

15. 함수와 이벤트

15-1 함수 알아보기

15-2 var를 사용한 변수의 특징

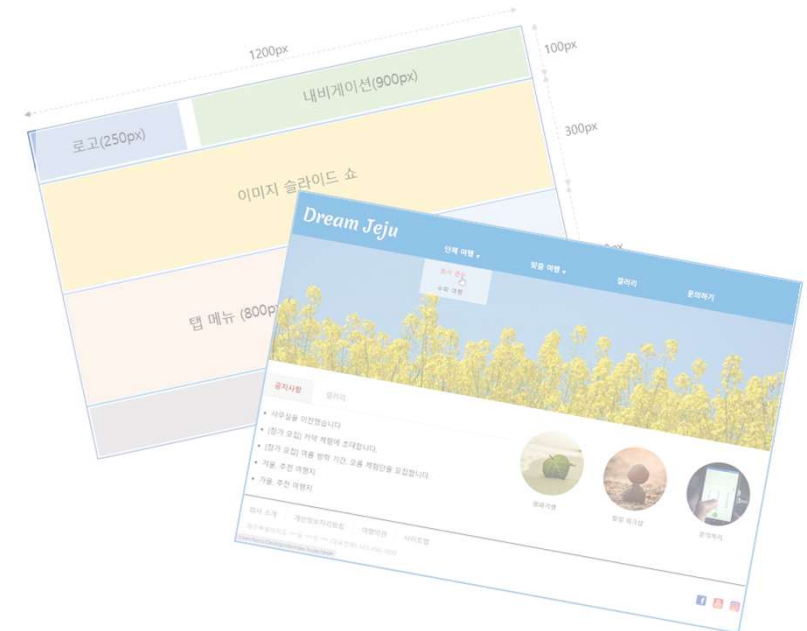
15-3 let와 const의 등장

15-4 재사용할 수 있는 함수 만들기

15-5 함수 표현식

15-6 이벤트와 이벤트 처리기

15-7 DOM을 이용한 이벤트 처리기



- ECMA
 - European Computer Manufacturers Association
 - Ecma International
 - ECMA-262 - ECMAS크립트 언어 규격 (자바스크립트 기반)

ECMAScript Editions

| Ver | Official Name | Description |
|-----|--|---|
| ES1 | ECMAScript 1 (1997) | First edition |
| ES2 | ECMAScript 2 (1998) | Editorial changes |
| ES3 | ECMAScript 3 (1999) | Added regular expressions Added try/catch Added switch Added do-while |
| ES4 | ECMAScript 4 | Never released |
| ES5 | ECMAScript 5 (2009) Read More | Added "strict mode" Added JSON support Added String.trim() Added Array.isArray() Added Array iteration methods Allows trailing commas for object literals |
| ES6 | ECMAScript 2015 Read More | Added let and const Added default parameter values Added Array.find() Added Array.findIndex() |
| | ECMAScript 2016 Read More | Added exponential operator (**) Added Array.includes() |
| | ECMAScript 2017 Read More | Added string padding Added Object.entries() Added Object.values() Added async functions Added shared memory Allows trailing commas for function parameters |
| | ECMAScript 2018 Read More | Added rest / spread properties Added asynchronous iteration Added Promise.finally() Additions to RegExp |
| | ECMAScript 2019 Read More | String.trimStart() String.trimEnd() Array.flat() Object.fromEntries Optional catch binding |
| | ECMAScript 2020 Read More | The Nullish Coalescing Operator (??) |

함수 알아보기

함수란

- 동작해야 할 목적대로 명령을 묶어 놓은 것
- 각 명령의 시작과 끝을 명확하게 구별할 수 있음
- 묶은 기능에 이름을 붙여서 어디서든 같은 이름으로 명령을 실행할 수 있음
- 자바스크립트에는 이미 여러 함수가 만들어져 있어서 가져다 사용할 수 있음

예) alert()

함수의 선언 및 호출

함수 선언 : 어떤 명령을 처리할지 미리 알려주는 것

```
기본형 function 함수명() {  
    명령  
}
```

함수 호출 : 선언한 함수를 사용하는 것

```
기본형 함수명( ) 또는 함수명(변수)
```

Do it! 함수를 사용해 두 수 더하기 예제 파일 15\using-function.html

(... 생략 ...)

```
<script>  
function addNumber() {  
    var num1 = 2;  
    var num2 = 3;  
    var sum = num1 + num2;  
    alert("결과값: " + sum);  
}  
  
addNumber();  
addNumber();  
</script>  
(... 생략 ...)
```

함수를 선언

함수를 2번 호출

127.0.0.1:5500 내용:
결과값: 5

확인

127.0.0.1:5500 내용:
결과값: 5

확인

알림 창에서
결과값이 두 번 나타납니다.

var를 사용한 변수의 특징

스코프 : 변수가 적용되는 범위

스코프에 따라 지역 변수(로컬 변수)와 전역 변수(글로벌 변수)로 나뉨

지역 변수

- 함수 안에서 선언하고 함수 안에서만 사용함
- var과 함께 변수 이름 지정

Do it! var 예약어로 지역 변수 선언하기 예제 파일 15\var-1.html

```
(... 생략 ...)  
<script>  
  fu addNumber() {  
    var sum = 10 + 20; // 지역 변수 선언  
  }  
  addNumber();  
  
  console.log(sum); // 지역 변수를 사용  
</script>  
(... 생략 ...)
```

변수 sum 적용 범위

오류 발생

전역 변수

- 스크립트 소스 전체에서 사용함
- 함수 밖에서 선언하거나 함수 안에서 var 없이 선언

Do it! var 예약어를 사용한 지역 변수와 전역 변수 예제 파일 15\var-2.html

```
(... 생략 ...)  
<script>  
  function addNumber() {  
    v sum = 10 + 20; // 지역 변수 선언  
    multi = 10 * 20; // 전역 변수 선언  
  }  
  addNumber();  
  console.log(multi); // 전역 변수를 사용  
</script>  
(... 생략 ...)
```

200

var를 사용한 변수의 특징

var 변수와 호이스팅

```
Do it! 변수와 호이스팅 예제 파일 15\var-3.html

<script>
  var x = 10;

  function displayNumber() {
    console.log("x is " + x);
    console.log("y is " + y);
    var y = 20;
  }

  displayNumber();
</script>
```

호이스팅

- 변수를 뒤에서 선언하지만, 마치 앞에서 미리 앞에서 선언한 것처럼 인식함
- 함수 실행문을 앞에 두고 선언 부분을 뒤에 두더라도 앞으로 끌어올려 인식함

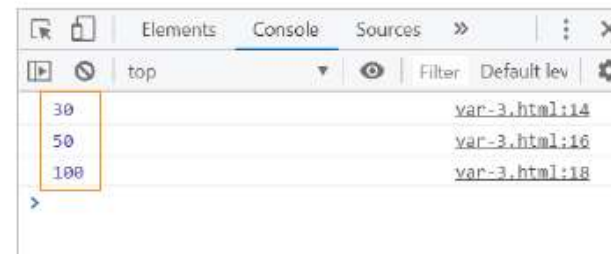
재선언과 재할당이 가능하다

- 재선언 : 이미 선언한 변수를 다시 선언할 수 있음
 - 재할당 : 같은 변수에 다른 값을 할당할 수 있음
- 재선언과 재할당이 가능하면 실수로 변수를 잘못 조작할 확률이 높아짐

```
Do it! var 예약어를 사용한 변수의 재할당과 재선언 예제 파일 15\var-4.html

<script>
  function addNumber(num1, num2) {
    return num1 + num2; // 2개의 수 더하기
  }

  var sum = addNumber(10, 20); // sum 변수 선언, 함수 호출
  console.log(sum);
  sum = 50; // sum 변수 재할당
  console.log(sum);
  var sum = 100; // sum 변수 재선언
  console.log(sum);
</script>
```



let과 const의 등장

let을 사용한 변수의 특징

- 블록 변수 – 블록({ }) 안에서만 사용할 수 있다
→ 전역 변수는 변수 이름과 초깃값만 할당하면 됨

전역 변수

```
(... 생략 ...)  
function calcSum(n) {  
  sum = 0;  
  for(let i = 1; i < n + 1; i++) {  
    sum += i;  
  }  
}  
calcSum(10);  
console.log(sum);  
(... 생략 ...)
```

블록 변수

- 재할당은 가능하지만 재선언은 할 수 없다
- 호이스팅이 없다

const를 사용한 변수의 특징

- 상수 – 변하지 않는 값을 선언할 때 사용
- 재선언, 재할당할 수 없음

자바스크립트 변수, 이렇게 사용하자

- 전역 변수는 최소한으로 사용
- var 변수는 함수의 시작 부분에서 선언
- for 문에서 카운터 변수는 var보다 let 변수로 선언
- ES6를 사용한다면 var보다 let를 사용하는 것이 좋다

| 예약어 | 선언하지 않고 사용할 경우 | 재선언 | 재할당 |
|-------|----------------|-----|-----|
| var | 오류 없음(호이스팅 발생) | ○ | ○ |
| let | 오류 발생 | × | ○ |
| const | 오류 발생 | × | × |

- typeof()
- 자료형
 - number
 - string
 - Boolean
 - undefined , null
- 심볼
 - ES6에 새롭게 추가된 원시 유형의 자료형
 - 심볼의 가장 큰 특징은 유일성을 보장한다는 것
 - 심볼은 객체 프로퍼티의 키로 사용할 수 있다

자료형 변환 -1

- 자바스크립트는 다른 언어와 다르게 프로그램 실행 중에 자료형이 변환되는 언어
- 자동으로 형이 변환될 때에도 있다 → 이런 상황을 미리 알아 두지 않으면 오류를 발생시키기도 하고, 처음에 예상했던 것과 다른 결과가 나올 수도 있습니다.

C 언어나 자바 등 일반 프로그래밍 언어

- 변수를 선언할 때 변수의 자료형을 결정
- 자료형에 맞는 값만 변수에 저장 가능
- 자료형으로 인한 프로그램의 오류 방지 가능

자바스크립트

- 변수를 선언할 때 자료형 지정하지 않음
- 변수에 값을 저장할 때 자료형 결정
- 편리하긴 하지만 변수를 일관성 있게 유지하기 힘들다

```
one = "20"    // 문자열  
two = 10      // 숫자형
```

자료형 변환 -2

- 변수에 값을 저장할 때 자료형이 결정되기도 하지만
- 연산을 할 때 자료형이 자동으로 변환된다. 주의해야 함!
- 문자열을 사칙 연산에 사용하면 자동으로 숫자형으로 변환됨
- 숫자와 문자열을 연결하면 숫자가 문자열로 변환됨
- 숫자형 변경하기...
 - 문자열 뿐만 아니라 null과 undefined를 포함해서 모든 자료형을 숫자로 변환할 수 있다
 - parseInt(), parseFloat()....
 - Number()

| 기존 유형 | 변환 결과 |
|-----------|--------------------|
| true | 1 |
| false | 0 |
| 숫자 | 숫자 |
| null | 0 |
| undefined | NaN |
| 정수 문자열 | 정수(맨 앞에 0이 있으면 제거) |
| 실수 문자열 | 실수(맨 앞에 0이 있으면 제거) |
| 16진수 문자열 | 10진수 |
| 빈 문자열 | 0 |
| 위 상황 외 | NaN |

자료형 변환 -3

- 문자열로 변환
 - toString()

자바스크립트 변수 사용법

- 전역 변수를 최소화
- var 변수는 함수의 시작 부분에서 선언
- for문에서 카운터 변수를 사용할 때는 var -> let....
- ES6 var -> let

4. 재사용할 수 있는 함수 만들기

매개 변수와 인수, return 문

- 매개 변수 : 하나의 함수를 여러 번 실행할 수 있도록 실행할 때마다 바뀌는 값을 변수로 처리한 것
- 인수 : 함수를 실행할 때 매개 변수 자리에 넘겨주는 값

Do it! 매개변수를 사용한 함수 선언하고 호출하기

```
(... 생략 ...)  
function addNumber(num1, num2) { ❶  
    var sum = num1 + num2; ❷  
    return sum; ❸  
}  
var result = addNumber(2, 3); // 함수 호출 ❹  
document.write("두 수를 더한 값: " + result); ❺  
(... 생략 ...)
```

- ❶ 자바스크립트 해석기가 function이라는 예약어를 만나면 함수를 선언하는 부분이라는 걸 인식하고 함수 블록({ })을 해석합니다. 아직 실행하지 않습니다.
- ❷ addNumber(2, 3)을 만나면 해석해 두었던 addNumber() 함수를 실행합니다.
- ❸ addNumber() 함수에서 2는 num1로, 3은 num2로 넘기고 더한 값을 sum 변수에 저장합니다.
- ❹ 함수 실행이 모두 끝나면 결과값 sum을 함수 호출 위치, 즉 var result로 넘깁니다.
- ❺ 넘겨받은 결과값을 result라는 변수에 저장합니다.
- ❻ result 변수에 있는 값을 화면에 표시합니다.

함수 표현식

익명 함수

- 함수 이름이 없는 함수
- 함수 자체가 식이므로 함수를 변수에 할당할 수도 있고 다른 함수의 매개변수로 사용할 수도 있음



Do it! 익명 함수 실행하기

```
(... 생략 ...)  
var sum = f (a, b) {  
    return a + b;  
}  
document.write("함수 실행 결과: " + sum(10, 20) );  
(... 생략 ...)
```

즉시 실행 함수

- 함수를 실행하는 순간 자바스크립트 해석기에서 함수를 해석함
- 식 형태로 선언하기 때문에 함수 선언 끝에 세미콜론(;) 붙임

기본형 (function() {
명령
})();

또는

기본형 (function(매개변수) {
명령
(인수));



Do it! 매개변수가 있는 즉시 실행 함수 만들기

```
(... 생략 ...)  
<script>  
    (function(a, b) {  
        sum = a + b;  
    })(100, 200);  
    document.write("함수 실행 결과: " + sum);  
</script>  
(... 생략 ...)
```

매개변수
인수

함수 표현식

화살표 함수

- ES6 이후 사용하는 => 표기법
- 익명 함수에서만 사용할 수 있음

기본형 (매개변수) => { 함수 내용 }

```
const hi = function() {  
  return alert("안녕하세요?");  
}
```



```
const hi = () => { return alert("안녕하세요");};
```

return 생략해서



```
const hi = () => alert("안녕하세요");
```

```
let hi = function(user) {  
  document.write (user + "님, 안녕하세요?");  
}
```



```
let hi = user => { document.write (user + "님, 안녕하세요?"); }
```

```
let sum = function(a, b) {  
  return a + b;  
}
```



```
let sum = (a, b) => a + b;
```

콜백 함수

콜백 함수는 다른 함수의 인자로 사용하는 함수

함수 이름을 사용해 콜백 함수 실행하기

addEventListener() 함수는 아직 배우지 않았지만, addEventListener() 함수 안에 display() 함수를 인자로 사용한다는 점만 알아두자.

이 때 display 뒤에 괄호가 없다는 점 기억하기!

```
const btn = document.querySelector("button");    // 버튼 요소 가져옴

function display() {
  alert("클릭했습니다.");
}

btn.addEventListener("click", display);          // 버튼 클릭하면 display 함수 실행
```


이벤트와 이벤트 처리기

이벤트

- 웹 브라우저나 사용자가 행하는 동작
- 웹 문서 영역안에서 이루어지는 동작만 가리킴
- 주로 마우스나 키보드를 사용할 때, 웹 문서를 불러올 때, 폼에 내용을 입력할 때 발생

표 15-1 마우스 이벤트

| 종류 | 설명 |
|-----------|---|
| click | 사용자가 HTML 요소를 클릭할 때 이벤트가 발생합니다. |
| dblclick | 사용자가 HTML 요소를 더블클릭할 때 이벤트가 발생합니다. |
| mousedown | 사용자가 요소 위에서 마우스 버튼을 눌렀을 때 이벤트가 발생합니다. |
| mousemove | 사용자가 요소 위에서 마우스 포인터를 움직일 때 이벤트가 발생합니다. |
| mouseover | 마우스 포인터가 요소 위로 옮겨질 때 이벤트가 발생합니다. |
| mouseout | 마우스 포인터가 요소를 벗어날 때 이벤트가 발생합니다. |
| mouseup | 사용자가 요소 위에 놓인 마우스 버튼에서 손을 뗄 때 이벤트가 발생합니다. |

표 15-2 키보드 이벤트

| 종류 | 설명 |
|----------|-----------------------------|
| keydown | 사용자가 키를 누르는 동안 이벤트가 발생합니다. |
| keypress | 사용자가 키를 눌렀을 때 이벤트가 발생합니다. |
| keyup | 사용자가 키에서 손을 뗄 때 이벤트가 발생합니다. |

표 15-3 문서 로딩 이벤트

| 종류 | 설명 |
|--------|---|
| abort | 문서가 완전히 로딩되기 전에 불러오기를 멈췄을 때 이벤트가 발생합니다. |
| error | 문서 가 정확히 로딩되지 않았을 때 이벤트가 발생합니다. |
| load | 문서 로딩이 끝나면 이벤트가 발생합니다. |
| resize | 문서 화면 크기가 바뀌었을 때 이벤트가 발생합니다. |
| scroll | 문서 화면이 스크롤되었을 때 이벤트가 발생합니다. |
| unload | 문서에서 벗어날 때 이벤트가 발생합니다. |

표 15-4 폼 이벤트

| 종류 | 설명 |
|--------|--|
| blur | 폼 요소에 포커스를 잃었을 때 이벤트가 발생합니다. |
| change | 목록이나 체크 상태 등이 변경되면 이벤트가 발생합니다. <input>, <select>, <textarea> 태그에서 사용합니다. |
| focus | 폼 요소에 포커스가 놓였을 때 이벤트가 발생합니다. <label>, <select>, <textarea>, <button> 태그에서 사용합니다. |
| reset | 폼이 리셋되었을 때 이벤트가 발생합니다. |
| submit | submit 버튼을 클릭했을 때 이벤트가 발생합니다. |

이벤트와 이벤트 처리기

이벤트 처리기

- 이벤트가 발생했을 때 처리하는 함수
- 이벤트 핸들러(event handler)라고도 함



- 이벤트가 발생한 HTML 태그에 이벤트 처리기를 직접 연결

기본형 <태그 on이벤트명 = "함수명">

Do it! 버튼을 클릭하면 알림 창 표시하기 예제 파일 15\event-1.html

```
(... 생략 ...)  
<body>  
  <ul>  
    <li><a href="#" onclick="alert('버튼을 클릭했습니다.')">Green</a></li>  
    <li><a href="#" onclick="alert('버튼을 클릭했습니다.')">Orange</a></li>  
    <li><a href="#" onclick="alert('버튼을 클릭했습니다.')">Purple</a></li>  
  </ul>  
(... 생략 ...)
```

The screenshot shows the web application in two states. On the left, three buttons labeled 'Green', 'Orange', and 'Purple' are displayed. The 'Green' button is highlighted with an orange border. On the right, an alert dialog box is shown with the text '127.0.0.1:5500 내준 버튼을 클릭했습니다.' (Received click on the button at 127.0.0.1:5500). The dialog has a '확인' (OK) button.


실습

- 버튼을 클릭해서 상세 설명 표시하거나 닫기
 - P. 544

DOM을 이용한 이벤트 처리기

DOM을 사용하면 자바스크립트가 주인이 되어 HTML의 요소를 가져와서 이벤트 처리기를 연결

기본형 웹 요소.onclick = 함수;

 **Do it!** 버튼 클릭해서 글자색 바꾸기 예제 파일 15\event-3.html

(... 생략 ...)

```
<body>
  <button id="change">글자색 바꾸기</button>
  <p>Reprehenderit ..... laborum quis.</p>
```

방법 1

방법 3

방법 2

// 방법 1: 웹 요소를 변수로 지정 & 미리 만든 함수 사용

```
var changeBtn = document.querySelector("#change");
changeBtn.onclick = changeColor;
```

```
function changeColor() {
  document.querySelector("p").style.color = "#f00";
}
```

함수 이름 뒤에 괄호가 없다는 것을 기억하세요!

// 방법 3: 함수를 직접 선언

```
document.querySelector("#change").onclick = function() {
  document.querySelector("p").style.color = "#f00";
};
```

// 방법 2: 웹 요소를 따로 변수로 만들지 않고 사용


```
document.querySelector("#change").onclick = changeColor;
```

```
function changeColor() {
  document.querySelector("p").style.color = "#f00";
}
```

실습-모달박스

- 모달 박스 modal box 란
 - 화면에 내용이 팝업 되면서 기타 내용은 블러 처리 되어 팝업된 내용에만 집중할 수 있게 해 주는 창

01 06\modal.html 문서에는 <div id="modal-box">~</div> 사이에 모달 박스에 표시할 내용이 미리 준비되어 있습니다. 웹 브라우저 창에서 이 문서를 확인해 보면 버튼과 프로필 내용이 함께 표시되어 있는데, 버튼을 클릭할 때마다 프

 모달 박스에 들어갈 내용은 웹 문서에 함께 들어 있어야 합니다.

로필을 보여 주거나 감추는 문서로 바뀌 보겠습니다.



실습-모달박스

02 먼저 프로필 내용을 모달 박스 형태로 만들어야 합니다. 06\css\modal.css 파일을 열고 다음의 소스를 추가하세요. 소스 위치는 button 스타일 다음이 좋습니다.

```
06\css\modal.css

button {
  :
}

#modal-box {
  position:fixed;
  top:0;
  left:0;
  bottom:0;
  right:0;
  background-color: rgba(0,0,0,0.6);
  display:flex;
  justify-content: center;
  align-items: center;
}
```

모달 박스의 위치 고정하기

화면의 가운데에 모달 박스 배치하기

실습-모달박스

03 CSS 파일을 저장한 후 웹 브라우저 창에서 06\modal.html 문서를 다시 한번 확인해 보면 모달 박스 형태로 표시될 것입니다.



실습-모달박스

04 모달 박스가 제대로 만들어졌나요? 그렇다면 처음에는 화면에서 감추었다가 버튼을 클릭할 때 표시해 보겠습니다. 앞에서 설명했던 `classList.toggle()`을 사용하기 위해 `#modal-box` 스타일에서 `display` 속성을 `none`으로 바꿉니다. 그러면 처음에 웹 문서를 불러올 때 모달 박스는 화면에 보이지 않습니다. 이렇게 감춘 모달 박스는 버튼을 클릭했을 때 화면에 표시해야 하므로 화면에 보여 주는 스타일을 따로 만들어야 합니다. 여기서는 이것을 `.active`라는 새로운 클래스 스타일로 만듭니다.

```
06\css\modal.css

button {
  :
}

#modal-box {
  position:fixed;
  top:0;
  left:0;
  bottom:0;
  right:0;
  background-color: rgba(0,0,0,0.6);
  display:none;
  justify-content: center;
  align-items: center;
}

#modal-box.active {
  display:flex;
}
```


실습-모달박스

05 이제 06\js 폴더에 새로운 스크립트 파일을 만들고 웹 문서에 연결하는 것은 간단하죠?
06\js 폴더에 modal.js 파일을 만들고 06\modal.html 문서에 스크립트 파일을 연결합니다.

```
06\modal.html

<body>
  <button id="open">프로필 보기</button>

  <div id="modal-box">
    ⋮
  </div>

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

실습-모달박스

06 [프로필 보기] 버튼을 가져와서 open이라는 변수에 저장하고 모달 박스는 modalBox 변수로 저장합니다. 모달 박스에서 [닫기] 버튼을 클릭했을 때 모달 박스가 사라져야 하므로 모달 박스의 [닫기] 버튼도 close 변수로 저장합니다. [열기] 버튼을 클릭하면 .active 스타일이 추가되었다가 [닫기] 버튼을 클릭하면 .active가 삭제되도록 지정하면 됩니다. 06\js\modal.js 파일에 다음의 소스를 작성하고 저장합니다.

06\js\modal.js

```
const open = document.querySelector("#open");
const modalBox = document.querySelector("#modal-box");
const close = document.querySelector("#close");

open.addEventListener("click", () => {
  modalBox.classList.toggle("active"); // 클릭하면 .active 스타일을 토글합니다.
});
```

실습-모달박스

07 웹 브라우저 창에서 06\modal.html 문서를 확인해 보세요. [프로필 보기] 버튼을 클릭하면 모달 박스가 나타나고, 모달 박스의 [닫기] 버튼(✕)을 클릭하면 모달 박스가 사라질 것입니다.

