타이머가 정상 종료되었을 때의 소스와 `resetTimer()` 함수 부분이 거의 비슷하게 반복되는 것을 볼 수 있습니다. 반복되는 부분을 새로운 함수로 만들어 프로그램을 좀 더 효율적으로 작성해 보겠습니다.

1. 타이머가 종료되었을 때의 소스와 타이머를 리셋했을 때의 소스는 `#display` 영역 부분만 제외하면 똑같습니다.



2. 여기에서 중복되는 부분을 `clearTimer()`라는 함수로 묶고, `else`문에 있는 ‘타이머 종료’라는 텍스트 표시만 따로 해주면 됩니다. 중복되는 소스를 함수로 묶어 여러 곳에서 간단하게 사용하는 것을 ‘코드 재사용’이라고 하는데, 여기에서는 `clearTimer()`라는 함수로 묶었지만 코드 재사용 방법은 개발자의 평소 개발 습관이나 다른 소스와의 연관성에 따라 다양한 방법이 있을 수 있습니다.



3. `clearTimer()` 라는 새로운 함수를 만들기 위해 `timer.js` 파일 끝에 다음 소스를 추가합니다.

```
function clearTimer( ) {

}
```

4. `resetTimer()` 함수 안의 소스들을 모두 선택한 후 [Ctrl+X]를 눌러 선택한 부분을 잘라내어, `clearTimer()` 함수 안에 붙여 넣습니다.

```
function clearTimer( ) {
    clearInterval(timer);
    document.querySelector("display").innerText = "";
    document.querySelector('startMin').value = "";
    document.querySelector("startSec").value = "";
}
```

5. `clearTimer()` 함수를 좀더 일반적으로 사용하기 위해 타이머 이름을 매개변수로 받는 함수로 수

정합니다. `clearInterval(timer)`라고 하면 'timer'라는 타이머 하나만 종료하지만 매개변수로 받으면 어떤 타이머든지 타이머 이름을 넘겨받아 타이머를 종료할 수 있습니다.

```
function clearTimer(t) { // 매개변수 t를 이용해 타이머 이름을 받음
  clearInterval(t); // 넘겨받은 타이머 종료
  document.querySelector("display").innerText = "";
  document.querySelector('startMin').value = "";
  document.querySelector("startSec").value = "";
}
```

6. `countTimer()` 함수의 `else`문에서 중복되는 소스를 수정하고 `resetTimer()` 함수 소스를 다음과 같이 수정합니다. 여기에서 '타이머 종료'를 표시하는 소스보다 `clearTimer(timer)`를 앞에 둔 이유는, `clearTimer()` 함수에서 `#display` 영역을 지워버리기 때문입니다. [Ctrl+S]를 눌러 소스를 저장합니다. 소스가 잘 작동되는지 웹 브라우저에서 확인해 보세요.

```
    else {
        clearTimer(timer);
        document.querySelector('#display').innerText = "타이머 종료";
    }
}

function resetTimer() {
    clearTimer(timer);
}
```