



함수는 블랙박스다





## ✓ 함수란?

- 함수는 블랙박스다.
- 함수는 값이다.
- 함수는 객체다.

## ✓ 함수의 정의

- 함수 표현식
- 함수 선언식

## ✓ 함수의 리턴

- return 있을 때 : 결과값 리턴
- return 없을 때 : undefined 리턴

## ✓ 함수의 숨겨진 매개 변수

- arguments : 매개변수들은 arguments에 담겨서 전달된다.

## ✓ 유효 범위(scope)

- 유효 범위(scope)는 함수다
- scope chain





# 함수 vs 메서드

<script>

```
var add = function (a, b) {  
    return a + b;  
}  
var result = add(1, 3); // 4  
console.log( result ); // (출력값) 4
```

함수 : 전역 컨텍스트에서 정의된 function

```
var obj = {  
    add : function (a, b) { return a + b; }  
}  
var result = obj.add(1,3); // 4  
console.log( result ); // (출력값) 4
```

메서드 : 객체 안에서 정의된 function

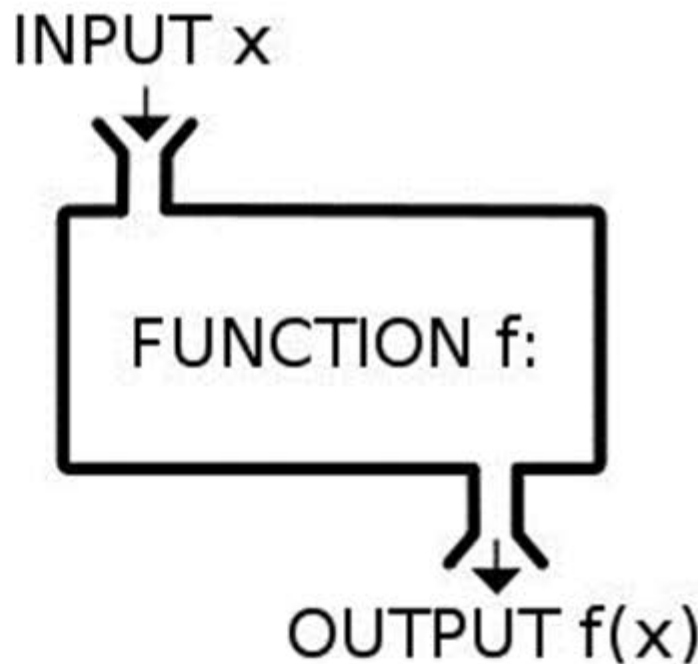
</script>





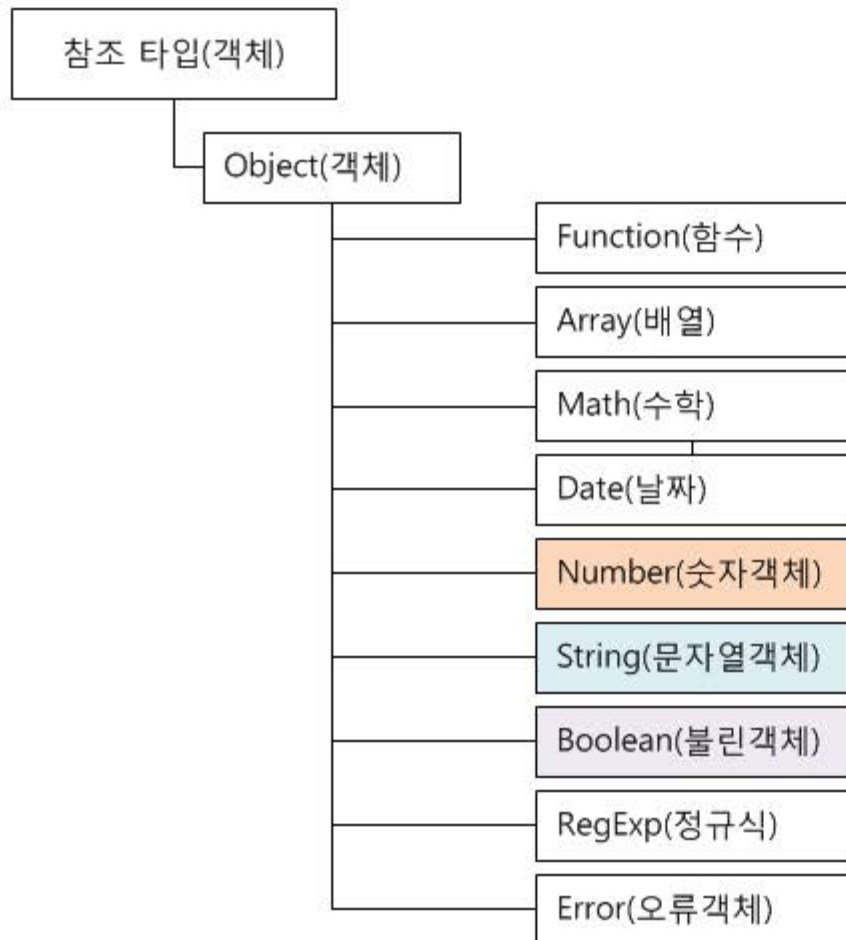
# 함수란?

- 함수는 블랙 박스다.
  - 함수는 입력한 값을 받아 처리한 결과를 반환한다
  - ex) `alert()`, `prompt()`
- 함수는 값이다.
  - 변수에 함수 대입 가능
  - 프로퍼티에 함수 대입 가능(메서드)
  - 배열 요소로 함수 사용 가능
  - 매개변수로 함수 사용 가능
  - 리턴값으로 함수 사용 가능(클로저)
- 함수는 객체다
  - 함수는 프로퍼티를 가질 수 있다.
  - 함수는 메서드를 가질 수 있다.
  - 코드영역을 갖는다.





# 참조 타입



- 배열

```
var array1 = [ 2, 3, 4.5, 6.78 ];  
var array2 = [ 'a', 'abc', 'ef' ];  
var array3 = [ 2, 'abc', true ];
```

- 함수

```
var log = function () {  
    console.log('It is a function');  
};
```

- 객체

```
var obj = { name : 'you', age:30 };
```







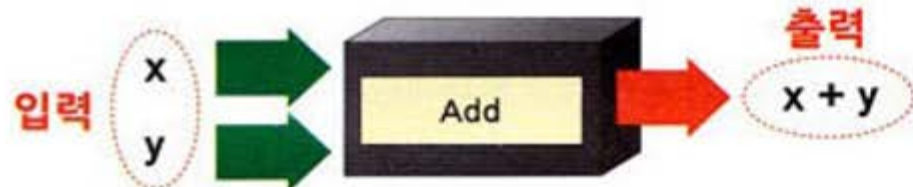
# 함수는 블랙박스다.

- 함수는 특정 작업을 수행하기 위해 불려지는 블랙박스

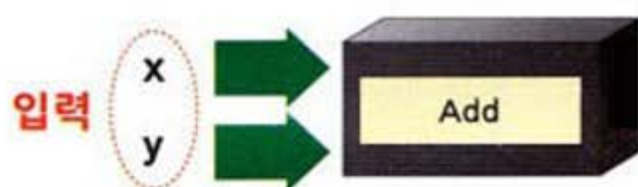
함수의 내용은 알 수 없다



함수의 내용은 알 수 없다



입력은 여러 개가 될 수 있다. 출력은 반드시 하나이다.



입력은 여러 개가 될 수 있다.

- 반복 수행되는 문장을 함수로 작성하면 **코드의 재사용**이 가능하다.



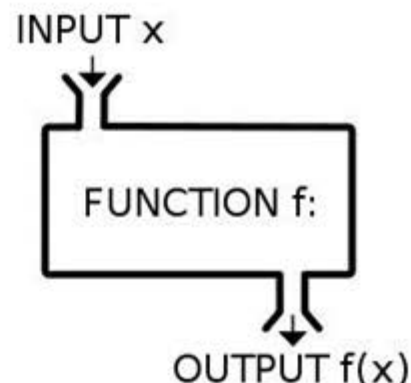


# 함수 정의-선언식

- 선언식 함수 정의: 파싱시 함수가 정의됨

```
function add(a, b) {  
    var sum = a + b;  
    return sum;  
}
```

```
var sum = add(1, 3); // 4
```



## 일반적인 함수 선언

이름

매개변수

```
function name(parameters){  
    statements  
}
```

구문





# 함수 정의-표현식

- 표현식 함수 정의: 실행시 함수가 정의됨

```
var add = function (a, b) {  
    var sum = a + b;  
    return sum;  
}  
  
var sum = add(1, 3); // 4
```
- 선언식은 실행 시 표현식으로 변경된다. **함수 정의시는 표현식 방식을 사용.**

## 함수 리터럴

- 명령문 대신 표현식에서 선언하는 이름이 지정되지 않은 함수
- 함수를 임시로 사용해야 하거나 표현식을 대신 사용할 수 있는 코드에서 함수를 사용해야 하는 경우에 유용

```
var foo = function () {};
```





# 함수 정의

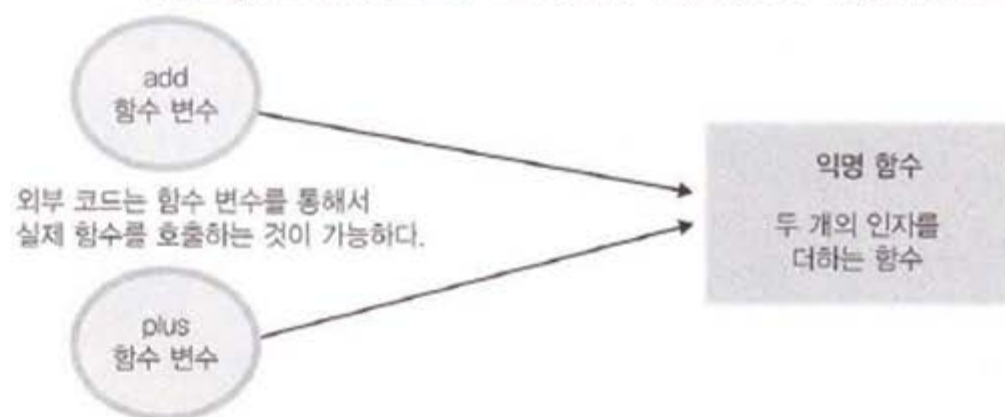
- 함수 정의는 표현식 방식을 사용하라.
  - 함수 호이스팅 방지

add와 plus 함수 변수는 두 개의 인자를 더하는 동일한 익명 함수를 참조한다.

```
// add() 함수 표현식  
var add = function (x, y) {  
    return x + y;  
};
```

```
var plus = add;
```

```
console.log(add(3,4)); // (출력값) 7  
console.log(plus(5,6)); // (출력값) 11
```





# 함수 정의 - 매개변수

- 매개변수들
  - 함수를 호출할 때 함수에서 사용할 데이터를 전달
  - Parameter argument는 순서대로 해당 parameter로 대입됨

```
var add = function (x, y) { // callee
    var sum = x + y;
    return sum;
}
```

Formal-parameter list

```
// caller
var a, b;
...
var result = add(a, b);
console.log(result);
```

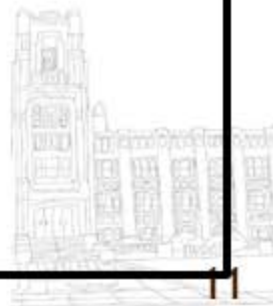
Actual-parameter list



```
<script>
  // 선언식 함수 정의
  function add1(a, b) {
    return a + b;
  }
  var result = add1(1, 3);    // 4
  console.log( result );    // (출력값) 4

  // 표현식 함수 정의
  var add2 = function (a, b) {
    return a + b;
  }
  var result = add2(1, 3);    // 4
  console.log( result );    // (출력값) 4

</script>
```



<script>

// 1부터 10까지의 합계 구하기

55

console.log('1부터 ' + end + '까지의 합계 = ' +

// 1부터 100까지의 합계 구하기

5005

console.log('10부터 ' + end + '까지의 합계 = ' + sum);

// 100부터 1000까지의 합계 구하기

495550

반복 되는  
코드를  
줄이려면  
어떻게  
해야할까?



&lt;script&gt;

```
var getSum = function (start, end) {  
    var sum = 0;  
    for (var i = start; i <= end; i=i+1) {  
        sum = sum + i;  
    }  
    return sum;  
};
```

// 1부터 10까지의 합계 구하기

```
var start = 1;  
var end = 10;  
var sum = getSum(start, end);  
console.log(start + '부터 ' + end + '까지의 합계 = ' + sum);
```

// 10부터 100까지의 합계 구하기

```
console.log(start + '부터 ' + end + '까지의 합계 = ' + sum);
```

// 100부터 1000까지의 합계 구하기





산술 연산을 출력하는 프로그램을 작성하시오. 단, 각각의 연산은 하나의 독립된 함수로 구현하시오

1. 두 개의 숫자를 입력 받는다.
2. 덧셈, 뺄셈, 곱셈, 나눗셈 함수를 만든다.
3. 덧셈, 뺄셈, 곱셈, 나눗셈 함수 호출 결과를 출력한다.

■ 실행결과예시

```
First num : 2  
Second num : 4  
Add : 6  
Sub : -2  
Mul : 8  
Div : 0.5
```



# 숨겨진 매개변수:arguments

- arguments 는 유사 배열 객체다.
  - arguments 는 length 프로퍼티와 [] 연산자만 사용 가능
  - 배열 메서드 사용시 에러 발생
- 함수를 호출할 때 매개변수들은 arguments 에 담겨서 전달된다.

```
var add = function (a, b) {  
  // arguments 객체 출력  
  console.dir(arguments);  
  
  return a+b;  
}  
  
console.log( add(1 )      );  
console.log( add(1, 2)    );  
console.log( add(1, 2, 3) );
```

```
▼ Arguments[1] ⓘ  
  0: 1  
  ▶ callee: function add(a, b)  
    length: 1  
  ▶ Symbol(Symbol.iterator): function values()  
  ▶ __proto__: Object
```

NaN

```
▼ Arguments[2] ⓘ  
  0: 1  
  1: 2  
  ▶ callee: function add(a, b)  
    length: 2  
  ▶ Symbol(Symbol.iterator): function values()  
  ▶ __proto__: Object
```

3

```
▼ Arguments[3] ⓘ  
  0: 1  
  1: 2  
  2: 3  
  ▶ callee: function add(a, b)  
    length: 3  
  ▶ Symbol(Symbol.iterator): function values()  
  ▶ __proto__: Object
```

3



# 숨겨진 매개변수:arguments

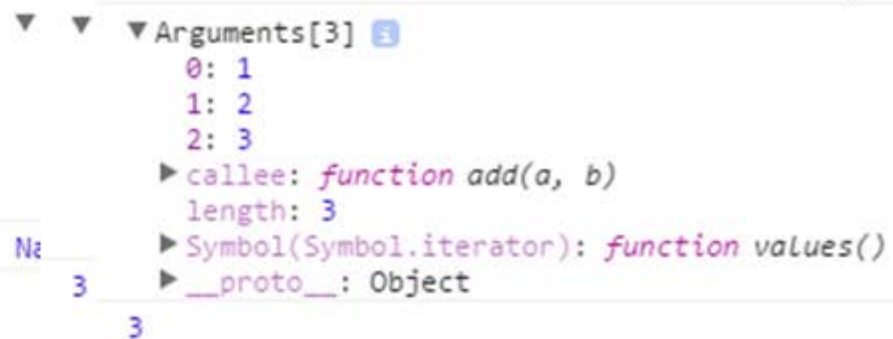
- arguments 는 유사 배열 객체다.
  - arguments 는 length 프로퍼티와 [] 연산자만 사용 가능
  - 배열 메서드 사용시 에러 발생
- 함수를 호출할 때 매개변수들은 arguments 에 담겨서 전달된다.

```
var sum = function () {  
    var result = 0;  
  
    for (var i = 0; i < arguments.length; i++) {  
        result += arguments[i];  
    }  
  
    return result;  
}  
  
console.log( sum(1,2,3) ) ;           // 6  
console.log( sum(1,2,3,4,5,6,7,8,9) ); // 45
```

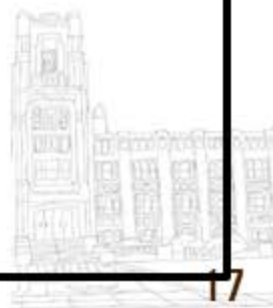


&lt;script&gt;

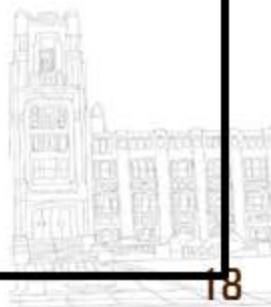
```
var add = function (a, b) {  
  // arguments 객체 출력  
  console.dir(arguments);  
  return a+b;  
}
```



```
console.log( add(1 )      ); // (출력값) NaN  
console.log( add(1, 2)    ); // (출력값) 3  
console.log( add(1, 2, 3) ); // (출력값) 3
```



```
var sum = function () {  
    var result = 0;  
    for (var i = 0; i < arguments.length; i++) {  
        result += arguments[i];  
    }  
    return result;  
}  
  
console.log( sum(1,2,3) );           // (출력값) 6  
console.log( sum(1,2,3,4,5,6,7,8,9) ); // (출력값) 45  
</script>
```

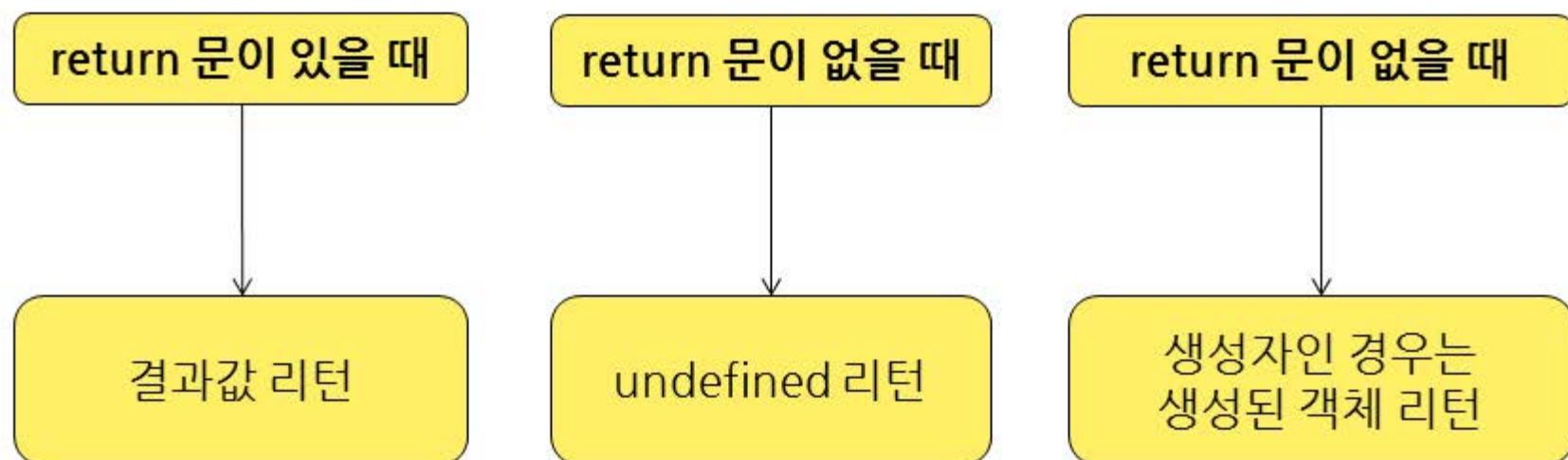






# 함수 리턴

- 함수는 항상 리턴값을 반환한다.
- 리턴값은 최대 한 개만 가능하다.




return 문을 생략하지 마시오

```
<script type="text/javascript">
  /* * return 문 있을 때 - 결과값 리턴 */
  var add = function(x,y) {
    var result = 0;
    result = x + y;
    return result;
  };

  var result = add(1,2);
  console.log( result );    // (출력값) 3

  /* * return 문 없을 때 - undefined 리턴 */
  var noreturn = function noreturn() {
    console.log('This function has no return statement.');
```

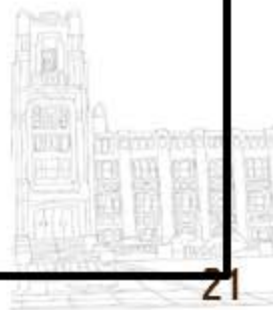


```
  };

  var result = noreturn();
  console.log( result );    // (출력값) undefined
</script>
```

```
<script type="text/javascript">
  /* * return 문 없을 때 - 생성 객체 리턴 */
  function Person(name, age, gender) {
    this.name = name;
    this.age = age;
    this.gender = gender;
  };

  var foo = new Person('foo', 30, 'man' );
  console.log(foo);
</script>
```

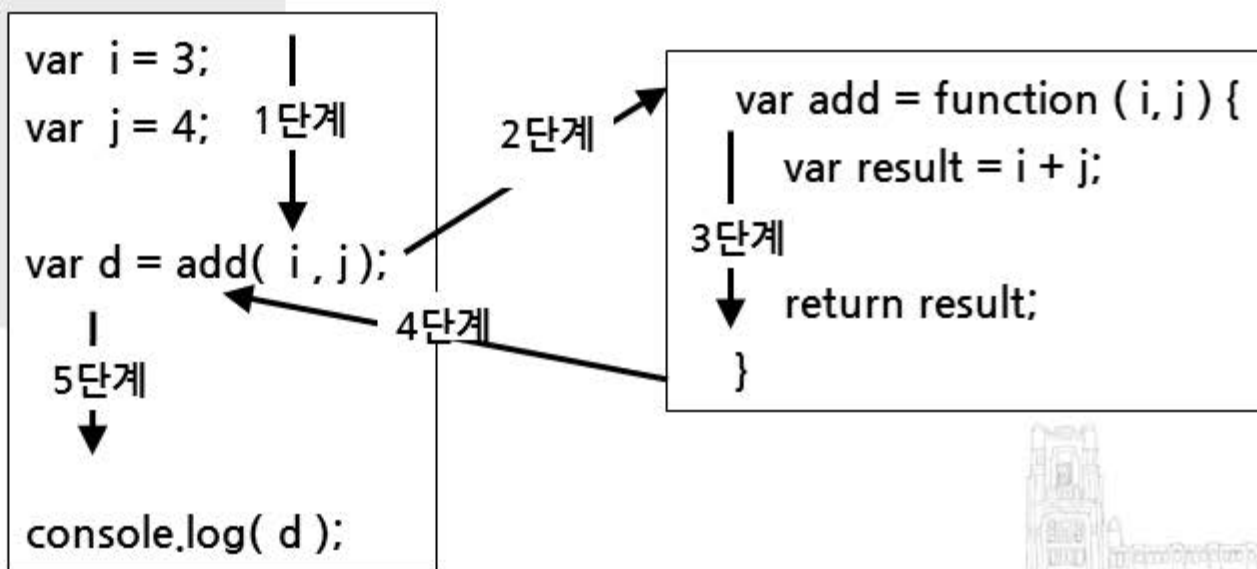




# 함수의 호출 순서

```
var add = function ( i, j ) {  
    var result = i + j;  
    return result;  
}
```

```
var i = 3;  
var j = 4;  
var d = add( i, j );  
console.log( d );
```





# 변수의 유효 범위(Scope)

- 변수의 유효 범위
  - 블록( { } ) 단위의 유효 범위 : C, C++, JAVA
  - 함수 단위의 유효 범위 : JavaScript
- 자바스크립트에서 변수의 유효 범위는 **함수**다

```
console.log( x ); // ???  
var x = 1;  
console.log( '-----' ); // ???  
  
var outer = function () {  
    console.log( x ); // ???  
    console.log( y ); // ???  
    var y = 5;  
    console.log( y ); // ???  
}  
outer();
```





&lt;script &gt;

```
console.log( x ); // ???  
var x = 1;  
console.log( '-----' ); // ???
```

변수의  
호이스팅

```
var outer = function () {  
  console.log( x ); // ???  
  console.log( y ); // ???  
  var y = 5;  
  console.log( y ); // ???  
}
```

변수의  
호이스팅

```
outer();
```

&lt;/script&gt;

globalscope

- x

outer()

outer scope

- y

scope  
chain

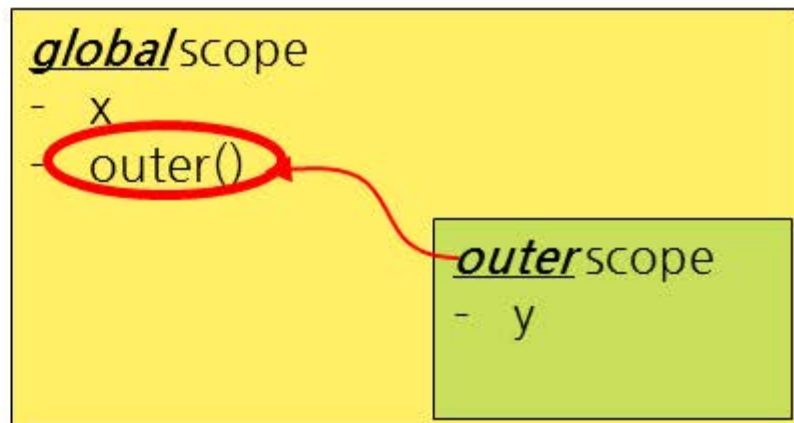
```
<script >
  console.log( x );
  var x = 1;
  console.log( x );

  var outer = function () {
    console.log( x );
    console.log( y );

    if(true) {
      var y = 5;
    }

    console.log( y );
  };

  outer();
</script>
```





# 변수의 유효 범위(Scope)

- 변수의 종류
  - 전역 변수 == 글로벌 스코프에서 사용 가능한 변수
  - 지역 변수 == 로컬 스코프에서만 사용 가능한 변수
  - 매개변수
  - 프로퍼티

```
var scope = '글로벌';
```

```
function getValue() {
```

```
var scope = '로컬';
```

```
return scope;
```

```
}
```

```
document.writeln(getValue());
```

```
document.writeln(scope);
```

**로컬 스코프 =**  
로컬 변수 scope의  
유효범위

**글로벌 스코프 =**  
글로벌 변수 scope의  
유효범위

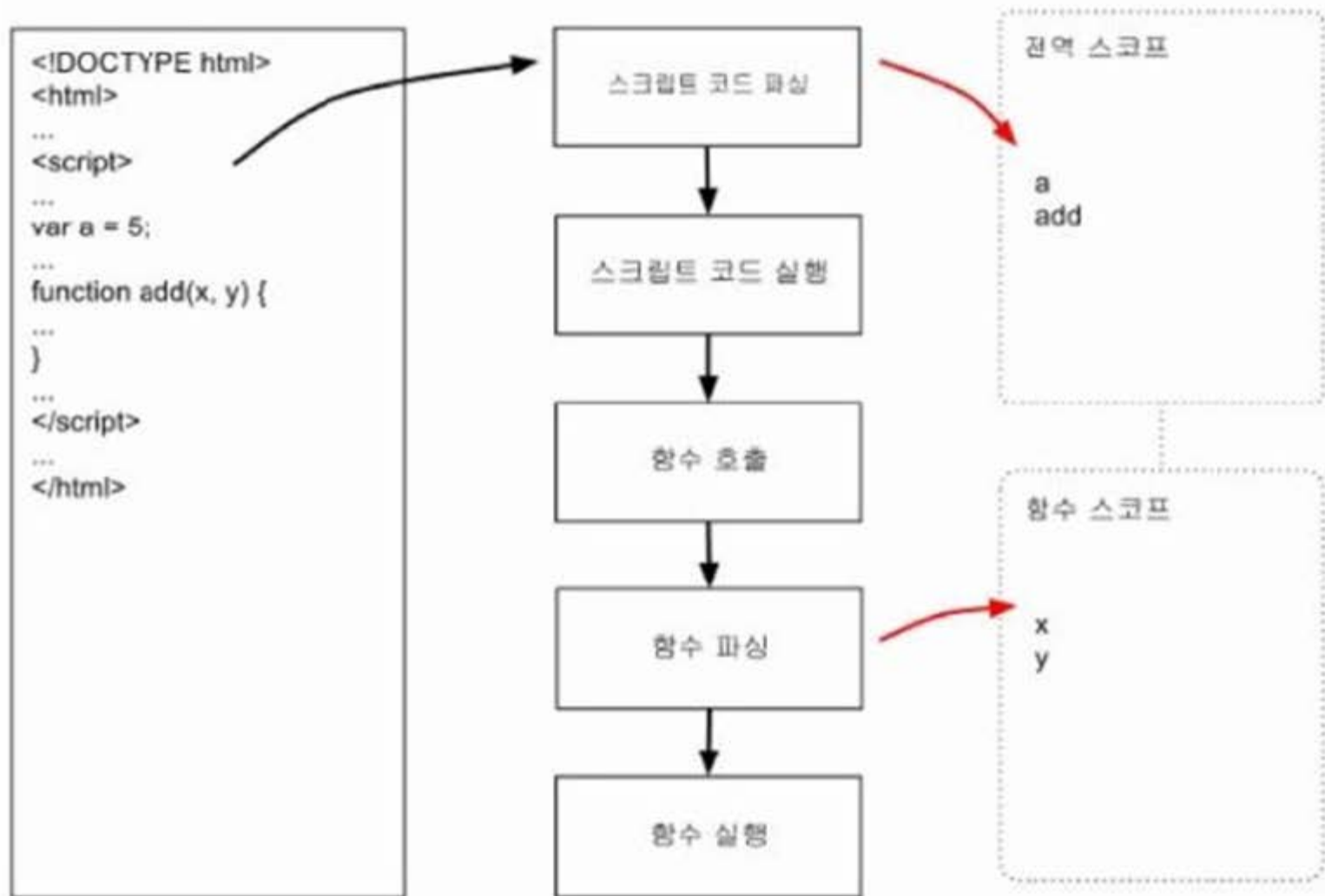
scope: 글로벌 변수

scope: 로컬 변수





# 전역 변수



```
<script>
  var foo = function ( ) {
    var a=3, b=5;

    var bar = function ( ) {
      var b=7, c=11;

      console.log( a, b, c ) ;
      a = a + b + c;
      console.log( a, b, c ) ;
    };

    console.log( a, b ) ; // ? ?
    bar();
    console.log( a, b ) ; // ? ?
  };

  foo();
</script>
```



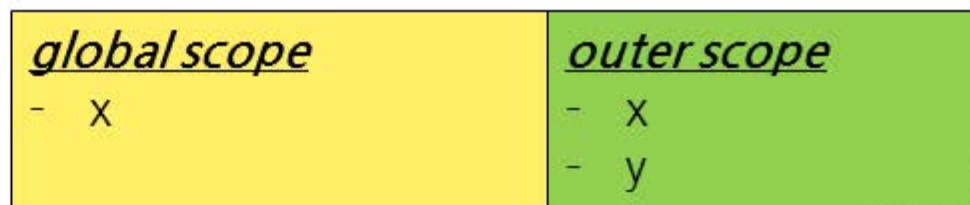
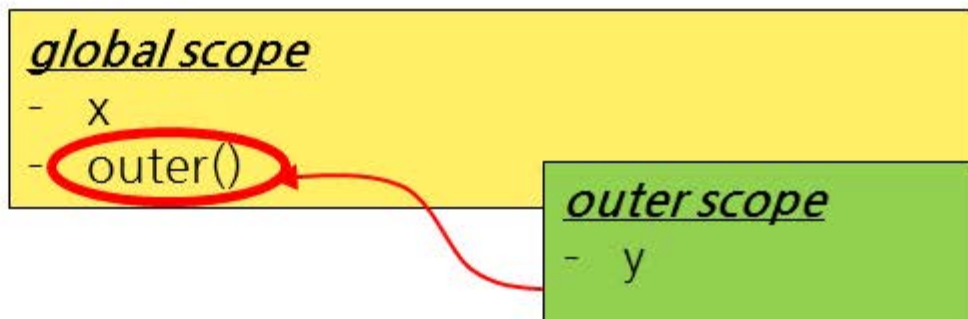


# 변수의 유효 범위(Scope) 체인

```
var x = 1;
```

```
var outer = function () {  
    var y = 2;  
  
    console.log( x );  
}
```

```
console.log( outer() );
```



```
<script type="text/javascript">
```

```
  var x = 'global';
```

```
  var f = function ( ) {
```

```
    console.log(x);
```

// ?

```
    console.log(window.x);
```

// ?

```
    console.log(this.x);
```

// ?

```
    var y = 'local';
```

```
    console.log(y);
```

// ?

```
  }
```

```
  var g = function ( ) {
```

```
    console.log(x);
```

// ?

```
    console.log(window.x);
```

// ?

```
    var x = 'local';
```

```
    console.log(x);
```

// ?

```
  }
```

```
  f();
```

```
  console.log('-----');
```

```
  g();
```

```
</script>
```

