Js의 arguments는 배열처럼 쓸 수 있지만 진짜 배열은 아니다.

변수나 함수나 다 값으로 간주함

*객체와 함수

var obj = new Object( ) ; →

key	value

1.빈 객체 생성
2.object ( ) 생성자에게 초기화 시킴

var obj = function ( ) { console.log("_____"); →

function 은 객체는 객체이지만 함수 body가 보관 됨

key	value
console.log("_____");	

← 1.빈 객체 생성
← 2.object ( ) 생성자에게 초기화 시킴
← 3.함수 body 보관

함수(객체)
object + 함수 body)

★호이스팅★

스크립트 태그 안에서 자바스크립트를 실행할 때 변수선언과 함수정의를 먼저 실행한다.

*closure

call
→ createGreeting ( )
- name = " 홍길동 "
- message = "홍길동님 반갑습니다"
- f = function

createGreeting ( ) →
의 효율이 끝나서 message 변수가
사라지더라도 closure는 문제없이
실행된다.

stack

createGreeting ( )		
name	message	f
"홍길동	홍길동님 반갑습니다	function

→ message는 바깥함수의 로컬 변수를 가르키는것이 아니다.
→ closure 정의

function() {
console.log(message);
}

message
"홍길동님 반갑습니다"

closure가 사용하는 바깥 함수의
로컬 변수를 복제하여 보관한다.

★★복제한 변수를 가르킨다.★★

*closure

생성 및 리턴 test1
createGreeting ("홍길동") →

function() { console.log(message); }
message
"홍길동님 반갑습니다"

← closure만 접근 가능한 전용 메모리

생성 및 리턴 test2
createGreeting ("임꺽정") →

function() { console.log(message); }
message
"임꺽정님 반갑습니다"

← closure만 접근 가능한 전용 메모리

*객체에 closure를 담아 리턴하기

call
→ createBox ( ) 객체 생성 box1 →

key	value
setValue	value = v
getValue	return value

value
100

← closure들이 공유할 복제 변수를 담는 메모리

box1.setvalue(100)
box1.getvalue() → return 100
box1.value → undefined
closure가 공유하는 자원이지 박스가 공유하는게 아님

call
→ createBox ( ) 객체 생성 box2 →

key	value
setValue	value = v
getValue	return value

value
200

← closure들이 공유할 복제 변수가 들어 있는 메모리

box2.setvalue(200) → return 200

클로저 형제는 클로저 메모리를 공유한다.

*함수 - 애로우(arrow)

//익명함수

var f1 = function() {
return "안녕";
};
console.log(f1());

// 애로우(arrow) 함수 : 초간단 함수 정의 문법
// 1) function 키워드를 제거한다.

```
// 2) 대신 파라미터 블록과 코드 블록 사이에 화살표를 추가한다.
var f2 = () => {
    return "안녕";
};
console.log(f2());
```

JSON 별도의 파일 포맷

문자열을 반드시 " " 적어야 한다.