

---

# 3장. 노드의 자바스크립트와 친해지기

# 강의 주제 및 목차

## 강의 주제

자바스크립트에서 알아야 하는 중요한 내용이 있나요?

## 목 차



1

자바스크립트의 객체와 함수 이해하기

2

배열 이해하기

3

콜백 함수 이해하기

4

프로토타입 객체 만들기

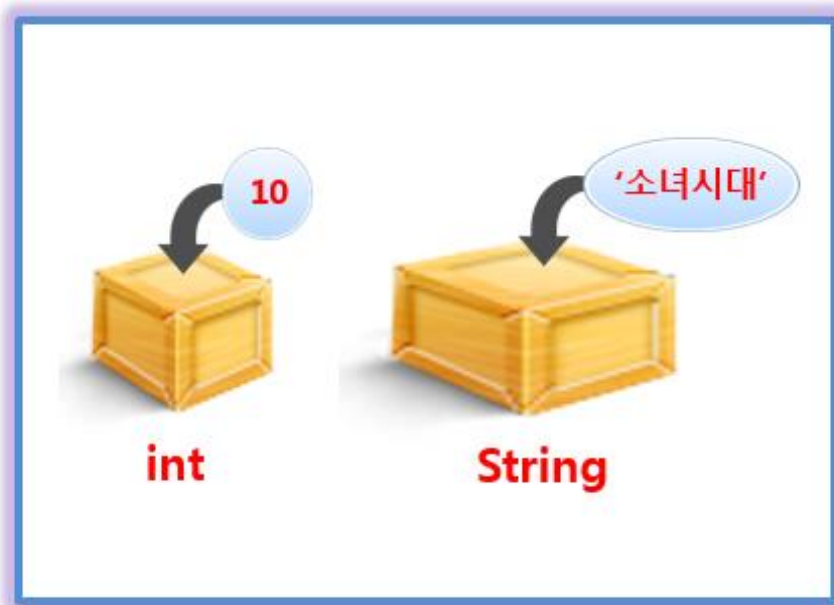
1.

자바스크립트의 객체와 함수 이해하기

# 자바와 자바스크립트의 변수 타입 비교

- 자바는 자료형(타입)을 명시하는 언어
- 자바스크립트는 자료형을 명시하지 않는 언어
- 내부에서는 자료형에 따라 변수 상자의 크기가 달라짐

자바



자바스크립트



# 자바스크립트의 자료형

- boolean, number, string 이 있으며, 그 외에 undefined, null, Object 자료형이 있음

자료형	설명
Boolean	[기본 자료형] true와 false의 두 가지 값을 가지는 자료형
Number	[기본 자료형] 64비트 형식의 IEEE 754 값이며 정수나 부동소수 값을 가지는 자료형 몇 가지 상징적인 값을 가질 수 있음: NaN(숫자가 아님), +무한대(Number.MAX_VALUE로 확인), -무한대(Number.MIN_VALUE로 확인)
String	[기본 자료형] 문자열 값을 가지는 자료형
undefined	값을 할당하지 않은 변수의 값
null	존재하지 않는 값을 가리키는 값
Object	객체를 값으로 가지는 자료형 객체는 속성들을 담고 있는 가방(Collection)으로 볼 수 있으며, 대표적인 객체로 Array나 Date를 들 수 있음

# 변수 만들기

---

- 변수 앞에는 var 키워드를 붙임

```
var age = 20;  
console.log('나이 : %d', age);  
var name = '소녀시대';  
console.log('이름 : %s', name);
```

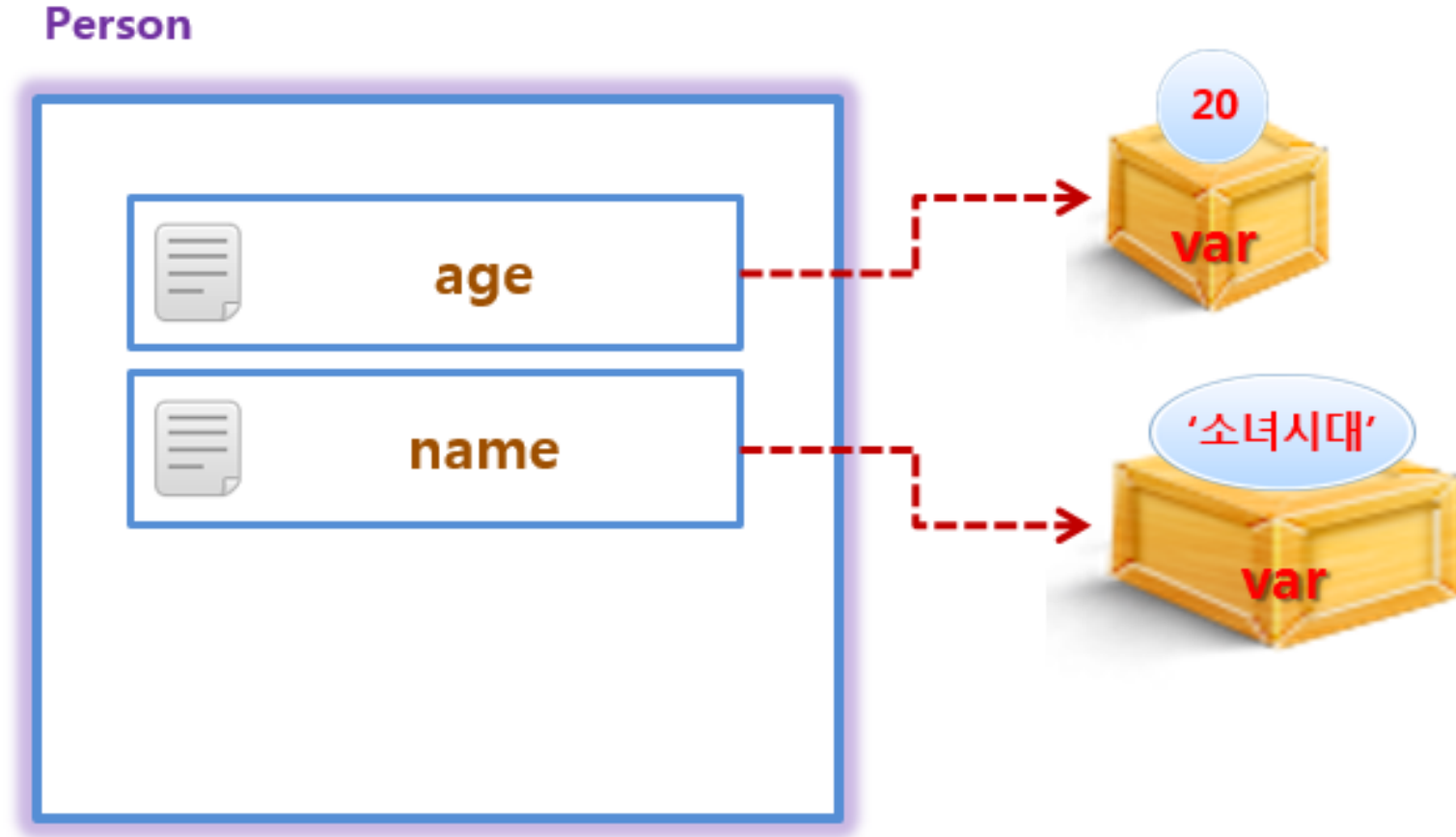
## ch03\_test1.js 자바스크립트의 변수 타입

---

```
var name;  
console.log('name : ' + name);  
  
var age = 20;  
console.log('나이 : %d', age);  
  
var name = '소녀시대';  
console.log('이름 : %s', name);
```

# 자바스크립트의 객체

- 속성들이 이름 - 값 의 형태로 들어가 있음





# 객체 만들기

- 객체는 { } 기호를 이용해 만듦
- 객체 안의 속성은 . 연산자를 이용해 접근하거나 객체이름 뒤에 [ ] 를 붙이고 그 안에 속성 이름을 문자열로 넣어 접근할 수 있음

```
var Person = {};  
Person["age"] = 20;  
Person["name"] = '소녀시대';  
Person.mobile = '010-1000-1000';  
console.log('나이 : %d', Person.age);  
console.log('이름 : %s', Person.name);  
console.log('전화 : %s', Person["mobile"]);
```

## ch03\_test2.js 자바스크립트의 객체 타입

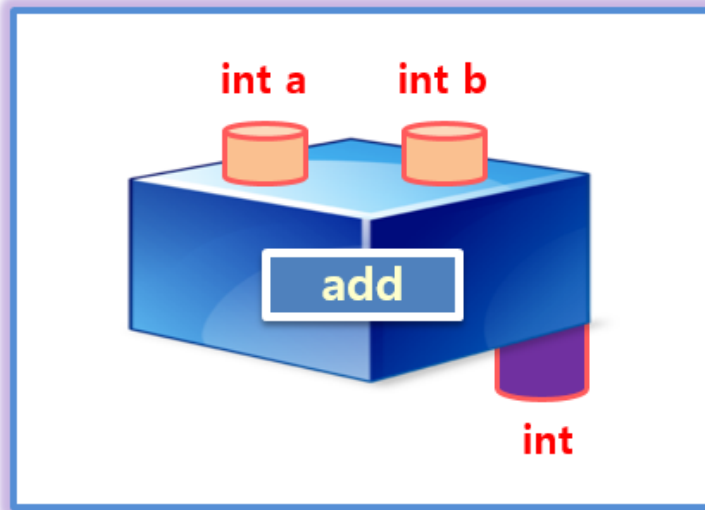
```
var Person = {};  
  
Person['age'] = 20;  
Person['name'] = '소녀시대';  
Person.mobile = '010-1000-1000';  
  
console.log('나이 : %d', Person.age);  
console.log('나이 : %d', Person['age']);  
console.log('이름 : %s', Person.name);  
console.log('이름 : %s', Person['name']);  
console.log('전화 : %s', Person['mobile']);  
console.log('전화 : %s', Person.mobile);
```

- 객체는 중괄호를 이용해 만들어지며 그 안에 속성을 추가할 수 있음

# 자바와 자바스크립트의 함수 비교

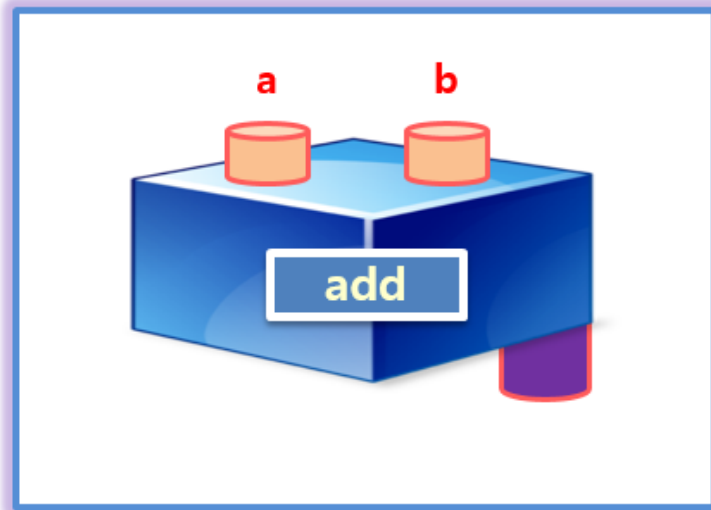
- 파라미터의 타입과 반환되는 값의 타입을 명시하지 않음
- 함수 앞에는 function 키워드를 붙임

자바



```
int add(int a, int b) { ... }
```

자바스크립트

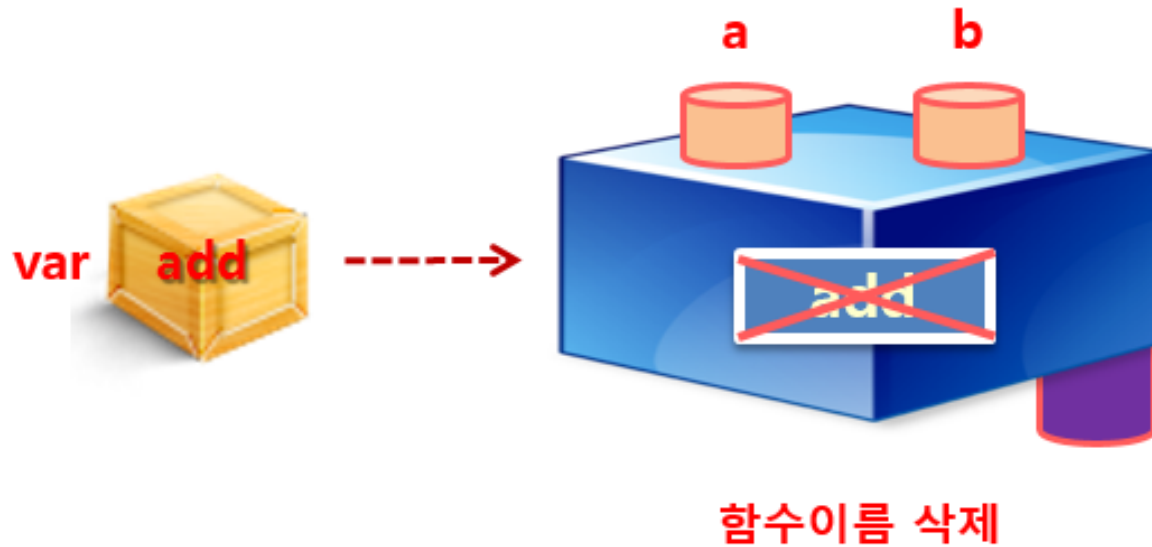


```
add( a, b) { ... }
```

↑  
function

# 함수를 변수에 할당하기

- 자바스크립트에서는 함수를 일급 객체(First class object)로 다룸
- 따라서, 함수가 변수에 할당될 수 있음
- 변수에 할당될 경우 두 가지 이름으로 함수를 호출할 수 있으므로 원래의 함수 이름을 생략하고 익명함수(Anonymous Function)라고 부름



```
var add = function ( a, b ) { ... };
```

# 함수 만들어 실행하기

- 함수를 만들고 실행할 수 있음
- 선언문 (Declaration)

```
function add(a, b) {  
    return a + b;  
}  
var result = add(10, 10);  
console.log('더하기 (10, 10) : %d', result);
```

## ch03\_test3.js 자바스크립트의 함수

```
function add(a, b) {  
    return a + b;  
}  
  
var result = add(10, 10);  
  
console.log('더하기 (10, 10) : %d', result);
```

- 자바와 같은 타입 기반 언어의 함수에서 타입만 제외한 형태

# 함수 만들어 변수에 할당하기

- 변수 이름으로 호출 가능

```
var add = function (a, b) {  
    return a + b;  
};  
var result = add(10, 10);  
console.log('더하기 (10, 10) : %d', result);
```

## ch03\_test4.js 익명 함수

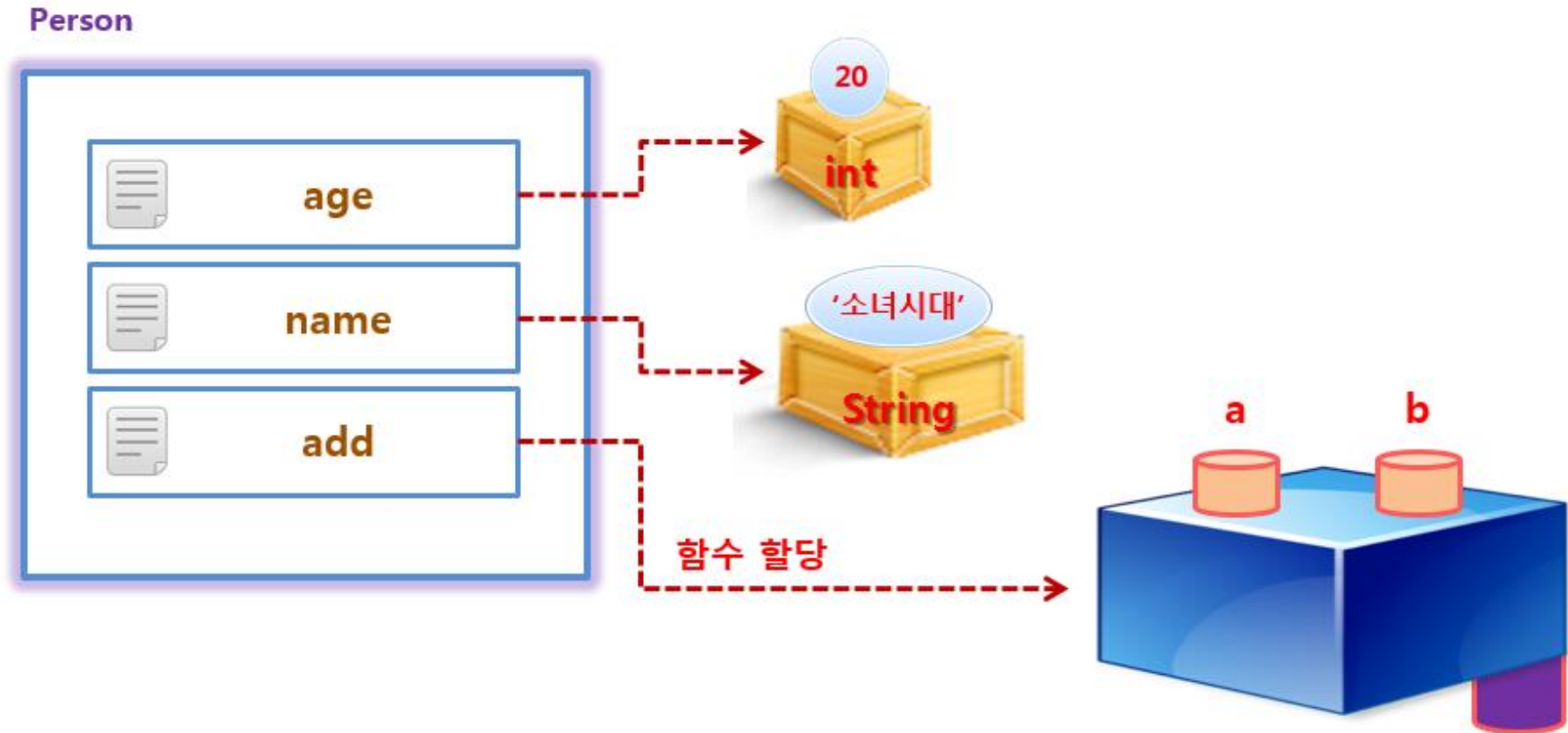
```
function add(a, b) {  
    return a + b;  
}  
  
var result = add(10, 10);  
  
console.log('더하기 (10, 10) : %d', result);
```

- 변수의 값으로 함수가 할당될 수 있음
- 변수의 값으로 함수를 구분하므로 함수의 이름을 넣을 필요가 없음
- 함수 선언문이 아니라 함수 표현식으로 추가함



# 객체의 속성으로 함수 할당하기

- 객체의 속성도 변수처럼 처리되므로 함수 할당 가능



# 객체의 속성에 함수 할당

```
var Person = {};  
Person["age"] = 20;  
Person["name"] = '소녀시대';  
Person.add = function(a, b) {  
    return a + b;  
};  
console.log('더하기 : %d', Person.add(10, 10));
```

## ch03\_test5.js 객체의 속성으로 함수 할당

```
var Person = {};  
  
Person['age'] = 20;  
Person['name'] = '소녀시대';  
Person.add = function(a, b) {  
    return a + b;  
};  
  
console.log('더하기 : %d', Person.add(10,10));
```

- 변수의 값으로 함수가 할당될 수 있음
- 변수의 값으로 함수를 구분하므로 함수의 이름을 넣을 필요가 없음
- 함수 선언문이 아니라 함수 표현식으로 추가함

## ch03\_test6.js 객체의 속성으로 함수 할당

```
var Person = {};  
  
Person['age'] = 20;  
Person['name'] = '소녀시대';  
  
var oper = function(a, b) {  
    return a + b;  
};  
  
Person['add'] = oper;  
  
console.log('더하기 : %d', Person.add(10,10));
```

- 함수를 변수에 할당한 후 속성으로 할당할 수 있음

# 객체를 만들 때 속성 할당하기

```
var Person = {  
  age: 20,  
  name: '소녀시대',  
  add: function(a, b) {  
    return a + b;  
  }  
};  
console.log('더하기 : %d', Person.add(10, 10));
```

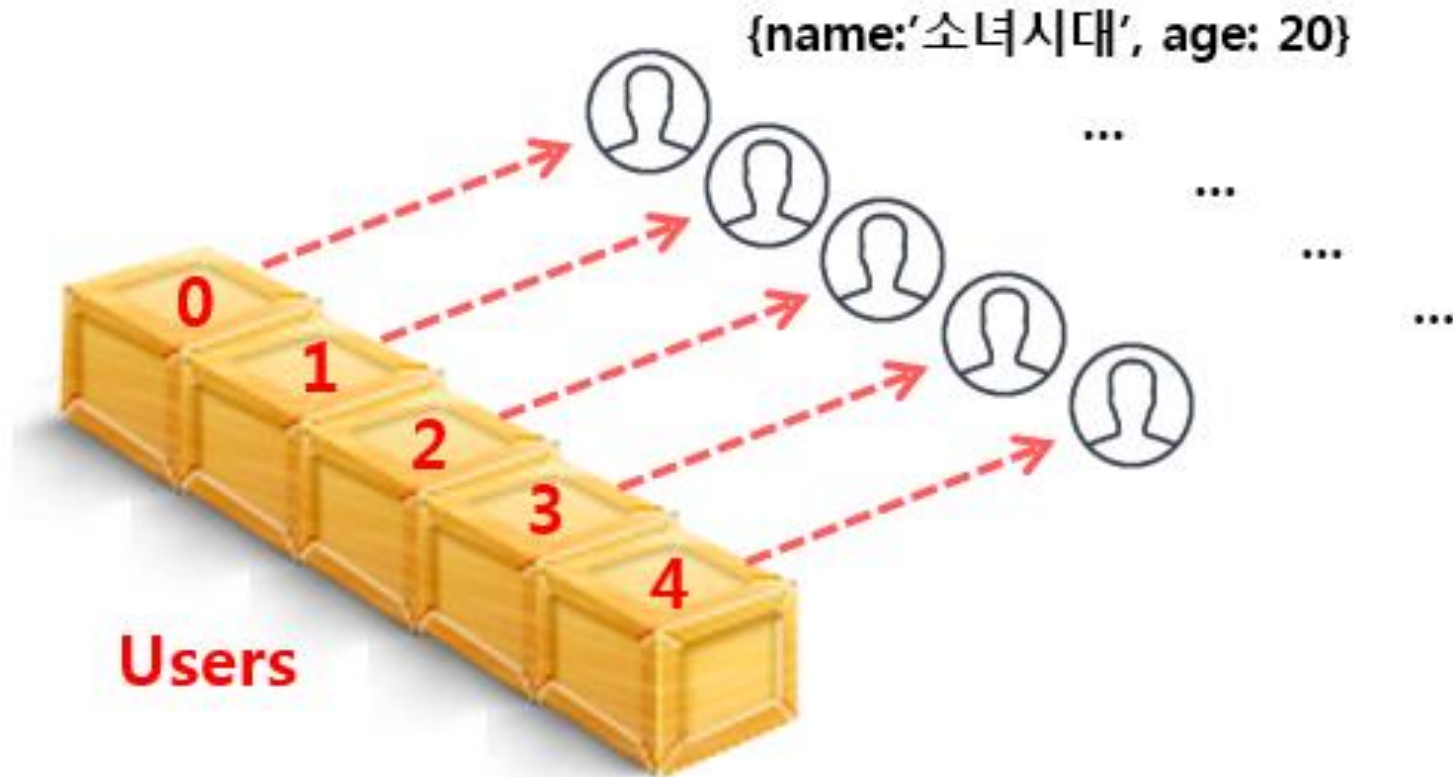
```
var Person = {  
    age: 20,  
    name: '소녀시대',  
    add: function(a, b) {  
        return a + b;  
    }  
};  
  
console.log('더하기 : %d', Person.add(10,10));
```

2.

배열 이해하기

# 배열이 만들어지는 모양

- 배열은 여러 개의 데이터를 하나의 변수에 담아둘 수 있는 방법
- 배열의 요소는 대괄호를 이용해 접근할 수 있음

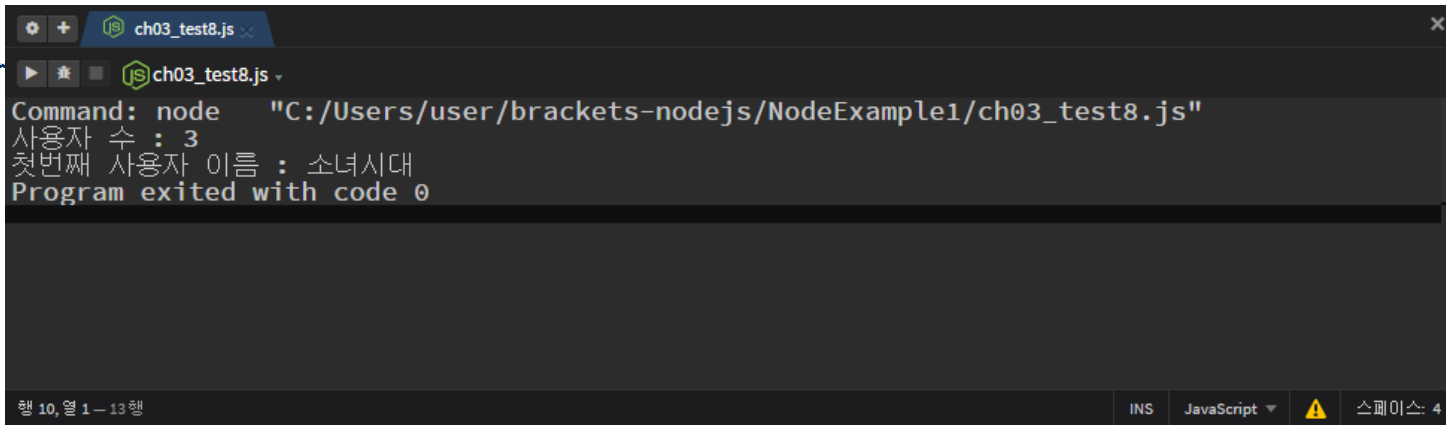




# 배열에 원소 추가

- push 함수를 호출하여 배열의 마지막에 원소를 추가할 수 있음

```
var Users = [{name:'소녀시대', age:20}, {name:'걸스데이', age:22}];  
Users.push({name:'티아라', age:23});  
console.log('사용자 수 : %d', Users.length);  
console.log('첫 번째 사용자 이름 : %s', Users[0].name);
```



```
ch03_test8.js  
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch03_test8.js"  
사용자 수 : 3  
첫번째 사용자 이름 : 소녀시대  
Program exited with code 0  
행 10, 열 1 - 13 행  
INS JavaScript 스페이스: 4
```

## ch03\_test8.js 배열 만들고 요소 추가하기

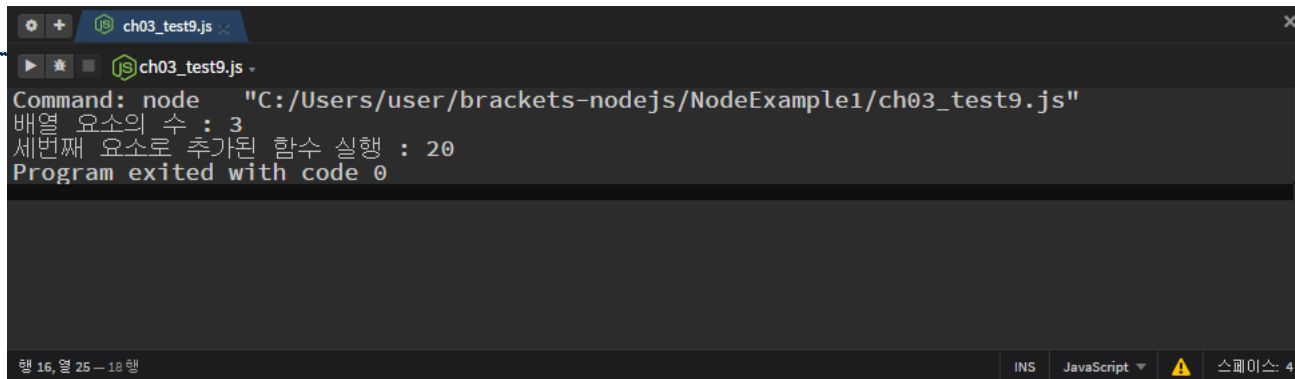
---

```
var Users = [{name:'소녀시대',age:20},{name:'걸스데이',age:22}];  
  
Users.push({name:'티아라',age:23});  
  
console.log('사용자 수 : %d', Users.length);  
console.log('첫번째 사용자 이름 : %s', Users[0].name);
```

# 배열 원소로 함수 추가

- 변수의 자료형과 상관없이 배열에 추가 가능

```
var Users = [{name:'소녀시대', age:20}, {name:'걸스데이', age:22}];  
var add = function(a, b) {  
    return a + b;  
};  
Users.push(add)  
console.log('배열 요소의 수 : %d', Users.length);  
console.log('세 번째 요소로 추가된 함수 실행 : %d', Users[2](10, 10));
```



```
ch03_test9.js  
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch03_test9.js"  
배열 요소의 수 : 3  
세 번째 요소로 추가된 함수 실행 : 20  
Program exited with code 0  
행 16, 열 25 - 18 행  
INS JavaScript 스페이스: 4
```

ch03\_test9.js

## ch03\_test9.js 배열과 배열 안의 요소에 함수 할당하기

```
var Users = [{name:'소녀시대',age:20},{name:'걸스데이',age:22}];

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

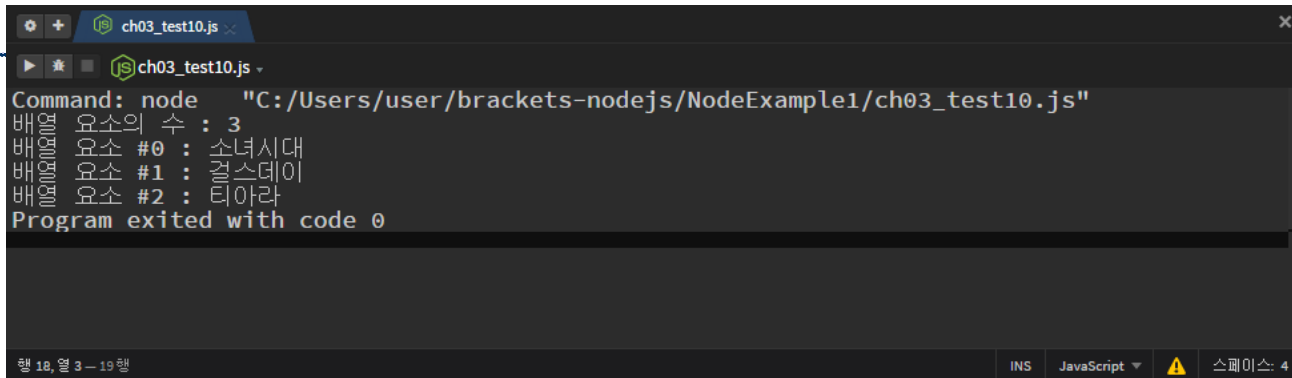
Users.push(add);

console.log('배열 요소의 수 : %d', Users.length);
console.log('세번째 요소로 추가된 함수 실행 : %d', Users[2](10,10));
```

# 배열의 원소를 하나씩 확인하기

- for 문에서 index를 사용하는 방법

```
var Users = [{name:'소녀시대', age:20}, {name:'걸스데이', age:22},  
{name:'티아라', age:23}];  
console.log('배열 요소의 수 : %d', Users.length);  
  
for (var i = 0; i < Users.length; i++) {  
    console.log('배열 요소 #' + i + ' : %s', Users[i].name);  
}
```



```
ch03_test10.js  
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch03_test10.js"  
배열 요소의 수 : 3  
배열 요소 #0 : 소녀시대  
배열 요소 #1 : 걸스데이  
배열 요소 #2 : 티아라  
Program exited with code 0  
행 18, 열 3 - 19 행  
INS JavaScript 스페이스: 4
```

ch03\_test10.js~

# forEach 구문 사용하기

- forEach 구문을 이용해 배열 원소를 하나씩 확인할 수 있음

```
console.log('forEach 구문 사용하기');  
Users.forEach(function(item, index) {  
    console.log('배열 요소 #' + index + ' : %s', item.name);  
});
```

## ch03\_test10.js 배열 안의 모든 요소를 하나씩 확인하기

```
var Users =
[ {name:'소녀시대',age:20}, {name:'걸스데이',age:22}, {name:'티아라',age:23}
];

console.log('배열 요소의 수 : %d', Users.length);
for (var i = 0; i < Users.length; i++) {
    console.log('배열 요소 #' + i + ' : %s', Users[i].name);
}

console.log('forEach 구문 사용하기');
Users.forEach(function(item, index) {
    console.log('배열 요소 #' + index + ' : %s', item.name);
});
```

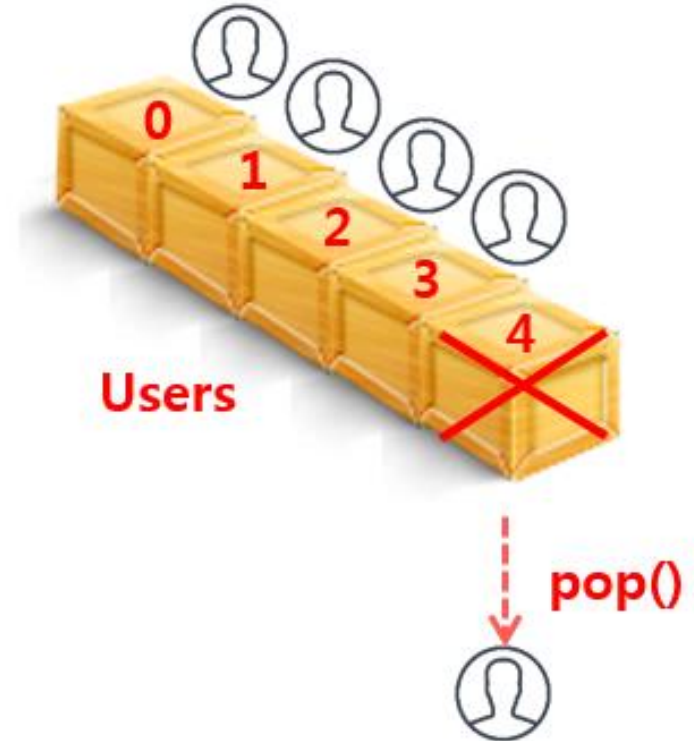
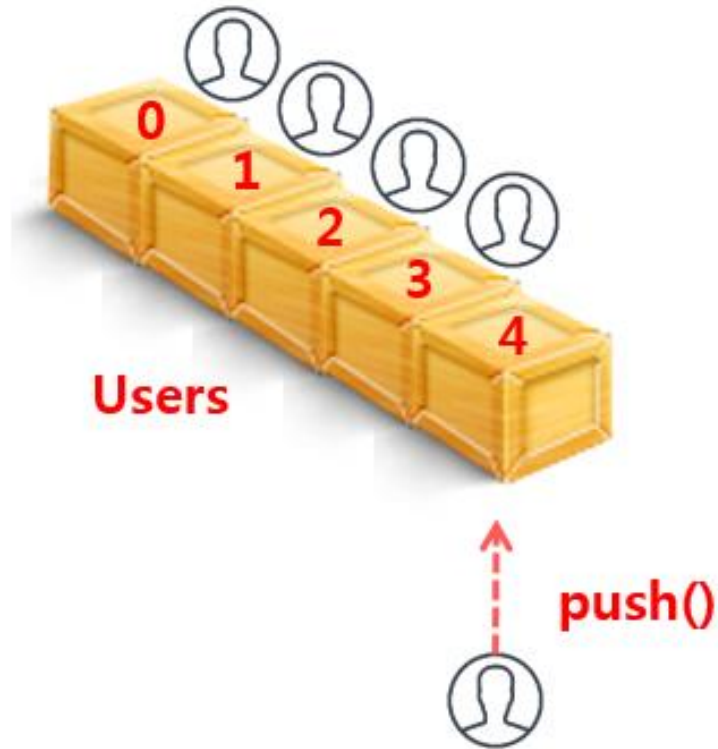
# 배열에 값 추가 및 삭제하기

- 배열의 끝에 원소를 추가하거나 삭제할 때 : push, pop
- 배열의 앞에 원소를 추가하거나 삭제할 때 : unshift, shift
- 여러 개의 원소를 한꺼번에 추가하거나 삭제할 때 : splice

속성 / 메소드 이름	설명
push(object)	배열의 끝에 요소를 추가합니다.
pop( )	배열의 끝에 있는 요소를 삭제합니다.
unshift( )	배열의 앞에 요소를 추가합니다.
shift( )	배열의 앞에 있는 요소를 삭제합니다.
splice(index, removeCount [,object])	여러 개의 객체를 요소로 추가하거나 삭제합니다.
slice(index, copyCount)	여러 개의 요소를 잘라내어 새로운 배열 객체로 만듭니다.



# push와 pop



배열 끝에 요소를 추가하거나 삭제하기  
push()와 pop() 메소드 사용

ch03\_test11.js

## ch03\_test11.js 배열 끝에 요소를 추가하거나 삭제하기

```
var Users = [{name:'소녀시대',age:20},{name:'걸스데이',age:22}];  
  
console.log('push() 호출 전 배열 요소의 수 : %d', Users.length);  
  
Users.push({name:'티아라',age:23});  
  
console.log('push() 호출 후 배열 요소의 수 : %d', Users.length);  
  
Users.pop();  
  
console.log('pop() 호출 후 배열 요소의 수 : %d', Users.length);
```

- push()와 pop() 메소드 사용

# unshift와 shift



배열 앞에 요소를 추가하거나 삭제하기  
35 `shift()`와 `unshift()` 메소드 사용

## ch03\_test12.js 배열 앞에 요소를 추가하거나 삭제하기

```
var Users = [{name:'소녀시대',age:20},{name:'걸스데이',age:22}];

console.log('unshift() 호출 전 배열 요소의 수 : %d', Users.length);

Users.unshift({name:'티아라',age:23});

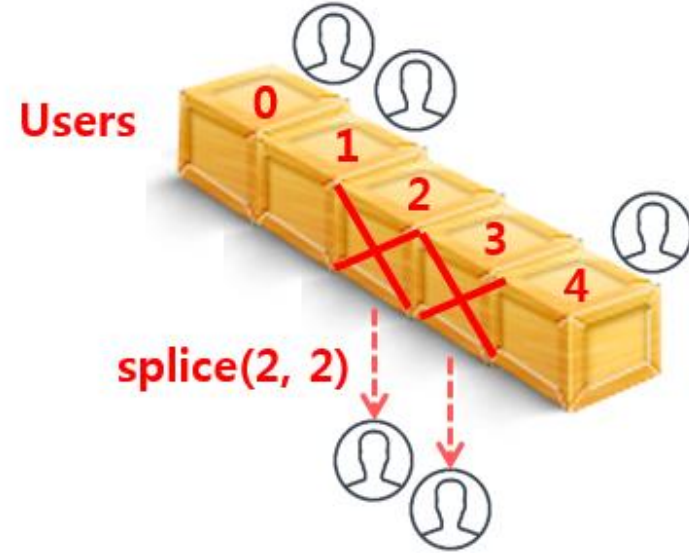
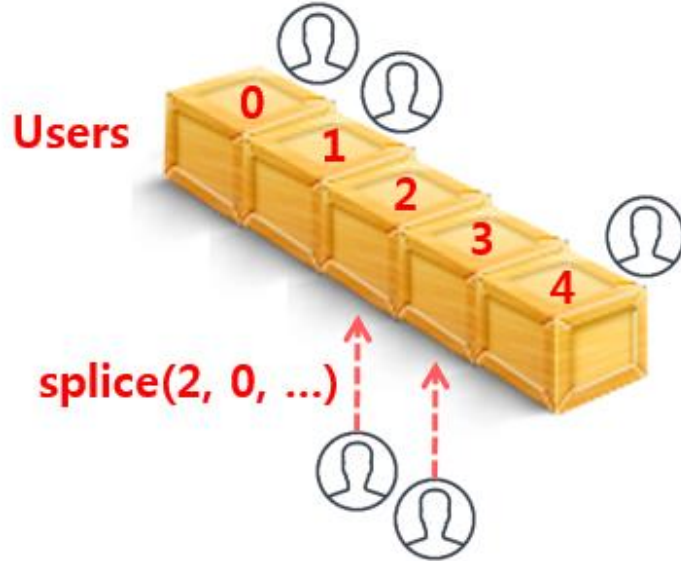
console.log('unshift() 호출 후 배열 요소의 수 : %d', Users.length);
console.dir(Users);

Users.shift();

console.log('shift() 호출 후 배열 요소의 수 : %d', Users.length);
console.dir(Users);
```

- shift()와 unshift() 메소드 사용

# splice



```
Users.splice(1, 0, {name:'애프터스쿨', age:25});  
console.log('splice()로 요소를 인덱스 1에 추가한 후');  
console.dir(Users);  
Users.splice(2, 1);  
console.log('splice()로 인덱스 2의 요소를 1개 삭제한 후');  
console.dir(Users);
```

배열의 요소 삭제하기, 요소들을 추가하거나 삭제하기

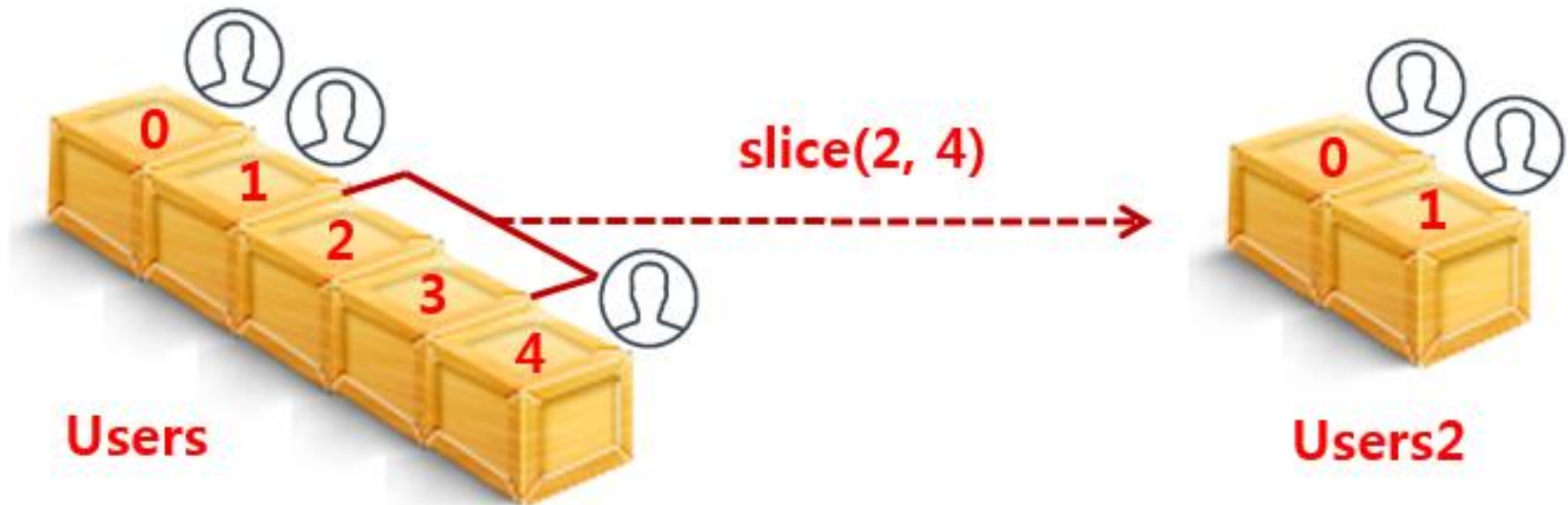
37 delete 키워드 사용, splice() 메소드 사용

## ch03\_test13.js 배열의 요소 삭제하기, 요소들을 추가하거나 삭제하기

```
var Users =  
[  
  {name:'소녀시대',age:20},  
  {name:'걸스데이',age:22},  
  {name:'티아라',age:23}  
];  
console.log('delete 키워드로 배열 요소 삭제 전 배열 요소의 수 : %d',  
Users.length);  
delete Users[1];  
console.log('delete 키워드로 배열 요소 삭제 후');  
console.dir(Users);  
  
Users.splice(1, 0, {name:'애프터스쿨',age:25});  
console.log('splice()로 요소를 인덱스 1에 추가한 후');  
console.dir(Users);  
  
Users.splice(2, 1);  
console.log('splice()로 인덱스 1의 요소를 1개 삭제한 후');  
console.dir(Users);
```

- delete 키워드 사용, splice() 메소드 사용

# slice



배열 일부를 잘라내어 새로운 객체로 만들기  
`slice()` 메소드 사용



## ch03\_test14.js 배열 일부를 잘라내어 새로운 객체로 만들기

```
var Users =  
[  
  {name:'소녀시대',age:20},  
  {name:'걸스데이',age:22},  
  {name:'티아라',age:23},  
  {name:'애프터스쿨',age:25}],  
console.log('배열 요소의 수 : %d', Users.length);  
console.log('원본 Users');  
console.dir(Users);  
  
var Users2 = Users.slice(1, 3);  
  
console.log('slice()로 잘라낸 후 Users2');  
console.dir(Users2);  
  
var Users3 = Users2.slice(1);  
  
console.log('slice()로 잘라낸 후 Users3');  
console.dir(Users3);
```

- slice() 메소드 사용

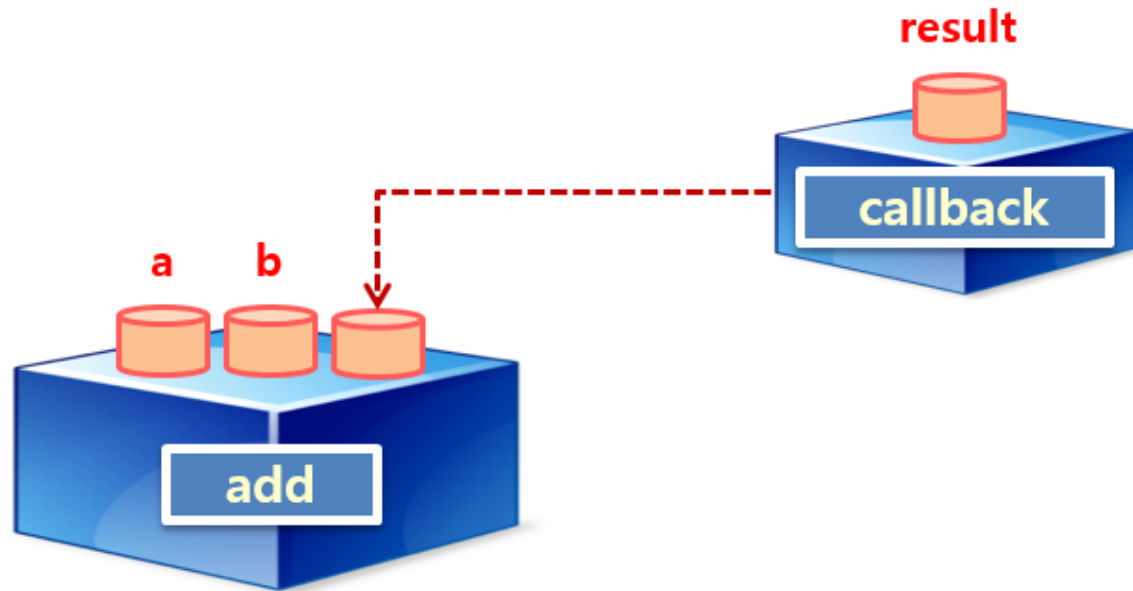


3.

콜백 함수 이해하기

# 함수를 파라미터로 전달하기

- 변수에 함수를 할당할 수 있으므로 함수를 호출할 때 파라미터로 다른 함수 전달 가능



콜백함수를 파라미터로 전달

```
function add( a, b, callback) {  
    var result = a + b;  
    callback(result);  
}
```

# 콜백 함수를 파라미터로 전달하기

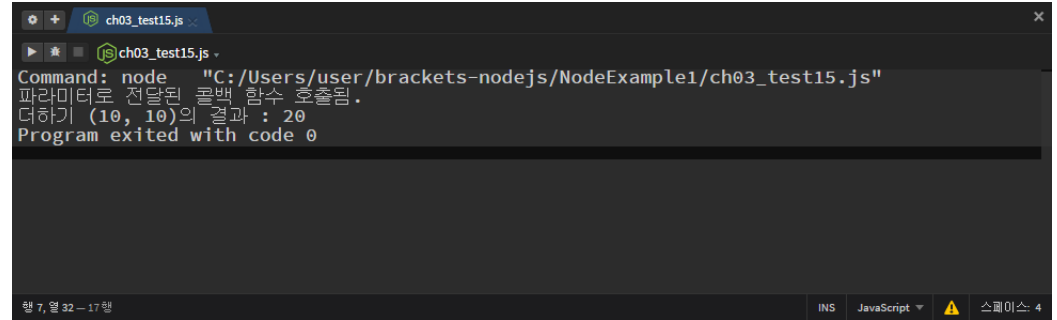
- 콜백함수 : 함수를 파라미터로 전달했을 때 특정 시점에 그 함수를 실행시켜 주는 경우

```
function add(a, b, callback) {  
    var result = a + b;  
    callback(result);  
}
```

```
add(10, 10, function(result) {  
    console.log('파라미터로 전달된 콜백 함수 호출됨.');
```

더하기 (10, 10)의 결과 : 20

```
    console.log('더하기 (10, 10)의 결과 : %d', result);  
});
```



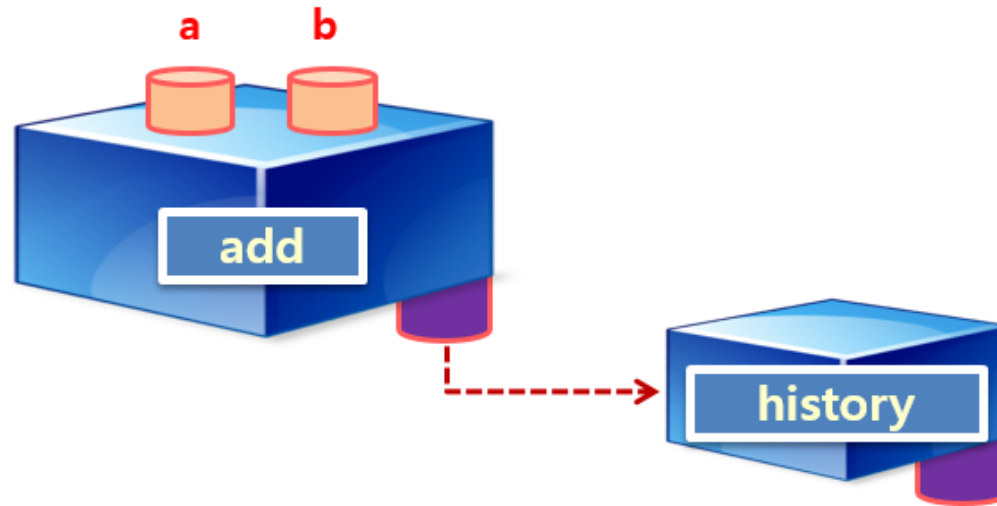
```
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch03_test15.js"  
파라미터로 전달된 콜백 함수 호출됨.  
더하기 (10, 10)의 결과 : 20  
Program exited with code 0
```

ch03\_test15.js

```
function add( a, b, callback) {  
    var result = a + b;  
    callback(result);  
}  
  
add(10, 10, function(result) {  
    console.log('파라미터로 전달된 콜백 함수 호출됨.');    console.log('더하기 (10, 10)의 결과 : %d', result);  
});
```

# 함수에서 반환하는 값이 함수인 경우

- 함수의 결과를 반환할 때 함수를 반환할 수 있음



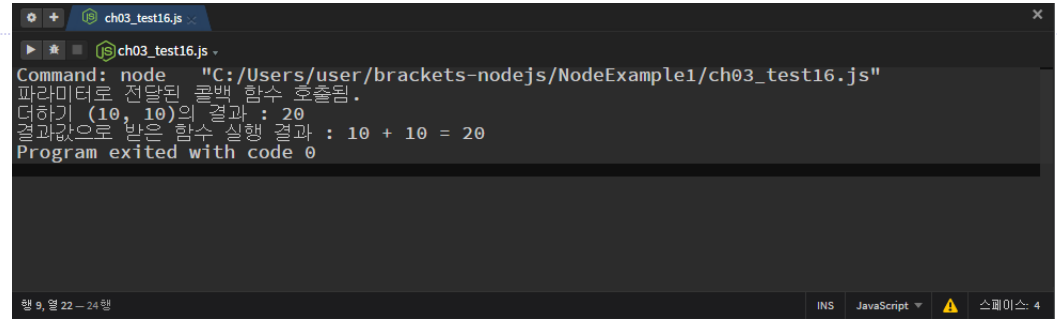
```
function add( a, b, callback) {  
  var result = a + b;  
  callback(result);  
  
  var history = function() {  
    return a + ' + ' + b + ' = ' + result;  
  };  
  return history;  
}
```

함수를 리턴

# 더하기 함수를 실행했을 때 기록을 남겨두었다가 출력하기

- 함수를 실행했을 때 함수를 반환 받고 반환된 함수를 실행하여 결과를 확인하는 방법

```
function add(a, b, callback) {  
  var result = a + b;  
  callback(result);  
  var history = function() {  
    return a + ' + ' + b + ' = ' + result;  
  };  
  return history;  
}  
  
var add_history = add(10, 10, function(result) {  
  console.log('파라미터로 전달된 콜백 함수 호출됨.');  console.log('더하기 (10, 10)의 결과 : %d', result);  
});  
  
console.log('결과 값으로 받은 함수 실행 결과 : ' + add_history());
```



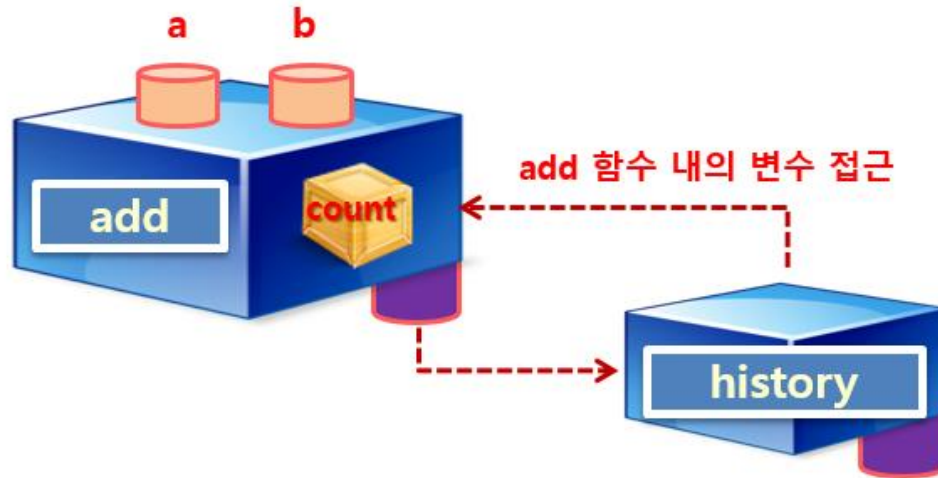
```
ch03_test16.js  
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch03_test16.js"  
파라미터로 전달된 콜백 함수 호출됨.  
더하기 (10, 10)의 결과 : 20  
결과값으로 받은 함수 실행 결과 : 10 + 10 = 20  
Program exited with code 0  
행 9, 열 22 - 24 행  
INS JavaScript 스킴아스: 4
```

ch03\_test16.js

```
function add(a, b, callback) {  
    var result = a + b;  
    callback(result);  
  
    var history = function() {  
        return a + ' + ' + b + ' = ' + result;  
    };  
    return history;  
}  
  
var history = add(10, 10, function(result) {  
    console.log('파라미터로 전달된 콜백 함수 호출됨.');    console.log('더하기 (10, 10)의 결과 : %d', result);  
});  
  
console.log('결과값으로 받은 함수 실행 결과 : ' + history());
```

# 함수에서 반환된 함수 안에서 변수에 접근하는 경우

- 처음 실행한 함수 안에서 접근하던 변수는 반환된 함수에서 계속 접근 가능



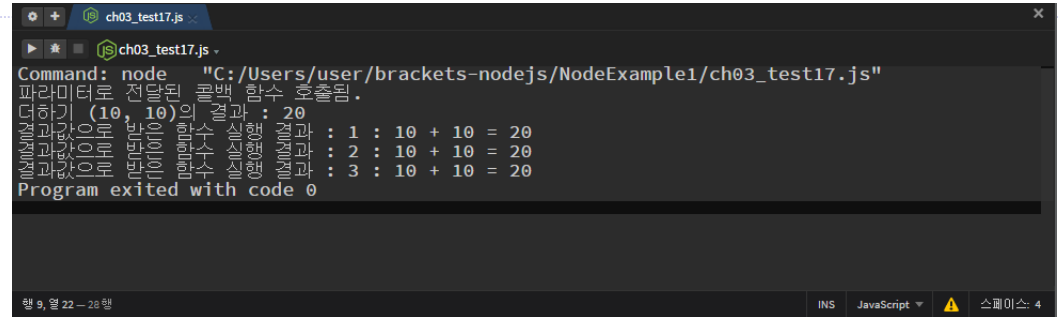
```
function add( a, b, callback) {  
  var count = 0;  
  
  var history = function() {  
    count++;  
    return count + ' : ' + a + ' + ' + b + ' = ' + result;  
  };  
  return history;  
}
```



# 반환된 함수에서 함수 내부의 변수 접근

- 반환된 함수에서 이 함수를 반환했던 함수 내부의 변수를 접근하는 방법

```
function add(a, b, callback) {  
    var result = a + b;  
    callback(result);  
    var count = 0;  
    var history = function() {  
        count++;  
        return count + ':' + a + ' + ' + b + ' = ' + result;  
    };  
    return history;  
}  
  
var add_history = add(10, 10, function(result) {  
    console.log('파라미터로 전달된 콜백 함수 호출됨.');    console.log('더하기 (10, 10)의 결과 : %d', result);  
});  
  
console.log('결과 값으로 받은 함수 실행 결과 : ' + add_history());  
console.log('결과 값으로 받은 함수 실행 결과 : ' + add_history());  
console.log('결과 값으로 받은 함수 실행 결과 : ' + add_history());
```



```
ch03_test17.js  
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch03_test17.js"  
파라미터로 전달된 콜백 함수 호출됨.  
더하기 (10, 10)의 결과 : 20  
결과 값으로 받은 함수 실행 결과 : 1 : 10 + 10 = 20  
결과 값으로 받은 함수 실행 결과 : 2 : 10 + 10 = 20  
결과 값으로 받은 함수 실행 결과 : 3 : 10 + 10 = 20  
Program exited with code 0  
행 9, 열 22 - 28 행  
INS JavaScript 스킴아스: 4
```

ch03\_test17.js

```
function add(a, b, callback) {  
    var result = a + b;  
    callback(result);  
    var count = 0;  
    var history = function() {  
        count++;  
        return count + ' : ' + a + ' + ' + b + ' = ' + result;  
    };  
    return history;  
}  
var history = add(10, 10, function(result) {  
    console.log('파라미터로 전달된 콜백 함수 호출됨.');    console.log('더하기 (10, 10)의 결과 : %d', result);  
});  
console.log('결과값으로 받은 함수 실행 결과 : ' + history());  
console.log('결과값으로 받은 함수 실행 결과 : ' + history());  
console.log('결과값으로 받은 함수 실행 결과 : ' + history());
```

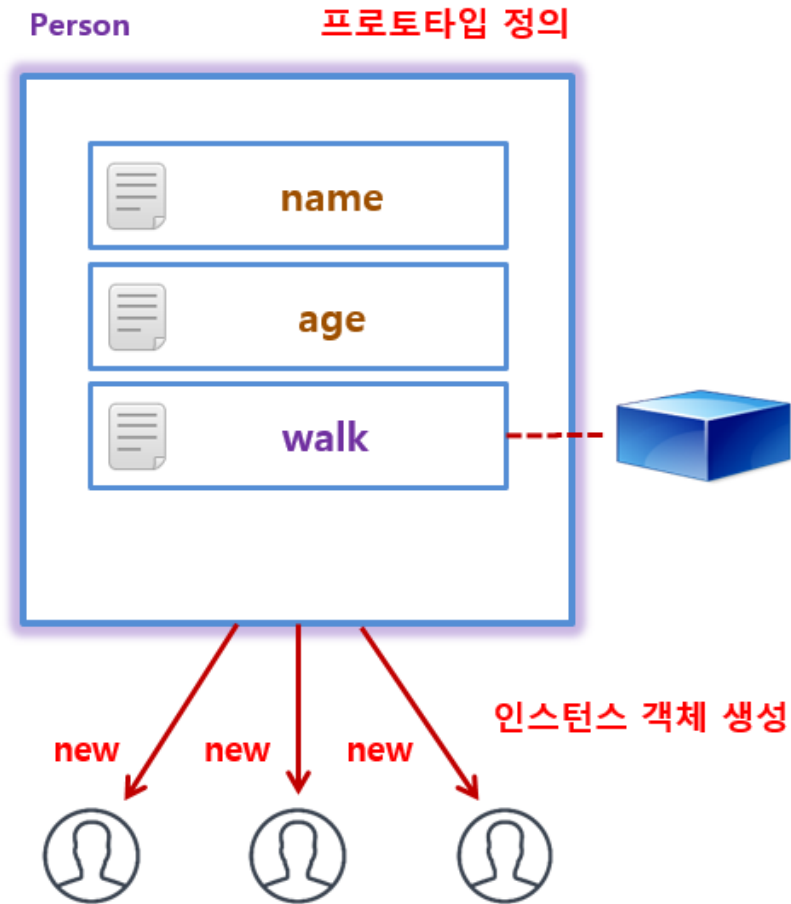
- 내부함수에서 외부 함수의 변수에 접근 했을 때 그 값이 그대로 유지 되는 것

4.

프로토타입 객체 만들기

# 자바스크립트에서 객체지향 방식으로 만들기

- 객체의 원형을 프로토타입(Prototype)이라 하고 이 프로토타입이 클래스(Class)의 역할을 함
- 프로토타입을 만들고 new 연산자를 이용해 새로운 객체를 만들어낼 수 있음



# Person 프로토타입을 만들고 객체 생성하기

- Person이라는 이름의 함수를 만들고 프로토타입 객체로 사용
- new 연산자를 사용하는 시점에 생성자 함수로 동작

```
function Person(name, age) {  
    this.name = name;  
    this.age = age;  
}
```

```
Person.prototype.walk = function(speed) {  
    console.log(speed + 'km 속도로 걸어갑니다.');
```

```
var person01 = new Person('소녀시대', 20);
```

## ch03\_test18.js   프로토타입 객체 만들기

```
function Person(name, age) {  
    this.name = name;  
    this.age = age;  
}  
  
Person.prototype.walk = function(speed) {  
    console.log(speed + 'km 속도로 걸어갑니다.');}  
  
var person01 = new Person('소녀시대', 20);  
var person02 = new Person('걸스데이', 22);  
  
console.log(person01.name + ' 객체의 walk(10)을 호출합니다.');person01.walk(10);
```

# Person 안에 자동으로 만들어지는 prototype 속성

- 함수를 만들 때 자동으로 생성됨
- 함수를 prototype 속성에 추가하면 new 연산자를 이용해 만든 객체에서 공용으로 사용

