

# 리액트 1주차 스터디

20 황수연

## <1장. 자바스크립트 문법>

### 01. 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.

### 02. 변수와 상수

- 변수 선언
  - 1) **let** 키워드: 같은 이름의 변수는 다시 선언하지 못하지만, 다른 블록 범위 내에서는 똑 같은 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.

### 03. 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.

### 04. 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

## 05. 함수

```
JS index.js x
1 // function 키워드 통해 함수 생성
2 function add(a, b) {
3   return a + b;
4 }
5
6 const sum = add(1, 2);
7 console.log(sum);
8
```

Console 0 Problems

Console was cleared

3

```
JS index.js x
1 // name이란 매개변수를 넣으면 콘솔 창에 결과 출력
2 function hello(name) {
3   console.log("Hello, " + name + "!");
4 }
5 hello("Suyeon");
6
```

Console 0 Problems

Console was cleared

Hello, Suyeon!

```
JS index.js x
1 // 문자열 결합을 위해 +연산자 말고
2 // ES6의 템플릿 리터럴 사용하기
3 // 출력할 변수를 {}로 감싸고 앞에 $추가
4 function hello(name) {
5   console.log(`Hello, ${name}!`);
6 }
7 hello("World");
8
```

Console 0 Problems

Console was cleared

Hello, World!

```
JS index.js
1 // 함수 선언 방식으로 화살표 함수 문법 사용하기
2 // 화살표 좌측에는 매개변수, 우측에는 코드 블록
3 const add = (a, b) => {
4   return a + b;
5 };
6
7 console.log(add(1, 2));
8
9 // 코드 블록 내에서 바로 return 하는 경우 아래와 같이 써도 됨
10 const add_ = (a,b) => a + b;
11 console.log(add_(2, 2));
12
```

Console 0 Problem

Console was cleared

3

4

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

## 06. 객체

```
JS index.js x
1 // 객체 선언 방식
2 const dog = {
3   name: "멍멍이",
4   age: 2
5 };
6
7 console.log(dog.name);
8 console.log(dog.age);
9
```

Console 0 Problem

Console was cleared

멍멍이

2

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

```
JS index.js
1 // 함수에서 객체를 매개변수로 받기
2 const ironMan = {
3   name: "토니 스타크",
4   actor: "로버트 다우니 주니어",
5   alias: "아이언맨"
6 };
7
8 const captainAmerica = {
9   name: "스티븐 로저스",
10  actor: "크리스 에반스",
11  alias: "캡틴 아메리카"
12 };
13
14 function print(hero) {
15   const text = `${hero.alias}(${hero.name})
16   역할을 맡은 배우는 ${hero.actor} 입니다.`;
17   console.log(text);
18 }
19
20 print(ironMan);
21 print(captainAmerica);
22
```

Console 0 Problems 0

Console was cleared

아이언맨(토니 스타크)  
역할을 맡은 배우는 로버트 다우니 주니어 입니다.

캡틴 아메리카(스티븐 로저스)  
역할을 맡은 배우는 크리스 에반스 입니다.

```
JS index.js x
1 // 객체 비구조화 할당
2 const ironMan = {
3   name: "토니 스타크",
4   actor: "로버트 다우니 주니어",
5   alias: "아이언맨"
6 };
7
8 const captainAmerica = {
9   name: "스티븐 로저스",
10  actor: "크리스 에반스",
11  alias: "캡틴 아메리카"
12 };
13
14 // 파라미터 단계에서 객체에서 값을 추출해서 새로운 상수로 선언
15 function print({ alias, name, actor }) {
16   const text = `${alias}(${name}) 역할을 맡은 배우는 ${actor} 입니다.`;
17   console.log(text);
18 }
19
20 print(ironMan);
21 print(captainAmerica);
22
```

Console 0 Problems 0 Filter All

Console was cleared

아이언맨(토니 스타크) 역할을 맡은 배우는 로버트 다우니 주니어 입니다.

캡틴 아메리카(스티븐 로저스) 역할을 맡은 배우는 크리스 에반스 입니다.

```
JS index.js x
1 // 객체 안에 함수 넣기
2 const dog = {
3   name: "멍멍이",
4   sound: "멍멍!",
5   say: function say() {
6     console.log(this.sound);
7   }
8 };
9
10 dog.say();
11
```

Console 0

멍멍!

- 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

```
JS index.js •
1 const numbers = {
2   _a: 1,
3   _b: 2,
4   sum: 3,
5   calculate() {
6     console.log('calculate');
7     this.sum = this._a + this._b;
8   },
9   get a() {
10    return this._a;
11  },
12  get b() {
13    return this._b;
14  },
15  set a(value) {
16    console.log('a가 바뀝니다. ');
17    this._a = value;
18    this.calculate();
19  },
20  set b(value) {
21    console.log('b가 바뀝니다. ');
22    this._b = value;
23    this.calculate();
24  }
25 };
26
27 console.log(numbers.sum);
28 numbers.a = 5;
29 numbers.b = 7;
30 numbers.a = 9;
31 console.log(numbers.sum);
32 console.log(numbers.sum);
33 console.log(numbers.sum);
```

Browser Tests

https://2pj25.csb.app/

Console 0 Problems 0

Console was cleared

3

a가 바뀝니다.

calculate

b가 바뀝니다.

calculate

a가 바뀝니다.

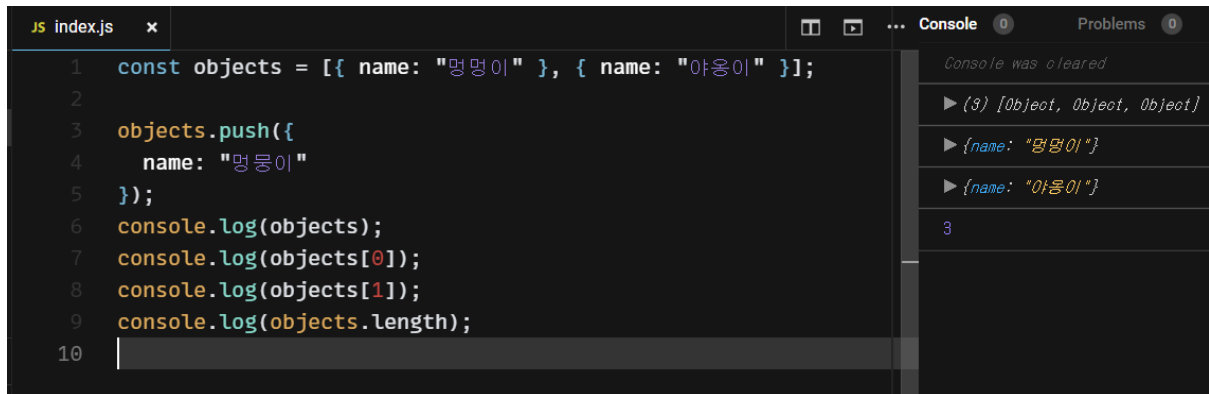
calculate

16

- 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

## 07.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



The screenshot shows a VS Code editor with a file named 'index.js'. The code defines an array 'objects' containing two objects, adds a third object using 'push', and logs the array and its elements. The console on the right shows the output: an array of three objects, followed by the first two objects individually, and the number 3 representing the array's length.

```
1 const objects = [{ name: "멍멍이" }, { name: "아옹이" }];
2
3 objects.push({
4   name: "멍멍이"
5 });
6 console.log(objects);
7 console.log(objects[0]);
8 console.log(objects[1]);
9 console.log(objects.length);
10
```

Console Output:

- Console was cleared
- {3} [Object, Object, Object]
- {name: "멍멍이"}
- {name: "아옹이"}
- 3

## 08.반복문

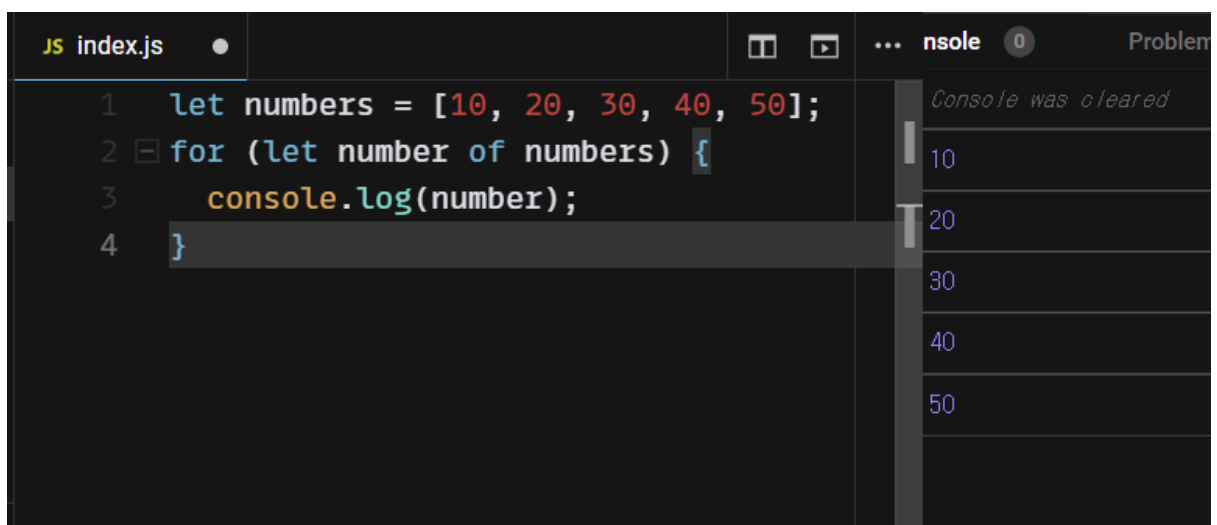
- For문:

```
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**



The screenshot shows a VS Code editor with a file named 'index.js'. The code uses a 'for...of' loop to iterate over an array of numbers and log each element. The console on the right shows the output: the numbers 10, 20, 30, 40, and 50, each on a new line.

```
1 let numbers = [10, 20, 30, 40, 50];
2 for (let number of numbers) {
3   console.log(number);
4 }
```

Console Output:

- Console was cleared
- 10
- 20
- 30
- 40
- 50

- 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

```

1  const doggy = {
2    name: '멍멍이',
3    sound: '멍멍',
4    age: 2
5  };
6
7  for (let key in doggy) {
8    console.log(`${key}: ${doggy[key]}`);
9  }

```

Console

Console was cleared

name: 멍멍이

sound: 멍멍

age: 2

- 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  function biggerThanThree(numbers) {
2    /* 구현해보세요 */
3    const array = [];
4    for (let i = 0; i < numbers.length; i++) {
5      if (numbers[i] > 3) {
6        array.push(numbers[i]);
7        console.log(numbers[i]);
8      }
9    }
10   return array;
11 }
12
13 const numbers = [1, 2, 3, 4, 5, 6, 7];
14 console.log(biggerThanThree(numbers)); // [4, 5, 6, 7]
15 export default biggerThanThree;
16

```

Console

Console was cleared

4

5

6

7

▶ (4) [4, 5, 6, 7]

## 09. 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 변환을 주는 함수를 전달한다. (=변화함수) 변화함수는 꼭 이름을 붙일 필요는 없고 매개변수 안에서 작성해도 된다.

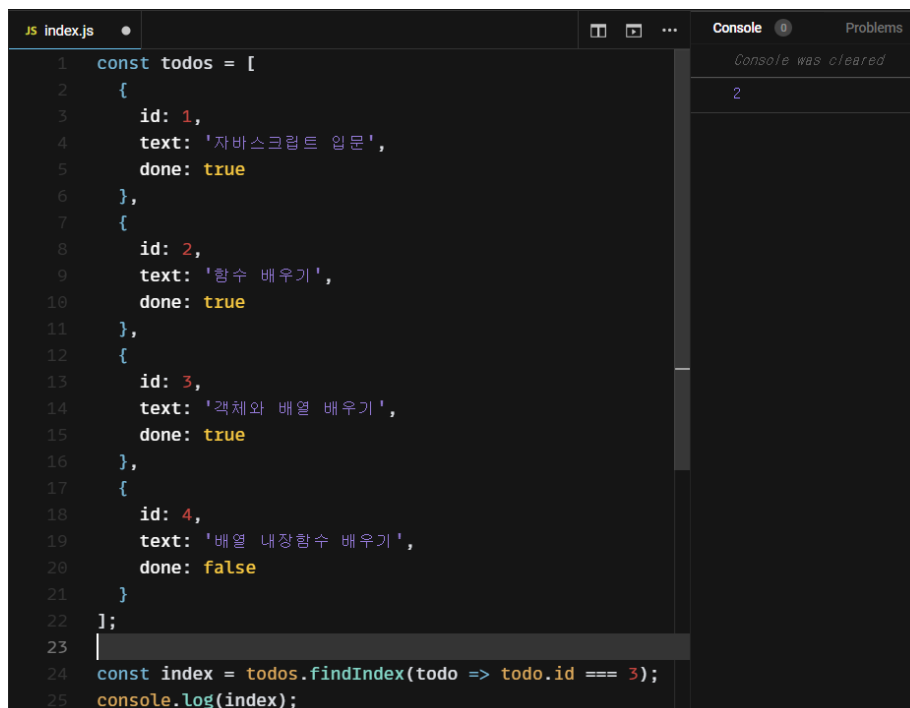
```
const array = [1, 2, 3, 4, 5, 6, 7, 8];

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.



```
JS index.js
1 const todos = [
2   {
3     id: 1,
4     text: '자바스크립트 입문',
5     done: true
6   },
7   {
8     id: 2,
9     text: '함수 배우기',
10    done: true
11  },
12  {
13    id: 3,
14    text: '객체와 배열 배우기',
15    done: true
16  },
17  {
18    id: 4,
19    text: '배열 내장함수 배우기',
20    done: false
21  }
22 ];
23
24 const index = todos.findIndex(todo => todo.id === 3);
25 console.log(index);
```

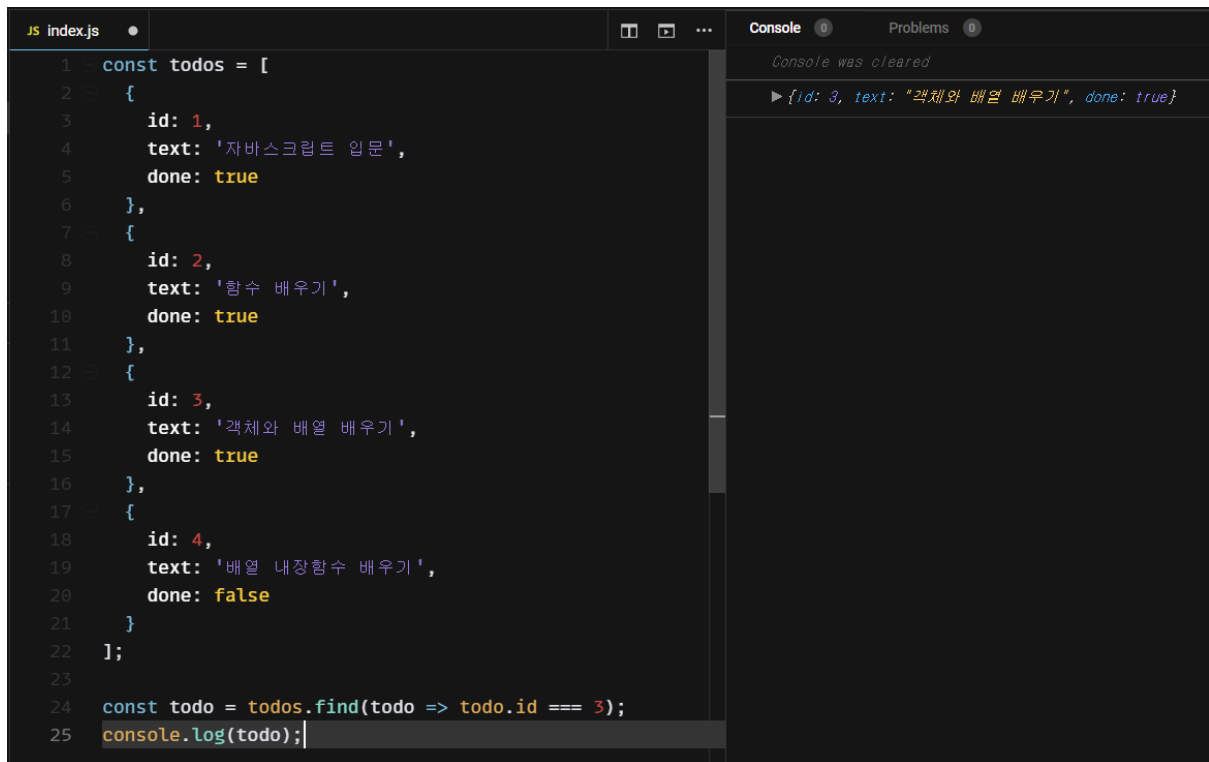
Console

Console was cleared

2



- find: 찾아낸 값 자체를 반환한다.



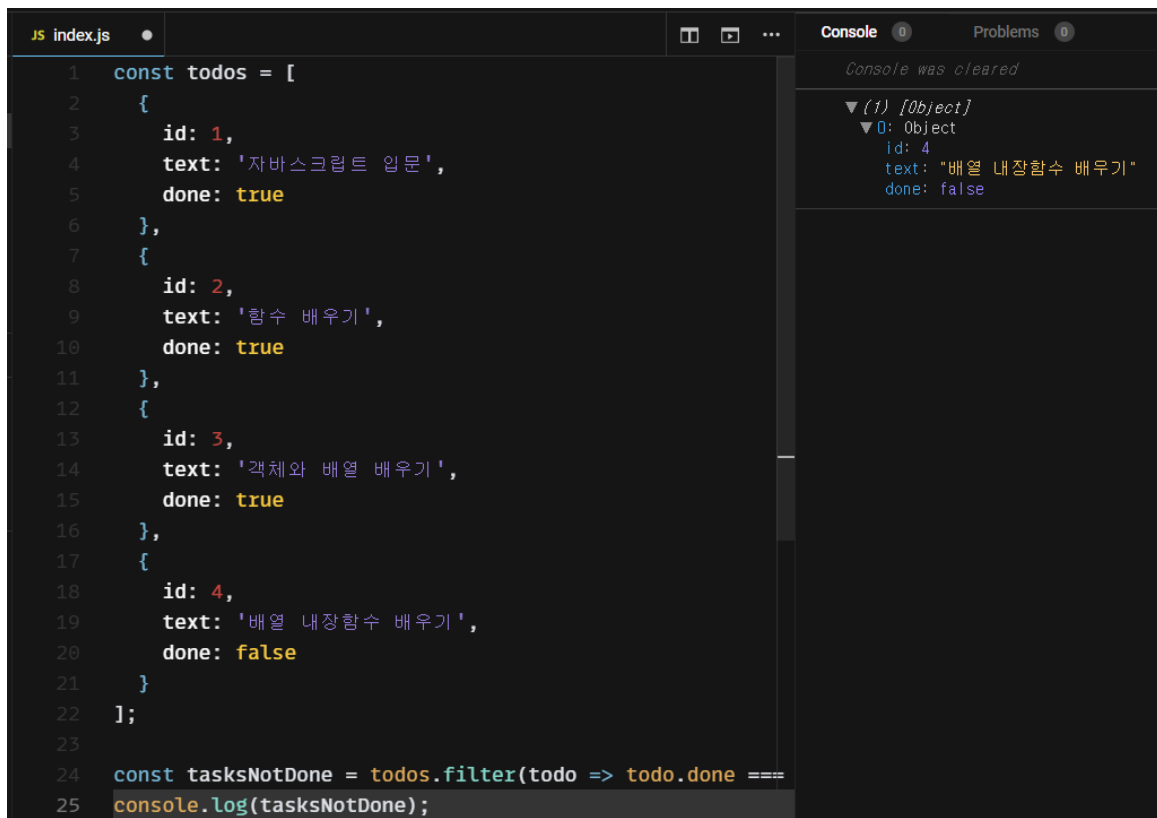
```
1 const todos = [
2   {
3     id: 1,
4     text: '자바스크립트 입문',
5     done: true
6   },
7   {
8     id: 2,
9     text: '함수 배우기',
10    done: true
11  },
12  {
13    id: 3,
14    text: '객체와 배열 배우기',
15    done: true
16  },
17  {
18    id: 4,
19    text: '배열 내장함수 배우기',
20    done: false
21  }
22 ];
23
24 const todo = todos.find(todo => todo.id === 3);
25 console.log(todo);
```

Console

Console was cleared

▶ {id: 3, text: "객체와 배열 배우기", done: true}

- 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.



```
1 const todos = [
2   {
3     id: 1,
4     text: '자바스크립트 입문',
5     done: true
6   },
7   {
8     id: 2,
9     text: '함수 배우기',
10    done: true
11  },
12  {
13    id: 3,
14    text: '객체와 배열 배우기',
15    done: true
16  },
17  {
18    id: 4,
19    text: '배열 내장함수 배우기',
20    done: false
21  }
22 ];
23
24 const tasksNotDone = todos.filter(todo => todo.done ===
25 console.log(tasksNotDone);
```

Console

Console was cleared

▼ (1) [Object]

▼ 0: Object

id: 4

text: "배열 내장함수 배우기"

done: false

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

```
JS index.js x [Icons] Console 0 Problem
1 const numbers = [10, 20, 30, 40];
2 const index = numbers.indexOf(30); // 30의 인덱스 알아내기
3 numbers.splice(index, 1);
4 console.log(numbers);
```

Console was cleared  
▶ (3) [10, 20, 40]

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

```
1 const numbers = [10, 20, 30, 40];
2 const sliced = numbers.slice(0, 2); // 0부터 시작해서 2전까지
3
4 console.log(sliced); // [10, 20]
5 console.log(numbers); // [10, 20, 30, 40]
6
```

Console was cleared  
▶ (2) [10, 20]  
▶ (4) [10, 20, 30, 40]

- **shift**와 **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.

```
1 const numbers = [10, 20, 30, 40];
2 const value = numbers.shift(); // shift
3 console.log(value);
4 console.log(numbers);
5
6 const popped = numbers.pop(); // pop
7 console.log(popped);
8 console.log(numbers);
9
10 numbers.unshift(100); // unshift
11 console.log(numbers);
```

Console was cleared  
10  
▶ (3) [100, 20, 30]  
40  
▶ (3) [100, 20, 30]  
▶ (3) [100, 20, 30]

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

```
1 const arr1 = [1, 2, 3];
2 const arr2 = [4, 5, 6];
3 const concated = arr1.concat(arr2);
4
5 console.log(concated);
```

Console was cleared  
▶ (6) [1, 2, 3, 4, 5, 6]

- **join**: 배열 안의 값들을 문자열 형태로 합친다.

```
1 const array = [1, 2, 3, 4, 5];
2 console.log(array.join()); // 1,2,3,4,5
3 console.log(array.join(" ")); // 1 2 3 4 5
4 console.log(array.join(", ")); // 1, 2, 3, 4, 5
5
```

Console was cleared  
1,2,3,4,5  
1 2 3 4 5  
1, 2, 3, 4, 5

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

```
1 const numbers = [1, 2, 3, 4, 5];
2 let sum = numbers.reduce((accumulator, current) => {
3   console.log({ accumulator, current });
4   return accumulator + current;
5 }, 0);
6
7 console.log(sum);
```

Console was cleared

- ▶ {accumulator: 0, current: 1}
- ▶ {accumulator: 1, current: 2}
- ▶ {accumulator: 3, current: 3}
- ▶ {accumulator: 6, current: 4}
- ▶ {accumulator: 10, current: 5}

15

- 퀴즈:

```
1 function countBiggerThanTen(numbers) {
2   /* 구현해보세요 */
3   let cnt = 0;
4   numbers.forEach((number) => {
5     if (number > 10) cnt++;
6   });
7   return cnt;
8 }
9
10 const count = countBiggerThanTen([1, 2, 3, 5, 10, 20, 30, 40, 50, 60]);
11 console.log(count); // 5
```

Console was

5

## 10. 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 키워드**를 사용한다.

```
1 function Animal(type, name, sound) {
2   this.type = type;
3   this.name = name;
4   this.sound = sound;
5   this.say = function () {
6     console.log(this.sound);
7   };
8 }
9
10 const dog = new Animal("개", "멍멍이", "멍멍");
11 const cat = new Animal("고양이", "야옹이", "야옹");
12
13 dog.say();
14 cat.say();
```

Console was cleared

멍멍

야옹

- **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.

```
1 function Animal(type, name, sound) {
2   this.type = type;
3   this.name = name;
4   this.sound = sound;
5 }
6
7 Animal.prototype.say = function() {
8   console.log(this.sound);
9 };
10 Animal.prototype.sharedValue = 1;
11
12 const dog = new Animal('개', '멍멍이', '멍멍');
13 const cat = new Animal('고양이', '야옹이', '야옹');
14
15 dog.say();
16 cat.say();
17
18 console.log(dog.sharedValue);
19 console.log(cat.sharedValue);
```

Console was cleared

멍멍
야옹
2 1

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

```
1 function Animal(type, name, sound) {
2   this.type = type;
3   this.name = name;
4   this.sound = sound;
5 }
6
7 Animal.prototype.say = function() {
8   console.log(this.sound);
9 };
10 Animal.prototype.sharedValue = 1;
11
12 function Dog(name, sound) {
13   Animal.call(this, '개', name, sound);
14 }
15 Dog.prototype = Animal.prototype;
16
17 function Cat(name, sound) {
18   Animal.call(this, '고양이', name, sound);
19 }
20 Cat.prototype = Animal.prototype;
21
22 const dog = new Dog('멍멍이', '멍멍');
23 const cat = new Cat('야옹이', '야옹');
24
25 dog.say();
26 cat.say();
```

Console was cleared

멍멍

야옹

새로 만든 Dog 와 Cat 함수에서 `Animal.call`을 호출하고 있다. 여기서 첫번째 인자에는 `this`를 넣어 주어야 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.

```
1 class Animal {
2   constructor(type, name, sound) {
3     this.type = type;
4     this.name = name;
5     this.sound = sound;
6   }
7   say() {
8     console.log(this.sound);
9   }
10 }
11
12 class Dog extends Animal {
13   constructor(name, sound) {
14     super('개 ', name, sound);
15   }
16 }
17
18 class Cat extends Animal {
19   constructor(name, sound) {
20     super('고양이 ', name, sound);
21   }
22 }
23
24 const dog = new Dog('멍멍이 ', '멍멍 ');
25 const cat = new Cat('야옹이 ', '야옹 ');
26
27 dog.say();
28 cat.say();
```

Console was cleared

멍멍

야옹

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>

### 01. 삼항 연산자

- 사용법: 조건 ? true일 때 : false일 때
- 특정 조건에 따라 값이 달라야 하는 상황일 때 유용하다. 이 경우 text값이 달라야 하는 상황일 때.

```
1 const array = [];  
2 let text = array.length === 0 ? '배열이 비어있습니다' : '배열이 비어있지 않습니다.';  
3  
4 console.log(text);
```

Console was cleared  
배열이 비어있습니다

- 중첩하여 사용가능하지만 가독성이 떨어지기 때문에 중첩은 피하는 것이 좋다.

### 02. 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 if 문을 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)