6일 (화)

1. Iterator 실습 퀴즈 - Mini Linter (문단 편집기)

```
let story = 'Last weekend, I took literally the most beautiful bike ride of my life. The route is called "The 9W to Nyack" and it actu

let overusedWords = ['really', 'very', 'basically'];

let unnecessaryWords = ['extremely', 'literally', 'actually' ];

const storyWords = story.split(' '); // story 리스트로 전환

console.log(storyWords.length); // story 단어 갯수 세기

//story 에서 unnecessaryWord 필터링하기
const betterWords = storyWords.filter(word => {
    if (!unnecessaryWords.includes(word)) {
      return word    //includes 메서드
    }
  }
);
// overWord 개수 세기
let overWordsCount = 0;

betterWords.forEach(word => {
  if (overusedWords.includes(word)) {
    overWordsCount++;
  }
});

// 문장 갯수 세기
let sentenceCount = 0;

betterWords.forEach(word => {
  if (word[word.length - 1] === '.' || word[word.length - 1] === '!') { // . 이랑 ! 가 들어가면 1문장으로 치고 더함
    sentenceCount++
  }
});

//문장 갯수 출력
console.log(sentenceCount);
// overword 갯수 출력
console.log(overWordsCount);
// 정리된 문장 배열 합치기
console.log(betterWords.join(' '));
```

2. Codecademy JavaScript - Classes

- **Introduction**

클래스를 만드는 법을 배울 것이다. 클래스란 비슷한 객체를 빠르게 생산할 수 있는 툴이라고 보면 된다. 클래스는 중복코드를 줄여주고 디버깅에 소요되는 시간을 줄여준다.

main.js

```
class Dog {
  constructor(name) {
    this._name = name;
    this._behavior = 0;
  }

  get name() {
    return this._name;
  }
  get behavior() {
    return this._behavior;
  }

  incrementBehavior() {
    this._behavior ++;
  }
}

const halley = new Dog('Halley');
console.log(halley.name); // Print name value to console
console.log(halley.behavior); // Print behavior value to
console
halley.incrementBehavior(); // Add one to behavior
console.log(halley.name); // Print name value to console
console.log(halley.behavior); // Print behavior value to
console
```

```
Halley
0
Halley
1
```

```
let halley = {
  _name: 'Halley',
  _behavior: 0,

  get name() {
    return this._name;
  },

  get behavior() {
    return this._behavior;
  },

  incrementBehavior() {
    this._behavior++;
  }
}
```

위 클래스는 이러한 객체를 아주 빠르게 생산해낸다.

- **Constructor 메서드 (생성자 함수)**

자스에서 클래스의 새로운 인스턴스를 생성할 때 항상 constructor 메서드를 사용한다. 클래스 이름은 첫글자 대문자이며 낙타표기법을 쓴다. 생성자 함수에서의 this 는 그 클래스의 인스턴스를 참조(가리킨다.)한다.

```
class Dog {
  constructor(name, gender) {
    this.name = name;   // ' , ' 를 안써도 됨
    this.gender = gender;
}
```

- **Instance 인스턴스**

인스턴스는 클래스에 의해 만들어진 클래스의 메서드와 프로퍼티 이름을 가진 객체이다.

인스턴스를 만드는 방법 →

```
class Dog {
  constructor(name) {
    this.name = name;
    this.behavior = 0;
  }
}
 // const 로 객체 이름을 변수로 지정하면서 = new + 클래스이름(생성자함수 인자)
const halley = new Dog('Halley');
```

```
console.log(halley.name); // Log the name value saved to halley
// Output: 'Halley'
```

- **Methods**

클래스에서는 ' , ' 를 안 써도된다.

```
class Surgeon {
  constructor(name, department) {
    this._name = name;
    this._department = department;
    this._remainingVacationDays = 20;
  }
  get name() {
    return this._name;
  }

  get department() {
    return this._department;
  }

  get remainingVacationDays() {
    return this._remainingVacationDays;
  }

  takeVacationDays(daysOff) {
    this._remainingVacationDays = this._remainingVacationDays - daysOff;
    return this._remainingVacationDays;
  }

}

const surgeonRomero = new Surgeon('Francisco Romero', 'Cardiovascular');
const surgeonJackson = new Surgeon('Ruth Jackson', 'Orthopedics');

```

- **Method Calls - 클래스에서 함수 콜**

객체에서와 똑같이 부르면 된다.

```
class Dog {
  constructor(name) {
    this._name = name;
    this._behavior = 0;
  }

  get name() {
    return this._name;
  }

  get behavior() {
    return this._behavior;
  }

  incrementBehavior() {
    this._behavior++;
  }
}

const halley = new Dog('Halley');
```

```
let nikko = new Dog('Nikko'); // Create dog named Nikko
nikko.incrementBehavior(); // Add 1 to nikko instance's behavior
let bradford = new Dog('Bradford'); // Create dog name Bradford
console.log(nikko.behavior); // Logs 1 to the console
console.log(bradford.behavior); // Logs 0 to the console
```

- **Inheritance - 1 - 상속**

상속은 개발자들이 입력해야할 코드를 줄여줄 수 있는 역할을 한다. 즉, 프로퍼티나 메소드가 비슷한 객체끼리(자식클래스(subclass) 는 부모클래스(superclass) 에게 비슷한 부분을 상속받는다.

위에서 배운 Dog 클래스와 아주 유사한 Cat 클래스를 보자.

```
class Cat {
  constructor(name, usesLitter) {
    this._name = name;
    this._usesLitter = usesLitter;
    this._behavior = 0;
  }

  get name() {
    return this._name;
  }

  get usesLitter() {
    return this._usesLitter;
  }

  get behavior() {
    return this._behavior;
  }

  incrementBehavior() {
    this._behavior++;
  }
}
```

Dog 클래스와 다른 점은 usesLitter 뿐 나머지는 모두 같다. 이럴경우 수퍼클래스로부터 상속받는것이다.

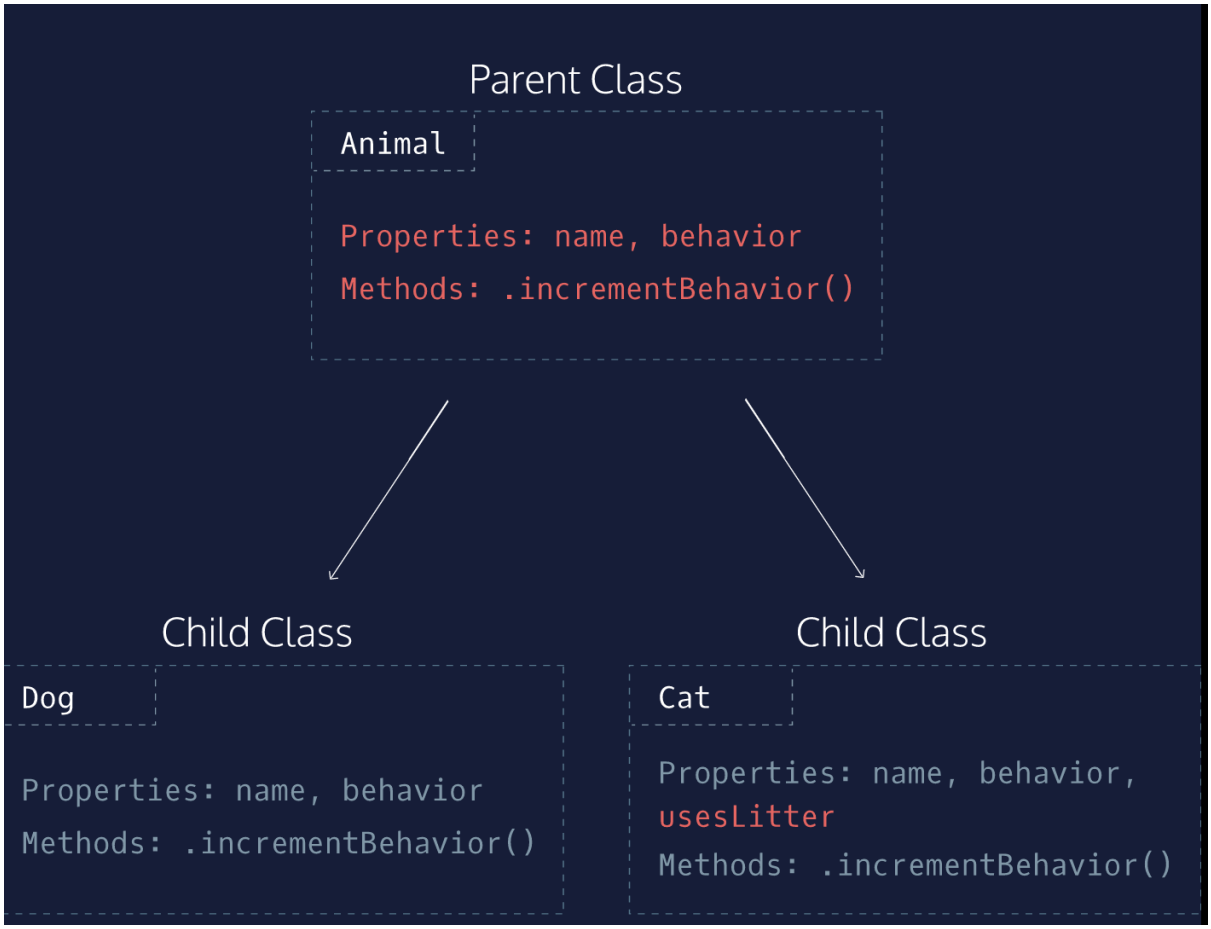
```
class Animal {
  constructor(name) {
    this._name = name;
    this._behavior = 0;
  }

  get name() {
    return this._name;
  }

  get behavior() {
    return this._behavior;
  }

  incrementBehavior() {
    this._behavior++;
  }
}
```

부모클래스 Animal 로 Dog 와 Cat 클래스를 모두 아우른다.



- **상속 - 2 -**

객체에 대한 정보를 얻은 후 직접 부모클래스 작성해보기

```
class HospitalEmployee {
  constructor(name) {
    this._name = name;
    this._remainingVacationDays = 20;
  }

  get name() {
    return this._name;
  }

  get remainingVacationDays() {
    return this._remainingVacationDays;
  }

  takeVacationDays(dayOff) {
    this._remainingVacationDays -= dayOff;
    return this._remainingVacationDays;
  }
}


```

- **상속 - 3 - extends 메소드**

이제 수퍼클래스를 이용해서 서브클래스를 만들어보자.

```
class Cat extends Animal {
  constructor(name, usesLitter) {
    super(name);
    this._usesLitter = usesLitter;
  }
}
```

서브클래스 Cat 을 만드는 과정이다.

extends 키워드를 사용하고 , 서브 클래스에 개별적으로 필요한 프로퍼티를 constructor 에 설정한다.

여기서 주의할 가장 중요한 점은 super( ) 클래스이다. 수퍼클래스의 this._name = name; 부분을 super 메소드로 상속받아 간단하게 작성하였다. 만약 super( ) 메소드를 사용하려면 constructor 의 첫번째 줄에 즉, this 키워드보다 먼저 작성해야 레퍼런스 오류가 안난다. 이점을 유의하자!!

```
19
20 class Nurse extends HospitalEmployee {
21   constructor(name, certifications) {
22     super(name);
23     this._certifications = certifications;
24   }
25 }
26
27
28 const nurseOlynyk = new Nurse('Olynyk', ['Trauma', 'Pediatrics']);
```

- **상속 - 4 -**

지금까지 부모클래스로부터 프로퍼티를 상속받고 개별적인 프로퍼티를 만드는 것을 배웠으니 이제 메소드에 대해서 배워보자.

extends 로 상속받은 자식클래스는 부모클래스의 모든 게터와 메소드들도 같이 상속받으므로 바로 사용이 가능하다.

```
const nurseOlynyk = new Nurse('Olynyk', ['Trauma',
'Pediatrics']);

nurseOlynyk.takeVacationDays(5);

console.log(nurseOlynyk.remainingVacationDays);
// 15
```

- **상속 - 5 -**

자식클래스도 자신의 프로퍼티, 게터, 세터, 그리고 메서드를 가질 수 있다. constructor 바로 밑에 정의하면 된다.

부모클래스를 변경해서 그 밑에 자식클래스의 메소드나 프로퍼티를 한번에 변경할 수 있다???

```
class Nurse extends HospitalEmployee {
  constructor(name, certifications) {
    super(name);
    this._certifications = certifications;
  }

  get certifications() {
    return this._certifications;
  }

  addCertification(newCertification) {
    this._certifications.push(newCertification);
  }

}
```

```
const nurseOlynyk = new Nurse('Olynyk', ['Trauma',
'Pediatrics']);
nurseOlynyk.takeVacationDays(5);
console.log(nurseOlynyk.remainingVacationDays);

nurseOlynyk.addCertification('Genetics');

console.log(nurseOlynyk.certifications);
```

- **Static Methods - 정적 메서드**

개인적인 인스턴스로는 이용이 불가능하지만, 직접적인 메서드 콜로 이용이 가능한 클래스를 살펴보자.

Date 클래스가 그 예이다. .now( ) 메서드는 정적이다. 그래서 다른 자식클래스의 인스턴스로 추가하지 못하고 Date 클래스에서 직접 호출해야한다.

```
class Animal {
  constructor(name) {
    this._name = name;
    this._behavior = 0;
  }

  static generateName() {
    const names = ['Angel', 'Spike', 'Buffy', 'Willow', 'Tara'];
    const randomNumber = Math.floor(Math.random()*5);
    return names[randomNumber];
  }
}
```

static 메소드이다. 이는 자식클래스에 상속하지 않는 메서드이며 사용하려면 무조건 Animal.generateName( ); 으로 써서 사용해야한다.

```
console.log(Animal.generateName()); // returns a name
```

You cannot access the `.generateName()` method from instances of the `Animal` class or instances of its subclasses (See below).

```
const tyson = new Animal('Tyson');
tyson.generateName(); // TypeError
```

오늘의 단어

- populate : 거주하다, 살다
- capitalize : 대문자로 쓰다.
- capitalize on/upon sth : ~을 활용하다, 기회로 삼다.
- prepend : 접두어를 붙이다. 다른것에 표현이나 구문을 덧붙이다.
- moot : (가능성이 너무 적어)고려할 가치가 없는