고급 주제

속성 순서

사실 javascript 객체의 멤버 변수란 것은 정확한 표현이 아니다.
멤버 변수가 아니고 속성(property)라고 표현하는 것이 맞다.
단순한 변수가 아니기 때문이다.

편의상 멤버 변수와 호칭을 섞어서 사용하도록 하겠다.

숫자 속성명이 가능하다.

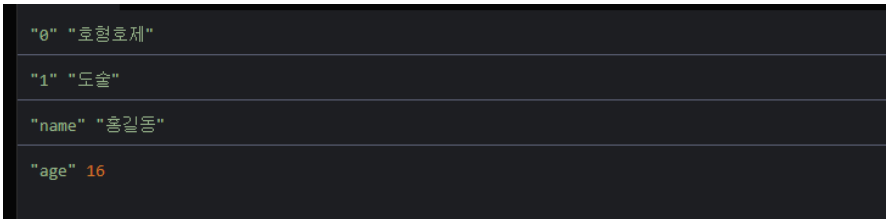
```
let person = { name: "홍길동", age: 16 }

person["age"] = 20
person[0] = "호형호제"
person[1] = "도술"

console.log(person); //{ '0': '호형호제', '1': '도술', name:
'홍길동', age: 20 }
console.log(person.age); //20
console.log(person["age"]); //20
console.log(person[0]) //호형호제
console.log(person.length) //undefined
```

person 객체는 배열이 아니다.
person[0] 이것은 배열의 0 번째 원소로 해석할 수 없고,
person 객체의 '0' 속성으로 해석해야 한다.

```
let person = { name: "홍길동", age: 16, 1: "도술", 0: "호형호
제"  }

console.log(person) //{ '0': '호형호제', '1': '도술', name:
'홍길동', age: 16 }
console.log(person[0]) //호형호제
```

```
let a = ["one", "two", "three", "four"];
for (let s of a) // 배열의 원소가 하나씩 꺼내지며 반복한다
    console.log(s); //one two three four

let person = { name: "홍길동", age: 16, 1: "도술", 0: "호형호
제"  }
for (let k in person) // 객체의 속성명이 하나씩 꺼내지며 반복한다
    console.log(k); // 0 1 name age
```

```
let person = { name: "홍길동", age: 16, 1: "도술", 0: "호형호
제"  }

for (let key in person)
    console.log(key, person[key])
```



객체의 속성을 for in 반복문으로 탐색할 수 있다.

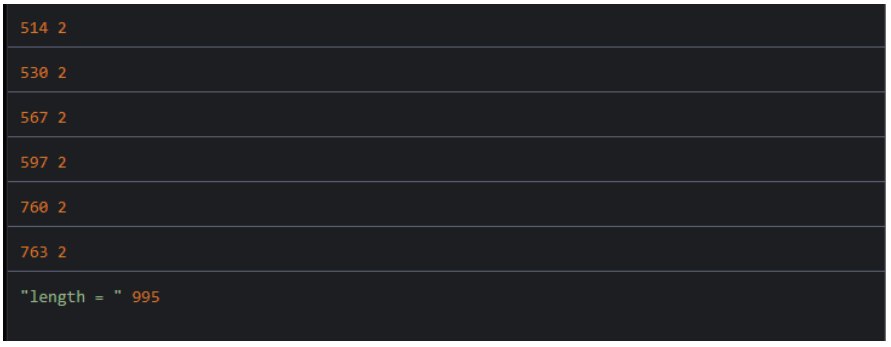
---

1 이상 1000 이하의 수를 랜덤하게 100 개를 생성했을 때,
두 번 이상 생성된 수를 출력하는 코드를 구현하자.

```
const MAX = 1000, MIN = 1;
let count = [];
```

```
for (let i = 0; i < 100; ++i) {
  let a = Math.floor(Math.random() * (MAX - MIN + 1) + MI
N);
  if (typeof count[a] == "undefined") count[a] = 1;
  else count[a] += 1;
}

for (let i = 0; i < count.length; ++i)
  if (count[i] >= 2)
    console.log(i, count[i]);

console.log("length = ", count.length);
```

count 배열을 이용하여, 각 수의 생성된 횟수를 센다.
마지막으로 count 배열의 크기를 출력한다.



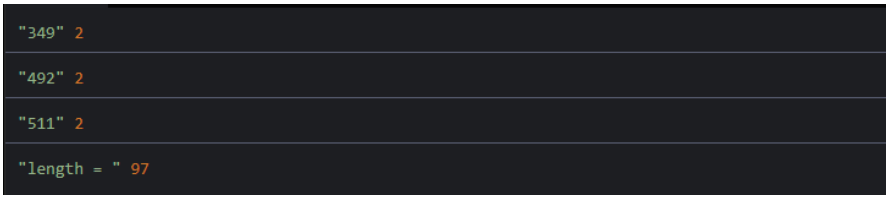
count 변수가 배열이고, 랜덤하게 생성된 값이 배열의 인덱스로 사용되기 때문에,
count 배열의 크기는, 랜덤하게 생성된 값의 최대값 + 1 이다.

```
const MAX = 1000, MIN = 1;
let count = { };

for (let i = 0; i < 100; ++i) {
  let a = Math.floor(Math.random() * (MAX - MIN + 1) + MI
N);
  if (typeof count[a] == "undefined") count[a] = 1;
  else count[a] += 1;
```

```
}

for (let key in count)
  if (count[key] >= 2)
    console.log(key, count[key]);

console.log("length = ", Object.keys(count).length);
```

count 객체를 Map 처럼 활용하여, 각 수의 생성된 횟수를 센다.
마지막으로 count 객체의 크기를 출력한다.

객체의 속성명이 Map의 key 역할을 하고, 속성값이 Map의 value 역할을 한다.



`for (let key in count)`

count 객체의 속서 각각에 대해서 반복한다.
반복할 때 마다 속성명이 key 변수에 대입된다.

`if (count[key] >= 2)`

count 객체의 key 속성값이 2 이상이면 true

```
const MAX = 1000, MIN = 1;
let count = { };

for (let i = 0; i < 100; ++i) {
  let a = Math.floor(Math.random() * (MAX - MIN + 1) + MI
N);
  count[a] = count[a] ? count[a] + 1 : 1;
}

for (let key in count)
  if (count[key] >= 2)
```

```
    console.log(key, count[key]);

console.log("length = ", Object.keys(count).length);
```

좀 더 간결하게 구현하였다.

count[a] ? count[a] + 1 : 1

count[a] 값이 undefined 이거나 0 이면,

조건식이 false가 되고, 이 표현식의 값이 1 이다.

count[a] 값이 0보다 큰 숫자이면

조건식이 true가 되고, 이 표현식의 값은 count[a] + 1 이다.

```
const MAX = 1000, MIN = 1;
let count = { };

for (let i = 0; i < 100; ++i) {
  let a = Math.floor(Math.random() * (MAX - MIN + 1) + MI
N);
  count[a] = (count[a] || 0) + 1;
}

for (let key in count)
  if (count[key] >= 2)
    console.log(key, count[key]);

console.log("length = ", Object.keys(count).length);
```

좀 더 간결하게 구현하였다.

(count[a] || 0)

count[a] 값이 false에 해당하는 값이면 (undefined 이거나 0 이면),

이 표현식의 값은 0 이다.

count[a] 값이 true에 해당하는 값이면 (0보다 큰 수이면),

이 표현식의 값이 count[a] 이다.

클래스

클래스 정의

```
person1.js
class Person {
  constructor(s, i) {
    this.name = s;
    this.age = i;
  }
}

let person = new Person("홍길동", 16);
console.log(person); //Person { name: '홍길동', age: 16 }
```

`class Person {`
Java 언어와 같은 방법으로 객체를 생성할 수 있다.
Person 클래스를 정의하고, Person 클래스로부터 객체를 생성할 수 있다.

`constructor(s, i) {`
생성자의 이름은 언제나 constructor 이다.
파라미터 변수 타입은 언제나 생략한다.

```
this.name = s;
this.age = i;
```

멤버 변수를 따로 선언할 필요 없다.
생성자에서 값을 대입할 때 멤버 변수가 생성된다.

`let person = new Person("홍길동", 16);`
Person 클래스의 객체를 생성한다.

this 생략 불가능

javascript 메소드에서 this를 생략할 수 없다.

메소드 구현

```
class Rectangle {
  constructor(w, h) {
    this.width = w;
    this.height = h;
  }

    area() {
    return this.width * this.height;
  }
}

let rectangle = new Rectangle(10, 15);
console.log(rectangle); //Rectangle { width: 10, height: 15
}
console.log(rectangle.area()); //150
```

```
area() {
  return this.width * this.height;
}
```

Rectangle 클래스에 area 메소드를 구현한다.
메소드의 리턴 타입은 언제나 생략한다.
javascript 언어는 변수의 타입, 파라미터 변수의 타입, 리턴 타입을 명시할 수 없다.

```
console.log(rectangle.area());
```

area 메소드의 리턴 값을 출력한다.

---

static 메소드 구현

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

    static distance(point1, point2) {
    let dx = point1.x - point2.x;
    let dy = point1.y - point2.y;
    return Math.hypot(dx, dy);
  }
}

let p1 = new Point(10, 15);
let p2 = new Point(25, 40);

console.log(Point.distance(p1, p2));
```

`static distance(point1, point2) {`

메소드 앞에 static 키워드를 붙이면, 그 메소드는 static 메소드가 된다.
static 메소드의 특징은, 현재 객체를 의미하는 this 키워드를 사용할 수 없다는 점이다.

this 키워드를 사용할 수 없기 때문에,
this 객체의 멤버 변수나, this 객체의 메소드를 호출할 수 없다.
this 객체가 아닌 다른 객체의 멤버 변수나 메소드를 사용하는 것은 가능하다.

상속 구현

```
class Figure {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }
```

```
    move(dx, dy) {
    this.x += dx;
    this.y += dy;
  }
}

class Rectangle extends Figure {
  constructor(x1, y1, x2, y2) {
    super(x1, y1);
    this.x2 = x2;
    this.y2 = y2;
  }

    move(dx, dy) {
    super.move(dx, dy);
    this.x2 += dx;
    this.y2 += dy;
  }
}

let rect = new Rectangle(10, 10, 20, 25);
rect.move(5, 10);
console.log(rect);
```

`class Rectangle extends Figure {`

Figure 클래스를 상속하여 Rectangle 클래스를 구현하였다.
부모 클래스의 멤버 변수와 메소드가 자식 클래스에 상속된다.

`super(x1, y1);`

부모 클래스의 생성자를 호출한다.

`super.move(dx, dy);`

부모 클래스의 move 메소드를 호출한다.

---

getter / setter

```
class Rectangle {
  constructor(x1, y1, x2, y2) {
    this.x1  = x1;
    this.y1  = y1;
    this.x2  = x2;
    this.y2  = y2;
  }

    getWidth() {
    return Math.abs(this.x2 - this.x1);
  }

    setWidth(width) {
    this.x2 = this.x1 + width;
  }

    getHeight() {
    return Math.abs(this.y2 - this.y1);
  }

    setHeight(height) {
    this.y2 = this.y1 + height;
  }
}

let rectangle = new Rectangle(5, 5, 20, 25);
rectangle.setWidth(30);
rectangle.setHeight(30);
console.log(rectangle);
console.log(rectangle.getWidth(), rectangle.getHeight());
```

Rectangle 클래스에는 x1, y1, x2, y2 멤버 변수만 정의되었다.

getWidth, setWidth, getHeight, setHeight 메소드는
이 x1, y1, x2, y2 멤버 변수를 사용하여 구현되었다.

위와 같이 get 메소드와 get 메소드를 구현하는 방법 보다 더 세련된 문법이 제공된다.

```
class Rectangle {
  constructor(x1, y1, x2, y2) {
    this.x1  = x1;
    this.y1  = y1;
    this.x2  = x2;
    this.y2  = y2;
  }

    get width() {
    return Math.abs(this.x2 - this.x1);
  }

    set width(width) {
    this.x2 = this.x1 + width;
  }

    get height() {
    return Math.abs(this.y2 - this.y1);
  }

    set height(height) {
    this.y2 = this.y1 + height;
  }
}

let rectangle = new Rectangle(5, 5, 20, 25);
rectangle.width = 30;
rectangle.height = 30;
console.log(rectangle);
console.log(rectangle.width, rectangle.height);
```

getter / setter 매소드를 구현하여 width 속성과 height 속성을 정의하였다.

width, height 속성은 멤버 변수가 아니고, getter / setter 메소드로 구현되었지만,
외부에서 사용할 때, 마치 멤버 변수인 것 처럼 사용할 수 있다.

연습 문제

문제 1: 간단한 `Book` 클래스

> 책을 나타내는 `Book` 클래스를 작성하세요.
>
> 이 클래스는 `title`(제목)과 `author`(저자) 속성을 가지며,
>
> 다음 기능을 포함해야 합니다

- **생성자**: `title`과 `author`를 받아서 초기화합니다.
- **메서드**: `getDetails()` 메서드를 작성하여 책의 제목과 저자를 문자열로 반환합니다

예시

```
const book = new Book("해리포터와 마법사의 돌", "J.K. 롤링");
console.log(book.getDetails()); // "해리포터와 마법사의 돌 저자:
J.K. 롤링"
```

답

문제 2: `Person` 클래스와 `Employee` 클래스

> 사람을 나타내는 `Person` 클래스와, `Person` 클래스를 상속받는
> `Employee` 클래스를 작성하세요.

1. `Person` **클래스** :

- 속성: `name`(이름), `age`(나이)
- 메서드: `getIntroduction()` - 사람의 이름과 나이를 문자열로 반환합니다.

2. `Employee` **클래스** :
   - 상속: `Person` 클래스에서 상속 받습니다.
   - 추가 속성: `position`(직위)
   - 메서드: `getDetails()` - 직위와 사람의 소개를 문자열로 반환합니다.

예시

```
const employee = new Employee("홍길동", 30, "웹 개발자");
console.log(employee.getIntroduction()); // "저의 이름은 홍길동 입니다, 나이는 30살 입니다"
console.log(employee.getDetails()); // "직업은 웹 개발자이고 제 이름은 홍길동 입니다, 나이는 30살 입니다"
```

답