

# 5일 (월)

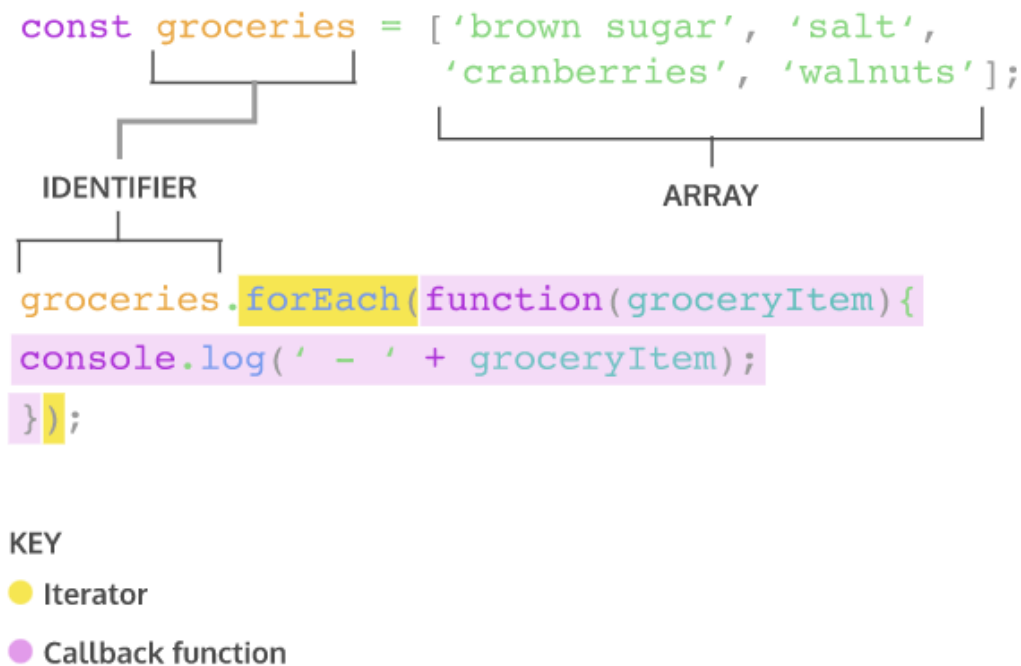
## 1. Codecademy JavaScript - Iterator 이어서

- **introduction**

보통 배열에서 원소를 조작하거나 값을 리턴하기 위해 쓰이는 메서드를 iterator 라고 부른다. 결국 배열 메서드 안에 또 다른 메서드가 들어감으로써 인자로 들어간 함수는 고차함수 (Higher Order Function = HOF)이며 콜백함수이다.

- **iteration 메서드 - .forEach() 메서드**

forEach() 는 배열내 모든 원소에 대해 같은 코드를 실행한다.



forEach() 의 리턴값은 항상 undefined 이며 기존 배열을 변경하지 않는다.

```
groceries.forEach(groceryItem => console.log(groceryItem));
```

```
function printGrocery(element){  
  console.log(element);  
}  
  
groceries.forEach(printGrocery);
```

- **iterator 메서드 - .map( ) 메서드**

배열과 함께 쓰이는 .map( ) 은 콜백함수를 갖으며, 콜백함수에 따른 새로운 배열을 리턴한다. 그러므로 새로운 변수를 지정해줘야한다.

```

1  const animals = ['Hen', 'elephant', 'llama', 'leopard',
2    'ostrich', 'Whale', 'octopus', 'rabbit', 'lion', 'dog'];
3
4  // Create the secretMessage array below
5  const secretMessage = animals.map(animal => {return animal
6    [0];} )
7
8  console.log(secretMessage.join(''));
9  //HelloWorld
10
11 const bigNumbers = [100, 200, 300, 400, 500];
12
13 // Create the smallNumbers array below
14 ▼ const smallNumbers = bigNumbers.map(number => {
15   return number / 100
16 });
17
18 console.log(smallNumbers);
19 // [ 1, 2, 3, 4, 5 ]

```

- **iterator 메서드 - .filter( ) 메서드**

.map( ) 과 같이 새로운 배열을 리턴한다. 하지만 조건에 맞는 원소만 골라내어 리턴한다. 콜백함수는 무조건 true 나 false 를 리턴해야하는 함수이어야 한다.

```
const words = ['chair', 'music', 'pillow', 'brick', 'pen', 'door'];

const shortWords = words.filter(word => {
  return word.length < 6;
});
```

```
console.log(words); // Output: ['chair', 'music', 'pillow', 'brick', 'pen', 'door'];
console.log(shortWords); // Output: ['chair', 'music', 'brick', 'pen', 'door']
```

- **iterator 메서드 - .findIndex( ) 메서드**

배열내에서 찾고자 하는 원소의 위치를 찾을 때 사용하는 메서드이다. 리턴값은 첫번째로 조건을 만족하는 인덱스이다. 만약 찾고자하는 값이 없다면 -1 을 리턴하게 된다. 인덱스를 리턴할 새로운 변수를 지정해야한다. 이 또한 true 나 false 를 리턴해야하는 함수여야한다.

```
const jumbledNums = [123, 25, 78, 5, 9];

const lessThanTen = jumbledNums.findIndex(num => {
  return num < 10;
});
```

```
console.log(lessThanTen); // Output: 3
```

If we check what element has index of 3:

```
console.log(jumbledNums[3]); // Output: 5
```

- **iterator 메서드 - .reduce( ) 메서드**

reduce( ) 메서드는 모든 배열의 원소를 이터레이팅 후 하나의 값을 리턴한다.

reduce 메서드는 인자 두 개를 가질 수 있으며 콜백함수는 무조건 입력해야하고, 두번째 인자는 accumulator 로 누적값의 값의 초기설정이 가능하다. 문자열도 가능하다.

```
//인자 하나만 들어간 예시
const numbers = [1, 2, 4, 10];

const summedNums = numbers.reduce((accumulator, currentValue) => {
  return accumulator + currentValue
})

console.log(summedNums) // Output: 17
```

```
//reduce 메서드에 인자 두개 들어간 예시 (accumulator: 100 )
const numbers = [1, 2, 4, 10];

const summedNums = numbers.reduce((accumulator, currentValue) => {
  return accumulator + currentValue
}, 100) // <- Second argument for .reduce()

console.log(summedNums); // Output: 117
```

- **추가적인 iterator Documentation**

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Array#Iteration\\_methods](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#Iteration_methods)

.some() 메서드 - 콜백함수 리턴값이 배열내에 하나라도 있는지 없는지 확인하는 메서드  
true, false 로 값을 리턴한다.

.every() 메서드 - 배열 내 모든 원소가 조건에 부합하는지 true, false 로 값을 리턴한다.

```
1  const words = ['unique', 'uncanny', 'pique', 'oxymoron', 'guise'];
2
3  // Something is missing in the method call below
4
5  ▼ console.log(words.some((word) => {
6    return word.length < 6;
7  }));
8
9  // Use filter to create a new array
10 ▼ const interestingWords = words.filter(word => {
11   return word.length > 5;
12 });
13
14
15 // Make sure to uncomment the code below and fix the incorrect code
   before running it
16
17 ▼ console.log(interestingWords.every((word) => {
18   return word.length > 5
19 } ));
20
```

- 상황에 맞는 올바른 이터레이터 선택하기

```

1  const cities = ['Orlando', 'Dubai', 'Edinburgh', 'Chennai',
2    'Accra', 'Denver', 'Eskisehir', 'Medellin', 'Yokohama'];
3
4  const nums = [1, 50, 75, 200, 350, 525, 1000];
5
6  // Choose a method that will return undefined
7  cities.forEach(city => console.log('Have you visited ' + city +
8    '?'));
9
10 // Choose a method that will return a new array
11 const longCities = cities.filter(city => city.length > 7);
12
13 // Choose a method that will return a single value
14 ▼ const word = cities.reduce((acc, currVal) => {
15   return acc + currVal[0]
16 }, "C");
17 console.log(word)
18
19 // Choose a method that will return a new array
20 const smallerNums = nums.map(num => num - 5);
21 console.log(smallerNums);
22
23 // Choose a method that will return a boolean value
24 console.log(nums.some(num => num < 0));
25

```

```
Have you visited Orlando?  
Have you visited Dubai?  
Have you visited Edinburgh?  
Have you visited Chennai?  
Have you visited Accra?  
Have you visited Denver?  
Have you visited Eskisehir?  
Have you visited Medellin?  
Have you visited Yokohama?  
CODECADEMY  
[ -4, 45, 70, 195, 345, 520, 995 ]  
false
```

## 2. Codecademy JavaScript - Objects

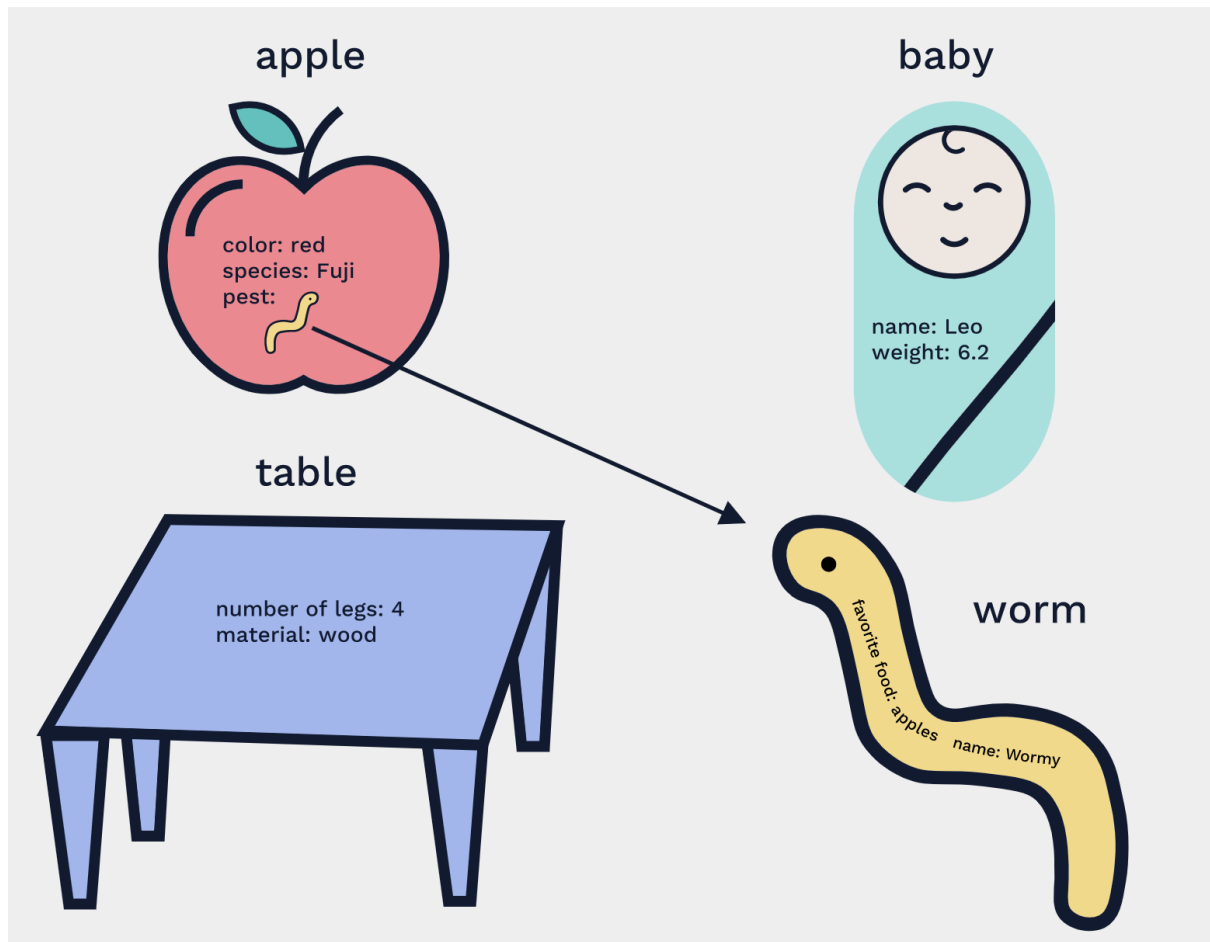
### Lesson 1. Objects



- **Introduction**

자바스크립트의 거의 모든 부분은 객체로 이루어져 있다. 자바스크립트에는 오직 7가지의 데이터 타입이 있다. (string(문자열), number, boolean, null, undefined, symbol 마지막으로 object)

객체를 제외한 나머지는 primitive data(원시데이터) type 이다.



객체 안에 객체 안에 객체가 가능하다.

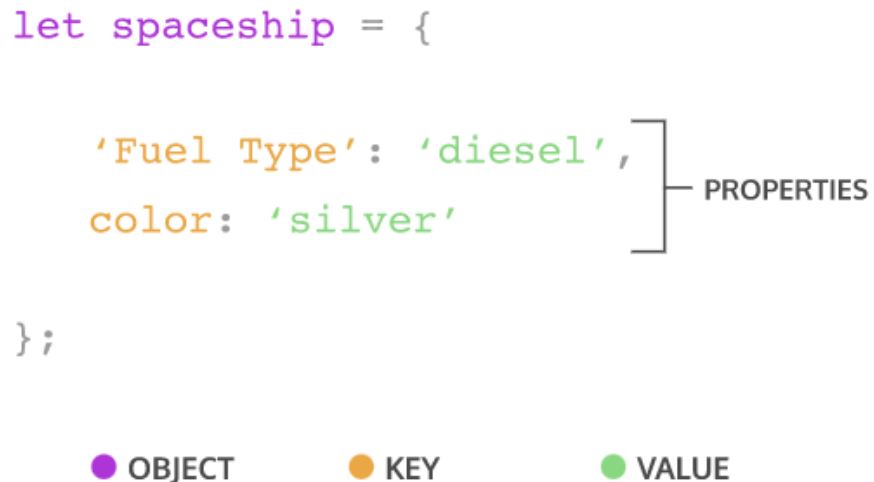
- **객체 만들기**

객체는 변수로 지정이 가능하다.

```
let spaceship = {}; // spaceship is an empty object
```

객체 내 데이터는 순서가 없다. key-value 쌍으로 저장된다. 키는 변수이름이며 키의 값을 가진 메모리의 위치를 가리킨다. 값(value) 는 어떠한 데이터도 가질 수 있다.(객체, 함수등)

```
let spaceship = {  
  'Fuel Type': 'diesel',  
  color: 'silver'  
};
```



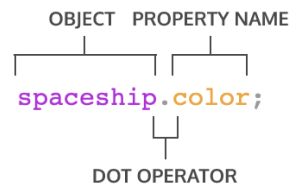
● OBJECT      ● KEY      ● VALUE

```
// An object literal with two key-value pairs  
let spaceship = {  
  'Fuel Type': 'diesel',  
  color: 'silver' //프로퍼티  
};
```

- **프로퍼티에 접근하기**

2가지 방법이 있다. 첫번째는 ( . )dot notation 쓰기

```
let spaceship = {
  homePlanet: 'Earth',
  color: 'silver'
};
spaceship.homePlanet; // Returns 'Earth',
spaceship.color; // Returns 'silver',
```



객체 내에 없는 프로퍼티에 접근하면 undefined 를 리턴한다.

```
1  ▼ let spaceship = {
2    homePlanet: 'Earth',
3    color: 'silver',
4    'Fuel Type': 'Turbo Fuel',
5    numCrew: 5,
6    flightPath: ['Venus', 'Mars', 'Saturn']
7  };
8
9  // Write your code below
10
11  const crewCount = spaceship.numCrew;
12  const planetArray = spaceship.flightPath;
13
14  console.log(crewCount);
15  console.log(planetArray);
16
```

두번째 접근 방법은 ( [ ] ) bracket notation 쓰기다. 배열뿐만 아니라 객체에서도 사용 가능하다.

OBJECT                  PROPERTY NAME

```
spaceship[ 'Fuel Type' ];
```

```
let spaceship = {
  'Fuel Type': 'Turbo Fuel',
  'Active Duty': true,
  homePlanet: 'Earth',
  numCrew: 5
};
spaceship['Active Duty']; // Returns true
spaceship['Fuel Type']; // Returns 'Turbo Fuel'
spaceship['numCrew']; // Returns 5
spaceship['!!!!!!!!!!!!!!']; // Returns undefined
```

```
let returnAnyProp = (objectName, propName) =>
  objectName[propName];

returnAnyProp(spaceship, 'homePlanet'); // Returns 'Earth'
```

이렇게 함수와 유용하게 쓰일 수 있다.

- 객체 내 프로퍼티 할당

객체는 mutable 이다. 계속해서 업데이트 가능

OBJECT      PROPERTY NAME      ASSIGNMENT OPERATOR      VALUE

`spaceship ['Fuel Type'] = 'vegetable oil';`

`spaceship.color = 'gold';`

```
const spaceship = {type: 'shuttle'};
spaceship = {type: 'alien'}; // TypeError: Assignment to constant variable.
spaceship.type = 'alien'; // Changes the value of the type property
spaceship.speed = 'Mach 5'; // Creates a new key of 'speed' with a value of 'Mach 5'
```

const 로 된 객체는 프로퍼티 값을 . 을 이용해서 바꾸거나 추가할 수 있다.

## 프로퍼티 삭제방법

```
const spaceship = {
  'Fuel Type': 'Turbo Fuel',
  homePlanet: 'Earth',
  mission: 'Explore the universe'
};

delete spaceship.mission; // Removes the mission property
```

delete 를 이용하여 프로퍼티를 삭제할 수 있다.

```
delete spaceship['Secret Mission'];
```

쿼트로 이루어진 key 는 [ ] 을 이용하여 접근해야한다.

- 메서드 - 객체내에 있는 함수를 메서드라고 한다.

```
const alienShip = {  
  invade: function () {  
    console.log('Hello! We have come to dominate your  
planet. Instead of Earth, it shall be called New  
Xaculon.')  
  }  
};
```

With the new method syntax introduced in ES6 we can omit the colon and the `function` keyword.

```
const alienShip = {  
  invade () {  
    console.log('Hello! We have come to dominate your  
planet. Instead of Earth, it shall be called New  
Xaculon.')  
  }  
};
```

```
alienShip.invade(); // Prints 'Hello! We have come to
dominate your planet. Instead of Earth, it shall be called
New Xaculon.'
```

예시

```
1  let retreatMessage = 'We no longer wish to
   conquer your planet. It is full of dogs,
   which we do not care for.';
2
3  // Write your code below
4  ▼ let alienShip = {
5  ▼    retreat() {
6      console.log(retreatMessage);
7    },
8
9  ▼    takeOff: function() {
10     console.log('Spim... Borp... Glix...
        Blastoff!');
11   }
12 };
13
14 alienShip.retreat();
15 alienShip.takeOff();
16
```

- **Nested Objects**

```
const spaceship = {
  telescope: {
    yearBuilt: 2018,
    model: '91031-XLT',
    focalLength: 2032
  },
  crew: {
    captain: {
      name: 'Sandra',
      degree: 'Computer Engineering',
      encourageTeam() { console.log('We got this!') }
    }
  },
  engine: {
    model: 'Nimbus2000'
  },
  nanoelectronics: {
    computer: {
      terabytes: 100,
      monitors: 'HD'
    },
    'back-up': {
      battery: 'Lithium',
      terabytes: 50
    }
  }
};
```

객체 안에 객체 또 객체 안에 객체가 가능하다.



```
const capFave = spaceship.crew.captain['favorite foods'][0];

spaceship.passengers = [{name : 'mingyu'}, {name : 'Haru'}];

const firstPassenger = spaceship.passengers[0];

console.log(firstPassenger);
// print { name : 'mingyu' }
```

중첩된 객체에서 값을 불러 오기 연습 예시

- **Pass by Reference - 메모리 참조형 타입 객체**

객체는 참조타입(reference type) 이다. 자료형에는 원시타입 과 참조타입 2가지로 나뉜다. 참조타입에는 객체, 배열, 함수 등 같은 Object 형식 타입이다.

이는 메모리에 값을 주소로 저장하고, 출력시 메모리 주소와 일치하는 값을 출력한다.

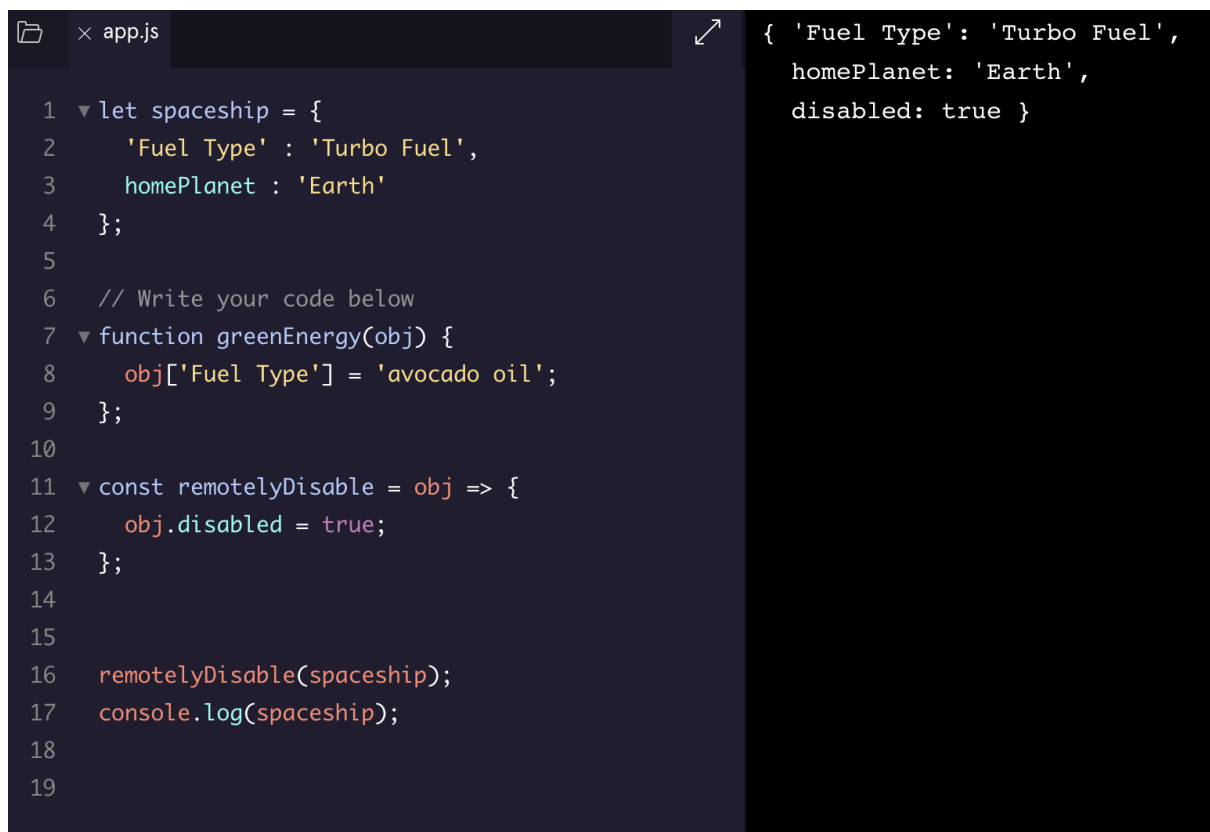
```
const origin = {
  a: 1,
  b: 2,
  c: 3,
  d: {
    q:1,
    w:2,
    e:3,
  },
}
const copy = origin;
copy.b = 1000;
console.log(origin.b); // result 1000
```

copy라는 변수에 할당된 것은 origin 의 메모리 주소이다. 그로인해 copy 값이 바뀌었지만 사실 origin 의 값을 바꾼 것이기에 origin 의 값도 변경된다.

원시타입은 참조없이 값이 변동되었을 때 메모리에 값을 그대로 저장한다. 헷갈림 x

const 객체임에도 내부의 값이 변경된다.

```
const a = 3100;
let b = a;
b = 3;
console.log(a); // result 3100
```



The screenshot shows a code editor with a file named 'app.js'. The code defines a spaceship object, a function to change its fuel type, and a function to disable it. The output on the right shows the state of the spaceship object after the disable function is called.

```
1 ▼ let spaceship = {
2   'Fuel Type' : 'Turbo Fuel',
3   homePlanet : 'Earth'
4 };
5
6 // Write your code below
7 ▼ function greenEnergy(obj) {
8   obj['Fuel Type'] = 'avocado oil';
9 };
10
11 ▼ const remotelyDisable = obj => {
12   obj.disabled = true;
13 };
14
15
16 remotelyDisable(spaceship);
17 console.log(spaceship);
18
19
```

```
{ 'Fuel Type': 'Turbo Fuel',
  homePlanet: 'Earth',
  disabled: true }
```

- **Looping Through Objects - for .. in**

반복문은 조건이 맞을 때까지 코드블록을 반복해주는 툴이다. 객체는 키와 밸류로 이루어진 non ordered 데이터이다. 즉, 인덱싱이 없다. 이에 대한 해결책으로 for..in.. 툴이 있다.

```
let spaceship = {
  crew: {
    captain: {
      name: 'Lily',
      degree: 'Computer Engineering',
      cheerTeam() { console.log('You got this!') }
    },
    'chief officer': {
      name: 'Dan',
      degree: 'Aerospace Engineering',
      agree() { console.log('I agree, captain!') }
    },
    medic: {
      name: 'Clementine',
      degree: 'Physics',
      announce() { console.log(`Jets on!`) } },
    translator: {
      name: 'Shauna',
      degree: 'Conservation Science',
      powerFuel() { console.log('The tank is full!') }
    }
  }
};
```

```

// write your code below
5 ▼ for (let crewMember in spaceship.crew) {
6     console.log(`${crewMember}: ${spaceship.crew[crewMember].name}`);
7 }
8
9 console.log();
10
11 ▼ for (let crewMember in spaceship.crew) {
12     console.log(`${spaceship.crew[crewMember].name}: ${spaceship.crew
13 [crewMember].degree}`);
14 }

```

for in 으로 원하는 객체의 프로퍼티 호출하기

```

captain: Lily
chief officer: Dan
medic: Clementine
translator: Shauna

Lily: Computer Engineering
Dan: Aerospace Engineering
Clementine: Physics
Shauna: Conservation Science

```

결과값

## Lesson 2. Advanced Objects

- Introduction

this 키워드, conveying privacy in JS Methods, getter and setter 정의하기 , 팩토리 함수 생성하기 , 디스트럭트링 기술 사용하기

- this 키워드

```
const goat = {
  dietType: 'herbivore',
  makeSound() {
    console.log('baaa');
  },
  diet() {
    console.log(dietType);
  }
};
goat.diet();
// Output will be "ReferenceError: dietType is not defined"
```

위와 같이 객체 내에 있는 프로퍼티를 객체 내 함수에 정의할 수 없다. 블록 내에 스코프가 dietType 을 자동으로 찾지 못한다.

```
const goat = {
  dietType: 'herbivore',
  makeSound() {
    console.log('baaa');
  },
  diet() {
    console.log(this.dietType);
  }
};

goat.diet();
// Output: herbivore
```

이와 같은 문제를 해결하기 위해 this 를 사용한다.

- **Arrow Function 과 this**

this 를 사용할때는 Arrow function 을 피하자 !!

→ arrow function 에선 this 가 글로벌 object 이거나 글로벌 스코프에 존재하는 object 이기때문에 자신의 객체 내 calling object가 안된다

\*\*\* Global Objects 전역객체

전역 객체 [object](#) 는 전역 범위 [global scope \(en-US\)](#) 에 항상 존재하는 객체를 의미합니다.

자바스크립트에는 전역 객체로 선언된 객체들이 항상 존재합니다. 웹브라우저에서 스크립트가 전역 변수를 생성할 때, 그것들은 전역 객체의 멤버로서 생성됩니다. (이것은 [Node.js](#) 에서는 예외입니다.) 전역 객체의 [interface](#) 는 스크립트가 실행되고 있는 곳의 실행 컨텍스트에 의존합니다. 예를 들어:

- 웹브라우저에 있는 스크립트가 특별히 백그라운드 작업으로 시작하지 않는 코드들은 그것의 전역 객체로써 [Window](#) 를 가집니다. 이것은 Web에 있는 자바스크립트 코드의 상당수가 그렇습니다.
- [Worker](#) 에서 실행하는 코드는 그것의 전역 객체로서 [WorkerGlobalScope \(en-US\)](#) 를 가집니다.
- [Node.js](#) 환경에서 실행하는 스크립트들은 [global](#) [↗](#) 로 호출되는 객체를 그것들의 전역 객체로 가집니다.

→ 모든 객체는 Window 객체 안에 존재한다.

\*\*\* Arrow function

---

## 화살표 함수

화살표 함수 표현(**arrow function expression**)은 [function 표현](#)에 비해 구문이 짧고 자신의 [this](#), [arguments](#), [super](#) 또는 [new.target](#)을 바인딩 하지 않습니다. 화살표 함수는 항상 **익명**입니다. 이 함수 표현은 메소드 함수가 아닌 곳에 가장 적합합니다. 그래서 생성자로서 사용할 수 없습니다.

---

\*\*\*바인딩이란?

## 바인딩이란 무엇인가~

bind : 결속시키다, 묶다

특정객체에서 실행되게끔 고정시키는 그런 역할이라 할 수 있겠다.

즉 바인딩이란

a라는 객체가 있고, 전역함수로써 혹은 b객체에 test()라는 메서드가 있다고 치자.

b.test()라고 실행되는게 보통이지만 이것을 마치 a객체에서 실행되게 ( a.test() 와 같은 효과 )하는 경우라 할 수 있다.

즉 메서드와 객체를 묶어놓는 것을 바인딩이라 하겠다.

- Privacy

객체 내 특정 프로퍼티를 변경하지 못하게 할 수 있는 기능이다. 특정 언어들은 빌트인 객체로 프라이버시를 갖고 있지만 자스는 없다. 그렇기에 자스개발자들은 '\_' 언더바를 이용한다.



```
const bankAccount = {  
  _amount: 1000  
}
```

In the example above, the `_amount` is not intended to be directly manipulated.

Even so, it is still possible to reassign

`_amount`:

```
bankAccount._amount = 1000000;
```

이런식으로 설정을 한다. 그러나 여전히 변경은 가능하다.

```
main.js
1  ▼ const robot = {
2    _energyLevel: 100,
3    ▼ recharge(){
4      this._energyLevel += 30;
5      console.log(`Recharged! Energy
    is currently at ${this._energyLevel}
    %.`)
6    }
7  };
8
9  robot._energyLevel = 'high';
10
11 robot.recharge();
```

```
Recharged! Energy is currently at
high30%.
```

객체내 energyLevel 을 'high' 로 바꿨지만 , 출력은 이상하게 30과 같이 그대로 더해져서 나왔다. 이것은 강제타입의 부작용이다.

- **Getters - get 메서드**

게터는 객체 내 프로퍼티를 얻거나 리턴하는 메서드이다. 그러나 단지 프로퍼티의 값만 얻는 것을 넘어 더 많은 것들을 한다.

```
const person = {
  _firstName: 'John',
  _lastName: 'Doe',
  get fullName() {
    if (this._firstName && this._lastName){
      return `${this._firstName} ${this._lastName}`;
    } else {
      return 'Missing a first name or a last name.';
    }
  }
}

// To call the getter method:
person.fullName; // 'John Doe'
```

#### — Getter 의 이점 —

게터는 프로퍼티를 얻을 때 데이터에 대한 액션을 수행한다.

게터는 조건문을 이용하여 다른 밸류를 리턴할 수 있다.

게터에서 콜링 오브젝트의 프로퍼티에 this를 사용하면서 접근할 수 있다.

우리의 코드기능이 다른 개발자가 이해하기 쉽게 한다.

하나 명심할 것은 게터(세터) 메서드는 값을 얻으려고하는 프로퍼티와 이름이 같을 수 없기 때문에 게터가 아닌 프로퍼티에 '\_' 언더스코어를 넣어주는것이다.

이후 프로퍼티를 호출할 때는 게터는 메서드가 아니기때문에 ( ) 를 넣지 않는다.

```

1 ▼ const robot = {
2   _model: '1E78V2',
3   _energyLevel: 100,
4 ▼ get energyLevel(){
5 ▼   if(typeof this._energyLevel === 'number') {
6     return 'My current energy level is ' + this._energyLevel
7 ▼   } else {
8     return 'System malfunction: cannot retrieve energy level'
9   }
10  }
11 };
12
13 console.log(robot.energyLevel);
14

```

- **Setters 메서드**

세터는 이미 정의된 프로퍼티를 재정의할 수 있는 메서드이다. 세터 메서드 ( )안에 변경할 값인 변수가 들어가야한다.

게터의 이점과 비슷하다. 하지만 세터 메서드가 존재하여도 직접적으로 프로퍼티를 바꾸는 것이 여전히 가능하다.

```

const person = {
  _age: 37,
  set age(newAge){
    if (typeof newAge === 'number'){
      this._age = newAge;
    } else {
      console.log('You must assign a number to age');
    }
  }
};

```

```
person.age = 40;  
console.log(person._age); // Logs: 40  
person.age = '40'; // Logs: You must assign a number to age
```

```
person._age = 'forty-five'  
console.log(person._age); // Prints forty-five
```

- **Factory Functions**

팩토리 평션은 말 그대로 공장처럼 많은 객체들을 리턴하는 함수이다. 이는 객체 커스터마이징이 가능하다.

```
const monsterFactory = (name, age, energySource,  
catchPhrase) => {  
  return {  
    name: name,  
    age: age,  
    energySource: energySource,  
    scare() {  
      console.log(catchPhrase);  
    }  
  }  
};
```

```
const ghost = monsterFactory('Ghouly', 251, 'ectoplasm', 'B00!');  
ghost.scare(); // 'B00!'
```

- **Property Value 속기법**

ES6 부터 도입된 destructuring 은 팩토리 함수를 만들 때 변수에 프로퍼티를 빠르게 입력해준다.

```
const monsterFactory = (name, age)
=> {
  return {
    name: name,
    age: age
  }
};
```

원래 팩토리 함수 정의

```
const monsterFactory = (name, age)
=> {
  return {
    name,
    age
  }
};
```

디스트럭터링 적용 → 변수만 입력하면 된다. 함수는 제외~

- **Destructured** 변수 할당

```
const vampire = {
  name: 'Dracula',
  residence: 'Transylvania',
  preferences: {
    day: 'stay inside',
    night: 'satisfy appetite'
  }
};
```

vampire 객체

```
const residence = vampire.residence;  
console.log(residence); // Prints  
'Transylvania'
```

객체 내 프로퍼티를 변수에 지정하고 싶을 때 - 기존 방법

```
const { residence } = vampire;  
console.log(residence); // Prints  
'Transylvania'
```

destructured 변수 할당 - 입력할 것이 훨씬 줄어 들었다.

```
const { day } = vampire.preferences;  
console.log(day); // Prints 'stay  
inside'
```

중첩된 객체의 프로퍼티도 가능하다.

- 빌트인 객체 메서드



```
▼ const robot = {  
  model: 'SAL-1000',  
  mobile: true,  
  sentient: false,  
  armor: 'Steel-plated',  
  energyLevel: 75  
};
```

`const array = Object.keys( obj )` → 객체 내 모든 키들을 배열로 리턴한다.

```
9    // What is missing in the following  
    method call?  
10   const robotKeys = Object.keys(robot)  
    ;  
11  
12   console.log(robotKeys);  
13
```

```
[ 'model', 'mobile', 'sentient',  
  'armor', 'energyLevel' ]
```

`const array = Object.entries( obj )` → 객체 내 모든 키와 값을 이차원 배열로 리턴한다.

```
// Declare robotEntries below this
line:
const robotEntries = Object.entries
(robot);

console.log();
console.log(robotEntries);
```

```
[ [ 'model', 'SAL-1000' ],
  [ 'mobile', true ],
  [ 'sentient', false ],
  [ 'armor', 'Steel-plated' ],
  [ 'energyLevel', 75 ] ]
```

`const array = Object.assign( { key: value... }, obj )` → obj 에 새로운 키와 값 쌍을 추가한다. 추가할 쌍을 변수로 지정하고 인자로 넣어도 된다. 위 2개와는 달리 객체를 출

력한다.

```
const newRobot = Object.assign(  
  {laserBlaster: true,  
  voiceRecognition: true}, robot);  
  
console.log();  
console.log(newRobot);
```

```
{ laserBlaster: true,  
  voiceRecognition: true,  
  model: 'SAL-1000',  
  mobile: true,  
  sentient: false,  
  armor: 'Steel-plated',  
  energyLevel: 75 }
```

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object) — 기타 Object 객체 메서드들 참고 !!

## 오늘의 영단어

- at one's disposal : ~의 마음대로
- at times : 때때로
- aptly : 적절히
- apt : 적절한
- beforehand : 사전에, 미리
- Nonetheless : 그럼에도 불구하고
- accumulator : 누산기, 축적기
- permeate : 액체, 기체 등이 스며들다 침투하다. 퍼지다.
- deceptively : 남을 속여서
- stuck with : ~을 억지로 떠맡다.
- inherently : 선천적으로, 본질적으로
- bind : 묶다.
- takeaway : 배운점, 가장 중요하다고 느낀 점 ; 테이크아웃 전문점
- coercion : 강요, 강제
- side - effect : 부작용
- workaround : 해결책