05

함수

목차

1. 함수 생성
2. 함수의 활용
3. 함수 매개 변수 초기화
4. 콜백 함수
5. 표준 내장 함수

함수

● 함수 정의

```
function 〈함수 이름〉(〈매개 변수〉) {
    〈함수 코드〉
    return 〈리턴 값〉
}
```

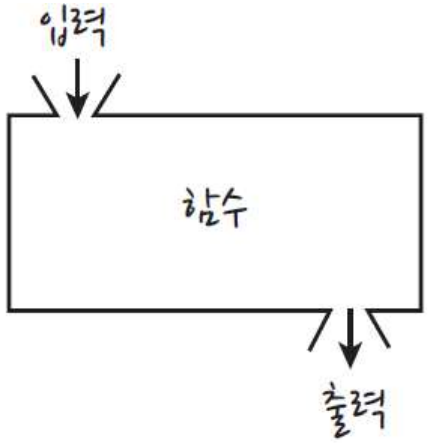


그림 5-4 함수 상자

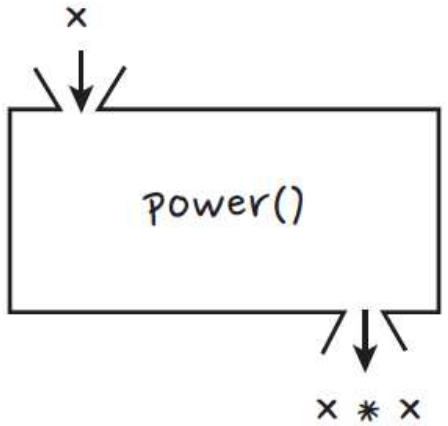
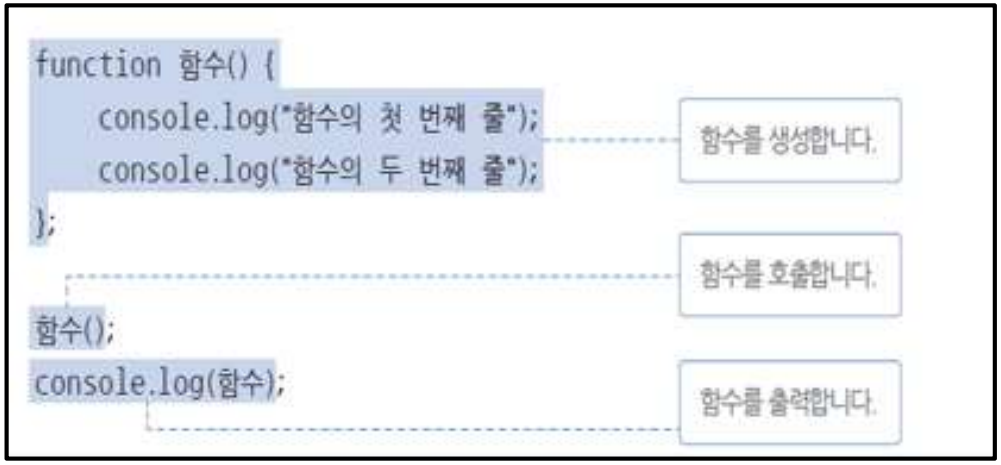


그림 5-5 power() 메소드

함수 생성

- **선언적 함수**
  - 이름을 붙여 함수를 생성

```
function <함수 이름>() { }
```



함수 생성

익명 함수

- 이름을 붙이지 않고 함수 생성
- 함수를 호출하면 함수 내부의 코드 묶음이 모두 실행

```
let 〈함수 이름〉 = function () { };
```

```
let 함수 = function () {
    console.log("함수의 첫 번째 줄");
    console.log("함수의 두 번째 줄");
};

함수();
console.log(함수);
```

함수를 생성합니다.

함수를 호출합니다.

함수 자체를 출력합니다.

익명 함수와 선언적 함수

- 익명 함수와 선언적 함수의 생성 순서

```
let 변수;
변수 = 10;
변수 = 20;

console.log(변수)
```

```
let 함수;
함수 = function () { console.log("첫 번째 함수"); };
함수 = function () { console.log("두 번째 함수"); };

함수();
```

함수 생성

- **화살표 함수[ECMAScript6]**

```
() => { }
```

  - '하나의 표현식을 리턴하는 함수'를 만들 때는 중괄호 생략 가능

```
let 함수 = () => {
    console.log("함수의 첫 번째 줄");
    console.log("함수의 두 번째 줄");
};

함수();
console.log(함수);
```

함수를 생성합니다.

함수를 호출합니다.

함수 매개 변수 초기화

- **함수 정의**
  - 모든 함수 가변인자를 가지는 함수로 정의 됩니다
  - 함수가 실행될 때 실행 컨텍스트에서는 함수 내부에 arguments 타입의 속성이 생성되고 함수 호출시 전달한 파라미터들이 저장 됩니다.
  - 함수에 정의된 매개변수의 개수보다 많은 매개변수로 호출하면 실행시에 무시
  - 함수에 정의된 매개변수의 개수보다 적은 매개변수로 호출하면 undefined로 전달

```
// 함수를 선언합니다.
function print(name, count) {
    console.log(`${name}이/가 ${count}개 있습니다.`)
}

// 함수를 호출합니다.
print("사과", 10);
print("사과");
```

함수 매개 변수 초기화

- 함수 매개 변수 초기화

```
// 함수를 선언합니다.
function print(name, count) {
    // 함수 매개 변수 초기화
    count = count || 1;

    // 함수 본문
    console.log(`${name}이/가 ${count}개 있습니다.`)
}

// 함수를 호출합니다.
print("사과");
```

콜백 함수

콜백 함수

- 함수를 매개변수로 전달하고, 함수를 리턴하는 함수를 정의할 수 있습니다 .
- 콜백 함수(callback function) : 다른 함수에 인자로 전달되어 특정 이벤트 또는 조건이 발생했을 때 호출되는 함수
  비동기적인 작업에서 많이 사용되며, 이벤트 처리, HTTP 요청, 타이머 설정 등 다양한 상황에서 활용됩니다.

```
// 함수를 선언합니다.
function callTenTimes(callback) {
    // 10회 반복합니다.
    for (let i = 0; i < 10; i++) {
        // 매개 변수로 전달된 함수를 호출합니다.
        callback();
    }
}

// 변수를 선언합니다.
callTenTimes(function () {
    console.log('함수 호출');
});
```

표준 내장 함수

- 표준 내장 함수
  - 자바스크립트에서 기본적으로 지원하는 함수
  - Global 객체의 메서드

함수	설명
parseInt()	문자열을 정수로 변환합니다.
parseFloat()	문자열을 실수로 변환합니다.

표준 내장 함수

- 표준 내장 함수

```
// 변수를 선언합니다.
let inputA = "52";
let inputB = "52.273";
let inputC = "1401동"

// parseInt() 함수의 기본적인 사용
console.log(parseInt(inputA))

// parseInt() 함수와 parseFloat() 함수의 차이
console.log(parseInt(inputB))
console.log(parseFloat(inputB))

// 문자열 뒤에 숫자가 아닌 문자가 포함되어 있을 때
console.log(parseInt(inputC));
```

표준 내장 함수

타이머 함수

- '특정 시간 후에' 또는 '특정 시간마다' 어떤 일을 할 때 사용
- 시간은 밀리초로 지정. 1초를 나타내려면 1000(밀리초)을 입력
- window(브라우저 최상위 객체)의 메서드

함수	설명
setTimeout(함수, 시간)	특정 시간 후에 함수를 실행합니다.
setInterval(함수, 시간)	특정 시간마다 함수를 실행합니다.

표준 내장 함수

타이머 함수

```
// 1초 후에
setTimeout(function () {
    console.log("1초가 지났습니다.");
}, 1000);

// 1초마다
setInterval(function () {
    console.log("1초 마다 호출됩니다.");
}, 1000);
```

- 종료 : Ctrl + C

표준 내장 함수

타이머 함수

- clearInterval( ) 함수

함수	설명
clearInterval(아이디)	특정 시간마다 실행하던 함수 호출을 정지합니다.

```
// 1초마다
let id = setInterval(function () {
    console.log("출력합니다.");
}, 1000);

// 3초 후에
setTimeout(function () {
    // 타이머를 제거합니다.
    clearInterval(id);
}, 3000);
```

익명 함수와 선언적 함수

```
함수 = function () { console.log("첫 번째 함수"); };
function 함수() { console.log("두 번째 함수"); };

함수();
```

예제 A	예제 B
함수 = function () { console.log("1"); }; 함수 = function () { console.log("2"); }; 함수();	함수 = function () { console.log("1"); }; function 함수() { console.log("2"); }; 함수();
예제 C	**예제 D**
function 함수() { console.log("1"); }; 함수 = function () { console.log("2"); }; 함수();	function 함수() { console.log("1"); }; function 함수() { console.log("2"); }; 함수();

익명 함수와 선언적 함수

- **익명 함수와 화살표 함수의 차이**
  - 내부에서 this 키워드가 가지는 의미

```
// 익명 함수 생성 후 곧바로 호출
(function () {
    console.log(this);
})();
```

```
// 화살표 함수 생성 후 곧바로 호출
(() => {
    console.log(this);
})();
```

함수 특성

- **콜백 함수(callback function)**
  - 함수를 매개변수로 전달하고, 함수를 리턴하는 함수를 정의 가능
  - 콜백 함수(callback function) : 다른 함수에 인자로 전달되어 특정 이벤트 또는 조건이 발생했을 때 호출되는 함수
  - 비동기적인 작업에서 많이 사용되며, 이벤트 처리, HTTP 요청, 타이머 설정 등 다양한 상황에서 활용됩니다.

```
function caller(param){
   for(var i=0;i<5;i++){
       param();
    }
}
function callee(){
    alert("callee");
}

caller(callee);//전달된 callee함수는 콜백함수입니다
```

클로저 함수

- **클로저(Closure)**
  - "A closure is the combination of a function and the lexical environment within which that function was declared."
  - 클로저는 함수와 그 함수가 선언됐을 때의 렉시컬 환경(Lexical environment)과의 조합이다.
  - 자신을 포함하고 있는 외부함수보다 내부함수가 더 오래 유지되는 경우, 외부 함수 밖에서 내부함수가 호출되더라도 외부함수의 지역 변수에 접근할 수 있는 함수

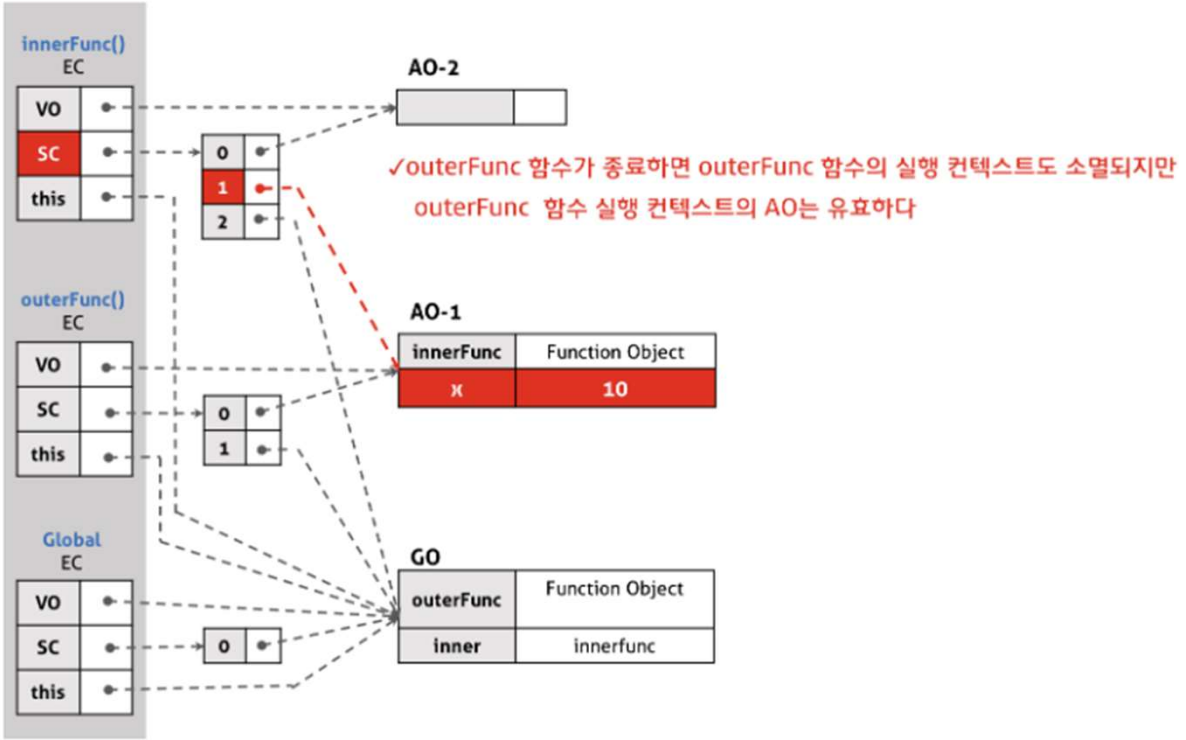
```
function outerFunc() {
  var x = 10;
  var innerFunc = function () { console.log(x); };
  return innerFunc;
}

var inner = outerFunc();
inner(); // 10
```

  - 클로저는 반환된 내부함수가 자신이 선언됐을 때의 환경(Lexical environment)인 스코프를 기억하여 자신이 선언됐을 때의 환경(스코프) 밖에서 호출되어도 그 환경(스코프)에 접근할 수 있는 함수
  - 클로저는 자신이 생성될 때의 환경(Lexical environment)을 기억하는 함수

클로저 함수

- **클로저(Closure)**
  - 클로저에 의해 참조되는 외부함수의 변수 즉 outerFunc 함수의 변수 x를 자유변수(Free variable)라고 부른다.
  - 클로저라는 이름은 자유변수에 함수가 닫혀있다(closed)라는 의미이며 자유변수에 엮여있는 함수라는 뜻이다.



실행 컨텍스트의 활성 객체(Activation object)와 클로저

클로저 함수

- **클로저(Closure)**
  - 클로저는 현재 상태를 기억하고 이 상태가 변경되어도 최신 상태를 유지해야 하는 상황에 매우 유용하다.
  - 자바스크립트에 클로저라는 기능이 없다면 상태를 유지하기 위해 전역 변수를 사용할 수 밖에 없다.
  - 전역 변수는 언제든지 누구나 접근할 수 있고 변경할 수 있기 때문에 많은 부작용을 유발해 오류의 원인이 되므로 사용을 억제해야 한다.

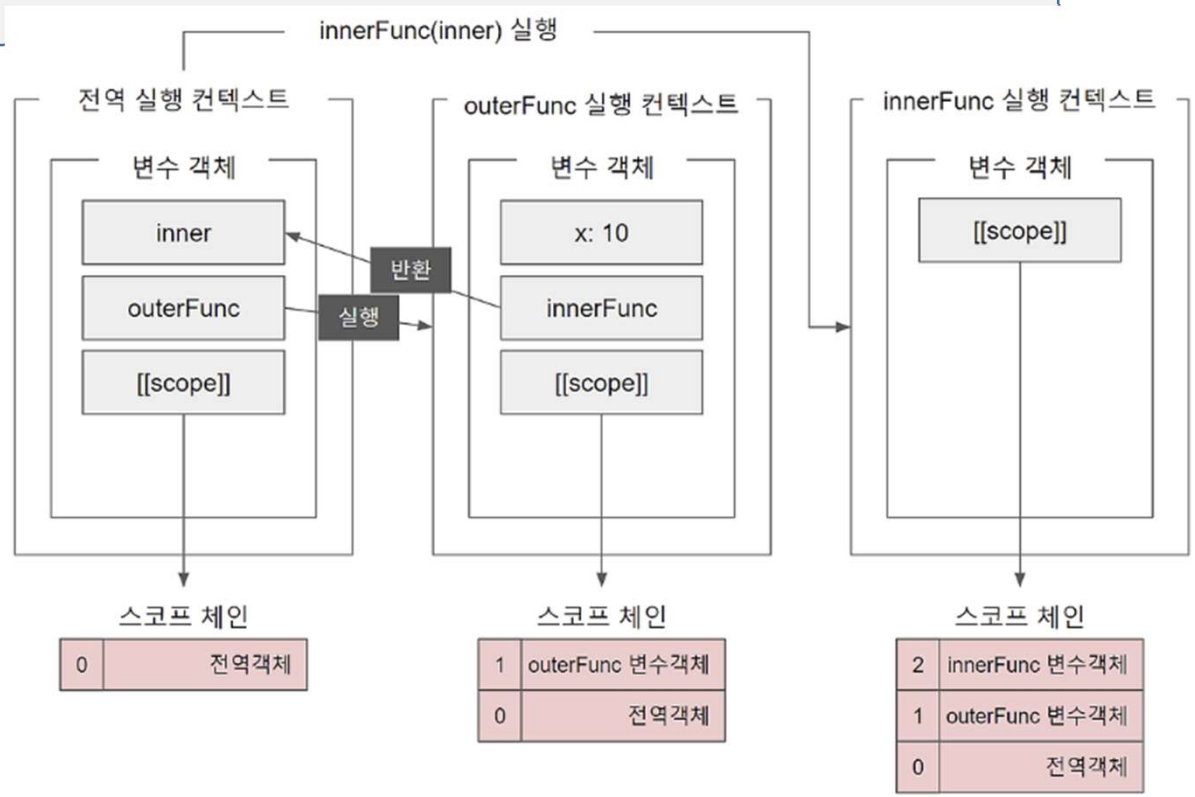
클로저 함수

- **클로저(Closure)**
  - 버튼이 클릭될 때마다 클릭한 횟수가 누적되어 화면에 표시되는 카운터 예제
  - 클릭된 횟수가 유지해야할 상태
  - 전역변수 counter는 전역 변수이기 때문에 언제든지 누구나 접근할 수 있고 변경할 수 있다.
  - increase 함수가 호출될 때마다 지역변수 counter를 0으로 초기화하기 때문에 언제나 1이 표시된다. 다시 말해 변경된 이전 상태를 기억하지 못한
  - 즉시실행함수는 호출된 이후 소멸되지만 즉시실행함수가 반환한 함수는 변수 increase에 할당되어 inclease 버튼을 클릭하면 클릭 이벤트 핸들러 내부에서 호출된다. 이때 클로저인 이 함수는 자신이 선언됐을 때의 렉시컬 환경인 즉시실행함수의 스코프에 속한 지역변수 counter를 기억한다. 따라서 즉시실행함수의 변수 counter에 접근할 수 있고 변수 counter는 자신을 참조하는 함수가 소멸될 때가지 유지된다.
  - 변수의 값은 누군가에 의해 언제든지 변경될 수 있어 오류 발생의 근본적 원인이 될 수 있다. 상태 변경이나 가변(mutable) 데이터를 피하고 불변성(Immutability)을 지향하는 함수형 프로그래밍에서 부수 효과(Side effect)를 최대한 억제하여 오류를 피하고 프로그램의 안정성을 높이기 위해 클로저는 적극적으로 사용된다.

클로저 함수

- **클로저(Closure)**

```
function outerFunc(){
 var x = 10;
 var innerFunc = function() { console.log(x);}
 return innerFunc;
}
var inner = outerFunc();
inner(); //10
```



실행 컨텍스트 (Execution Context)

- **실행 컨텍스트 (Execution Context)**
  - JavaScript 코드가 실행될 때 각 코드가 실행되는 환경을 관리하는 구조
- **전역 실행 컨텍스트 (Global Execution Context)**
  - 코드 실행 시 가장 먼저 생성되는 컨텍스트
  - 전역 객체 (window, global)와 연결됨
- **함수 실행 컨텍스트 (Function Execution Context)**
  - 함수가 호출될 때마다 생성됨
  - 함수 실행이 끝나면 제거됨
- **실행 컨텍스트 생성 과정**
  1. 함수가 호출되면 실행 컨텍스트가 생성
  2. 변수 객체 (Variable Object, VO) 생성 = > 변수, 함수 선언이 저장됨
  3. 스코프 체인 (Scope Chain) 생성 = > 현재 실행 컨텍스트 + 상위 컨텍스트 참조
  4. this 바인딩 (실행 환경에 맞게 this 지정)
  5. Execution Phase (실행 단계) = > 변수 및 함수가 실행되며 값 할당됨
  6. 소멸 단계 (Destruction) = > 함수 실행이 끝나면 실행 컨텍스트 제거됨

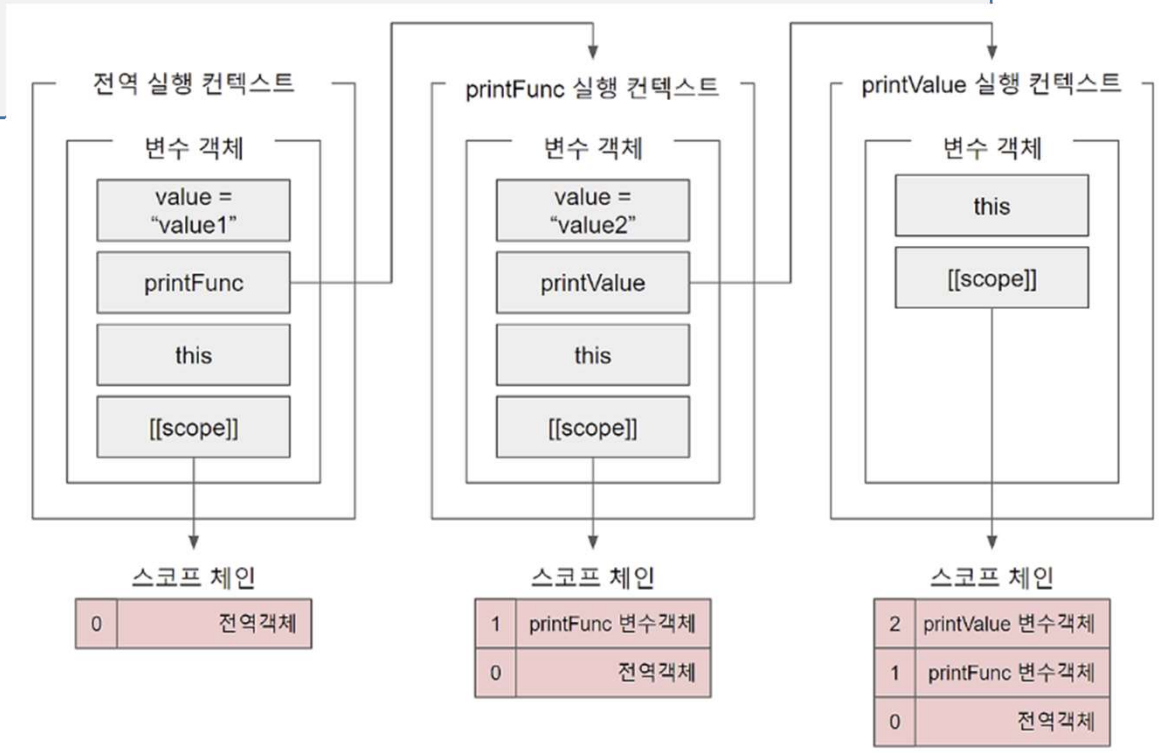
실행 컨텍스트 (Execution Context)

- **변수 객체 (VO)**
  - 실행 컨텍스트가 생성될 때 변수, 함수 선언을 저장하는 객체
- **활성 객체 (AO)**
  - 실행 컨텍스트가 실행되면서 초기화된 VO

클로저 함수

- **Scope Chain**

```
var value = "value1";
function printFunc(){
 var value = "value2";
 function printValue(){
 return value;
 }
console.log(printValue());
}
printFunc();
```



함수

- **summary**
  - 함수적 프로그래밍 언어 이면서 객체지향 언어입니다
  - 함수를 1급 객체로 취급함
  - 함수의 인수로 함수(기능)를 전달할 수 있습니다
  - 함수 내부에 함수를 정의할 수 있습니다.
  - 내부함수는 내부함수가 정의된 함수 내부에서만 호출할 수 있습니다.
  - 함수 내부에서 함수 definition을 리턴할 수 있습니다.
  - 선언적 함수 : 함수이름을 가지고 정의됩니다.
  - 익명 함수 : 이름 없는 함수 (변수에 저장)
  - 화살표함수(람다함수) : 이름X, function 선언, 하나의 표현식을 리턴하는 함수
  - 즉시실행함수 : 함수 표현식, definitio과 실행(1회성)
  - 모든 함수 가변인자를 가지는 함수로 정의됩니다
  - 함수에 정의된 매개변수의 개수보다 많은 매개변수로 호출하면 실행시에 무시된다
  - 함수에 정의된 매개변수의 개수보다 적은 매개변수로 호출하면 undefined로 전달된다
  - 함수가 실행될때 실행 컨텍스트에서는 함수 내부에 함수 속성 arguments객체 자동 생성되며 , 파라미터값들이 저장됨
  - arguments는 배열과 유사한 객체입니다.