

# 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



# 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("결괏값: " + sum);  
}  
  
addNumber();  
addNumber();  
</script>  
(... 생략 ...)
```

함수를 선언

함수를 2번 호출

127.0.0.1:5500 내용:  
결괏값: 5

확인

127.0.0.1:5500 내용:  
결괏값: 5

확인



알림 창에서  
결괏값이 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

uogcunj:를윤

## 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

JEBA:를윤

# var를 사용한 변수의 특징

## var 변수와 호이스팅

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

(... 생략 ...)
<script>
  var x = 10;

  function displayNumber() {
    console.log("x is " + x);
    console.log("y is " + y);
    var y = 20;
  }
  displayNumber();
</script>
(... 생략 ...)
```

### 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

(... 생략 ...)
<script>
  f 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);
  v sum = 100; // sum 변수 재선언
  console.log(sum);
</script>
(... 생략 ...)
```

	Elements	Console	Sources	>>	⋮	×
		top			Filter	Default lev
30						
50						
100						
>						

# 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

# 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
```

```
document.write("두 수를 더한 값: " + result); ❹
```

(... 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 (a, b)  
인수 (100, 200)

# 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;
```



# 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.

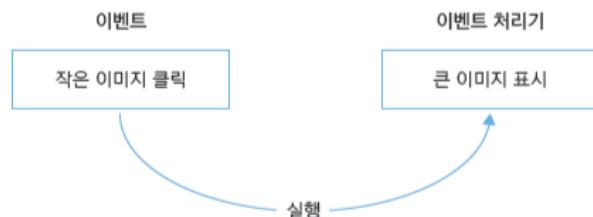
표 15-4 폼 이벤트

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>

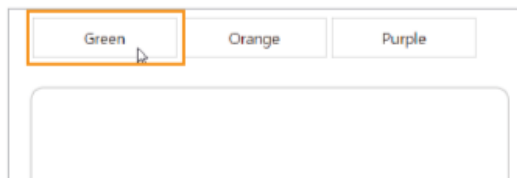
<li><a href="#" onclick="alert('버튼을 클릭했습니다.')">Green</a></li>

<li><a href="#" onclick="alert('버튼을 클릭했습니다.')">Orange</a></li>

<li><a href="#" onclick="alert('버튼을 클릭했습니다.')">Purple</a></li>

</ul>

(... 생략 ...)



# 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;
```

함수 이름 뒤에 괄호가 없다는 것을 기억하세요!

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

// 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";
}
```