**리액트 1주차 스터디**

20 황수연

**<1장. 자바스크립트 문법>**

1. Hello Javascript!

* 개발자 도구 Ctrl + Shift + I 를 눌러서 열 수 있고, 여기서 콘솔 창을 사용할 수 있다.
* 매번 개발자 도구 사용은 불편하므로 CodeSandBox 라는 사이트를 사용한다.

1. 변수와 상수

* 변수 선언

1) let 키워드: 같은 이름의 변수는 다시 선언하지 못하지만, 다른 블록 범위 내에서는 똑같은 이름으로 사용 가능하다.

2) var 키워드: 모던 자바스크립트에서는 더 이상 사용하지 않는 선언 방법이다. let과의 차이점으로는 같은 이름으로 여러 번 선언할 수 있다는 점이다.

* 상수 선언: const 키워드를 통해 선언할 수 있다. 상수로 선언하면 값은 바꿀 수 없다.
* 데이터 타입: 문자열은 따옴표로 감싸서 선언(대소 상관X), boolean(true, false), null 은 우리가 없다고 고의적으로 설정하는 값, undefined는 우리가 설정을 하지 않았기 때문에 없는 값을 의미한다.

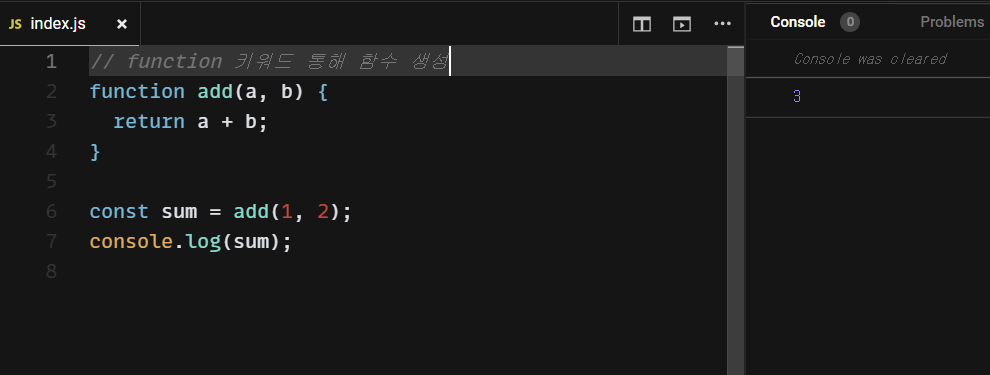
1. 연산자

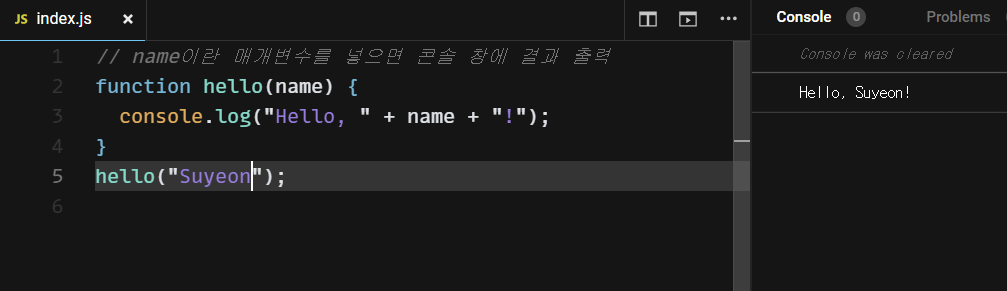
* 산술 연산자: 사칙연산과 같은 작업을 하는 연산자(+, -, \* /), a++, ++a와 같은 연산자.
* 대입 연산자: 특정 값에 연산을 한 값을 바로 설정할 때 사용( a+= 3;)
* 논리 연산자: NOT연산자(!), AND연산자, OR연산자(||) (연산 순서: NOT -> AND -> OR)
* 비교 연산자: === 는 타입 검사를 하지만, == 는 타입 검사를 하지 않는다.

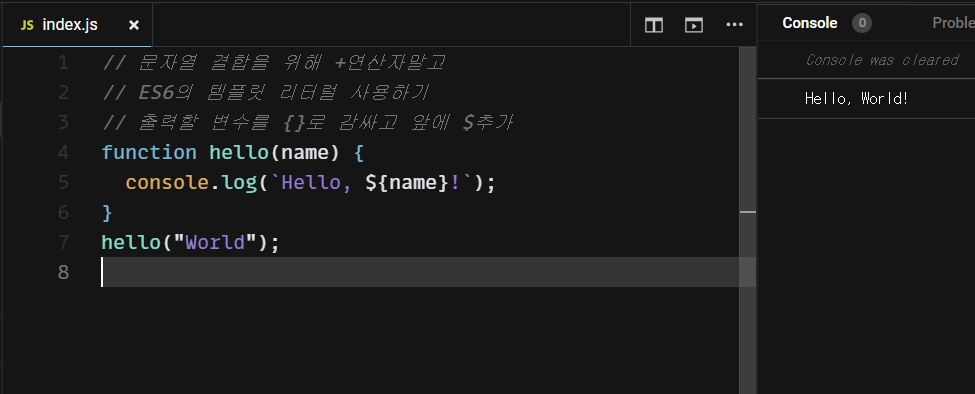
1. 조건문

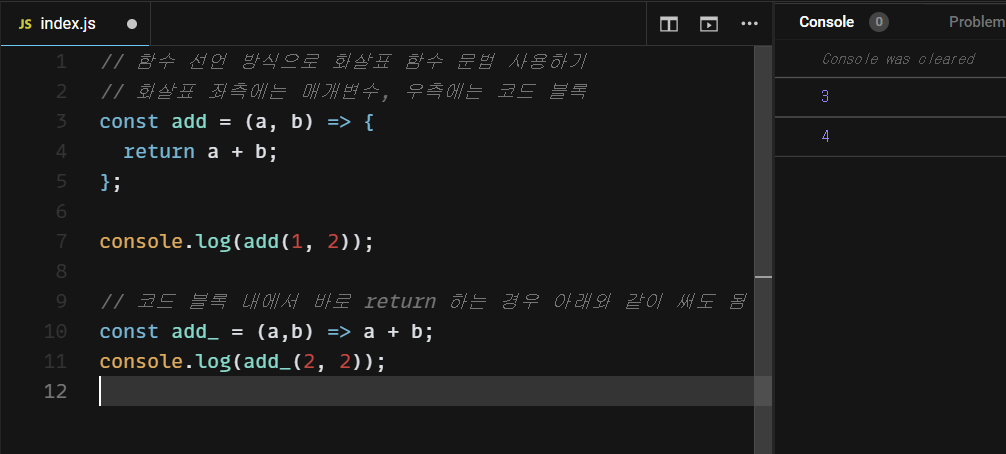
* if 문: if (조건) { 코드 ; }, 조건이 만족 될 때에만 특정 코드 실행
* if-else 문: 특정 조건이 만족 할 때와 만족하지 않을 때 다른 코드 실행
* if-else if 문: 여러 조건에 따라 다른 작업
* switch/case 문: 특정 값이 무엇이냐에 따라 다른 작업

1. 함수



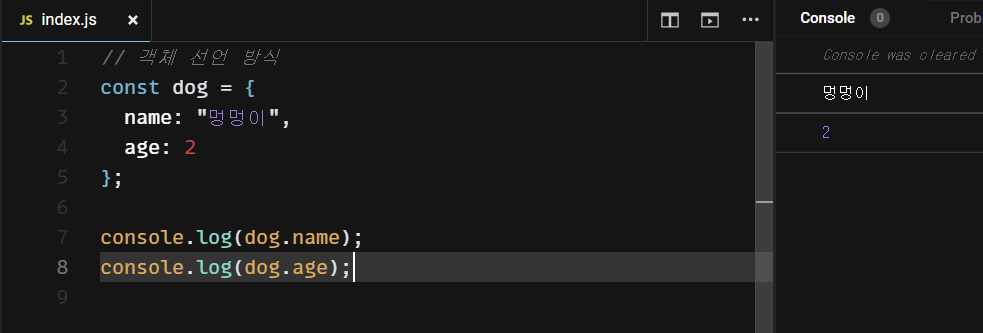




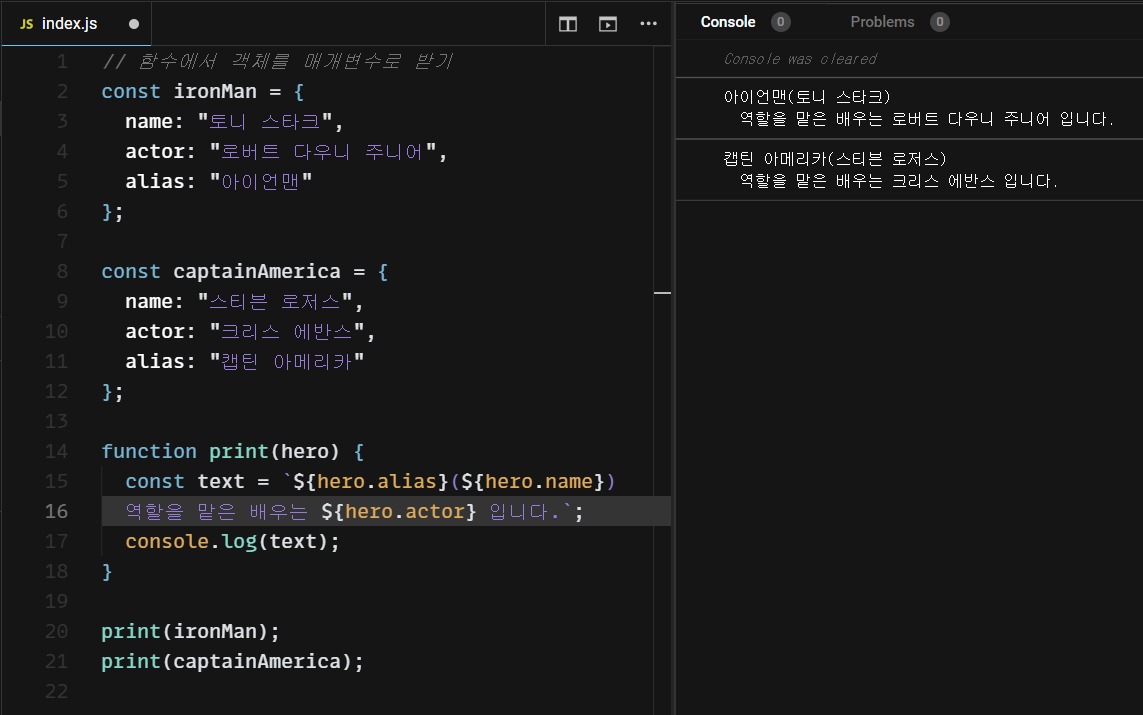


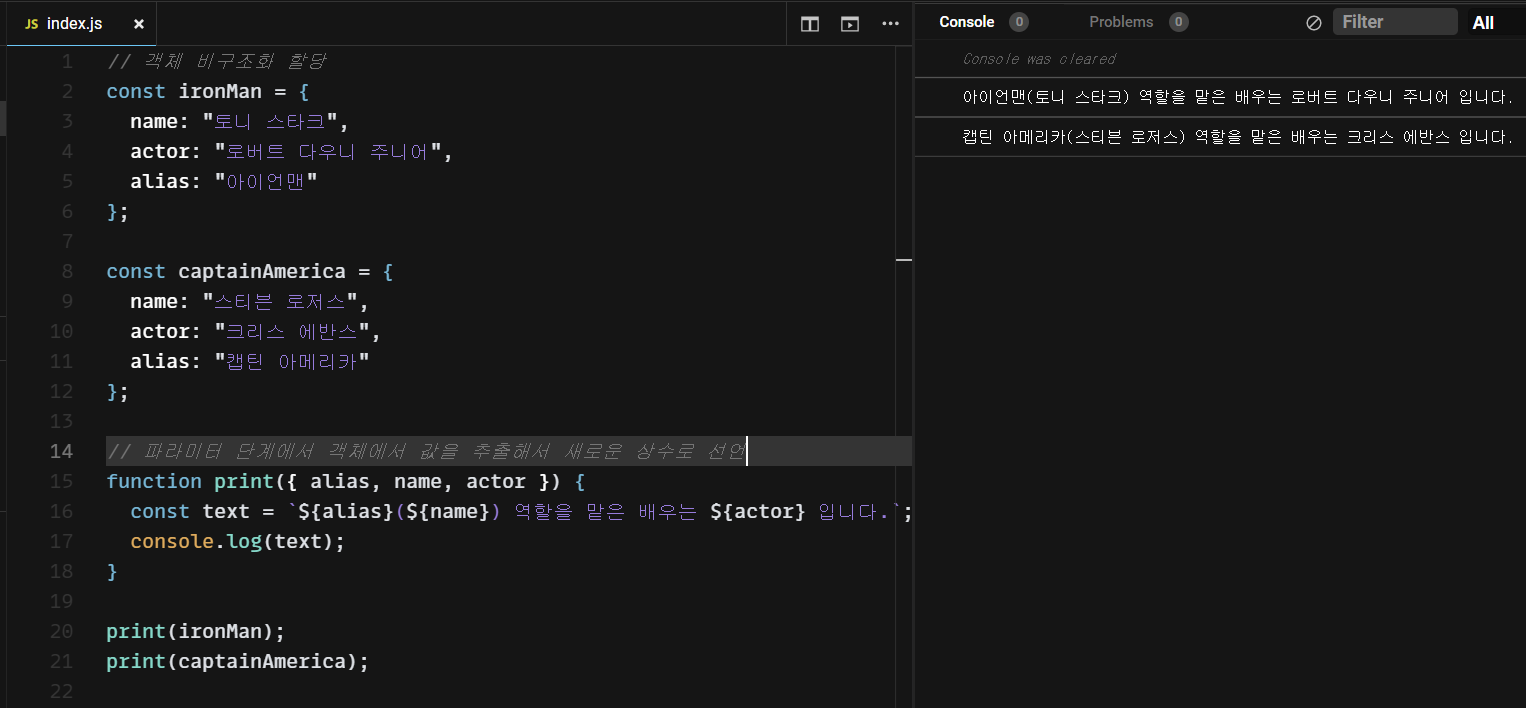
화살표 함수와 일반 function 으로 만든 함수의 차이점은 this 가 서로 다르다는 것이다.

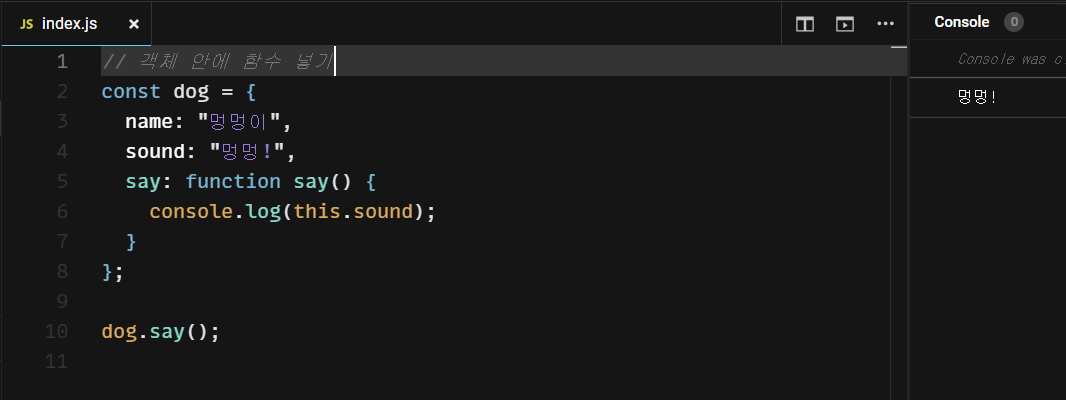
1. 객체



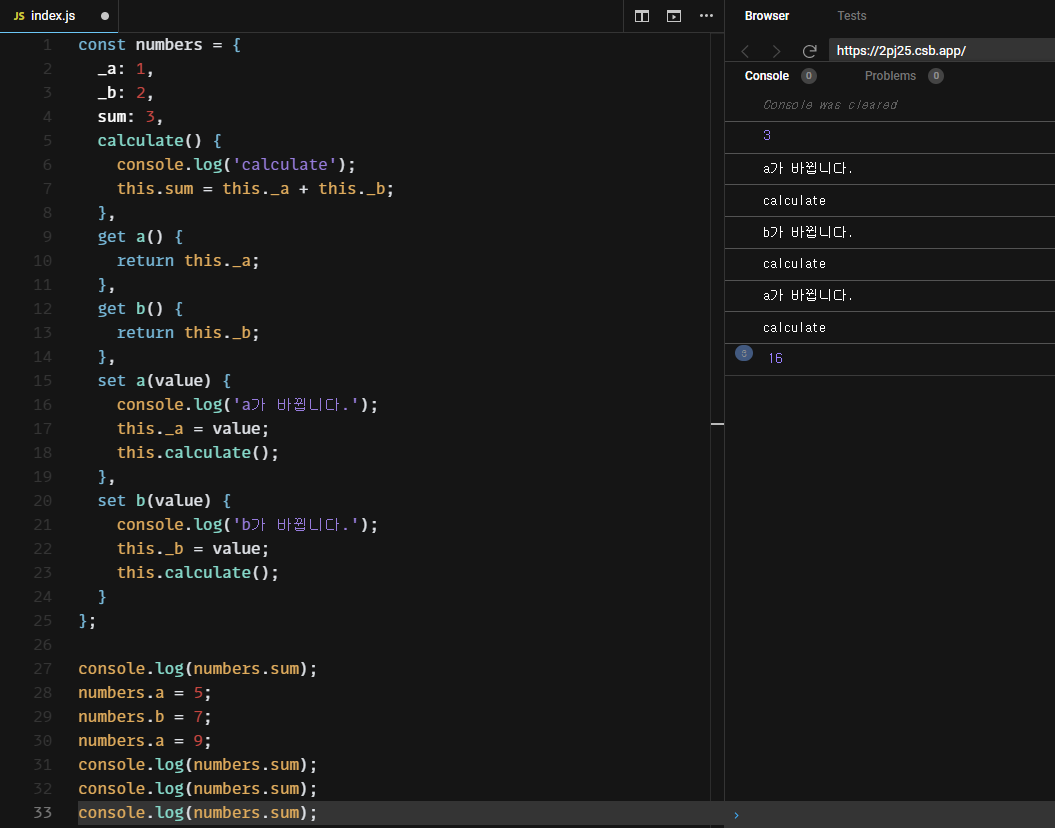
* 객체를 선언할 때에는 { 키 : 원하는 값 } 의 형태로 한다. 키에 해당하는 부분은 공백이 없어야 한다. 공백이 있어야 한다면 따옴표를 통해 문자열로 넣어준다.





 - 여기서 this는 자신이 속한 객체를 가리킨다. 객체 안에 함수를 넣을 때, 화살표 함수로 선언한다면 화살표 함수의 this는 자신이 속한 객체를 가리키지 않기 때문에 제대로 작동하지 않는다.

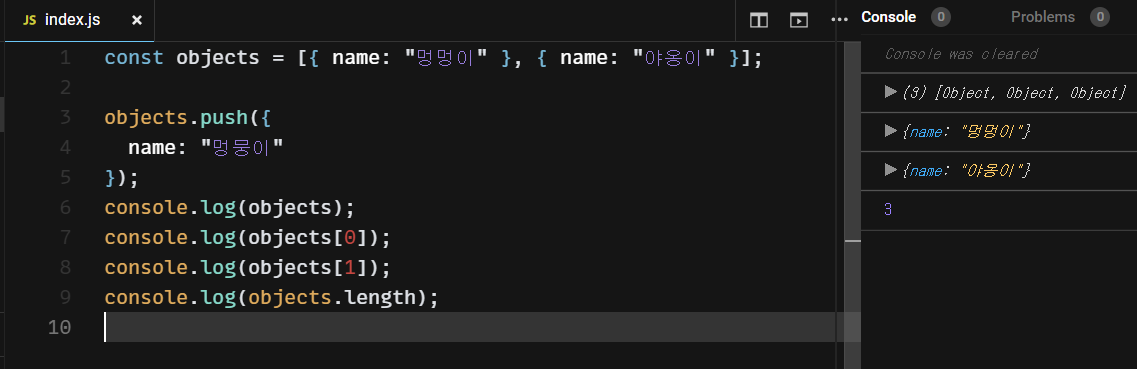
* Getter 함수와 setter 함수: 특정 값을 바꾸거나 조회하려 할 때 실행시키는 코드



* Setter 함수: numbers.a = 5 이렇게 값을 설정했을 때 5를 함수의 파라미터로 받아오게 된다.
* Getter 함수: 특정 값을 조회 할 때 우리가 설정한 함수로 연산한 값을 반환

1. 배열

* 객체 배열 생성: [] 로 감싼다. 배열의 n번째 항목을 조회 시 [n]으로 조회한다.
* 배열에 원소 추가: 배열 내장 함수인 push 함수 사용
* 배열의 크기: length 사용



1. 반복문

* For문:

for (초기 구문; 조건 구문; 변화 구문;) {

코드

}

* While 문:

let i = 0;

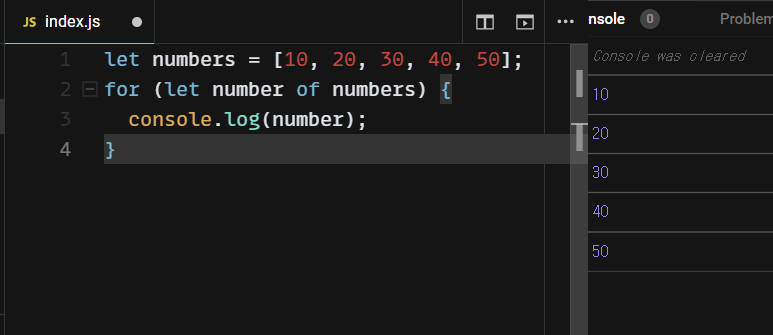
while (i < 10) {

console.log(i);

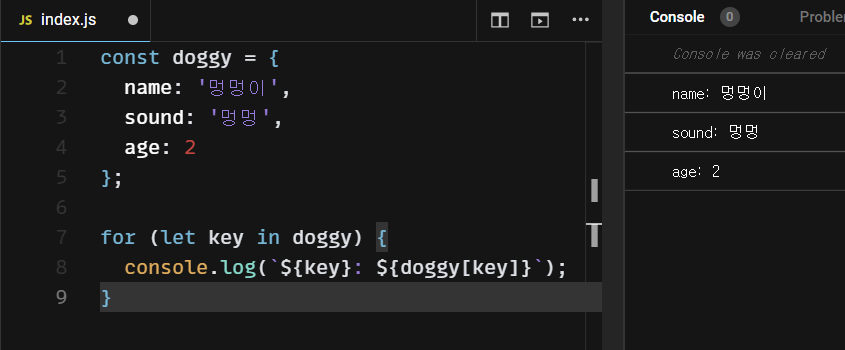
i++;

}

* For … of 문: 배열에 관한 반복문



* 객체에 대한 함수
* Object.entries : [[키, 값], [키, 값]]의 형태의 배열로 반환
* Object.keys : [키, 키, 키] 의 형태의 배열로 반환
* Object.values: [값, 값, 값]의 형태의 배열로 반환
* 객체를 위한 반복문 For … in 문



* Break와 continue: 반복문에서 사용, 반복문을 끝내거나 다음 반복문을 실행하게끔 한다.

for (let i = 0; i < 10; i++) {

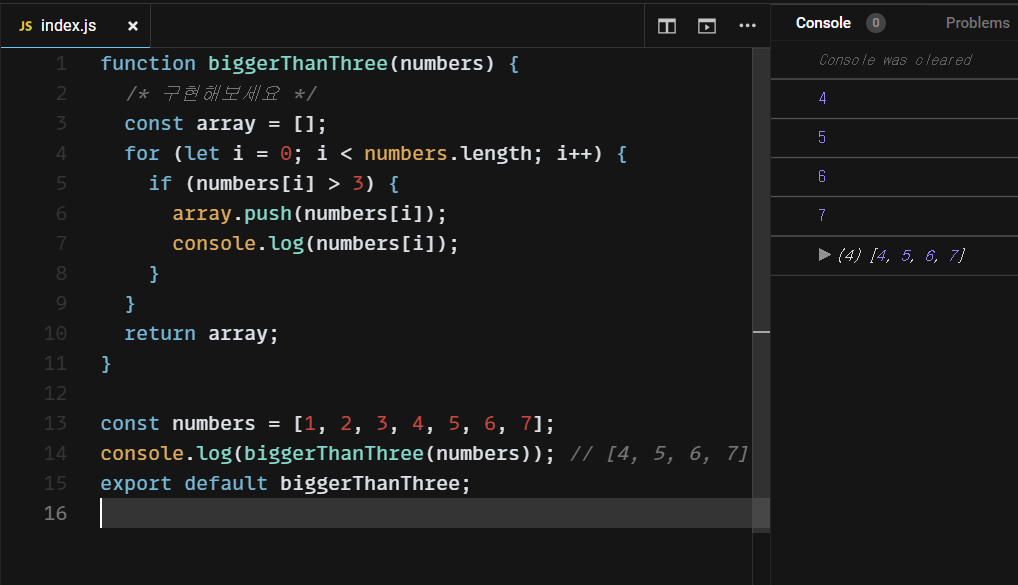
if (i === 2) continue; // 다음 루프를 실행

console.log(i);

if (i === 5) break; // 반복문을 끝내기

}

* 퀴즈:



1. 배열 내장 함수

* forEach문: 매개변수로 각 원소에 대해 처리하고 싶은 코드를 함수로 넣는다.
* 콜백함수: 함수 형태의 파라미터를 전달하는 것

const superheroes = ['아이언맨', '캡틴 아메리카', '토르', '닥터 스트레인지'];

superheroes.forEach(hero => {

console.log(hero);

});

* Map: 배열 안의 각 원소를 변환할 때 사용하며, 새로운 배열이 만들어진다. 파라미터로는 변화를 주는 함수를 전달한다. (=변화함수) 변화함수는 꼭 이름을 붙일 필요는 없고 매개변수 안에서 작성해도 된다.

const array = [1, 2, 3, 4, 5, 6, 7, 8];

const square = n => n \* n; // 변화함수

const squared = array.map(square);

// const squared = array.map(n => n \* n): 과 같이 작성 가능

console.log(squared);

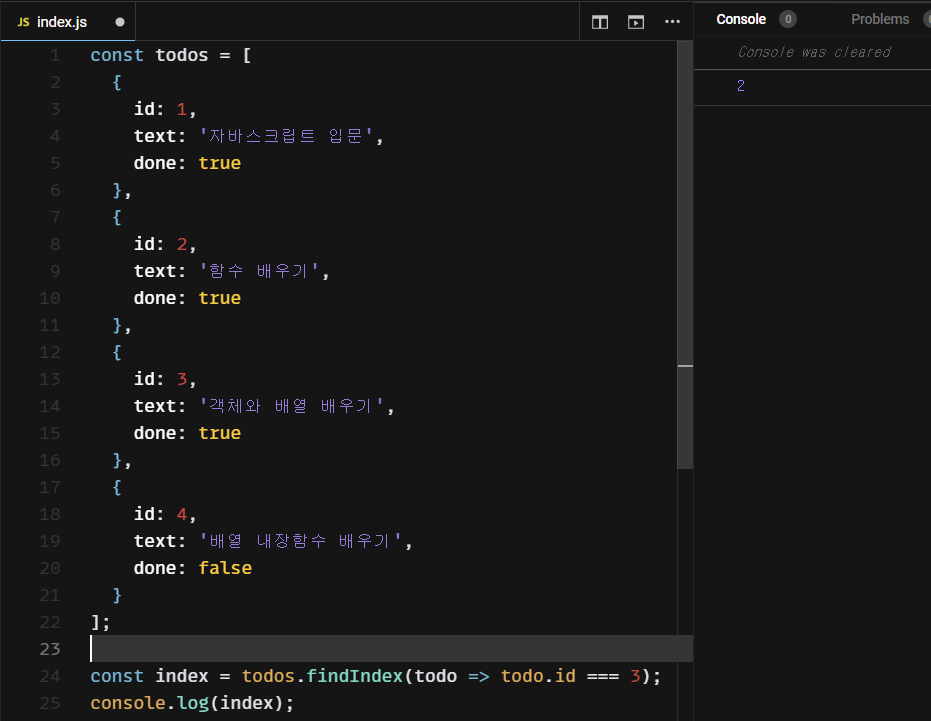
* indexOf: 원하는 항목이 몇 번째 원소인지 찾아주는 함수이다.

const superheroes = ['아이언맨', '캡틴 아메리카', '토르', '닥터 스트레인지'];

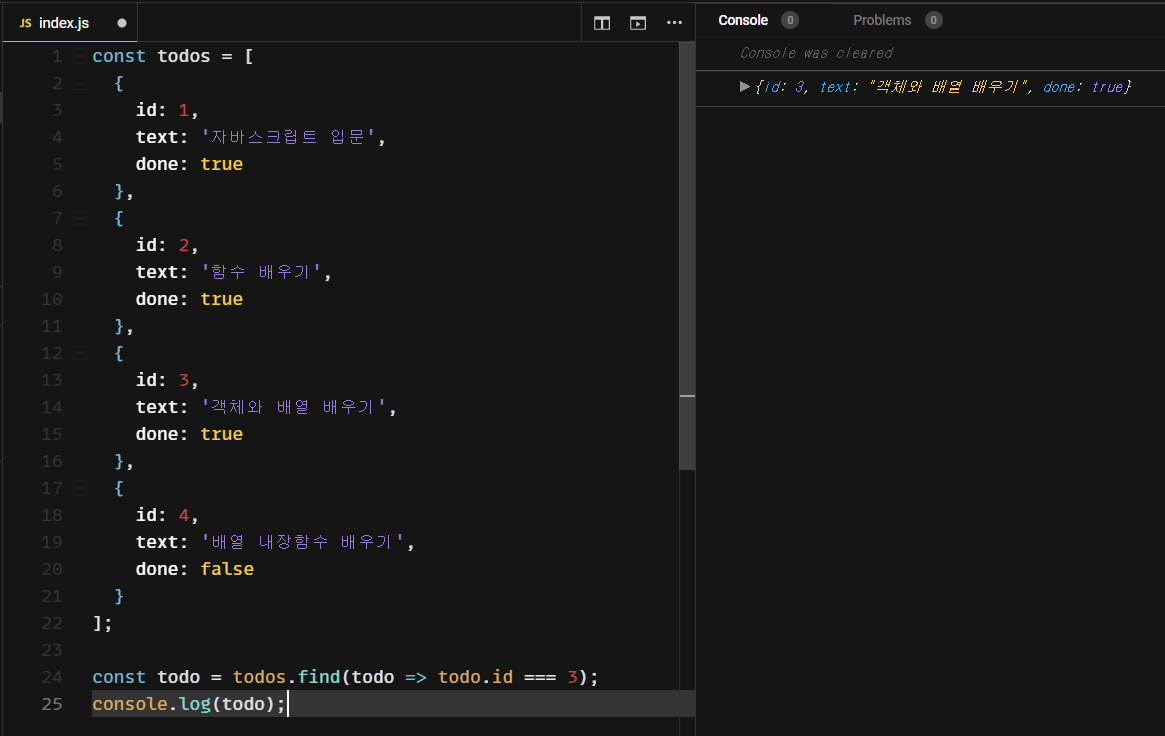
const index = superheroes.indexOf('토르');

console.log(index); // 결과는 2

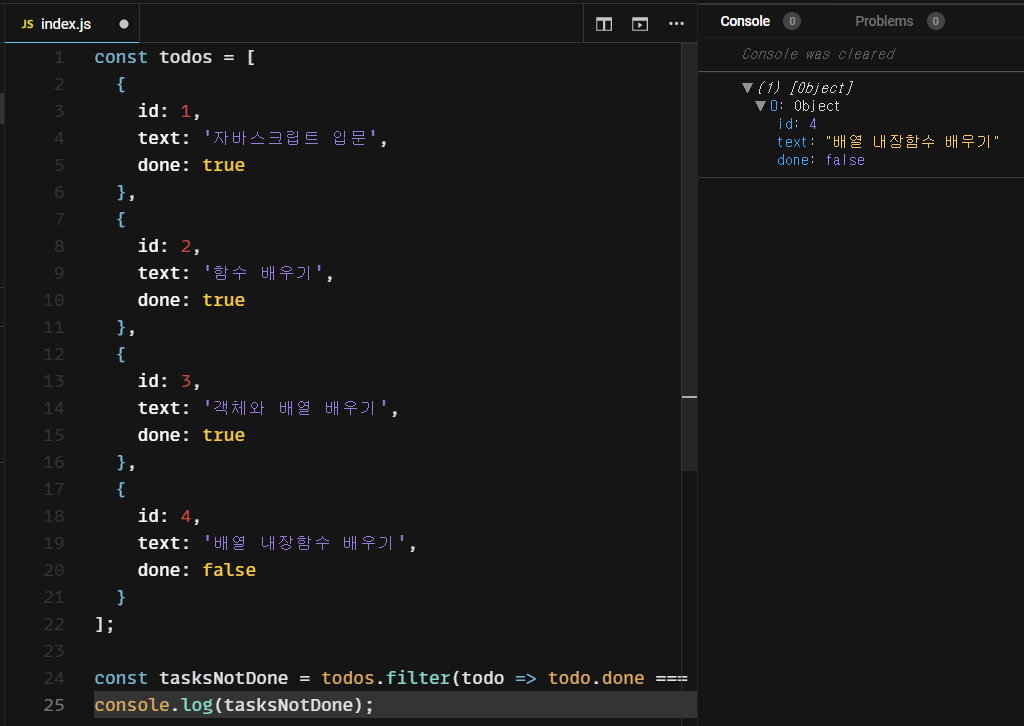
* findIndex: 객체 배열이거나 배열의 배열인 경우 indexOf 를 사용할 수 없다. 따라서 findIndex 함수에 검사하고자 하는 조건을 반환하는 함수를 넣어서 찾을 수 있다.



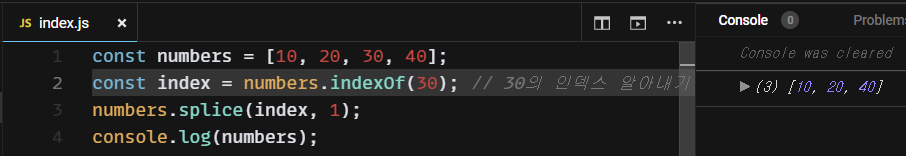
* find: 찾아낸 값 자체를 반환한다.



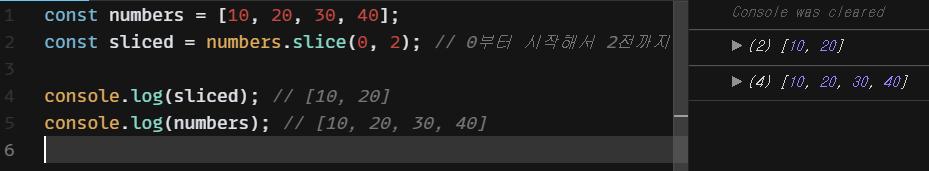
* filter: 특정 조건을 만족하는 값들만 따로 추출하여 새로운 배열을 만든다. 파라미터는 조건을 검사하는 함수이며, 이 함수의 파라미터로 각 원소의 값을 받아온다. 파라미터로 넣은 함수에서 true를 반환하면 새로운 배열에 따로 추출한다.



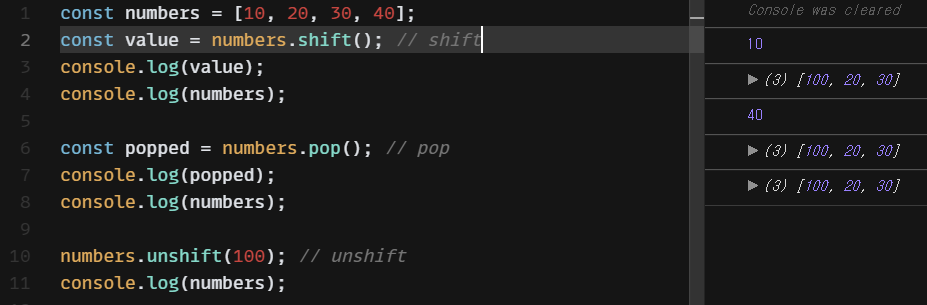
* splice: 배열에서 특정 항목을 제거할 때 사용한다. 첫번째 파라미터는 지우기 시작할 인덱스, 두번째 파라미터는 그 인덱스부터 몇 개를 지울지를 의미한다.



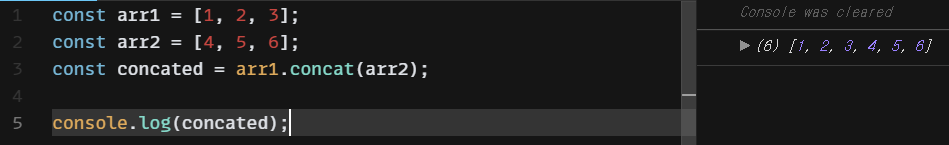
* slice: splice와 비슷하지만 기존의 배열을 건드리지 않는다는 점에서 다르다. 첫번째 파라미터는 어디서부터 자를지, 두번째 파라미터는 어디까지 자를지를 의미한다.



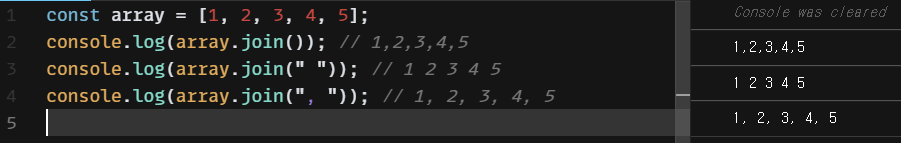
* shift와 pop: shift는 첫번째 원소를 배열에서 추출하며, 배열에서 해당 원소는 사라진다. Pop은 push의 반대로, 배열의 맨 마지막 항목을 추출한다.
* unshift: shift의 반대로, 맨 앞에 새 원소를 추가한다.



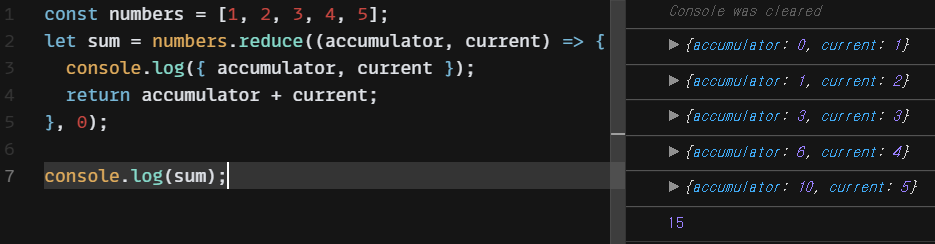
* concat: 여러 개의 배열을 하나로 합치며, 원본 배열은 바뀌지 않는다.



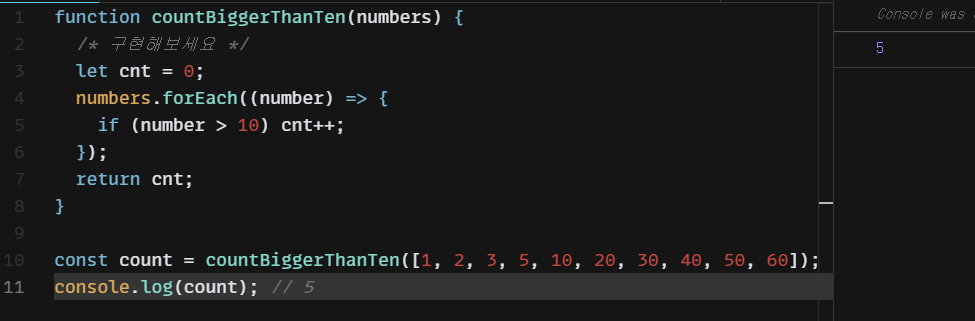
* join: 배열 안의 값들을 문자열 형태로 합친다.



* reduce: 주어진 배열에 대해 총 합을 구해야 하는 상황에서 유용하다. 첫번째 파라미터는 accumulator와 current를 파라미터로 가져와서 결과를 반환하는 콜백함수, 두번째 파라미터는 reduce 함수에서 사용할 초깃값이다. 여기서 accumulator는 누적된 값을 의미한다.

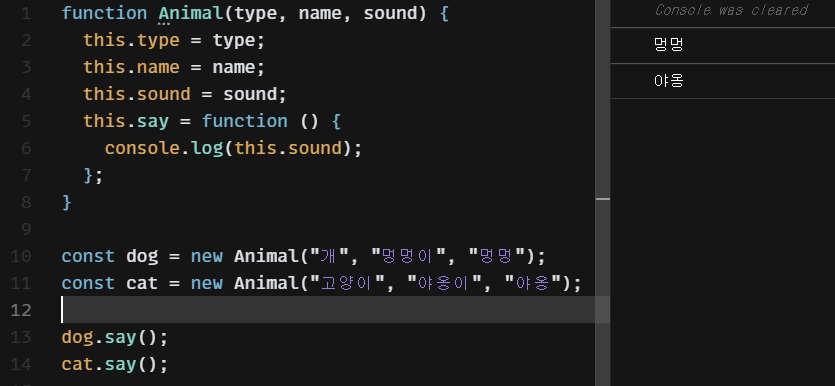


* 퀴즈:

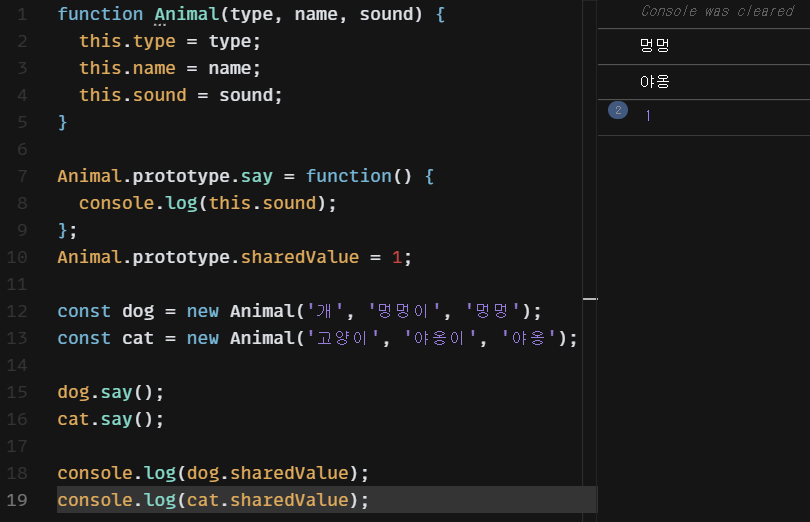


1. 프로토타입과 클래스

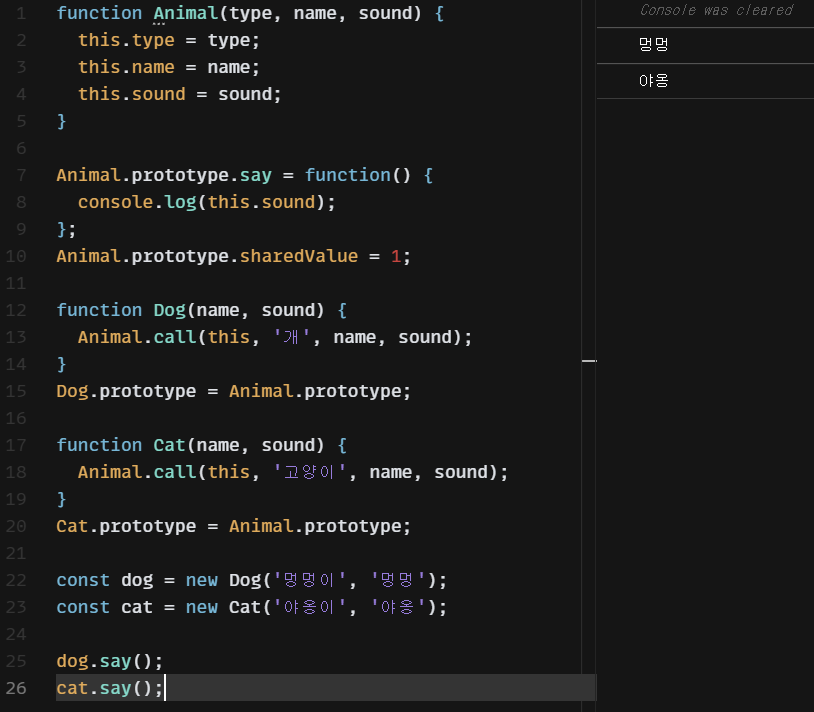
* 객체 생성자: 새로운 객체를 만들고 그 안에 넣고 싶은 값 혹은 함수들을 구현할 수 있게 해준다. 보통 함수의 이름은 대문자로 시작하고, 새로운 객체를 만들 때에는 new 키워드를 사용한다.



* 프로토타입: 같은 객체 생성자 함수를 사용하는 경우, 특정 함수 또는 값을 재사용할 수 있게 해준다. 객체 생성자 함수 아래에 .prototype.[원하는키] = 코드를 입력하여 설정할 수 있다.

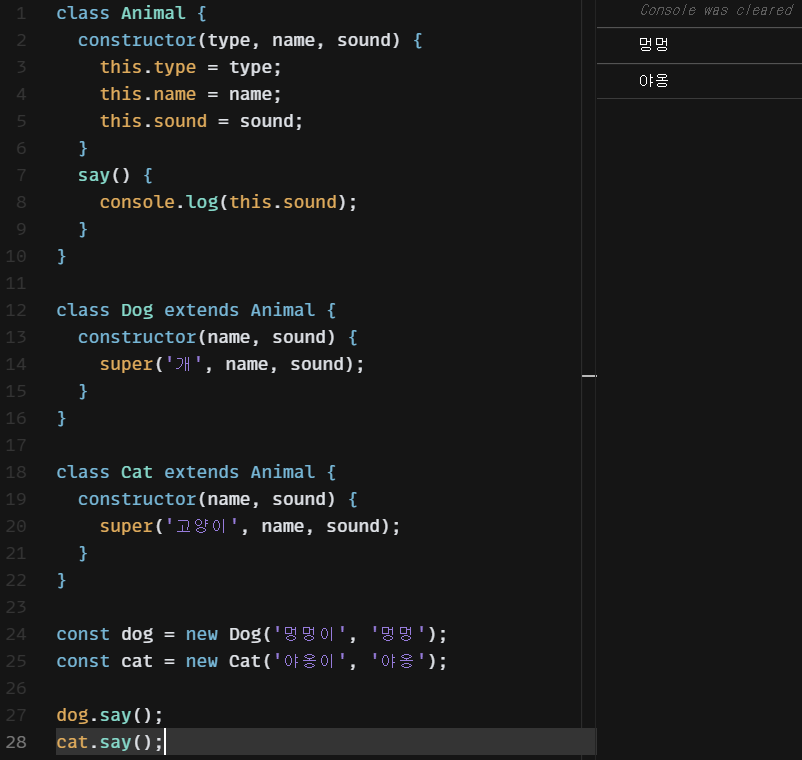


* 객체 생성자 상속받기: Cat과 Dog라는 새로운 객체 생성자를 Animal의 기능을 재사용하여 만든다고 가정했을 때,



새로 만든 Dog 와 Cat 함수에서 Animal.call을 호출하고 있다. 여기서 첫번째 인자에는 this를 넣어주어야 하고, 그 이후에는 Animal 객체 생성자 함수에서 필요로 하는 파라미터를 넣어주어야 한다. 추가적으로 prototype 을 공유해야 하기 때문에 상속받은 객체 생성자 함수를 만들고 나서 prototype 값을 Animal.prototype 으로 설정해준다.

* 클래스: ES6부터 추가된 class 문법을 통해 객체 생성자로 구현했던 코드를 더 명확하고 깔끔하게 구현할 수 있고, 상속도 쉽게 할 수 있다.

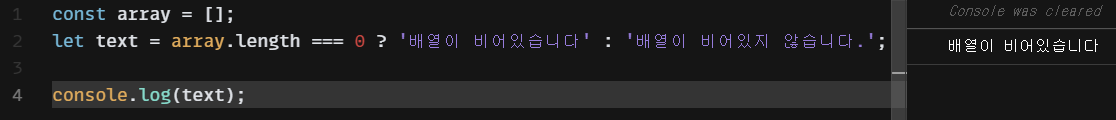


여기서 say라는 함수를 클래스 내부에 선언하였는데, 클래스 내부의 함수는 ‘메서드’라 부른다. 메서드를 만들면 자동으로 prototype으로 등록된다. 상속을 할 때는 extends 키워드를 사용하며, constructor에서 사용하는 super() 함수가 상속받은 클래스의 생성자를 가리킨다.

**<2장. 알고 있으면 유용한 자바스크립트 문법>**

1. 삼항 연산자

* 사용법: 조건 ? true일때 : false일때
* 특정 조건에 따라 값이 달라야 하는 상황일 때 유용하다. 이 경우 text값이 달라야 하는 상황일 때.



* 중첩하여 사용가능하지만 가독성이 떨어지기 때문에 중첩은 피하는 것이 좋다.

1. Truthy and Falsy

* Truthy: true같은 거… Falsy: false 같은 거… 라고 이해하면 된다.
* 객체를 파라미터로 받는 함수의 경우, 파라미터를 전달받지 않거나 그 값이 null인 경우 if문을 통해 따로 다뤄주어야 한다. if(person === undefined || person === null) 이러한 경우 다음과 같이 축약해서 작성할 수 있다. if(!person) 이러한 코드가 작동하는 이유는 undefined 와 null은 falsy 한 값이기 때문이다. Falsy한 값 앞에 느낌표를 붙여주면 true로 전환된다.
* Falsy한 값: undefined, null, 0, ‘ ‘, NaN(Not A Number)(이외 모든 값은 truthy한 값이다)