자바스크립트 게임 프로젝트

기초문법

서재원 @librarywon

객체

-일반 자료형과 달리 다양한 데이터를 가능

-키로 구분된 데이터 집합이나 복잡한 개체(entity)를 저장 가능

객체


```
let user = {      // 객체
  name: "John",   // 키: "name",  값: "John"
  age: 30         // 키: "age", 값: 30
};
```

ex)문자열과 정수가 하나의 객체 안에서 존재 가능

객체 생성의 두가지 방법

```
1  let user = new Object(); // '객체 생성자' 문법
2  let user = {};  // '객체 리터럴' 문법
```

객체 구조

```
객체 이름 = {
프로퍼티1 (key :value)
프로퍼티2
}
```

프로퍼티

‘키 (key): 값(value)’ = 으로 구성

키(key) -> 문자형
값(value) -> 모든 자료형 허용

객체


```
let user = {       // 객체
  name: "John",    // 키: "name",  값: "John"
  age: 30          // 키: "age", 값: 30
};
```

기존 변수 받아와 객체 생성

```
1  function makeUser(name, age) {
2    return {
3      name: name,
4      age: age,
5      // ...등등
6    };
7  }
8
9  let user = makeUser("John", 30);
10 alert(user.name); // John
```

단축 프로퍼티

```
function makeUser(name, age) {
  return {
    name, // name: name 과 같음
    age,  // age: age 와 같음
    // ...
  };
}
```

```
let user = {
  name,  // name: name 과 같음
  age: 30
};
```

객체 프로퍼티 값 얻기

```
1  // 프로퍼티 값 얻기
2  alert( user.name ); // John
3  alert( user.age ); // 30
```

객체 프로퍼티 추가/수정

```
1  user.isAdmin = true;
```

기존에 있던 프로퍼티 -> 수정
기존에 없던 프로퍼티 -> 추가

객체 프로퍼티 삭제

```
1  delete user.age;
```

객체 in 연산자 활용

```
1  "key" in object
```

객체 안에 “key” 값이 존재?

객체 in 연산자 활용 예시

```
let user = { name: "John", age: 30 };

alert( "age" in user ); // user.age가 존재하므로 true가 출력됩니다.
alert( "blabla" in user ); // user.blabla는 존재하지 않기 때문에 false가 출력됩니다.
```

for ~ in 반복문

```
for (key in object) {
  // 각 프로퍼티 키(key)를 이용하여 본문(body)을 실행합니다.
}
```

기존에 알던 for(; ;) 와 완전히 다름

객체


```
let user = {
  name: "John",
  age: 30,
  isAdmin: true
};

for (let key in user) {
  // 키
  alert( key );  // name, age, isAdmin
  // 키에 해당하는 값
  alert( user[key] ); // John, 30, true
}
```

for (let key in user)

= for (let q in user)

‘key’는 변수 다른 이름 사용해도 괜찮음

문제1

다음 각 동작을 한 줄씩, 코드로 작성해보세요.

1. 빈 객체 `user`를 만듭니다.
2. `user`에 키가 `name`, 값이 `John`인 프로퍼티를 추가하세요.
3. `user`에 키가 `surname`, 값이 `Smith`인 프로퍼티를 추가하세요.
4. `name`의 값을 `Pete`로 수정해보세요.
5. `user`에서 프로퍼티 `name`을 삭제하세요.

문제2

객체에 프로퍼티가 하나도 없는 경우 `true`, 그렇지 않은 경우 `false`를 반환해주는 함수 `isEmpty(obj)`를 만들어 보세요.

아래와 같이 동작해야 합니다.

```
let schedule = {};

alert( isEmpty(schedule) ); // true

schedule["8:30"] = "get up";

alert( isEmpty(schedule) ); // false
```

문제3

모든 팀원의 월급에 대한 정보를 담고 있는 객체가 있다고 해봅시다.

```
let salaries = {
  John: 100,
  Ann: 160,
  Pete: 130
}
```

모든 팀원의 월급을 합한 값을 구하고, 그 값을 변수 `sum`에 저장해주는 코드를 작성해보세요. `sum`엔 `390`이 저장되어야겠죠?

주의: `salaries`가 비어있다면 `sum`에 `0`이 저장되어야 합니다.

객체

문제4 (응용)

객체 `obj`의 프로퍼티 값이 숫자인 경우 그 값을 `두 배` 해주는 함수 `multiplyNumeric(obj)`을 만들어보세요.

예시:

```
// 함수 호출 전
let menu = {
  width: 200,
  height: 300,
  title: "My menu"
};

multiplyNumeric(menu);

// 함수 호출 후
menu = {
  width: 400,
  height: 600,
  title: "My menu"
};
```

`multiplyNumeric`은 아무것도 반환하지 않아도 괜찮습니다. 객체 자체를 수정해주기만 하면 됩니다.

힌트) `typeof`를 사용하면 프로퍼티 값이 숫자인지 확인할 수 있습니다.

객체 메서드 만들기

```
let user = {
  name: "John",
  age: 30
};

user.sayHi = function() {
  alert("안녕하세요!");
};

user.sayHi(); // 안녕하세요!
```

객체 메서드 단축 구문

```
1  // 아래 두 객체는 동일하게 동작합니다.
2
3  user = {
4    sayHi: function() {
5      alert("Hello");
6    }
7  };
8
9  // 단축 구문을 사용하니 더 깔끔해 보이네요.
10 user = {
11   sayHi() { // "sayHi: function()"과 동일합니다.
12     alert("Hello");
13   }
14 };
```

매서드와 this

'this'는 매서드를 호출할 때 사용된 객체를 가리킴

객체


```
1  let user = {
2    name: "John",
3    age: 30,
4
5    sayHi() {
6      // 'this'는 '현재 객체'를 나타냅니다.
7      alert(this.name);
8    }
9
10 };
11
12 user.sayHi(); // John
```

user.sayHi() 실행 동안

this 는 user(객체)를 나타냄

객체


```
let user = { name: "John" };
let admin = { name: "Admin" };

function sayHi() {
  alert( this.name );
}

// 별개의 객체에서 동일한 함수를 사용함
user.f = sayHi;
admin.f = sayHi;

// 'this'는 '점(.) 앞의' 객체를 참조하기 때문에
// this 값이 달라짐
user.f(); // John  (this == user)
admin.f(); // Admin  (this == admin)

admin['f'](); // Admin (점과 대괄호는 동일하게 동작함)
```

호출한 객체가 중요

동일한 함수여도
다른 this 값 발생

문제

`calculator`라는 객체를 만들고 세 메서드를 구현해 봅시다.

- `read()`에선 프롬프트 창을 띄우고 더할 값 두 개를 입력받습니다. 입력받은 값은 객체의 프로퍼티에 저장합니다.
- `sum()`은 저장된 두 값의 합을 반환합니다.
- `mul()`은 저장된 두 값의 곱을 반환합니다.

```
let calculator = {
  // ... 여기에 답안 작성 ...
};

calculator.read();
alert( calculator.sum() );
alert( calculator.mul() );
```

힌트

```
let calculator = {
  sum() {
    [redacted]
  },

  mul() {
    [redacted]
  },

  read() {
    this.a = +prompt('첫 번째 값:', 0);
    this.b = +prompt('두 번째 값:', 0);
  }
};

calculator.read();
alert( calculator.sum() );
alert( calculator.mul() );
```

객체 생성자 함수

```
1  function User(name) {
2    this.name = name;
3    this.isAdmin = false;
4  }
5
6  let user = new User("보라");
7
8  alert(user.name); // 보라
9  alert(user.isAdmin); // false
```

1. 빈 객체 만들어 this에 할당
2. 프로퍼티 name,isAdmin추가
3. this 반환

생성자를 사용하는 이유?

new User("호진"), new User("지민") 이용하면
손쉽게 사용자 객체를 만들기 가능

일일이 객체를 만드는 방법보다 훨씬 간단

재사용 가능한 객체 코드 구현

클래스의 기본 문법

```
1  class MyClass {
2    // 여러 메서드를 정의할 수 있음
3    constructor() { ... }
4    method1() { ... }
5    method2() { ... }
6    method3() { ... }
7    ...
8  }
```

클래스의 예시

```
1  class User {
2
3    constructor(name) {
4      this.name = name;
5    }
6
7    sayHi() {
8      alert(this.name);
9    }
10
11 }
12
13 // 사용법:
14 let user = new User("John");
15 user.sayHi();
```

new User(name) 으로 호출
-> 객체 생성

constructor()
-> 객체 초기화

user객체가 sayHi() 매서드 호출

클레스도 프로퍼티 추가 가능

```
1  class User {
2    name = "보라";
3  }
4
5  let user = new User();
6  alert(user.name); // 보라
7  alert(User.prototype.name); // undefined
```

다음 시간은 Canvas