10주차 JavaScript

상태	완료 (승인 X)
설명	함수

함수

코드를 작성하다 보면 유사한 동작을 하는 코드가 여러 곳에서 필요할 때가 많은데 함수를 활용하면 중복되는 코드들을 스크립트 어디에서든지 호출하여 사용할 수 있다.

우리는 앞서 다양한 예시에서 `alert(message)`, `prompt(message, default)`, `confirm(question)`과 같은 내장 함수를 사용해 보았는데, 이번엔 위와같은 함수들을 직접 만들어 보는 방법에 대해 배워보자.

함수 선언

함수는 아래와 같은 방법으로 만들어낼 수 있다.

```
function showMessage() {
  alert( '안녕하세요!' );
}
```

만약 함수에 매개변수(인자) 가 있다면 아래와 같이 작성할 수 있다.

```
function name(parameter1, parameter2, ... parameterN) {
  // 함수 본문
}
```

새롭게 정의한 함수는 함수 이름 옆에 괄호를 붙여 호출할 수 있다. 즉, 복잡한 로직을 중복해서 사용해야하는 경우에 함수를 활용하면 쉽게 작성할 수 있게 된다.

```
function showMessage() {
  alert( '안녕하세요!' );
}

showMessage();
showMessage();
```

또한 함수를 활용하면 유지보수가 쉬워진다. showMessage의 `안녕하세요!` 를 수정하고 싶을때 함수를 사용하지 않았다면 모든 부분을 다 수정해야 하지만 함수를 활용했다면 함수안의 코드만 변경하면 되기 때문이다.

지역 변수

함수 내에서 선언한 변수인 지역 변수(local variable)는 함수 안에서만 접근할 수 있다.

```
function showMessage() {
  let message = "안녕하세요!"; // 지역 변수alert( message );
}

showMessage(); // 안녕하세요!

alert( message );
// ReferenceError: message is not defined (message는 함수 내 지역 변수이기 때문에 에러가 발생합니다.)
```

💡 **영역 ❓**

함수 내에서 선언한 변수를 확인하려면 해당 함수의 영역을 파악해야 한다. 함수의 영역은 소괄호 `{ }` 로 나타내며 해당 영역이 함수의 영역이 된다.

영역은 함수 뿐 아니라 `if` `for` `while` 문 등에서도 사용되었으며, 마찬가지로 `{ }` 안의 영역은 각 문법의 공간이 된다.

외부 변수

함수 내부에서 함수 외부의 변수인 외부 변수(outer variable)에 접근할 수 있다.

```
let userName = 'John';

function showMessage() {
  let message = 'Hello, ' + userName;
  alert(message);
}

showMessage(); // Hello, John
```

함수에선 외부 변수에 접근하는 것뿐만 아니라, 수정도 할 수 있다. 즉, 함수안에서는 외부 변수를 사용할 수 있지만 함수 외부에선 함수내의 변수를 사용할 수 없다.

```
let userName = 'John';

function showMessage() {
  userName = "Bob"; // (1) 외부 변수를 수정함

  let message = 'Hello, ' + userName;
  alert(message);
}

alert( userName ); // 함수 호출 전이므로 John 이 출력됨

showMessage();

alert( userName ); // 함수에 의해 Bob 으로 값이 바뀜
```

외부 변수는 지역 변수가 없는 경우에만 사용할 수 있으며, 만약에 외부변수와 내부변수의 이름이 같다면 내부변수는 외부변수를 가리고 내부변수만 사용되게 된다.

```
let userName = 'John';

function showMessage() {
  let userName = "Bob";
// 같은 이름을 가진 지역 변수를 선언합니다.let message = 'Hello, ' + userName; // Bobalert(message);
}

// 함수는 내부 변수인 userName만 사용합니다,
showMessage();
```

```
alert( userName ); // 함수는 외부 변수에 접근하지 않습니다. 따라서 값이 변경되지 않고, John이 출력됩니다.
```

매개변수 (인자)

매개변수(parameter)를 이용하면 임의의 데이터를 함수 안에 전달할 수 있으며 매개변수는 *인자(parameter)* 라고 불리기도 한다.

아래 예시에서 함수 showMessage는 매개변수 `from` 과 `text`를 가진다.

```
function showMessage(from, text) { // 인자: from, text
  alert(from + ': ' + text);
}

showMessage('Ann', 'Hello!'); // Ann: Hello! (*)
showMessage('Ann', "What's up?"); // Ann: What's up? (**
```

`(*)`, `(**)`로 표시한 줄에서 함수를 호출하면, 함수에 전달된 인자는 지역변수 `from`과 `text`에 복사되며 그 후 함수는 지역변수에 복사된 값을 사용하게 된다.

전역변수와 매개변수는 같은 이름으로 존재할 수도 있다. 이 경우 마치 동명이인처럼 동작하게 되며, 전역변수의 값이 매개변수로 복사된 후, 위에서 설명했던 방식과 동일하게 동작한다.

```
function showMessage(from, text) {

  from = '*' + from + '*';
// "from"을 좀 더 멋지게 꾸며줍니다.alert( from + ': ' + text );
}

let from = "Ann";

showMessage(from, "Hello"); // *Ann*: Hello

// 함수는 복사된 값을 사용하기 때문에 바깥의 "from"은 값이 변경되지 않습니다.
alert( from ); // Ann
```

인수와 매개변수의 차이점에 대해서 한번 다시 알아보고 가자.

1️⃣ 매개변수는 함수 선언 방식 괄호 사이에 있는 변수(선언 시 쓰이는 용어).

2️⃣ 인수는 함수를 호출할 때 매개변수에 전달되는 값(호출 시 쓰이는 용어).

기본값

함수 호출 시 매개변수에 인수를 전달하지 않으면 그 값은 `undefined`가 된다.

위에서 정의한 함수 `showMessage(from, text)`는 매개변수가 2개지만, 아래와 같이 인수를 하나만 넣어서 호출할 수 있으며 문법적으로 오류가 나지 않는다.

```
showMessage("Ann");
```

두 번째 매개변수에 값을 전달하지 않았기 때문에 `text`엔 `undefined`가 할당되고 따라서 에러 없이 `"Ann: undefined"`가 출력되게 된다.

매개변수에 값을 전달하지 않아도 그 값이 `undefined`가 되지 않게 하려면 함수를 선언할 때 `=`를 사용해 '기본값(default value)'을 설정해주는 방법이 있다.

```
function showMessage(from, text = "no text given") {
  alert( from + ": " + text );
}

showMessage("Ann"); // Ann: no text given
```

이젠 `text`가 값을 전달받지 못해도 `undefined` 대신 기본값 `"no text given"`이 할당되게 된다.

반환 값

함수를 호출했을 때 함수를 호출한 그곳에 특정 값을 반환하게 할 수 있으며 이를 반환 값이라고 한다

인수로 받은 두 값을 더해주는 간단한 함수를 만들어 반환 값에 대해 알아보도록 하자.

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

let result = sum(1, 2);
alert( result ); // 3
```

`return`은 함수 내 어디서든 사용할 수 있으며 실행 흐름이 `return`을 만나면 함수 실행은 즉시 중단되고 함수를 호출한 곳에 값을 반환하게 된다.

아래와 같이 함수 하나에 여러 개의 `return`문이 올 수도 있다. 어떤 `return`이던지 만나게 되면 함수는 바로 종료되게 된다.

```
function checkAge(age) {
  if (age >= 18) {
    return true;} else {
    return confirm('보호자의 동의를 받으셨나요?');}
}

let age = prompt('나이를 알려주세요', 18);

if ( checkAge(age) ) {
  alert( '접속 허용' );
} else {
  alert( '접속 차단' );
}
```

아래와 같이 지시자 `return`만 명시하는 것도 가능하며 함수가 바로 종료되게 된다.

```
function showMovie(age) {
  if ( !checkAge(age) ) {
    return;}

  alert( "영화 상영" ); // (*)
  // ...
}
```

위 예시에서, `checkAge(age)`가 `false`를 반환하면, `(*)`로 표시한 줄은 실행이 안 되기 때문에 함수 `showMovie`는 즉시 종료되게 된다.

> 💡 return 사용 이유 ❓
>
> 이전에 함수 외부에서는 내부의 값을 접근할 수 없다고 했었는데, 내부에서 변경된 값을 외부로 전달해주기 위해서 `return`을 사용한다.
>
> 자바스크립트에서 반환값은 어떤 자료형이든 들어갈 수 있다.

함수 이름짓기

함수는 어떤 `동작`을 수행하기 위한 코드를 모아놓은 것이기에 이름을 동사로 시작하게 짓는것이 좋다. 함수 이름은 가능한 한 간결하고 명확해야 한다. 즉 , 코드를 읽는 사람은 함수 이름만 보고도 함수가 어떤 기능을 하는지 알 수 있어야 한다.

아래 접두어들은 함수를 만들 때 자주 사용되는 어휘들이다.

- `"get…"` – 값을 반환함
- `"calc…"` – 무언가를 계산함
- `"create…"` – 무언가를 생성함
- `"check…"` – 무언가를 확인하고 불린값을 반환함

```
showMessage(..)     // 메시지를 보여줌
getAge(..)          // 나이를 나타내는 값을 얻고 그 값을 반환함
calcSum(..)         // 합계를 계산하고 그 결과를 반환함
createForm(..)      // form을 생성하고 만들어진 form을 반환함
checkPermission(..) // 승인 여부를 확인하고 true나 false를 반환함
```

요약

함수 선언 방식으로 함수를 만들 수 있다.

```
function 함수이름(복수의, 매개변수는, 콤마로, 구분합니다) {
  /* 함수 본문 */
}
```

1️⃣ 함수에 전달된 매개변수는 복사된 후 함수의 지역변수가 된다.

2️⃣ 함수는 외부 변수에 접근할 수 있지만 함수 바깥에서 함수 내부의 지역변수에 접근하는 건 불가능하다.

3️⃣ 함수는 값을 반환할 수 있으며 값을 반환하지 않는 경우는 반환 값이 `undefined`가 된다.

깔끔하고 이해하기 쉬운 코드를 작성하려면 함수 내부에서 외부 변수를 사용하는 방법 대신 지역 변수와 매개변수를 활용하는 게 좋다.

함수 이름을 지을 땐 아래와 같은 규칙을 따르는 것이 좋다.

- 함수 이름은 함수가 어떤 동작을 하는지 설명할 수 있어야 한다. 이렇게 이름을 지으면 함수 호출 코드만 보아도 해당 함수가 무엇을 하고 어떤 값을 반환할지 바로 알 수 있기 때문이다.
- 함수는 동작을 수행하기 때문에 이름이 주로 동사이다.
- `create…`, `show…`, `get…`, `check…` 등의 잘 알려진 접두어를 사용해 이름을 지을 수 있으며 접두어를 사용하면 함수 이름만 보고도 해당 함수가 어떤 동작을 하는지 파악할 수 있어야 한다.

문제 1

아래 함수는 매개변수 `age`가 `18`보다 큰 경우 `true`를 반환합니다.

그 이외의 경우는 컨펌 대화상자를 통해 사용자에게 질문한 후, 해당 결과를 반환합니다.

```
function checkAge(age) {
  if (age > 18) {
    return true;
  } else {
    // ...
    return confirm('보호자의 동의를 받으셨나요?');
  }}
```

위 코드에서 `else`문을 삭제해도 기존 코드와 동일하게 작동할까요?

```
function checkAge(age) {
  if (age > 18) {
    return true;
  }
  // ...
  return confirm('보호자의 동의를 받으셨나요?');}
```

문제 2

아래 함수는 매개변수 `age`가 `18`보다 큰 경우 `true`를 반환합니다.

그 이외의 경우는 컨펌 대화상자를 통해 사용자에게 질문한 후, 해당 결과를 반환합니다.

```
function checkAge(age) {
  if (age > 18) {
    return true;
  } else {
    return confirm('보호자의 동의를 받으셨나요?');
  }
}
```

`if`문을 사용하지 않고 동일한 동작을 하는 함수를 한 줄에 작성해보세요.

아래 조건을 충족하는 해답 2개를 작성해야 합니다.

1. 물음표 연산자 `?`를 사용하여 본문을 작성
2. OR 연산자 `||`를 사용하여 본문을 작성

문제 3

`a`와 `b` 중 작은 값을 반환해주는 함수, `min(a,b)`을 만들어보세요.

만든 함수는 아래와 같이 동작해야 합니다.

```
min(2, 5) == 2
min(3, -1) == -1
min(1, 1) == 1
```

문제 4

`x`의 `n`제곱을 반환해주는 함수, `pow(x,n)`를 만들어보세요. `x`의 `n` 제곱은 `x`를 `n`번 곱해서 만들 수 있습니다.

```
pow(3, 2) = 3 * 3 = 9
pow(3, 3) = 3 * 3 * 3 = 27
pow(1, 100) = 1 * 1 * ...* 1 = 1
```

프롬프트 대화상자를 띄워 사용자로부터 `x`와 `n`을 입력받고 `pow(x,n)`의 반환 값을 보여주는 코드를 작성해 보세요.

주의사항: `n`은 `1` 이상의 자연수이어야 합니다. 이외의 경우엔 자연수를 입력하라는 얼럿 창을 띄워주어야 합니다.

함수 표현식

다른 언어를 배워 본 경험이 있다면 해당 부분을 조금 자세히 읽어보고 만약 이번 교재가 언어를 배우는 처음이라면 조금 어려운 내용을 다루니 가볍게 읽어보도록 하자.

자바스크립트는 함수를 특별한 종류의 값으로 취급하며 다른 언어에서처럼 "특별한 동작을 하는 구조"로 취급되지 않는다. 우리는 방금 함수를 아래와 같은 방식으로 만들었다.

```
function sayHi() {
  alert( "Hello" );
}
```

자바스크립트에서 함수는 값이기에 위와 같은 함수 선언식 방식 외에 *함수 표현식(Function Expression)* 을 사용해서 함수를 만들 수 있다. 즉, 함수를 값처럼 변수에 할당할 수 있다.

```
let sayHi = function() {
  alert( "Hello" );
};.
```

위 예시를 간단한 말로 풀면 다음과 같다.

“함수를 만들고 그 함수를 변수 `sayHi`에 할당하기”

함수는 값이기 때문에 `alert`를 이용하여 함수 코드를 출력할 수도 있다.

```
function sayHi() {
  alert( "Hello" );
}

alert( sayHi ); // 함수 코드가 보임
```

마지막 줄에서 `sayHi`옆에 괄호가 없기 때문에 함수는 실행되지 않게 된다. 괄호가 있어야만 함수가 호출된다는 걸 꼭 기억하자.

변수를 복사해 다른 변수에 할당하는 것처럼 함수를 복사해 다른 변수에 할당하는것도 당연히 가능하다.

```
function sayHi() {   // (1) 함수 생성
  alert( "Hello" );
}

let func = sayHi;    // (2) 함수 복사

func(); // Hello     // (3) 복사한 함수를 실행(정상적으로 실행됩니다)!
sayHi(); // Hello    //     본래 함수도 정상적으로 실행됩니다.
```

함수 `sayHi`는 아래와 같이 함수 표현식을 사용해 정의할 수 있다.

```
let sayHi = function() {
  alert( "Hello" );
};

let func = sayHi;
// ...
```

동작 결과는 동일하다.

함수 표현식 끝의 `;`

함수 표현식의 끝에 왜 세미 콜론 `;`이 붙는지 의문이 들 수 있다. 함수 선언문에는 세미 콜론 이 없다.

```
function sayHi() {
  // ...
}

let sayHi = function() {
  // ...
};
```

이유는 생각보다 간단하다.

`if { ... }`, `for { }`, `function f { }` 같이 중괄호로 만든 코드 블록 끝엔 `;`이 없어도 된다.

함수 표현식은 `let sayHi = ...;`과 같은 구문 안에서 값의 역할을 하게 된다. 코드 블록이 아니고 값처럼 취급되어 변수에 할당되기 때문에, 함수때문이 아닌 단지 구문의 끝이기 때문에 `;`이 붙게 된다.

콜백 함수

함수가 값으로 취급되기 때문에 함수의 인자로 넘겨주는것도 가능하다. 인자로 넘겨지는 함수를 콜백함수라고 하며, 꽤나 어려운 내용이니 원리정도만 알아보고 넘어가자.

매개변수가 3개 있는 함수, `ask(question, yes, no)`를 작성해보자.

함수는 반드시 `question(질문)`을 해야 하고, 사용자의 답변에 따라 `yes()` 나 `no()`를 호출하게 된다.

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}function showOk() {
  alert( "동의하셨습니다." );
}

function showCancel() {
  alert( "취소 버튼을 누르셨습니다." );
}

// 사용법: 함수 showOk와 showCancel가 ask 함수의 인수로 전달됨
ask("동의하십니까?", showOk, showCancel);
```

위와 같이 코드를 작성하면 아주 깔끔하게 코드를 작성하는것이 가능하다.

함수 `ask`의 인수, `showOk`와 `showCancel`은 *콜백 함수* 또는 *콜백*이라고 부르게 된다.

화살표 함수

함수 표현식보다 단순하고 간결한 문법으로 함수를 만들 수 있는 방법이 있다.

바로 화살표 함수(arrow function)를 사용하는 것입니다. 화살표 함수라는 이름은 문법의 생김새를 차용해 지어졌다.

```
let func = (arg1, arg2, ...argN) => expression
```

이렇게 코드를 작성하면 인자 `arg1..argN`를 받는 함수 `func`이 만들어 진다. 함수 `func`는 화살표(`=>`) 우측의 `표현식(expression)`을 평가하고, 평가 결과를 반환한다.

아래 함수를 축약해서 만들 수 있다고 생각하면 쉽게 이해할 수 있다.

```
let func = function(arg1, arg2, ...argN) {
  return expression;
};
```

좀 더 구체적인 예시를 살펴보자.

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

/* 위 화살표 함수는 아래 함수의 축약 버전입니다.

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

alert( sum(1, 2) ); // 3
```

- 인수가 하나밖에 없다면 인수를 감싸는 괄호를 생략할 수 있으며 괄호를 생략하면 코드 길이를 더 줄일 수 있다.

  ```
  let double = n => n * 2;
  // let double = function(n) { return n * 2 }과 거의 동일합니다.alert( double(3) ); // 6
  ```

- 인수가 하나도 없을 땐 괄호를 비워놓으면 되지만, 이 때 괄호는 생략할 수 없다.

```
let sayHi = () => alert("안녕하세요!");

sayHi();
```

화살표 함수는 함수 표현식과 같은 방법으로 사용할 수 있다.

```
let age = prompt("나이를 알려주세요.", 18);

let welcome = (age < 18) ?
  () => alert('안녕') :
  () => alert("안녕하세요!");

welcome();
```

화살표 함수를 처음 접하면 이해가 잘 안될 수 있지만, 눈법이 눈에 익기 시작하면 정말 간단하게 함수를 활용할 수 있게 되니 화살표 함수를 사용하는 연습을 해보자.

본문이 여러중인 화살표함수

본문이 여러줄이라면 위에 소개했던 함수와 마찬가지로 중괄호 `{ }` 를 활용하면 된다.

```
let sum = (a, b) => {  // 중괄호는 본문 여러 줄로 구성되어 있음을 알려줍니다.
  let result = a + b;
  return result; // 중괄호를 사용했다면, return 지시자로 결괏값을 반환해주어야 합니다.};

alert( sum(1, 2) ); // 3
```

문제 1

함수 표현식을 사용해 만든 아래 함수를 화살표 함수로 바꿔보세요.

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}

ask(
  "동의하십니까?",
  function() { alert("동의하셨습니다."); },
  function() { alert("취소 버튼을 누르셨습니다."); }
);
```