

12장 함수

모던 자바스크립트 Deep Dive



목차

1. 함수란
2. 함수 리터럴
3. 함수 정의
4. 함수 호출
5. 참조에 의한 전달과 외부 상태의 변경
6. 다양한 함수의 형태

1. 함수란

함수: 일련의 과정을 문(statement)으로 구현하고 코드 블록으로 감싸서 하나의 실행 단위로 정의한 것

```
function add(x, y) {  
    return x + y;  
}  
add(2, 5);
```

매개변수

반환값

인수

1. 함수란

함수: 일련의 과정을 문(statement)으로 구현하고 코드 블록으로 감싸서 하나의 실행 단위로 정의한 것

함수 정의	[function add(x, y) { return x + y; }
함수 호출	[add(2, 5);

2. 함수 리터럴

리터럴: 사람이 이해할 수 있는 문자 또는 약속된 기호를 사용해 값을 생성하는 표기 방식

```
let f = function add (x, y) {  
  return x + y;  
};
```

함수 선언문과 같이 function 키워드, 함수 이름, 매개변수 목록, 함수 몸체로 구성됨

2. 함수 리터럴

함수 이름

- 함수 몸체 내에서만 참조 가능
- 이름이 있는 함수 -> 기명 함수
- 이름이 없는 함수 -> 익명(무명) 함수

3. 함수 정의

- 함수 선언문
- 함수 표현식
- Function 생성자 함수
- 화살표 함수(ES6)

3. 함수 정의 - 함수 선언문

```
function add(x, y) {  
    return x + y;  
}
```


3. 함수 정의 - 함수 선언문

함수 리터럴과 달리 함수 이름 생략 불가능

```
function (x, y) {  
    return x + y;  
}
```

```
// SyntaxError: Function statements require  
a function name
```

3. 함수 정의 - 함수 선언문

함수 선언문은 '표현식이 아닌 문'

```
function (x, y) {  
    return x + y;  
}
```

```
// SyntaxError: Function statements require  
a function name
```

3. 함수 정의 - 함수 선언문

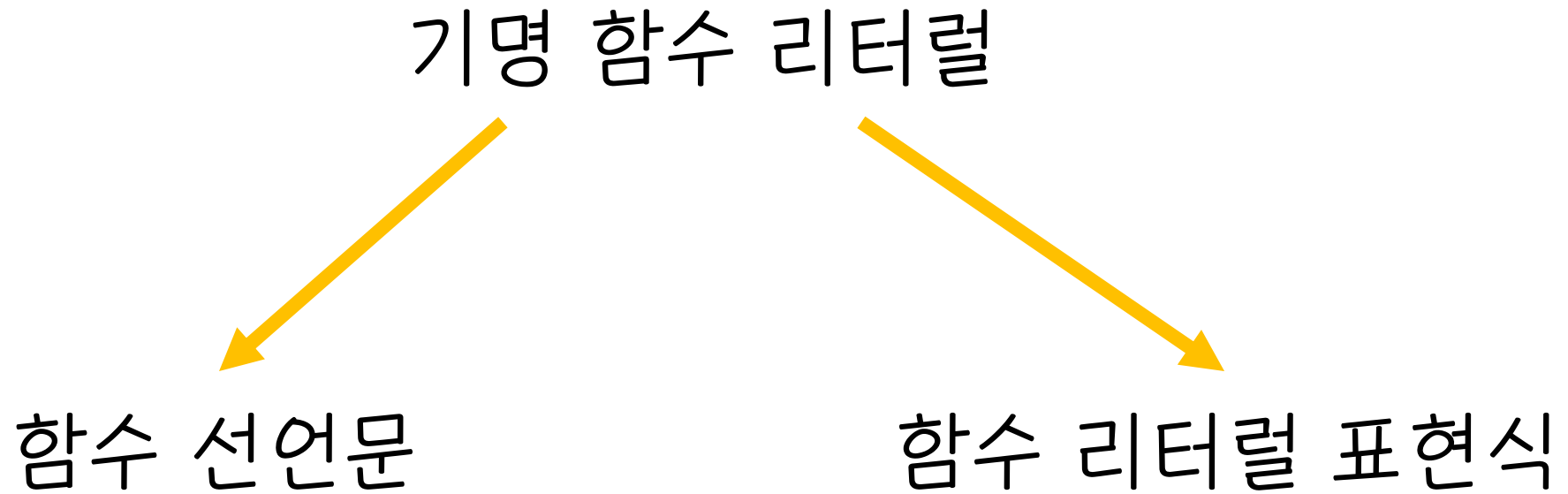
함수 선언문은 '표현식이 아닌 문' -> 변수에 할당할 수 X

```
let add = function add(x, y) {  
    return x + y;  
};  
add(2, 5);
```

자바스크립트 엔진이 코드의 문맥에 따라 다르게 해석

3. 함수 정의 - 함수 선언문

자바스크립트 엔진이 코드의 문맥에 따라 다르게 해석



3. 함수 정의 - 함수 선언문

- 함수 선언문

```
function foo() {console.log('foo');}
```

```
foo();
```

```
// foo
```

- 함수 리터럴 표현식

```
(function bar() {console.log('bar');});
```

```
bar();
```

```
// ReferenceError: bar is not defined
```

3. 함수 정의 - 함수 선언문

- 함수 리터럴 표현식

```
(function bar() {console.log('bar');});  
bar();  
// ReferenceError: bar is not defined
```

함수 몸체 외부에서 함수 이름으로 함수 호출 불가

3. 함수 정의 - 함수 선언문

- 함수 선언문

```
function foo() {console.log('foo');}  
foo();  
// foo
```

자바스크립트 엔진이 함수 이름과 동일한 이름의 식별자를 암묵적으로 생성

3. 함수 정의 - 함수 선언문

함수는 함수이름으로 호출하는 것이 아니라, 함수 객체를 가리키는 식별자로 호출함

식별자 함수 이름

```
let add = function add(x, y) {  
    return x + y;  
};  
add(2, 5);
```

식별자

3. 함수 정의 - 함수 표현식

함수 객체를 변수에 할당

```
let add = function add(x, y) {  
    return x + y;  
};
```

3. 함수 정의 - 함수 표현식

```
let add = function foo(x, y) {  
    return x + y;  
};
```

```
console.log(add(2, 5));  
// 7
```

```
console.log(foo(2, 5));  
// ReferenceError: foo is not defined
```

3. 함수 정의 - 함수 생성 시점과 함수 호이스팅

- 함수 선언문으로 정의한 함수는 함수 선언문 이전에 호출 가능
- 함수 표현식으로 정의한 함수는 함수 표현식 이전에 호출 불가능

➡ 함수의 생성 시점이 다르기 때문

3. 함수 정의 - 함수 생성 시점과 함수 호이스팅

함수 선언문

- 함수 호이스팅
- 런타임 이전에 식별자 생성
- 식별자는 함수 객체로 초기화

함수 표현식

- 변수 호이스팅
- 런타임 이전에 식별자 생성
- var 키워드로 생성된 변수는 undefined로 초기화

3. 함수 정의 - Function 생성자 함수

바람직하지 않은 방식

```
let add =  
    new Function('x', 'y', 'return x + y');
```

3. 함수 정의 - 화살표 함수

```
let add = (x, y) => x + y;
```

4. 함수 호출 - 매개변수와 인수

값을 함수 외부에서 내부로 전달할 때 -> 매개변수를 통해 인수를 전달

매개변수

```
function add(x, y) {  
    return x + y;  
}
```

인수 전달

```
let result = add(2, 5);
```

인수

The diagram illustrates the process of passing arguments to a function. A yellow box highlights the parameters 'x' and 'y' in the function definition 'function add(x, y)'. Another yellow box highlights the arguments '2' and '5' in the function call 'let result = add(2, 5);'. A yellow arrow originates from the arguments box and points to the parameters box, with the text '인수 전달' (Argument Passing) written next to it. The word '매개변수' (Parameter) is written above the parameters box, and '인수' (Argument) is written below the arguments box.

4. 함수 호출 - 매개변수와 인수

매개변수

- 매개변수에 인수 순서대로 할당
- 함수 몸체 내부에서 변수와 동일하게 취급
- 함수 몸체 내부에서만 참조 가능

4. 함수 호출 - 매개변수와 인수

함수는 매개변수의 개수와 인수의 개수가 일치하는지 체크하지 x

```
function add(x, y) {  
    return x + y;  
}  
console.log(add(2));  
// NaN
```

인수가 할당되지 않은 매개변수의 값은 undefined

4. 함수 호출 - 매개변수와 인수

함수는 매개변수의 개수와 인수의 개수가 일치하는지 체크하지 x

```
function add(x, y) {  
    return x + y;  
}  
console.log(add(2, 5, 10));  
// 7
```

매개변수보다 인수가 더 많은 경우 초과된 인수는 무시됨

4. 함수 호출 - 매개변수와 인수

매개변수 기본값 사용


```
function add(a = 0, b = 0, c = 0) {  
    return a + b + c;  
}
```

4. 함수 호출 - 매개변수와 인수

- 매개변수의 개수는 적을수록 좋음
- 이상적인 함수는 한 가지 일만 해야 하며 가급적 작게 만들어야 함

5. 참조에 의한 전달과 외부 상태의 변경

```
function change(a, b) {  
  a += 100;  
  b.name = 'Kim';  
}
```

 a는 원시 값 전달받고 b는 객체 전달받음

```
let num = 100;  
let person = { name: 'Lee' };
```

5. 참조에 의한 전달과 외부 상태의 변경

원시 값은 값 자체가 복사되어 전달

```
change(num, person);
```

객체는 참조 값이 복사되어 전달

```
console.log(num);
```

```
//100 → 원본 훼손되지 x
```

```
console.log(person);
```

```
// { name: "Kim" } → 원본 훼손됨
```

6. 다양한 함수의 형태 - 즉시 실행 함수

함수 정의와 동시에 즉시 호출되는 함수

```
(function () {  
    let a = 3;  
    let b = 5;  
    return a * b;  
})();
```

6. 다양한 함수의 형태 - 즉시 실행 함수

일반 함수처럼 인수를 전달할 수 있음

```
let res = (function (a, b) {  
    return a * b;  
})(3, 5);  
  
console.log(res);  
// 15
```


6. 다양한 함수의 형태 - 재귀함수

자기 자신을 호출하는 행위 (재귀 호출)를 수행하는 함수

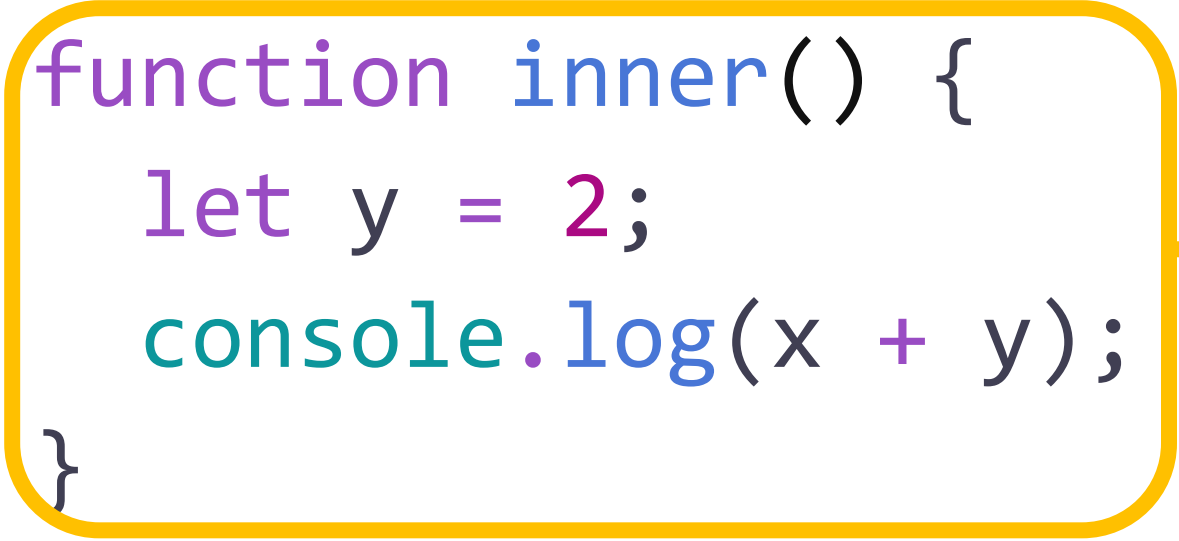
```
function countdown(n) {  
  if (n > 0) return;  
  console.log(n);  
  countdown(n - 1);  
}  
countdown(10);
```

6. 다양한 함수의 형태 - 재귀함수

- 자신을 무한 재귀 호출 -> 재귀 호출을 멈출 수 있는 **탈출 조건** 필요
- 반복문보다 재귀 함수가 더 직관적으로 이해하기 쉬울 때만 사용

6. 다양한 함수의 형태 - 중첩 함수 (내부 함수)

```
function outer() {  
  let x = 1;  
  function inner() {  
    let y = 2;  
    console.log(x + y);  
  }  
  inner();  
}  
outer();
```



중첩 함수

6. 다양한 함수의 형태 - 중첩 함수 (내부 함수)

- 함수 내부에 정의된 함수는 중첩 함수 (내부 함수)
- 중첩 함수를 포함하는 함수는 외부 함수
- If문이나 for문 등의 코드 블록에서 함수를 정의하는 것은 바람직 x

6. 다양한 함수의 형태 - 콜백함수

```
function repeat(n, f) {  
  for (let i = 0; i < n; i++) {  
    f(i);  
  }  
}
```

`f(i);` → `i`를 전달하면서 `f`를 호출

```
let logAll = function (i) {  
  console.log(i);  
};
```

`repeat(5, logAll);` → 반복 호출할 함수를 인수로 전달

6. 다양한 함수의 형태 - 콜백함수

```
function repeat(n, f) {  
  for (let i = 0; i < n; i++) {  
    f(i);  
  }  
}
```

→ 고차 함수

```
let logAll = function (i) {  
  console.log(i);  
};
```

→ 콜백 함수

6. 다양한 함수의 형태 - 콜백함수

콜백 함수를 익명 함수 리터럴로 정의하면서 곧바로 고차 함수에 전달

```
repeat(5, function (i) {  
    if (i % 2) console.log(i);  
});
```

6. 다양한 함수의 형태 - 순수 함수와 비순수 함수

- 순수 함수: 부수 효과가 없는 함수
- 비순수 함수: 부수 효과가 있는 함수

6. 다양한 함수의 형태 - 순수 함수와 비순수 함수

- 순수 함수 -> 외부 상태에 의존 x, 외부 상태 변경 x

```
let count = 0;
```

```
function increase(n) {  
    return ++n;  
}
```

함수 increase는 동일한 인수가 전달되면 언제나 동일한 값 반환

6. 다양한 함수의 형태 - 순수 함수와 비순수 함수

- 비순수 함수 -> 외부 상태에 의존 o, 외부 상태 변경 o

```
let count = 0;
```

```
function increase() {  
    return ++count;  
}
```

외부 상태인 count가 increase 함수에 의해 변화