세종사이버대학교

03

함수의 다양한 정의 방법

학습내용

01 함수의 정의 방법

02 함수 선언문 vs. 표현식

03 화살표 함수(ES6)

04 Function 생성자 함수

학습목표

- 자바스크립트에서 함수를 정의하는 다양한 방법(선언문, 표현식, 화살표 함수, 생성자 함수)을 비교하여 설명할 수 있다.
- 함수 선언문과 함수 표현식의 차이점(호이스팅, 실행 시점 등)을 예제를 통해 구분할 수 있다.
- 화살표 함수의 문법적 특징을 이해하고, 상황에 따라 적절히 활용할 수 있다.
- Function 생성자 함수를 사용하여 동적으로 함수를 생성하고 실행할 수 있다.

01

함수의 정의 방법

1) 함수의 정의(define) 방법 종류

선언 방식	예제 코드	특징	지원 버전
함수 선언식 (Function Declaration)	function sayHi() { }	• 호이스팅 됨 • 가장 전통적인 방식임	ES3 이상 (기본 지원)
함수 표현식 (Function Expression)	const sayHi = function() { }	• 호이스팅 안됨 • 코드의 유연성, 가독성 높음	ES3 이상 (기본 지원)
화살표 함수 (Arrow Function)	const sayHi = () => { }	• this 바인딩 없음 • 간결한 문법임	ES6 (2015) 이상
Function 생성자 (Function Constructor)	const sayHi = new Function('console.log("Hi")')	• 동적으로 새로운 함수를 생성가능 • 보안/성능 이슈가 있음	ES3 이상 (기본 지원)

02

함수 선언문 vs. 표현식

1) 코드의 문맥에 따른 JavaScript 엔진의 함수 해석

{}은 중의적 해석 가능

{}은 코드 블록일 수도 있고, 객체 리터럴({})일 수도 있음

- {}이 단독으로 존재
  - ➡ JavaScript 엔진은 {} 을 블록문으로 해석
- {}이 값으로 평가되어야 할 문맥에서 피연산자로 사용될 경우
  - ➡ JavaScript 엔진은 {}을 객체 리터럴로 해석

1) 코드의 문맥에 따른 JavaScript 엔진의 함수 해석

{ }은 중의적 해석 가능

```
//블록문으로 오해되는 경우
function wrong() {
  return
  {
    message: “Oops”
  };
}
// 출력: undefined
console.log(wrong());
```

{ }는 블록문으로 해석되고, return은
세미콜론 자동 삽입(ASI)에 의해 줄에서
끝나버림 ➡ 실제로는 return;만 실행되고
{ message: “Oops” }는 무시됨

```
//권장코드
function safe() {
  return (
    { message: “safe return” }
  );
}
```

객체 리터럴로 안전하게 해석하려면?

- return과 객체 {} 사이에 줄바꿈 하지 않기
- return ({ ... })처럼 괄호로 감싸기

1) 코드의 문맥에 따른 JavaScript 엔진의 함수 해석

JavaScript 엔진은 코드의 문맥에 따라, **동일한 함수 리터럴을 함수 표현식 or 함수 선언문**으로 해석하는 경우가 있음

함수 표현도 중의적 해석 가능

- 함수 리터럴이 **단독**으로 사용
  - ➡ 함수 선언문(값으로 평가되지 않고, 실행만 되는 문장)으로 해석
- 함수 리터럴이 값처럼 평가되어야 하는 문맥

  예 **변수에 할당, 인자로 전달, 연산자 오른쪽** 등

  - ➡ 함수 리터럴 표현식(값으로 평가될 수 있는 문장)으로 해석

1) 코드의 문맥에 따른 JavaScript 엔진의 함수 해석

JavaScript 엔진은 코드의 문맥에 따라, **동일한 함수 리터럴을 함수 표현식 or 함수 선언문**으로 해석하는 경우가 있음

```
// function sayHello()는 문장의 //
시작이고, 선언문으로 해석됨

function sayHello() {
  console.log("Hello");
}

sayHello(); // 실행됨
```

함수 선언문으로는 해석 되는 경우

➡ 호이스팅(hoisting)되어 함수 정의 이전에도 호출 가능

```
//변수에 할당시
const greet = function() {
  console.log("Hi");
};
greet(); // 실행됨

// 괄호로 감싸서 피연산자로 사용 시
(function () {
  console.log("IIFE 실행");
})();
```

함수 표현식으로 해석되는 경우

➡ 호이스팅되지 않음 (선언 이후에만 호출 가능)

“함수가 생성되는 것은 동일하나, 호출방식에서 차이가 남”

```
// 1)함수 선언문으로 함수 호출
function foo() {
  console.log(“foo”); // foo
}
foo();

// 2) 함수 리터럴 표현식으로 함수 호출
(function bar() {
  console.log(“bar”); // ReferenceError: bar is not defined
});
bar(); // 호출 불가능 ➡ 함수 리터럴의 이름은 함수 내부에서만 호출 가능한 식별자이므로 외부에서 함수이름으로 호출할 수 없다.
```

- 그룹연산자() 내에 있는 함수는 함수 선언문이 아닌, 함수 리터럴 표현식으로 해석됨
- 함수 리터럴에서는 함수 이름을 생략할 수 있음(익명함수)

2) 함수 리터럴 표현식으로 함수 호출 시 메모리 구조

함수 리터럴 표현식으로 함수 호출 시 메모리 구조

함수 리터럴인 bar는 함수 몸체 내에서만 참조할 수 있는 식별자이므로 외부에서는 호출할 수 없음

함수 선언문으로 함수 호출 시 메모리 구조

생성된 함수 객체를 가리키는 유효한 식별자가 필요

➡ JavaScript 엔진은 생성된 함수를 호출하기 위해 함수 이름과 동일한 이름의 식별자를 암묵적으로 생성하고, 거기에 함수 객체를 할당

참고 https://github.com/youngminss/Docs-modernJS__deepDive/tree/master/09-Function

03

화살표 함수(ES6)

1) ES6(2015)에 도입된 간결한 함수 표현식

- function 키워드 없이 **간결하게** '기호'로 함수 정의

- 내부의 this, arguments, super를 상속(Lexical binding)
  - this를 자동으로 바인딩(정적 바인딩)

```
const add =  (a, b) => {
  return a + b;
};

//축약형 (한 줄 반환 시)
const square = x => x * x ;
```

축약형

04

Function 생성자 함수

1) Function 생성자 함수란?

- 문자열을 받아서 새로운 함수를 동적으로 생성
  - ➡ 보안상 위험할 수 있으며, 거의 사용되지 않음
- new Function(arg1, arg2, ..., body) 형식
- 내부적으로 eval()과 유사하게 동작해
  문자열로 된 JavaScript 코드를 받아 실행하는 함수
  - ➡ eval("표현식"): → 표현식을 실행하여 그 결과 값을 반환

1) Function 생성자 함수란?


```
//기본 코드
const sum = new Function('a', 'b', 'return a + b');
console.log(sum(2, 3)); // 출력: 5

// 동적 코드 생성 예
const args = ['x', 'y'];
const body = 'return x * y;';
const multiply = new Function(...args, body)
console.log(multiply(4, 5)); // 출력:20
```

문자열로 받은 코드를 런타임에 동적으로 함수로 만들 수 있음

2) Function 생성자 함수의 활용

Function 생성자 함수의 필요성

이유	설명	예시
동적으로 함수 생성	런타임에 문자열로 함수를 만들 수 있어, 사용자가 입력한 내용 등을 바탕으로 실시간 코드 생성 가능	new Function(“a”, “b”, “return a + b;”)
컴파일 타임이 아닌 런타임 생성	함수 선언문/표현식은 코드를 작성할 때(정적) 정의되지만, 생성자 함수는 코드 실행 중에 동적으로 생성	사용자가 입력한 수식 “x + y”를 계산하는 함수로 생성 가능
전역 스코프에서 실행	생성자 함수는 항상 전역 스코프에서 평가되기 때문에 특정한 글로벌 컨텍스트를 필요로 할 때 사용 가능	Function(“console.log(this)”) ➡ 전역에서 실행

eval보다 안전하고 명확	eval()처럼 문자열을 코드로 실행하지만, Function()은 함수로 제한되므로 더 예측 가능하고 안전	Function(“return 2 + 2”)() // 지역변수 접근 불가
		➡ eval(“2 + 2”)보다 범위 제한

2) Function 생성자 함수의 활용

- 주의 사항

  - 생성자 함수는 런타임에 파싱되므로 성능이 떨어짐

  - 디버깅 어려움

    ➡ 코드가 문자열이므로, IDE 자동완성, 정적 분석 등이 불가능

  - 사용자가 입력한 문자열을 직접 함수로 만들면 XSS(코드 삽입 공격) 위험이 발생함

학습 평가

Q1

다음 중 JavaScript에서 함수의 목적에 가장 부합하는 설명은 무엇인가?

1. 데이터를 나열하기 위해
2. 값을 보관하기 위해
3. 재사용 가능한 동작을 정의하기 위해
4. 변수명을 짓기 위해

학습 평가

Q1

다음 중 JavaScript에서 함수의 목적에 가장 부합하는 설명은 무엇인가?

1. 데이터를 나열하기 위해
2. 값을 보관하기 위해
3. 재사용 가능한 동작을 정의하기 위해
4. 변수명을 짓기 위해

정답 3

해설 함수는 여러 번 사용할 수 있는 동작(기능)을 정의하고 모듈화하는 데 사용됩니다.

학습 평가

Q2

다음 중 JavaScript 함수의 선언문으로 올바른 것은 무엇인가?

1. function sayHello() { console.log("Hi"); }
2. const function = sayHello() => {};
3. let = function sayHello {}
4. function: sayHello() {}

학습 평가

Q2

다음 중 JavaScript 함수의 선언문으로 올바른 것은 무엇인가?

1. function sayHello() { console.log("Hi"); }
2. const function = sayHello() => {};
3. let = function sayHello {}
4. function: sayHello() {}

정답 1

해설 함수 선언문은 function 함수이름() 형식이 올바릅니다.

학습 평가

Q3 함수에서 값을 반환하는 키워드는 무엇인가?

1. return
2. output
3. yield
4. result

학습 평가

Q3 함수에서 값을 반환하는 키워드는 무엇인가?

- ☑ return
- 2 output
- 3 yield
- 4 result

정답 1

해설 JavaScript 함수는 return을 통해 값을 반환합니다.

학습 평가

Q4

함수 호출과 매개변수 관련 설명 중 옳은 것은 무엇인가?

1. 함수는 매개변수를 반드시 받아야 실행된다.
2. 함수 호출 시 인자 개수가 다르면 오류가 난다.
3. 함수는 호출된 위치에서만 정의되어야 한다.
4. 함수는 호출할 때 인자를 전달해 동작을 지정할 수 있다.

학습 평가

Q4

함수 호출과 매개변수 관련 설명 중 옳은 것은 무엇인가?

1. 함수는 매개변수를 반드시 받아야 실행된다.
2. 함수 호출 시 인자 개수가 다르면 오류가 난다.
3. 함수는 호출된 위치에서만 정의되어야 한다.
4. ✓ 함수는 호출할 때 인자를 전달해 동작을 지정할 수 있다.

정답 4

해설 함수 리터럴을 사용하면 호출 시 인자를 통해 그 동작을 지정할 수 있다.

학습 평가

Q5

다음 중 표현식(Expression)이 아닌 문(Statement)은 무엇인가?

1. const x = 3 + 5;
2. “Hello” + “World”
3. let y;
4. x 〉 10

학습 평가

Q5

다음 중 표현식(Expression)이 아닌 문(Statement)은 무엇인가?

1. const x = 3 + 5;
2. “Hello” + “World”
3. let y;
4. x 〉 10

정답 3

해설 let y;는 값을 생성하지 않는 선언문입니다.

학습 평가

Q6 화살표 함수로 올바른 선언은 무엇인가?

1. const hi = => { console.log(“Hi”); }
2. const hi = () => { console.log(“Hi”); }
3. () => { const hi = console.log(“Hi”); }
4. const => () hi { console.log(“Hi”); }

학습 평가

Q6 화살표 함수로 올바른 선언은 무엇인가?

1. const hi = => { console.log("Hi"); }
2. const hi = () => { console.log("Hi"); }
3. () => { const hi = console.log("Hi"); }
4. const => () hi { console.log("Hi"); }

정답 2

해설 ()는 매개변수가 없음을 의미하며, =>는 화살표 함수 문법입니다.

학습 평가

Q7

다음 중 표현식이 아닌 문은 무엇인가?

1. 3 + 4
2. x = 10
3. if (x > 0) { console.log(x); }
4. “Hi” + “There”

학습 평가

Q7 다음 중 표현식이 아닌 문은 무엇인가?

1. 3 + 4
2. x = 10
3. ✓ if (x > 0) { console.log(x); }
4. “Hi” + “There”

정답 3

해설 if는 제어 흐름을 위한 문(Statement)입니다.
값은 반환하지 않습니다.

학습 평가

Q8 다음 중 '리터럴'이 아닌 것은 무엇인가?

1. "Hello"
2. 100
3. true
4. let a = 5;

학습 평가

Q8 다음 중 '리터럴'이 아닌 것은 무엇인가?

1. "Hello"
2. 100
3. true
4. let a = 5;

정답 4

해설 변수 선언문 자체는 리터럴이 아니며, 값의 저장 방식입니다.

학습 평가

Q9

함수 표현식과 함수 선언문의 가장 큰 차이점은 무엇인가?

1. 선언문은 호이스팅되고, 표현식은 그렇지 않다.
2. 표현식은 항상 더 빠르다.
3. 둘 다 결과적으로 동일하다.
4. 선언문은 const로만 정의할 수 있다.

학습 평가

Q9

함수 표현식과 함수 선언문의 가장 큰 차이점은 무엇인가?

- ✓ 선언문은 호이스팅되고, 표현식은 그렇지 않다.
- 2 표현식은 항상 더 빠르다.
- 3 둘 다 결과적으로 동일하다.
- 4 선언문은 const로만 정의할 수 있다.

정답 1

해설 함수 선언문은 코드 해석 시 메모리에 먼저 올라가지만, 표현식은 그렇지 않습니다.

학습 평가

Q10 다음 코드의 출력 결과는 어떻게 되는가?

```
const add = function(x, y) { return x + y; };
console.log(add(2, 3));
```

1. 5
2. undefined
3. “2 + 3”
4. NaN

학습 평가

Q10 다음 코드의 출력 결과는 어떻게 되는가?

```
const add = function(x, y) { return x + y; };
console.log(add(2, 3));
```

- ✓ 1 5
- 2 undefined
- 3 “2 + 3”
- 4 NaN

정답 1

해설 add(2, 3)은 2와 3을 더한 값인 5를 반환합니다.

학습정리

함수의 개념과 선언 방식

- 함수란?
  - 입력을 받아 작업을 수행하고 결과를 반환하는 코드 블록
  - function greet() { return “Hi”; }
- 함수 호출
  - 정의한 함수를 실행
  - greet(); //함수의 원형만 필요

학습정리

함수의 개념과 선언 방식

- 매개변수/인수
  - 매개변수 : 함수 선언 시 전달값
  - 입력인수 : 호출 시 전달값
  - 예시: function f(x) { } / f(3);
- return 키워드
  - 결과값을 함수 밖으로 전달
  - return x + y;

학습정리

함수의 개념과 선언 방식

- 함수 선언 방식 : 다양한 형식의 함수 정의

구분	특징	예시
선언문(Declaration)	호이스팅 가능/ 가장 기본	function hi() {}
표현식(Expression)	호이스팅 안됨 / 변수에 저장	const hi = function() {};
화살표 함수(Arrow)	간결, this 없음	const hi = () => {};
Function 생성자	동적 생성 / 일반적으로 지양	new Function('x', 'y', 'return x + y');

학습정리

값, 리터럴, 표현식, 문

구분	특징	예시
값(Value)	변수에 저장될 수 있는 데이터	10, “hello”, true, [1,2,3]
리터럴(Literal)	코드에 직접 적힌 값	5, “abc”, false, []
표현식(Expression)	하나의 값을 만들어내는 코드	3 + 4, “a” + “b”, x > 5
문(Statement)	실행 가능한 코드 단위, 명령문	if, for, let x = 10;
표현식인 문	표현식이면서 동시에 실행 가능한 문	let y = 2 * 3;
표현식이 아닌 문	값은 만들지 않지만 흐름 제어 담당	if (x > 0) { ... }, while (...) {}

세종사이버대학교

05주. 나만의 기능 만들기 : 여러 가지 함수

01

스코프(Scope)