함수

2020-01-19

이승진

**학습목표**

함수 기본 자료형

파라미터 값 생략

가변 파라미터

콜백함수

목차

[1. 함수 (function) 2](#_Toc37092407)

[1) 함수란? 2](#_Toc37092408)

[2) 함수 정의 2](#_Toc37092409)

[3) 함수 호출 3](#_Toc37092410)

[2. 파라미터 값 생략 4](#_Toc37092411)

[1) 파라미터 값 생략 가능 4](#_Toc37092412)

[2) 파라미터 값이 전달되었는지 확인 5](#_Toc37092413)

[3. 가변 파라미터 7](#_Toc37092414)

[1) 가변 파라미터 ES6 문법 7](#_Toc37092415)

[2) 가변 파라미터 옛날 문법 8](#_Toc37092416)

[4. 콜백 함수 9](#_Toc37092417)

[1) 함수 자료형 9](#_Toc37092418)

[2) 콜백 함수 전달 12](#_Toc37092419)

[3) 화살표 함수 15](#_Toc37092420)

[5. 타이머 18](#_Toc37092421)

[1) setTimeout 함수 18](#_Toc37092422)

[2) setInterval 함수 19](#_Toc37092423)

[6. 함수 호출 21](#_Toc37092424)

[1) 지역 함수 21](#_Toc37092425)

[2) 이름 없는 함수 호출 22](#_Toc37092426)

[3) 함수를 리턴 23](#_Toc37092427)

[7. 요약 24](#_Toc37092428)

[8. 과제 25](#_Toc37092429)

# 함수 (function)

## 함수란?

javascript 언어에서 함수는, C 언어의 함수와 유사하다.

javascript 함수의 독특한 부분은, 함수가 기본 자료형이라는 점이다.

C 언어의 함수나 Java의 메소드는, 실행 가능한 코드일 뿐, 자료형은 아니다.

함수가 기본 자료형이라는 말은, 함수가 어떤 값이라는 말이다.

그래서 함수를 변수에 대입할 수도 있고,

함수를 파라미터 값으로 전달할 수도 있고,

함수를 리턴 값으로 리턴할 수도 있다.

## 함수 정의

|  |
| --- |
| function hello(name) {      console.log("hello " + name);  } |

위 함수의 이름은 hello 이다.

파라미터 변수는 name 이다.

javascript 언어는 약타입 언어이므로, 파라미터 변수의 타입을 선언하지 않고,

파라미터 변수 이름만 선언한다.

|  |
| --- |
| function add(a, b) {      return a + b;  } |

위 함수의 이름은 add 이다.

파라미터 변수는 각각 a, b 이다.

add 함수의 리턴 값은 number 일 테니지만, 리턴 타입도 선언하지 않는다.

## 함수 호출

### 함수 호출 #1

|  |
| --- |
| function hello(name) {      console.log("hello " + name);  }  hello("홍길동"); |

함수를 호출하는 방법은 C 언어와 같다.

실행

|  |
| --- |
| hello 홍길동 |

### 함수 호출 #2

|  |
| --- |
| hello("홍길동");  function hello(name) {      console.log("hello " + name);  } |

실행

|  |
| --- |
| hello 홍길동 |

함수 호출이 위에 있고, 함수 정의가 아래 있어도 된다.

### 함수 호출 #3

|  |
| --- |
| function add(a, b) {      return a + b;  }  console.log(add(3, 4)); |

실행

|  |
| --- |
| 7 |

# 파라미터 값 생략

## 파라미터 값 생략 가능

### 예 #1

|  |
| --- |
| function hello(name) {      console.log("hello " + name);  }  hello(); |

실행

|  |
| --- |
| hello undefined |

hello 함수를 호출하면서 파라미터 값을 전달하지 않았다.

값이 전달되지 않은 파라미터 변수의 값은 undefined 이다.

### 예 #2

|  |
| --- |
| function add(a, b) {      return a + b;  }  console.log(add(3, 4))  console.log(add(3)); |

실행

|  |
| --- |
| 7  NaN |

파라미터 값이 한 개만 전달되기 때문에,

파라미터 변수 a의 값은 3 이고, 파라미터 변수 b의 값은 undefined 가 된다.

3 + undefined 표현식의 값은, 계산할 수 없으므로, NaN 이다.

## 파라미터 값이 전달되었는지 확인

### 예 #1 -

|  |
| --- |
| function add(a, b) {      if (b == undefined) b = 0;      return a + b;  }  console.log(add(3, 4))  console.log(add(3)); |

실행

|  |
| --- |
| 7  3 |

if (b == undefined) b = 0;

두 번째 파라미터가 전달되지 않은 경우에, b 변수 값은 undefined 이고,

이 if 문이 true 가 된다.

### 예 #2

|  |
| --- |
| function add(a, b) {      if (!b) b = 0;      return a + b;  }  console.log(add(3, 4))  console.log(add(3)); |

실행

|  |
| --- |
| 7  3 |

변수 b 값이 undefined 이면,

**if (b)** 조건식은 false 이고, **if (!b)** 조건식은 true 이다.

**if (b == undefine)** 대신에 **if (!b)** 를 사용할 수 있다.

### 예 #5

|  |
| --- |
| function add(a, b) {      return a + (b || 0);  }  console.log(add(3, 4))  console.log(add(3)); |

실행

|  |
| --- |
| 7  3 |

(b || 0) 표현식의 값

b 값이 숫자이면, 이 표현식의 값은 b 값이 된다.

b 값이 undefined 이면, 이 표현식의 값은 0 이다.

### (e1 || e2) 의 값은

e1 표현식의 값이 true 에 해당하는 값이면, 결과는 e1 값이고,

e1 표현식의 값이 false 에 해당하는 값이면, 결과는 e2 값이다.

# 가변 파라미터

## 가변 파라미터 ES6 문법

|  |
| --- |
| function sum(...a) {      let result = 0;      for (let i = 0; i < a.length; ++i)          result += a[i];      return result;  }  console.log(sum(1, 2, 3, 4));  console.log(sum(4));  console.log(sum()); |

실행

|  |
| --- |
| 10  4  0 |

sum 함수를 호출할 때 전달되는 파라미터 값 목록이, 배열 형태로 a 변수에 전달된다.

a 변수의 사용법은 배열의 사용법과 같다.

...a 부분의 문법의 이름은 spread syntax 이다.

## 가변 파라미터 옛날 문법

|  |
| --- |
| function sum() {      let result = 0;      for (let i = 0; i < arguments.length; ++i)          result += arguments[i];      return result;  }  console.log(sum(1, 2, 3, 4));  console.log(sum(4));  console.log(sum()); |

실행

|  |
| --- |
| 10  4  0 |

ES6 이전 옛날 문법에서는, arguments 키워드를 이용해서 구현해야 한다.

arguments 키워드는 모든 함수에서 사용할 수 있다.

arguments 키워드의 값은, 함수를 호출할 때 전달된 파라미터 값 목록이 배열로 만들어진 것이다.

# 콜백 함수

## 함수 자료형

javascript 언어에서 함수는 자료형(data type) 중 하나이다.

즉 함수도 자료(data) 이고, 값이다.

### function1.js

|  |
| --- |
| function add(a, b) {    return a + b;  }  let a = add(3, 4);  console.log(a);  let f = add;  console.log(typeof f);  let b = f(3, 4);  console.log(b); |

실행

|  |
| --- |
| 7  function  7 |

let a = add(3, 4);

변수 a 에는 add(3, 4) 함수의 리턴값 7이 대입된다.

let f = add;

변수 f 에 대입되는 값은 add 함수이다.

typeof f

변수 f 에 대입된 값은 함수이고, 그 값의 자료형은 function 이다.

let b = f(3, 4);

변수 f의 값인 함수를 호출한다. 즉 add 함수가 호출된다.

이 함수의 리턴값 7이 변수 b에 대입된다.

### function2.js

|  |
| --- |
| let f = function(a, b) {    return a + b;  }    console.log(typeof f);  let a = f(3, 4);  console.log(a); |

실행

|  |
| --- |
| function  7 |

let f = function(a, b) {

  return a + b;

}

변수 f 에 대입되는 값은, 노란색으로 칠한 부분이다. 즉 함수가 변수 f 에 대입된다.

typeof f

변수 f 에 대입된 값은 함수이고, 그 값의 자료형은 function 이다.

let b = f(3, 4);

변수 f의 값인 함수를 호출한다. 즉 add 함수가 호출된다.

이 함수의 리턴값 7이 변수 b에 대입된다.

### 동일한 코드

|  |
| --- |
| function add(a, b) {    return a + b;  }  let f = add; |

위 코드와 아래 코드는 동일하다.

변수 f에 함수가 대입된다.

|  |
| --- |
| let f = function(a, b) {    return a + b;  } |

### 동일한 코드

|  |
| --- |
| function add(a, b) {    return a + b;  } |

위 코드와 아래 코드는 동일하다.

변수 add에 함수가 대입된다.

|  |
| --- |
| let add = function(a, b) {    return a + b;  } |

## 콜백 함수 전달

### callback1.js

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | function test1(f) {    let result = f(3, 4);    console.log(result);  }  function add(a, b) {    return a + b;  }  function multiply(a, b) {    return a \* b;  }  test1(add);  test1(multiply); |

실행

|  |
| --- |
| 7  12 |

### 실행 순서

(줄1~4) test1 함수가 생성된다. (test1 변수에 함수가 대입된다.)

(줄6~8) add 함수가 생성된다. (add 변수에 함수가 대입된다.)

(줄10~12) multiply 함수가 생성된다. (multiply 변수에 함수가 대입된다.)

(줄14) test1(add);

test1 함수를 호출한다.

이때 파라미터 값으로 add 함수가 전달된다.

(줄1) function test1(f) {

test1 함수가 호출된다.

파라미터 변수 f 에 전달된 값은 add 함수이다.

(줄2) let result = f(3, 4);

add 함수가 호출되고, 그 리턴 값이 result 변수에 대입된다.

이 값은 7 이다.

(줄3) console.log(result);

화면에 7 이 출력된다.

(줄4) test1 함수 리턴.

(줄15) test1(multiply);

test1 함수를 호출한다.

이때 파라미터 값으로 multiply 함수가 전달된다.

(줄1) function test1(f) {

test1 함수가 호출된다.

파라미터 변수 f 에 전달된 값은 multiply 함수이다.

(줄2) let result = f(3, 4);

multiply 함수가 호출되고, 그 리턴 값이 result 변수에 대입된다.

이 값은 12 이다.

(줄3) console.log(result);

화면에 12 가 출력된다.

(줄4) test1 함수 리턴.

### 콜백 함수란?

콜백 함수(callback function)는 다른 함수의 파라미터 값으로 전달되어 호출되는 함수이다.

줄14에서 add 함수가 test1 함수의 파라미터 값으로 전달되어, test1 함수 내부에서 호출되었다.

줄15에서 multiply 함수가 test1 함수의 파라미터 값으로 전달되어, test1 함수 내부에서 호출되었다.

줄14의 add 함수, 줄15의 multiply 함수가 콜백 함수이다.

### callback2.js

|  |
| --- |
| function test1(f) {    let result = f(3, 4);    console.log(result);  }  let add = function(a, b) {    return a + b;  }  let multiply = function(a, b) {    return a \* b;  }  test1(add);  test1(multiply); |

위 콜백함수 예제를 위와 같이 구현할 수 있다.

실행 결과는 동일하다.

add 변수와 multiply 변수에 대입되는 값은 함수이다.

test1(add);

test1 함수가 호출된다.

파라미터 변수 f 에 전달된 값은 add 함수이다.

즉 파라미터 값으로 전달되는 것은 함수이다.

test1(multiply);

test1 함수를 호출한다.

이때 파라미터 값으로 add 함수가 전달된다.

즉 파라미터 값으로 전달되는 것은 함수이다.

### callback3.js

|  |
| --- |
| function test1(f) {    let result = f(3, 4);    console.log(result);  }  test1(function(a, b) {    return a + b;  });  test1(function(a, b) {    return a \* b;  }); |

위 콜백함수 예제를 위와 같이 구현할 수 있다.

실행 결과는 동일하다.

test1(function(a, b) {

  return a + b;

});

test1 함수를 호출한다.

노란색으로 표시한 부분이 파라미터 값으로 전달된다.

즉 파라미터 값으로 전달되는 것은 함수이다.

## 화살표 함수

화살표 함수는 콜백 함수를 좀 더 간결하게 구현하기 위한 문법다.

### arrow1.js

|  |
| --- |
| function test1(f) {    let result = f(3, 4);    console.log(result);  }  let add = (a, b) => {    return a + b;  }  let multiply = (a, b) => {    return a \* b;  }  test1(add);  test1(multiply); |

위 콜백함수 예제를 위와 같이 구현할 수 있다.

실행 결과는 동일하다.

실행

|  |
| --- |
| 7  12 |

add 변수와 multiply 변수에 대입되는 값은 함수이다.

function(a, b) {

  return a + b;

}

위 코드와 아래 코드는 동일한 함수를 구현한다.

(a, b) => {

  return a + b;

}

### arrow2.js

|  |
| --- |
| function test1(f) {    let result = f(3, 4);    console.log(result);  }  test1((a, b) => {    return a + b;  });  test1((a, b) => {    return a \* b;  }); |

위 콜백함수 예제를 위와 같이 구현할 수 있다.

실행 결과는 동일하다.

실행

|  |
| --- |
| 7  12 |

test1((a, b) => {

  return a + b;

});

test1 함수를 호출한다.

노란색으로 표시한 부분이 파라미터 값으로 전달된다.

즉 파라미터 값으로 전달되는 것은 함수이다.

### arrow3.js

|  |
| --- |
| function test1(f) {    let result = f(3, 4);    console.log(result);  }  let add = (a, b) => a + b;  let multiply = (a, b) => a \* b  test1(add);  test1(multiply); |

위 콜백함수 예제를 위와 같이 구현할 수 있다.

실행 결과는 동일하다.

화살표 함수의 본문이 어떤 값을 리턴하는 한 문장인 경우에,

다음과 같이 더 간결하게 구현할 수 있다.

(a, b) => a + b

위 코드와 아래 코드는 동일한 화살표 함수이다.

(a, b) => { return a + b; }

### arrow4.js

|  |
| --- |
| function test1(f) {    let result = f(3, 4);    console.log(result);  }  test1((a, b) =>  a + b);  test1((a, b) =>  a \* b); |

위 콜백함수 예제를 위와 같이 구현할 수 있다.

실행 결과는 동일하다.

# 타이머

## setTimeout 함수

|  |
| --- |
| let id = setTimeout(callback, delay, [arg1], [arg2], ...) |

### delay

밀리초 후에 callback 함수가 호출된다.

### arg1, arg2 ...

callback 함수를 호출할 때 파라미터 값들이다.

### 리턴값

등록된 callback 함수의 id 이다.

callback 함수 등록을 취소할 때, 이 id 값을 사용한다.

|  |
| --- |
| let id = setTimeout(...생략...);  clearTimeout(id); // 취소 |

### timeout1.js

|  |
| --- |
| function printTime(msg) {    console.log(msg, new Date());  }  setTimeout(printTime, 1000, "1초 후");  setTimeout(printTime, 2000, "2초 후");  setTimeout(printTime, 3000, "3초 후"); |

출력

|  |
| --- |
| 1초 후 2020-03-29T13:37:09.514Z  2초 후 2020-03-29T13:37:10.515Z  3초 후 2020-03-29T13:37:11.515Z |

## setInterval 함수

|  |
| --- |
| let id = setInterval(callback, delay, [arg1], [arg2], ...) |

### delay

밀리초 간격으로 callback 함수가 반복 호출된다.

### arg1, arg2 ...

callback 함수를 호출할 때 파라미터 값들이다.

### 리턴값

등록된 callback 함수의 id 이다.

callback 함수 등록을 취소할 때, 이 id 값을 사용한다.

|  |
| --- |
| let id = setInterval(...생략...);  clearInterval(id); // 취소 |

### interval1.js

|  |
| --- |
| function printTime(msg) {    console.log(msg, new Date());  }  setInterval(printTime, 1000, "1초 간격"); |

출력

|  |
| --- |
| 1초 간격 2020-03-29T13:39:26.969Z  1초 간격 2020-03-29T13:39:27.969Z  1초 간격 2020-03-29T13:39:28.969Z  1초 간격 2020-03-29T13:39:29.969Z  1초 간격 2020-03-29T13:39:30.970Z  1초 간격 2020-03-29T13:39:31.971Z  1초 간격 2020-03-29T13:39:32.971Z  1초 간격 2020-03-29T13:39:33.971Z  1초 간격 2020-03-29T13:39:34.972Z |

1초 간격으로 출력이 계속된다.

실행을 중단하려면

visual studio code에서 Ctrl+F5를 눌러서 실행한 경우 Shift+F5를 누르면 종료.

쉘에서 실행한 경우, Ctrl+C를 눌러서 종료

### interval2.js

|  |
| --- |
| setInterval(function (msg) {    console.log(msg, new Date());  }, 1000, "1초 간격"); |

노란색으로 칠한 부분이 setInterval 함수의 첫째 파라미터 값이다.

interval1.js 소스코드와 실행 결과가 동일하다.

### interval3.js

|  |
| --- |
| setInterval((msg) => console.log(msg, new Date()), 1000, "1초 간격"); |

노란색으로 칠한 부분이 setInterval 함수의 첫째 파라미터 값이다.

interval1.js, interval2.js 소스코드와 실행 결과가 동일하다.

# 함수 호출

## 지역 함수

### func3.js

|  |
| --- |
| function outterFunc() {    print("hello")    function print(msg) {      console.log(msg)    }  }  outterFunc()  // print("world")  여기서 호출할 수 없다. |

출력

|  |
| --- |
| hello |

outterFunc 함수 내부에 print 함수가 선언되었다.

print 함수는 outterFunc 함수 내부에서만 호출 할 수 있고, 외부에서는 호출할 수 없다.

### func4.js

|  |
| --- |
| function outterFunc() {    function print(msg) {      console.log(msg)    }    print("hello")  }  outterFunc() |

출력

|  |
| --- |
| hello |

outterFunc 함수 내부에 print 함수가 선언되었다.

print 함수 선언 위치와 print 함수 호출 위치의 순서는 상관없다.

## 이름 없는 함수 호출

### func5.js

|  |
| --- |
| let f = function (msg) {    console.log(msg)  }  f("hello") |

출력

|  |
| --- |
| hello |

노란색으로 칠한 부분은 함수이다.

그 함수를 변수 f에 대입한다.

하늘색으로 칠한 부분은 함수 호출을 위한 괄호이고, "hello" 문자열은 파라미터 값이다.

변수 f의 값인 함수이다. 이 함수가 호출된다.

위 코드에서 f 변수를 제거하면 아래의 코드가 된다.

### func6.js

|  |
| --- |
| (function (msg) {    console.log(msg)  })("hello") |

출력

|  |
| --- |
| hello |

노란색으로 칠한 부분은 함수이다.

이 함수에는 이름이 없다.

하늘색으로 칠한 부분은 함수 호출을 위한 괄호이고, "hello" 문자열은 파라미터 값이다.

노란색으로 칠한 부분은 함수이다. 이 함수가 호출된다.

## 함수를 리턴

### func7.js

|  |
| --- |
| function factory() {    return function (msg) {      console.log(msg)    }  }  let f = factory()  f("hello") |

factory 함수의 리턴 값은 함수이다. (노란색으로 칠한 부분)

이 리턴값을 변수 f에 대입한다.

하늘색으로 칠한 부분은 함수 호출을 위한 괄호이고, "hello" 문자열은 파라미터 값이다.

변수 f의 값인 함수이다. 이 함수가 호출된다.

### func8.js

|  |
| --- |
| function factory() {    return function (msg) {      console.log(msg)    }  }  factory()("hello") |

하늘색으로 칠한 부분은 함수 호출을 위한 괄호이고, "hello" 문자열은 파라미터 값이다.

factory 함수의 리턴 값은 함수이다. 이 함수가 호출된다.

# 요약

### 함수가 기본 자료형

javascript 에서 함수는 기본 자료형이고, 어떤 값이다.

그래서 함수를 변수에 대입할 수도 있고,

파라미터 값으로 전달할 수도 있고,

함수를 리턴 값으로 리턴할 수도 있다.

### 파라미터 값 생략

함수를 호출하면서 파라미터 값을 전달하지 않아도 된다.

값이 전달되지 않은 파라미터 변수의 값은 undefined 이다.

### spread syntax

|  |
| --- |
| function sum(...a) {  // 생략  } |

sum 함수를 호출할 때 전달되는 파라미터 값 목록이, 배열 형태로 a 변수에 전달된다.

a 변수의 사용법은 배열의 사용법과 같다.

### arguments 키워드

arguments 키워드는 모든 함수에서 사용할 수 있다.

arguments 키워드의 값은, 함수를 호출할 때 전달된 파라미터 값 목록이 배열로 만들어진 것이다.

### 동일한 코드

|  |
| --- |
| function add(a, b) {    return a + b;  } |

위 코드와 아래 코드는 동일하다.

변수 add에 함수가 대입된다.

|  |
| --- |
| let add = function(a, b) {    return a + b;  } |

### 콜백 함수

콜백 함수(callback function)는 다른 함수의 파라미터 값으로 전달되어 호출되는 함수이다.

### 화살표 함수

화살표 함수는 콜백 함수를 좀 더 간결하게 구현하기 위한 문법다.

### setTimeout 함수

|  |
| --- |
| let id = setTimeout(callback, delay, [arg1], [arg2], ...) |

**delay :** 밀리초 후에 callback 함수가 호출된다.

**arg1, arg2 ... :** callback 함수를 호출할 때 파라미터 값들이다.

**리턴값 :** 등록된 callback 함수의 id 이다. callback 함수 등록을 취소할 때, 이 id 값을 사용한다.

### setInterval 함수

|  |
| --- |
| let id = setInterval(callback, delay, [arg1], [arg2], ...) |

setTimeout 함수와 사용 방법이 거의 같다.

delay 간격으로 callback 함수가 반복 호출된다는 점이 다르다.

# 과제

### timer1.js

1초 간격으로 현재 시각을 출력하는 코드를 구현하시오.

10번만 반복 호출하고 종료 되도록(clearInterval) 구현하시오.

힌트: 호출 횟수는 전역 변수로 세어야 함.