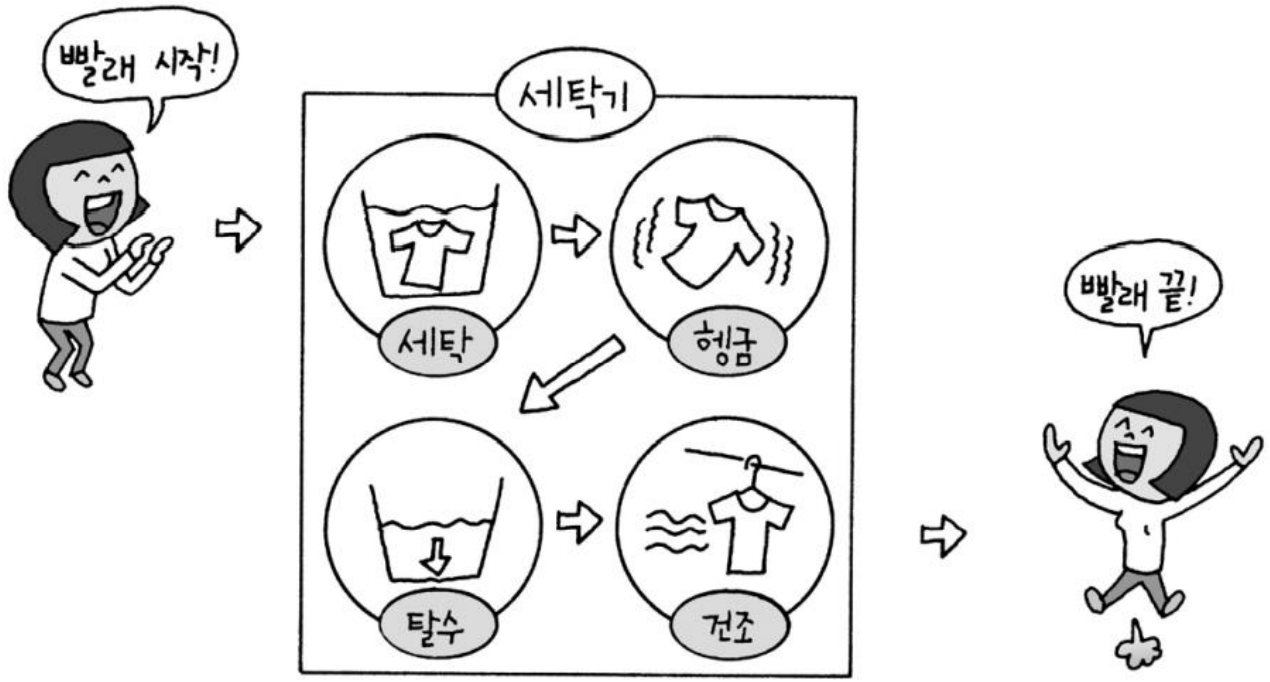
05. 함수와 이벤트

여러 동작을 묶은 덩어리, 함수



여러 동작을 묶은 덩어리, 함수

함수(function)를 사용하면 무엇이 좋을까?

- 각 명령의 시작과 끝을 명확하게 구별할 수 있다.
- 함수에 별도의 이름을 붙이면 같은 기능이 필요할 때마다 해당 함수를 실행할 수 있다.

함수 선언 (함수 정의)

- 함수가 어떤 명령을 처리해야 할지 미리 알려주는 것
- function 예약어를 사용하고, { } 안에 실행할 명령을 작성

```
> function addNumber( ) {
    var sum = 10 + 20;
    console.log(sum);
  }
```

함수 호출 (함수 실행)

- 함수 이름을 사용해 함수 실행

```
> addNumber( )
  30
```

함수 선언은 어디에 두어야 할까

함수를 선언해 놓기만 하면, 선언한 위치와 상관없이 함수를 실행할 수 있습니다

```
addNumber( );

function addNumber( ) {
    ……
}
```

함수 호출 소스를 선언 소스보다 앞에 작성한 예

```
function addNumber( ) {
    ……
}

addNumber( );
```

함수 호출 소스를 선언 소스 뒤에 작성한 예

let과 const

- let, const – ES6 버전 이후에 변수를 선언하는 예약어
- var가 있는데 왜 let과 const라는 예약어가 필요할까?

var 변수의 스코프(scope)

- 변수 이름 앞에 var를 붙이면 지역 변수 → 함수 안에서만 사용할 수 있는 변수
- var가 없으면 전역 변수 → 프로그램 전체에서 사용할 수 있는 변수

  → 실수로 var를 빼먹었다면?

let과 const

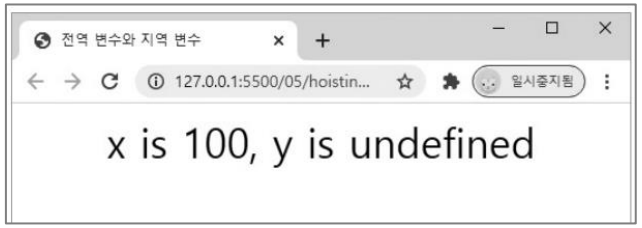
var 변수의 호이스팅

변수를 선언하기 전에 변수를 사용하면?

→ 오류가 생기지 않는다

→ 변수 선언이 앞에 있는 것처럼 끌어올려(hoisting) 인식한다.

```
1   var x = 100;
2
3   test();
4
5   function test() {
6      document.write("x is " + x + ", y is " + y);
7      var y = 200;
8   }
```



let과 const

var 변수의 재선언

이미 있는 변수를 다시 선언할 수 있다

→ 실수로 서로 다른 위치에서 같은 변수를 선언할 수 있다는 것은 문제

→ 재선언하면 이전 변수를 덮어쓰기 때문에 예상하지 못한 오류가 생길 수 있음.

let과 const

- let – 프로그램 안에서 값이 변하는 변수
- const – 프로그램 안에서 값이 변하지 않는 변수

변수의 스코프

let 변수와 const 변수는 블록 영역의 스코프

호이스팅 없음

변수를 선언하지 않고 사용하면 오류 발생

변수의 재선언 불가

같은 변수를 다시 선언하면 오류 발생

```
1   function calcSum(n) {
2       let sum = 0;   // 블록 변수 선언
3       for(let i = 1; i < n + 1; i++) {
4           sum += i;
5       }
6       console.log(sum);   // 블록 변수 사용
7   }
8
9   calcSum(10);
```

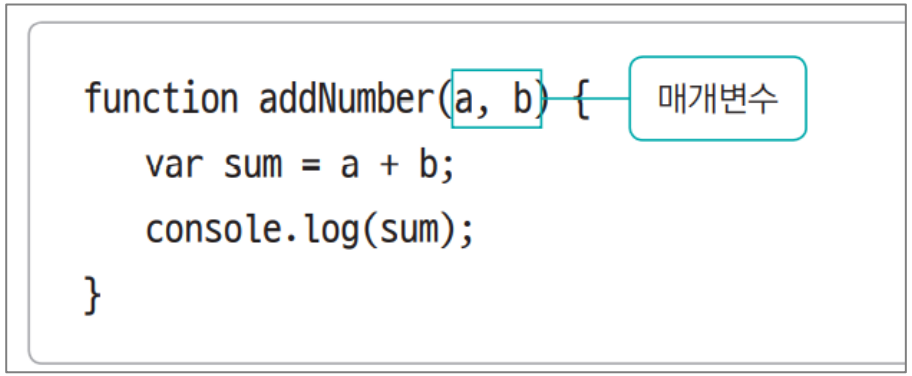
변수는 이렇게 사용하세요

- 전역 변수는 최소한으로 사용한다
- var 변수는 함수의 시작 부분에서 선언한다 (호이스팅 방지)
- for문의 카운터 변수는 블록 변수(let)를 사용하는게 좋다
- ES6를 사용한 프로그램이라면 var보다 let을 사용한다

여러 번 사용할 수 있는 함수 만들기

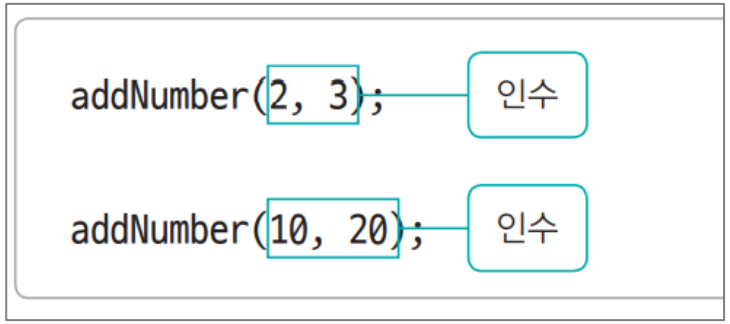
매개변수(parameter)

- 함수를 실행하기 위해 필요하다고 지정하는 값
- 함수를 선언할 때 함수 이름 옆의 괄호 안에 매개변수 이름을 넣음



인수(argument)

- 함수를 실행하기 위해 필요하다고 지정하는 값
- 함수를 실행할 때 매개변수로 넘겨주는 값



여러 번 사용할 수 있는 함수 만들기

return 문

- 함수를 실행한 결괏값을 함수 밖으로 넘기는 문
- 반환된 값은 함수를 실행한 소스 위치로 넘겨짐.

```
1   var num1 = parseInt(prompt("첫 번째 숫자는? "));
2   var num2 = parseInt(prompt("두 번째 숫자는? "));
3   var result = addNumber(num1, num2);
4   alert("두 수를 더한 값은 " + result + "입니다.");
5
6   function addNumber(a, b) {
7       var sum = a + b;
8       return sum;
9   }
```

❶ num1 변수와 num2 변수에 값을 저장합니다.
❷ num1과 num2 값을 가지고 addNumber( ) 함수를 호출합니다.
❸ 함수 선언부로 넘어와 함수를 실행하는데,
num1 값은 a 변수로, num2 값은 b 변수로 넘겨집니다.
❹ a 값과 b 값을 더해 sum 변수에 저장합니다.
❺ 결괏값 sum을 반환합니다.
❻ 반환된 값을 변수 result에 저장합니다.
❼ result 변숫값을 화면에 표시합니다.

함수 표현식

익명 함수

- 이름이 없는 함수
- 함수 자체가 '식(Expression)'이기 때문에 함수를 변수에 할당하거나 다른 함수의 매개변수로 사용할 수도 있음.

```
> var add = function(a, b) {    //함수 선언 후 변수 add에 할당
    return a + b;
  }
```

```
> var sum = add(10, 20);   //익명 함수 실행 후 결괏값을 변수 sum에 저장
> sum   //변수 sum 값 확인
< 30
```

함수 표현식

즉시 실행 함수

- 함수를 정의함과 동시에 실행하는 함수
- 즉시 실행 함수는 변수에 할당할 수 있고, 함수에서 반환하는 값을 변수에 할당할 수도 있음.

```
(function( ) {
    ......
 })( );
```

또는

```
(function( ) {
    ......
 }( )) ;
```

함수 표현식

즉시 실행 함수

매개변수와 인수를 사용한다면?

→ 매개변수는 function() 괄호 안에, 인수는 함수 끝에 있는 괄호 안에.

```
> var result = (function( ) {
     return 10 + 20;
  }( ));
> console.log(result);
< 30
```

```
> var result = (function(a, b) {      //매개변수 추가
     return a + b;
  } (10, 20));     //인수 추가
> console.log(result);
< 30
```

화살표 함수

ES6 이후 버전에는 => 표기법을 사용해 함수 선언을 간단하게 작성

매개변수가 없을 때

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

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

```
const hi = () => "안녕하세요? " ;
```

화살표 함수

매개변수가 있을 때

```
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) => { return a + b }
```

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

[미리보기] 버튼 클릭해서 상세 설명 열고 닫기

이벤트

이벤트

- 웹 브라우저나 사용자가 행하는 동작
- 사용자가 웹 문서 영역을 벗어나 하는 동작은 이벤트가 아님

이벤트 처리기

- 이벤트가 발생했을 때 어떤 함수를 실행할지 알려줌
- 태그 안에서 이벤트를 처리할 때는 "on"+"이벤트명" 사용 (예, 클릭하면 onclick 사용)

이벤트

마우스 이벤트

속성	설명
click	사용자가 HTML 요소를 마우스로 눌렀을 때 이벤트가 발생합니다.
dblclick	사용자가 HTML 요소를 마우스로 두 번 눌렀을 때 이벤트가 발생합니다.
mousedown	사용자가 요소 위에서 마우스 버튼을 누르는 동안 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 누르고 있던 마우스 버튼에서 손을 뗄 때 이벤트가 발생합니다.

폼 이벤트

속성	설명
blur	폼 요소에 포커스를 잃었을 때 이벤트가 발생합니다.
change	목록이나 체크 상태 등이 변경되었을 때 이벤트가 발생합니다(<input>, <select>, <textarea> 태그에서 사용합니다).
focus	폼 요소에 포커스가 놓였을 때 이벤트가 발생합니다(<label>, <select>, <textarea>, <button> 태그에서 사용합니다).
reset	폼이 다시 시작되었을 때 이벤트가 발생합니다.
submit	submit 버튼을 눌렀을 때 이벤트가 발생합니다.

키보드 이벤트

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

문서 로딩 이벤트

속성	설명
abort	웹 문서가 완전히 로딩되기 전에 불러오기를 멈췄을 때 이벤트가 발생합니다.
error	문서가 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	문서를 벗어날 때 이벤트가 발생합니다.

버튼 클릭해서 상세 설명 열고 닫기

실습 파일 :
05₩event.html, 05₩js₩event.js

상세 설명 여는 함수 & 닫는 함수 (05₩js₩event.js)

```
1   function showDetail() {
2       document.querySelector('#desc').style.display = "block"; //상세 설명 내용을 화면에 표시
3       document.querySelector('#open').style.display = "none"; //[상세 설명 보기] 버튼 감춤
4   }
5
6   function hideDetail() {
7       document.querySelector('#desc').style.display = "none"; //상세 설명 내용을 화면에서 감춤
8       document.querySelector('#open').style.display = "block";   //[상세 설명 보기] 버튼 표시
9   }
```

버튼 클릭해서 상세 설명 열고 닫기

버튼 클릭했을 때 함수 실행하기 (05₩event.html)

```
11  <div id="item">
12      <img src="images/flower1.jpg" alt="">
13      <button class="over" id="open" onclick="showDetail()">상세 설명 보기</button>
14      <div id="desc" class="detail">
15          <h4>민들레</h4>
16          <p>어디서나 매우 흔하게 보이는 …… 널리 퍼진다.</p>
17          <button id="close" onclick="hideDetail()">상세 설명 닫기</button>
18      </div>
```

❸ 함수 실행

❶ 버튼에서 click 이벤트가 발생하면

❷ click 이벤트 처리를 찾아