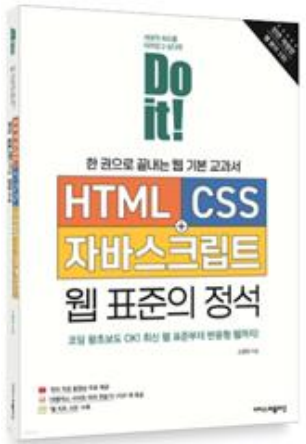
17. 함수와 이벤트



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

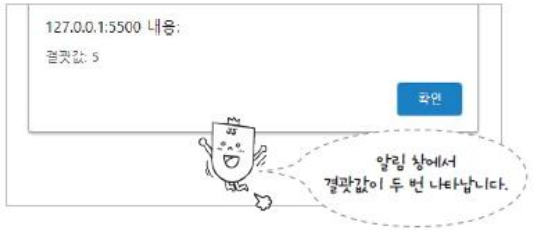
```
기본형  function 함수명() {
            명령
        }
```

함수 호출 : 선언한 함수를 사용하는 것

```
기본형  함수명() 또는 함수명(변수)
```

(예) 두 수를 더하는 함수 만들고 실행하기

```
<script>
  function addNumber() {
    let num1 = 2;
    let num2 = 3;
    let sum = num1 + num2;
    alert(`결괏값: ${sum}`);
  }
  addNumber();
  addNumber();
</script>
```



변수 스코프

스코프 : 변수가 적용되는 범위

var 예약어를 사용한 변수 ← 함수 스코프를 가진다

지역 변수

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

```
<script>
  var sum = 0;      // 전역 변수 선언
  function addNumber() {
    var result;     // 지역 변수 선언
    sum = 10 + 20;
    result = 10 * 20;
  }

  addNumber();
  console.log(sum);
  console.log(result);
</script>
```

오류 발생

전역 변수

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

```
<script>
  var sum = 0;    // 전역 변수 선언
  function addNumber() {
    sum = 10 + 20;
    result = 10 * 20;
  }
  addNumber();
  console.log(sum);
  console.log(result);
</script>
```

실수로 var 빼먹으면 전역 변수가 됨

var를 사용한 변수의 특징

var 변수와 호이스팅

```
<script>
  var x = 10;   // 전역 변수 선언
  function displayNumber() {
    console.log(`x is ${x}`);
    console.log(`y is ${y}`);
    var y = 20;   // 지역 변수 선언 및 값 할당
  }
displayNumber();
</script>
```

y 변수 값이 undefined가 됨

호이스팅

- 변수를 뒤에서 선언하지만, 마치 앞에서 미리 앞에서 선언한 것처럼 인식함

→ undefined 값으로 인해 엉뚱한 결과 발생

→ 아예 오류를 발생하는 것이 안전할 수 있음.

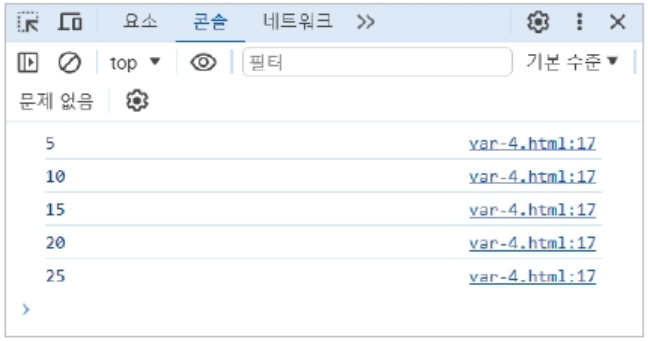
- 함수 실행문을 앞에 두고 선언 부분을 뒤에 두더라도 앞으로 끌어올려 인식함

재선언과 재할당이 가능하다

- 재선언 : 이미 선언한 변수를 다시 선언할 수 있음
- 재할당 : 같은 변수에 다른 값을 할당할 수 있음

→ 재선언이 가능하면 실수로 변수를 잘못 조작할 확률이 높아짐

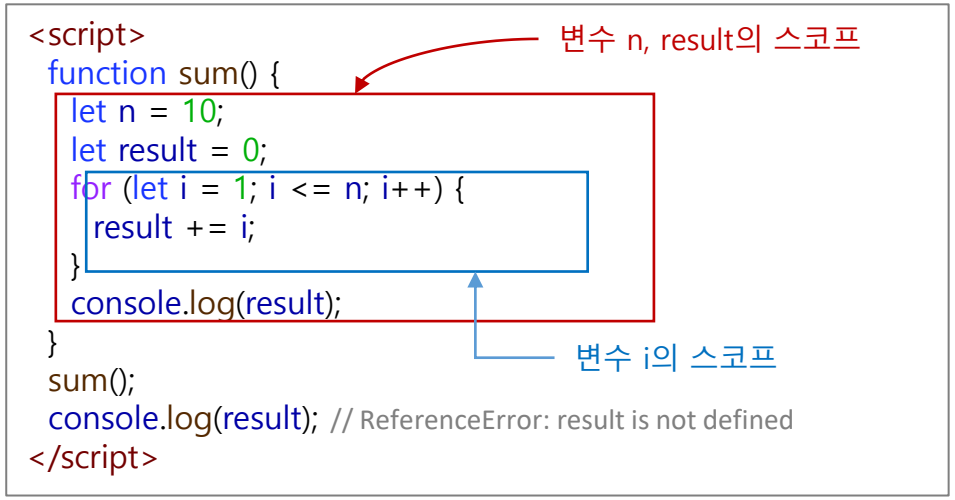
```
<script>
  var seed = 3; // 3의 배수를 만들자

  for (let i = 1; i <= 5; i++) {
    var seed = 5;  // 재선언
    var result = seed * i;;
    console.log(result);
  }
</script>
```



let과 const의 등장

let을 사용한 변수의 특징

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




```
<script>
  function sum() {
    let n = 10;
    let result = 0;
    for (let i = 1; i <= n; i++) {
      result += i;
    }
    console.log(result);
  }
  sum();
  console.log(result); // ReferenceError: result is not defined
</script>
```

- **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 변수를 써야 한다면 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

매개변수와 인수를 합쳐서 '인자 ' 라고도 부름

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

```
<script>
  function addNumber(num1, num2){ ❶
    let sum = num1 + num2; ❸
    return sum; ❹
  }
  let result = addNumber(2, 3);
  document.write(`두 수를 더 한 값 : ${result}`); ❻
</script>
```

❺ ❷

두 수를 더한 값: 5

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 결괏값 sum을 함수 호출 위치, 즉 var result로 넘깁니다.
❺ 넘겨받은 결괏값을 result라는 변수에 저장합니다.
❻ result 변수에 있는 값을 화면에 표시합니다.

재사용할 수 있는 함수 만들기

매개 변수 기본값 지정하기

ES6부터는 매개변수에 기본값을 지정할 수 있다.

함수를 실행할 때 변숫값을 넘겨 받지 못하면 기본값을 사용한다.

```
function multiple(a, b = 5, c = 10) { // b = 5, c = 10으로 기본값 지정
  return a * b + c;
}
```

```
multiple(5, 10, 20); // a = 5, b = 10, c = 20
multiple(10, 20); // a = 10, b = 20, c = 10(기본값)
multiple(30); // a = 30, b = 5(기본값), c = 10(기본값)
```

함수 표현식

익명 함수

- 함수 이름이 없는 함수
- 함수 자체가 식이므로 함수를 변수에 할당할 수도 있고 다른 함수의 매개변수로 사용할 수도 있음
- 변수에 할당된 익명 함수는 변수를 이용해 함수 실행

```
<script>
  let sum = function(a, b) {
    return a + b;
  }
  document.write(`함수 실행 결과 : ${sum(10, 20)}`);
</script>
```

함수 실행 결과 : 30

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);
```

```
<script>
  (function() {
    let userName = prompt("이름을 입력하세요.");
    document.write(`<p>안녕하세요? <span class="accent">${userName}</span>
님!</p>`);
  })();
</script>
```

```
<script>
  (function(a, b){   // 함수 선언을 위한 매개변수
     sum = a + b;
  }(100, 200));      // 마지막에 함수 실행을 위한 인수
  document.write(`함수 실행 결과 : ${sum}`);
  </script>
```

함수 표현식

화살표 함수

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

기본형 (매개변수) => { 함수 내용 }

매개변수 없을 경우

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

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

return 생략해서 

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

매개변수가 있을 경우

```
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 17-2 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 17-3 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 17-1 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.

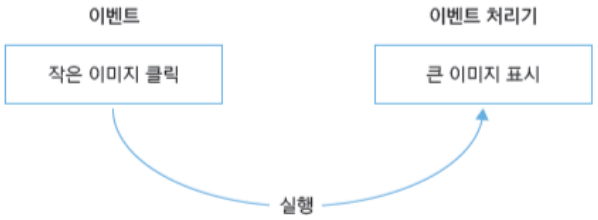
표 17-4 폼 이벤트

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">

(예) 버튼 클릭하면 알림 창 표시하기

```
<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>
```

(예) 버튼 클릭하면 배경색 바꾸기

```
<body>
  <ul>
    <li><a href="#" onclick="changeBg('green')">Green</a></li>
    <li><a href="#" onclick="changeBg('orange')">Orange</a></li>
    <li><a href="#" o ="changeBg('purple')">Purple</a></li>
  </ul>
  <div id="result"></div>

  <script>
    function changeBg(color) {
      let result = document.querySelector('#result');
      result.style.backgroundColor = color;
    }
  </script>
</body>
```