

10장

자바스크립트 함수 다루기

10.1 함수란

10.2 함수를 정의하는 방법

10.3 함수 기능 확장하기

10.4 함수의 특징 이해하기

10.5 즉시 실행 함수 사용하기

10.1 함수란

- 함수(function) : 어떤 목적을 가지고 작성한 코드를 모아 둔 블록문
- 함수를 정의한다 : 블록문을 function 키워드, 식별자, 소괄호와 함께 묶어 함수를 생성하는 것
- 함수를 정의하면,
 - 코드를 새로 작성할 필요 없이 정의한 함수를 호출하면 됨

예

```
function gugudan(){ // 함수 시작
  for(let i = 1; i <= 9; i++){
    console.log(`3 * ${i} = ${3 * i}`);
  }
} // 함수 끝
```

10.2 함수를 정의하는 방법

1. 함수 선언문으로 함수 정의하기

- function 키워드로 함수를 정의하는 방법

형식 `function 식별자(){}`

`gugudan();`

2. 함수 표현식으로 함수 정의하기

- 함수도 변수에 할당해 함수를 정의하는 방법
 - 네이밍 함수 : 변수에 할당하는 함수에 식별자가 있을 때
 - 익명 함수 : 변수에 할당하는 함수에 식별자가 없을 때

형식 `const 변수명 = function(){}; // 익명 함수`

`const 변수명 = function 식별자(){}; // 네이밍 함수`

10.2 함수를 정의하는 방법

10/02/anonymous_func.js

```
const gugudan = function(){  
  for(let i=1; i<=9; i++){  
    console.log(`3 * ${i} = ${3 * i}`);  
  }  
};  
gugudan(); // 함수 호출 문제없음
```

10/02/naming_func_call.js

```
const gugudan = function naming(){  
  for(let i = 1; i <= 9; i++){  
    console.log(`3 * ${i} = ${3 * i}`);  
  }  
};  
naming(); // 함수 호출
```

실행결과

ReferenceError: naming is not defined

10.2 함수를 정의하는 방법

3. 화살표 함수로 함수 정의하기

- ES6에서 추가된 함수 정의 방법
- 화살표를 사용해 함수를 정의하는 방법
- 익명 함수로만 정의할 수 있음

형식 `() => {};`

예

```
const gugudan = () => {  
  for(let i = 1; i <= 9; i++){  
    console.log(`3 * ${i} = ${3 * i}`);  
  }  
};  
gugudan();
```

10.3 함수 기능 확장하기

1. 매개변수와 인수

- 매개변수 : 함수가 호출될 때 전달받은 데이터를 할당하기 위해 함수에서 선언하는 변수
- 인수 : 정의한 함수를 호출할 때 전달하는 데이터

형식 // 함수 선언문

```
function 함수명(매개변수1, 매개변수2, ..., 매개변수N){}
```

// 함수 표현식

```
const 함수명 = function 식별자(매개변수1, ..., 매개변수N){};
```

// 화살표 함수

```
const 함수명 = (매개변수1, 매개변수2, ..., 매개변수N) => {};
```

// 함수 호출

```
함수명(인수1, 인수2, ..., 인수N);
```

10.3 함수 기능 확장하기

1. 매개변수와 인수 예제 실습

10/03/gugudan.js

```
function gugudan(① dan){  
  for(let i = 1; i <= 9; i++){  
    console.log(`${dan} * ${i} = ${dan * i}`);  
  }  
}  
  
gugudan(② 3); // 3단 출력  
gugudan(② 5); // 5단 출력  
gugudan(② 8); // 8단 출력
```

① 매개변수

② 인수

10.3 함수 기능 확장하기

2. 매개변수의 특징

- 데이터 전달
 - 함수를 호출하며 데이터를 전달해도 매개변수를 정의하지 않으면 데이터를 전달받지 못함(단, 오류가 발생하지는 않음)
 - 함수를 호출할 때 전달한 데이터와 매개변수는 일대일 매칭 관계가 형성
- 매개변수의 기본값 : undefined

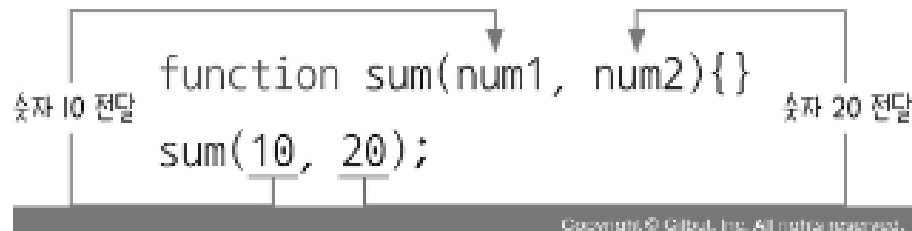



그림 10-1 데이터 전달 과정

10.3 함수 기능 확장하기

2. 매개변수의 특징

```
function sum(num1, num2){}
sum();
```



undefined undefined

그림 10-2 매개변수에 데이터를 전달하지 않는 경우

```
function sum(){}
sum(10, 20);
```



매개변수가 없어도 됩니다.
단, 전달받은 값을 활용할 수 없습니다.

그림 10-3 데이터를 전달하지만 매개변수가 없는 경우

10.3 함수 기능 확장하기

3. return 문

- 함수 내부에서 함수를 호출한 곳으로 데이터를 전달할 때

형식 `return` 식(또는 값)

예

```
function sum(num1, num2){  
  let result = num1 + num2;  
  return result;  
}  
  
const result = sum(10, 20);  
console.log("out: " + result); // out: 30
```

- 반환한다 : 함수 내부 변수인 `result`에 할당된 값, 즉 데이터가 `sum()` 함수를 호출한 곳으로 전달한다
- 반환값 : 반환된 데이터

- 화살표 함수에서 `}`를 생략하면 화살표 다음에 오는 코드는 `return` 문으로 처리됨

10.3 함수 기능 확장하기

3. return 문

10/03/add_func.js

```
function sum(num1, num2){  
  let result = num1 + num2;  
  console.log("inner: " + result);  
}  
sum(10, 20); // inner: 30
```

10/03/add_func.js

```
function sum(num1, num2){  
  let result = num1 + num2;  
}  
sum(10, 20);  
console.log("out:" + result); // ReferenceError: result is not defined
```

10.3 함수 기능 확장하기

3. 화살표 함수 return 문

- 화살표 함수에서 {}를 생략하면 화살표 다음으로 오는 코드는 return처리

10/03/return_arrow.js

```
const sum = (num1, num2) => num1 + num2;  
const result = sum(10, 20); // 30
```

10.4 함수의 특징 이해하기

1. 스코프

- 변수나 함수와 같은 참조 대상 식별자를 찾아내기 위한 규칙
- 함수 스코프
 - 함수에서 정의한 블록문만 스코프의 유효 범위로 인정하는 방식
 - 함수 내부는 지역 스코프, 함수 외부는 전역 스코프 영역이 됨
- 블록 스코프
 - {}로 구성된 블록문 기준으로 스코프의 유효 범위를 나누는 방식
 - **let과 const 키워드로 선언한 변수에 한해서만 적용**

10.4 함수의 특징 이해하기

- 전역 스코프

- 스코프와 상관없이 모두 참조 가능
- 전역 변수 : 전역 스코프에 선언한 변수

- 지역 스코프

- 함수 내부에 선언한 변수 a 는 함수 내부에서 참조 가능
- 지역 변수 : 지역 스코프에 선언한 변수

10.4 함수의 특징 이해하기

10/04/global.js

```
let a = 10; // 전역 스코프
function sum(){
  console.log(`함수 내부: ${a}`);
}
sum();
console.log(`함수 외부: ${a}`);
```

실행결과

함수 내부: 10

함수 외부: 10

```
function sum(){
  let a = 10; // 지역 스코프
  console.log(`함수 내부: ${a}`);
}
sum();
console.log(`함수 외부: ${a}`);
```

실행결과

함수 내부: 10

ReferenceError: a is not defined

10.4 함수의 특징 이해하기

10/04/block_scope.js

```
let a = 10;
{
  let b = 20;
  console.log(`코드 블록 내부 a: ${a}`);
  console.log(`코드 블록 내부 b: ${b}`);
}
console.log(`코드 블록 외부 a: ${a}`);
console.log(`코드 블록 외부 b: ${b}`);
```

실행결과

코드 블록 내부 a: 10

코드 블록 내부 b: 20

코드 블록 외부 a: 10

ReferenceError: b is not defined

10.4 함수의 특징 이해하기

10/04/block_scope2.js

```
var a = 10;
{
  var b = 20;
  console.log(`코드 블록 내부 a: ${a}`);
  console.log(`코드 블록 내부 b: ${b}`);
}
console.log(`코드 블록 외부 a: ${a}`);
console.log(`코드 블록 외부 b: ${b}`);
```

실행결과

코드 블록 내부 a: 10

코드 블록 내부 b: 20

코드 블록 외부 a: 10

코드 블록 외부 b: 20

10.4 함수의 특징 이해하기

- 참조 우선순위

- let, const 키워드는 같은 스코프 영역에서 중복 선언이 불가능
- 전역 스코프와 지역 스코프에 같은 식별자를 가지는 참조 대상이 있다면,
 - ✓ 먼저 같은 지역 스코프의 식별자를 참조
 - ✓ 같은 지역 스코프에서 참조할 식별자를 찾지 못할 때만 전역 스코프 참조

10/04/reference.js

```
let a = 10;
const b = 20;
function sum(){
  let a = 50;
  const b = 70;
  console.log(`함수 내부 a: ${a}`);
  console.log(`함수 내부 b: ${b}`);
}
sum();
```

실행결과

함수 내부 a: 50

함수 내부 b: 70

10.4 함수의 특징 이해하기

2. 함수 호이스팅

- 호이스팅 : 코드를 선언과 할당으로 나누었을 때, 선언부를 자신의 스코프 최상위로 끌어올리는 것
- 호이스팅의 대상
 - 함수 선언문
 - var 키워드를 사용한 함수 표현식
 - 화살표 함수 방식
- let이나 const 키워드로 선언했다면 호이스팅 자체가 되지 않음

10.4 함수의 특징 이해하기

2. 함수 호이스팅 예제

10/04/hoisting.js

```
console.log(num);  
var num = 10;
```



해석 방법

```
var num; // 선언부를 스코프 최상위로 끌어올림  
console.log(num); // undefined 출력  
num = 10;
```

10.4 함수의 특징 이해하기

2. 함수 호이스팅 예제(함수)

10/04/hoisting_func.js

```
printHello();  
function printHello(){  
  console.log("Hello");  
}
```



해석 방법

```
function printHello(){ // 함수 선언문을 최상위로 끌어올림  
  console.log("Hello");  
}  
printHello();
```

10.4 함수의 특징 이해하기

2. 함수 호이스팅 예제(함수 다른예제)

10/04/hoisting_express.js

```
printHello();  
var printHello = function printHello(){  
  console.log("Hello");  
}
```

실행결과

TypeError: printHello is not a function



해석 방법

```
var printHello;  
printHello();  
printHello = function printHello(){  
  console.log("Hello");  
}
```

10.5 즉시 실행 함수 사용하기

```
(function sum(a, b){  
    console.log(a + b);  
})(10, 20); // 30
```

- 즉시 실행 함수
 - 함수를 정의하면서 동시에 실행까지 하는 함수
 - 한 번 실행되고 나면 메모리에 데이터가 남아 있지 않음
 - 해당 식별자를 한 번도 사용되지 않은 것처럼 인식

형식 `(function(){})()`;

- 전역 스코프가 오염됐다 : 한 번만 사용할 함수인데, 식별자를 더 이상 사용할 수 없게 되었을 때

```
const init = function(){  
    console.log("initialized!");  
}
```

```
(function init(){  
    console.log("initialized!");  
})(); // initialized!  
init(); // ReferenceError: init is not define
```