Function

☑ Function

스크립트를 작성하다 보면 유사한 동작을 하는 코드가 여러 곳에서 필요할 때가 많다. 사용자가 로그인이나 로그아웃을 했을 때 안내 메시지를 보여주는 동작 같은 경우이다.

function는 프로그램을 구성하는 주요 '구성 요소(building block)' 이다. function를 이용하면 중복 없이 유사한 동작을 하는 코드를 여러 번 호출할 수 있다.

앞서 다양한 예시에서 console.log(message), prompt(message, default), confirm(question)과 같은 내장 function를 사용해 보았다. 이번에는 function를 직접 만드는 방법에 대해 학습하도록 한다.

❖ Function Declaration

Function Declaration(function 선언) 방식을 이용하면 function를 만들 수 있다(function 선언 방식은 function 선언문이라고 부르기도 한다).

function 선언 방식은 아래와 같이 작성할 수 있다.

```
function showMessage() {
  console.log( '안녕하세요!' );
}
```

function 키워드, function 이름, 괄호로 둘러싼 parameter를 차례로 써주면 function를 선언할 수 있다. 위 function에는 parameter가 없는데, 만약 parameter가 여러 개 있다면 각 parameter를 콤마로 구분해준다. 이어서 function를 구성하는 코드의 모임인 'function 본문(body)'을 중괄호로 감싸 붙여준다.

```
function name(parameters) {
  ...function 본문...
}
```

새롭게 정의한 function는 function 이름 옆에 괄호를 붙여 호출할 수 있다. showMessage(); 같이 한다.

[예시]

```
function showMessage() {
  console.log( '안녕하세요!' );
```

```
}

showMessage();
showMessage();
```

showMessage()로 function를 호출하면 function 본문이 실행된다. 위 예시에선 showMessage를 두 번 호출했으므로 console 창에 두 번 출력이 된다.

function의 주요 용도 중 하나는 중복 코드 피하기 이다. 위 예시를 통해 이를 확인해 보았다.

console 창에 보여줄 메시지를 바꾸거나 메시지를 보여주는 방식 자체를 변경하고 싶다면, function 본문 중 출력에 관여하는 코드 딱 하나만 수정해주면 된다.

❖ Local variable(지역 변수)

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

[예시]

```
function showMessage() {
  let message = "안녕하세요!"; // local variable

  console.log( message );
}

showMessage();   // 안녕하세요!

console.log( message ); // ReferenceError: message is not defined (message는 function 내 local variable 이기 때문에 에러가 발생한다.)
```

❖ Outer variable (외부 변수)

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

```
let userName = 'SmartIT';

function showMessage() {
  let message = 'Hello, ' + userName;
  console.log(message);
}

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

function에선 외부 변수에 접근하는 것뿐만 아니라, 수정도 할 수 있다.

[예시]

```
let userName = 'SmartIT';

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

  let message = 'Hello, ' + userName;
  console.log(message);
}

console.log( userName );  // function 호출 전이므로 John 이 출력됨

showMessage();

console.log( userName );  // function에 의해 Bob 으로 값이 바뀜
```

외부 변수는 local variable가 없는 경우에만 사용할 수 있다.

function 내부에 외부 변수와 동일한 이름을 가진 변수가 선언되었다면, 내부 변수는 외부 변수를 가린다. 예시를 살펴보자. function 내부에 외부 변수와 동일한 이름을 가진 지역 변수 userName가 선언되어 있다. 외부 변수는 내부 변수에 가려져 값이 수정되지 않았다.

```
let userName = 'SmartIT';

function showMessage() {
  let userName = "ICT";   // 같은 이름을 가진 local variable를 선언한다.

  let message = 'Hello, ' + userName;      // Bob
  console.log(message);
}

// function는 내부 변수인 userName만 사용한다,
showMessage();

// function는 외부 변수에 접근하지 않는다. 따라서 값이 변경되지 않고, John이 출력된다.
console.log( userName );
```

⊙ Global variable (전역 변수)

위 예시의 userName처럼, function 외부에 선언된 변수는 전역 변수(global variable) 라고 부른다.

global variable는 같은 이름을 가진 local variable에 의해 가려지지만 않는다면 모든 function에서 접근할 수 있다.
변수는 연관되는 function 내에 선언하고, global variable는 되도록 사용하지 않는 것이 좋다. 비교적 근래에 작성된 코드들은 대부분 global variable를 사용하지 않거나 최소한으로만 사용한다. 다만 프로젝트 전반에서 사용되는 데이터는 global variable에 저장하는 것이 유용한 경우도 있으니 이 점을 알아두도록 한다.

❖ Parameter (parameter)

parameter(parameter)를 이용하면 임의의 데이터를 function 안에 전달할 수 있다. parameter는 인수(argument) 라고 불리기도 한다(parameter와 argument는 엄밀히 같진 않다).

아래 예시에서 function showMessage는 parameter from 과 text를 가진다.

```
function showMessage(from, text) {        // parameter: from, text
  console.log(from + ': ' + text);
}

showMessage('Ann', 'Hello!');             // Ann: Hello!          (1)
showMessage('Ann', "What's up?");         // Ann: What's up?      (2)
```

(1), (2)로 표시한 줄에서 function를 호출하면, function에 전달된 인자는 local variable from과 text에 복사된다. 그 후 function는 local variable에 복사된 값을 사용한다.

예시 하나를 더 살펴보자. global variable from이 있고, 이 변수를 function에 전달하였다. function가 from을 변경하지만, 변경 사항은 외부 변수 from에 반영되지 않니다. function는 언제나 복사된 값을 사용하기 때문이다.

```
function showMessage(from, text) {
  from = '*' + from + '*';        // "from"을 좀 더 멋지게 꾸며준다.
  console.log( from + ': ' + text );
}

let from = "Ann";

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

// function는 복사된 값을 사용하기 때문에 바깥의 "from"은 값이 변경되지 않는다.
console.log( from );      // Ann
```

❖ Default value (기본값)

parameter에 값을 전달하지 않으면 그 값은 undefined가 된다.
예시를 통해 이에 대해 알아보자. 위에서 정의한 function showMessage(from, text)는 parameter가 2개 지만, 아래와 같이 인수를 하나만 넣어서 호출할 수 있다.

```
showMessage("SMU");
```

이렇게 코드를 작성해도 에러가 발생하지 않는다. 두 번째 parameter에 값을 전달하지 않았기 때문에 text엔 undefiend가 할당될 뿐이다. 따라서 에러 없이 "SMU: undefined"가 출력된다.

parameter에 값을 전달하지 않아도 그 값이 undefined가 되지 않게 하려면, '기본값(default value)'을 설정해주면 된다. parameter 오른쪽에 = 을 붙이고 undefined 대신 설정하고자 하는 기본값을 써주면 된다.

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

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

이젠 text가 값을 전달받지 못해도 undefined 대신 기본값 "no text given"이 할당된다.

위 예시에선 문자열 "no text given"을 기본값으로 설정했다. 그러나 아래와 같이 복잡한 표현식도 기본값으로 설정할 수도 있다.

```
function showMessage(from, text = anotherFunction()) {
  // anotherFunction()은 text값이 없을 때만 호출됨
  // anotherFunction()의 반환 값이 text의 값이 됨
}
```

⊙ parameter 기본값 평가 시점

JavaScript에선 function를 호출할 때마다 parameter 기본값을 평가한다. 물론 해당하는 parameter가 없을 때만 기본값을 평가한다.

위 예시에선 parameter text에 값이 없는 경우, showMessage()를 호출할 때마다 anotherFunction()이 호출된다.

❖ parameter 기본값을 설정할 수 있는 또 다른 방법

가끔은 function 선언부에서 parameter 기본값을 설정하는 것 대신 function가 실행되는 도중에 기본값을 설정하는 게 논리에 맞는 경우가 생기기도 한다.

이런 경우엔 일단 parameter를 undefined와 비교하여 function 호출 시 parameter가 생략되었는지를 확인한다.

```
function showMessage(text) {
  if (text === undefined) {
    text = '빈 문자열';
  }
  console.log(text);
}

showMessage(); // 빈 문자열
```

이렇게 if 문을 쓰는 것 대신 논리 연산자 ||를 사용할 수도 있다.

```
// parameter가 생략되었거나 빈 문자열("")이 넘어오면 변수에 '빈 문자열'이 할당된다.
function showMessage(text) {
  text = text || '빈 문자열';
  ...
}
```

이 외에도 모던 JavaScript Engine이 지원하는 nullish 병합 연산자(nullish coalescing operator) ??를 사용하면 0처럼 falsy로 평가되는 값들을 일반 값처럼 처리할 수 있어서 좋다.

```
// parameter 'count'가 넘어오지 않으면 'unknown'을 출력해주는 function
function showCount(count) {
  console.log(count ?? "unknown");
}

showCount(0);           // 0
showCount(null);        // unknown
showCount();            // unknown
```

❖ Return value (반환 값)

function를 호출했을 때 function를 호출한 그 곳에 특정 값을 반환하게 할 수 있다. 이때 이 특정 값을 반환 값(return value)이라고 부른다.

인수로 받은 두 값을 더해주는 간단한 function를 만들어 반환 값에 대해 알아보도록 한다.

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

let result = sum(1, 2);
console.log( result );    // 3
```

지시자 return은 function 내 어디서든 사용할 수 있다. 실행 흐름이 지시자 return을 만나면 function 실행은 즉시 중단되고 function를 호출한 곳에 값을 반환한다. 위 예시에선 반환 값을 result에 할당하였다.

아래와 같이 function 하나에 여러 개의 return문이 올 수도 있다.

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

let age = prompt('나이를 알려주세요', 18);

if ( checkAge(age) ) {
  console.log( '접속 허용' );
} else {
  console.log( '접속 차단' );
}
```

아래와 같이 지시자 return만 명시하는 것도 가능하다. 이런 경우는 function가 즉시 종료된다.

[예시]

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

```
  console.log( "영화 상영" );     // (1)
  // ...
}
```

위 예시에서, checkAge(age)가 false를 반환하면, (1)로 표시한 줄은 실행이 안 되기 때문에 function showMovie는 console 창을 보여주지 않는다.

⊙ return 문이 없거나 return 지시자만 있는 function는 undefined를 반환한다.

return 문이 없는 function도 무언가를 반환한다. undefined를 반환한다.

```
function doNothing() { /* empty */ }

console.log( doNothing() === undefined ); // true
```

return 지시자만 있는 경우도 undefined를 반환한다. return은 return undefined와 동일하게 동작한다.

```
function doNothing() {
  return;
}

console.log( doNothing() === undefined ); // true
```

⊙ return과 value(값) 사이에 절대 newline(줄)을 삽입하지 않는다.

반환하려는 값이 긴 표현식인 경우, 아래와 같이 지시자 return과 반환하려는 값 사이에 새 줄을 넣어 코드를 작성하고 싶을 수도 있다.

```
return
 (some + long + expression + or + whatever * f(a) + f(b))
```

JavaScript는 return문 끝에 세미콜론을 자동으로 넣기 때문에 이렇게 return문을 작성하면 안된다. 위 코드는 아래 코드처럼 동작한다.

```
return;
 (some + long + expression + or + whatever * f(a) + f(b))
```

따라서 반환하고자 했던 표현식을 반환하지 못하고 아무것도 반환하지 않는 것처럼 되어버린다.

표현식을 여러 줄에 걸쳐 작성하고 싶다면 표현식이 return 지시자가 있는 줄에서 시작하도록 작성해야 한다. 또는 아래와 같이 여는 괄호를 return 지시자와 같은 줄에 써줘도 괜찮다.

```
return (
  some + long + expression
  + or +
  whatever * f(a) + f(b)
  )
```

이렇게 하면 의도한 대로 표현식을 반환할 수 있다.

❖ Naming a function

function는 어떤 동작을 수행하기 위한 코드를 모아놓은 것이다. 따라서 function의 이름은 대개 동사이다. function 이름은 가능한 한 간결하고 명확해야 한다. function가 어떤 동작을 하는지 설명할 수 있어야 한다. 코드를 읽는 사람은 function 이름만 보고도 function가 어떤 기능을 하는지 힌트를 얻을 수 있어야 한다.

function가 어떤 동작을 하는지 축약해서 설명해주는 동사를 접두어로 붙여 function 이름을 만드는 게 관습이다. 다만, 팀 내에서 그 뜻이 반드시 합의된 접두어만 사용해야 한다.

"show"로 시작하는 function는 대개 무언가를 보여주는 function 이다.

이 외에 아래와 같은 접두어를 사용할 수 있다.

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

위 접두어를 사용하면 아래와 같은 function를 만들 수 있습니다.

```
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
```

접두어를 적절히 활용하면 function 이름만 보고도 function가 어떤 동작을 하고 어떤 값을 반환하는지 쉽게 알 수 있다.

⊙ function는 동작 하나만 담당해야 한다.

function는 function 이름에 언급되어있는 동작을 정확히 수행해야 한다. 그 이외의 동작은 수행해선 안된다.

독립적인 두 개의 동작은 독립된 function 두 개에서 나눠서 수행할 수 있게 해야 한다. 한 장소에서 두 동작을 동시에 필요로 하는 경우라도 나누어야 한다.(이 경우는 제3의 function를 만들어 그곳에서 두 function를 호출한다).

개발자들이 빈번히 하는 실수를 소개해 본다.

- ☞ getAge function는 나이를 얻어오는 동작만 수행해야 한다. console.log 창에 나이를 출력해주는 동작은 이 function에 들어가지 않는 것이 좋다.
- ☞ createForm function는 form을 만들고 이를 반환하는 동작만 해야 한다. form을 문서에 추가하는 동작이 해당 function에 들어가 있으면 좋지 않다.
- ☞ checkPermission function는 승인 여부를 확인하고 그 결과를 반환하는 동작만 해야 한다. 승인 여부를 보여주는 메시지를 띄우는 동작이 들어가 있으면 좋지 않다.

위 예시들은 접두어의 의미가 합의되었다고 가정하고 만들었다. 본인이나 본인이 속한 팀에서 접두어의 의미를 재합의하여 function를 만들 수도 있긴 하지만, 아마도 위 예시에서 사용한 접두어 의미와 크게 차이가 나진 않을 것이다. 어찌 되었든 접두어를 사용하여 function 이름을 지을 땐, 해당 접두어에 어떤 의미가 있는지 잘 이해하고 있어야 한다. 해당 접두어가 붙은 function가 어떤 동작을 하는지, 어떤 동작은 하지 못하는지 알고 있어야 한다. 접두어를 붙여 만든 모두 function는 팀에서 만든 규칙을 반드시 따라야 한다. 팀원들은 이 규칙을 충분히 이해하고 있어야 하며, 팀원들 사이에 이 규칙이 잘 공유되어야 한다.

⊙ 아주 짧은 이름

정말 빈번히 쓰이는 function 중에 이름이 아주 짧은 function가 있다.
jQuery Framework에서 쓰이는 function $와 Lodash 라이브러리의 핵심 function _ 같은 것이다.

이 function들은 지금까지 소개한 function 이름짓기에 관련된 규칙을 지키지 않고 있다. 예외에 속한다. function 이름은 간결하고 function가 어떤 일을 하는지 설명할 수 있게 지어야 한다.

❖ function == Comment(주석)

function는 간결하고, 한 가지 기능만 수행할 수 있게 만들어야 한다. function가 길어지면 function를 잘게 쪼갤 때가 되었다는 신호로 받아들어야 한다. function를 쪼개는 건 쉬운 작업은 아니다. 그러나 function를 분리해 작성하면 많은 장점이 있기 때문에 function가 길어질 경우엔 function를 분리해 작성할 것을 권장한다.

function를 간결하게 만들면 테스트와 디버깅이 쉬워진다. 그리고 function 그 자체로 주석의 역할까지 한다.

같은 동작을 하는 function, showPrimes(n)를 두 개 만들어 비교해 보자. showPrimes(n)은 n까지의 소수(prime numbers)를 출력해준다. 첫 번째 showPrimes(n)에선 레이블을 사용해 반복문을 작성하도록 한다.

```
function showPrimes(n) {
  nextPrime: for (let i = 2; i < n; i++) {

    for (let j = 2; j < i; j++) {
      if (i % j == 0) continue nextPrime;
    }

    console.log( i ); // prime(소수)
  }
}
```

두 번째 showPrimes(n)는 소수인지 아닌지 여부를 검증하는 코드를 따로 분리해 isPrime(n)이라는 function에 넣어서 작성했다.

```
function showPrimes(n) {
  for (let i = 2; i < n; i++) {
    if (!isPrime(i)) continue;

    console.log(i);  // a prime
  }
}

function isPrime(n) {
  for (let i = 2; i < n; i++) {
    if ( n % i == 0) return false;
  }
  return true;
}
```

두 번째 showPrimes(n)가 더 이해하기 쉽다. isPrime function 이름을 보고 해당 function가 소수 여부를 검증하는 동작을 한다는 걸 쉽게 알 수 있다. 이렇게 이름만 보고도 어떤 동작을 하는지 알 수 있는 코드를 자기 설명적(self-describing) 코드라고 부른다.

위와 같이 function는 중복을 없애려는 용도 외에도 사용할 수 있다. 이렇게 function를 활용하면 코드가 정돈되고 가독성이 높아진다.

요약하면 function 선언 방식으로 function를 만들 수 있다.

```
function name(parameter, delimied, by, comma) {
  /* function 본문 */
}
```

- ☞ function에 전달된 parameter는 복사된 후 function의 local variable가 된다.
- ☞ function는 외부 변수에 접근할 수 있다. 그러나 function 바깥에서 function 내부의 local variable에 접근하는 건 불가능하다.
- ☞ function는 값을 반환할 수 있다. 값을 반환하지 않는 경우는 반환 값이 undefined가 된다.

깔끔하고 이해하기 쉬운 코드를 작성하려면 function 내부에서 외부 변수를 사용하는 방법 대신 local variable와 parameter를 활용하는 게 좋다.

개발자는 parameter를 받아서 그 변수를 가지고 반환 값을 만들어 내는 function를 더 쉽게 이해할 수 있다. parameter 없이 function 내부에서 외부 변수를 수정해 반환 값을 만들어 내는 function는 쉽게 이해하기 힘들다.

function 이름을 지을 땐 아래와 같은 규칙을 따르는 것이 좋다.

- ☞ function 이름은 function가 어떤 동작을 하는지 설명할 수 있어야 한다. 이렇게 이름을 지으면 function 호출 코드만 보아도 해당 function가 무엇을 하고 어떤 값을 반환할지 바로 알 수 있다.
- ☞ function는 동작을 수행하기 때문에 이름이 주로 동사이다.
- ☞ create…, show…, get…, check… 등의 잘 알려진 접두어를 사용해 이름을 지을 수 있다. 접두어를 사용하면 function 이름만 보고도 해당 function가 어떤 동작을 하는지 파악할 수 있다.

function는 script를 구성하는 주요 구성 요소이다. 지금까진 다룬 내용은 function의 기본이다. 여기선 function를 만드는 방법, 사용하는 방법을 소개했는데 이 내용은 시작일 뿐이다.

[과제]

☞ "else"는 정말 필요한가?

★ 중요도: 4

아래 function는 parameter age가 18보다 큰 경우 true를 반환한다. 그 이외의 경우는 confirm 대화상자를 통해 사용자에게 질문한 후, 해당 결과를 반환한다.

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

위 코드에서 else문을 삭제해도 기존 코드와 동일하게 작동할까?

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

아니면 뭔가 변화가 있을까?

[해답]

No difference.

☞ '?'나 '||'를 사용하여 function 다시 작성하기

★ 중요도: 4

아래 function는 parameter age가 18보다 큰 경우 true를 반환한다. 그 이외의 경우는 confirm 대화상자를 통해 사용자에게 질문한 후, 해당 결과를 반환한다.

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

```
  }
}
```

if문을 사용하지 않고 동일한 동작을 하는 function를 한 줄에 작성해보자. 아래 조건을 충족하는 해답 2개를 작성해야 한다.

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

[해답]

```
① Using a question mark operator '?':
function checkAge(age) {
  return (age > 18) ? true : confirm('보호자의 동의를 받으셨나요?');
}

② Using OR || (the shortest variant):
function checkAge(age) {
  return (age > 18) || confirm('보호자의 동의를 받으셨나요?');
}
```

여기서 18세 이상인 괄호는 필요하지 않다. 더 나은 가독성을 위해 존재한다.

☞ min(a, b) function 만들기

★ **중요도:** 1

a 와 b 중 작은 값을 반환해주는 function, min(a,b)을 만들어보자. 만든 function는 아래와 같이 동작해야 한다.

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

[해답]

```
① if 사용:
function min(a, b) {
  if (a < b) {
    return a;
  } else {
    return b;
  }
```

```
}
```

② 물음표 연산자 '?'가 있는 솔루션:

```
function min(a, b) {
  return a < b ? a : b;
}
```

이 경우에는 a == b 의 경우 무엇을 반환할 지 중요하지 않다.

☞ pow(x,n) function 만들기

★ 중요도: 4

x의 n제곱을 반환해주는 function, pow(x,n)를 만들어보자. x의 n 제곱은 x를 n번 곱해서 만들 수 있다.

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

prompt 대화상자를 띄워 사용자로부터 x와 n을 입력받고 pow(x,n)의 반환 값을 보여주는 코드를 작성해보자. 주의사항: n은 1 이상의 자연수이어야 한다. 이외의 경우엔 자연수를 입력하라는 alert 창을 띄워주어야 한다.

[해답]

```
function pow(x, n) {
  let result = x;

  for (let i = 1; i < n; i++) {
    result *= x;
  }

  return result;
}

let x = prompt("x?", '');
let n = prompt("n?", '');

if (n < 1) {
  alert(`${n}은 양의 정수이어야 한다.`);
} else {
  alert( pow(x, n) );
}
```