Object Prototype Extensions and Super Calls

get and set the prototype of an object, using the 'super' keyword to make code more concise

The `Object` API has been extended with two methods to get and set the prototype of an object:

- `Object.getPrototypeOf(o)` returns the prototype of `o`
- `Object.setPrototypeOf( o, proto )` sets the prototype of `o` to `proto`

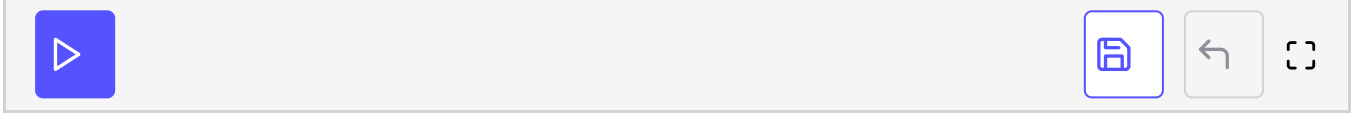
```
let proto = {
    whoami() { console.log('I am proto'); }
};

let obj = {
    whoami() {
        super.whoami();
        console.log('I am obj');
    }
};

console.log( Object.getPrototypeOf( obj ) );
//{}

Object.setPrototypeOf( obj, proto );

obj.whoami();
// I am proto
// I am obj
```



The following points are worth noting:

- The prototype of an object is `Object` by default.
- `Object.setPrototypeOf` can change this prototype to any object
- The `super` call saves some typing. Without `super`, we would have to write the following:

```
Object.getPrototypeOf( obj ).whoami.call( this );
```

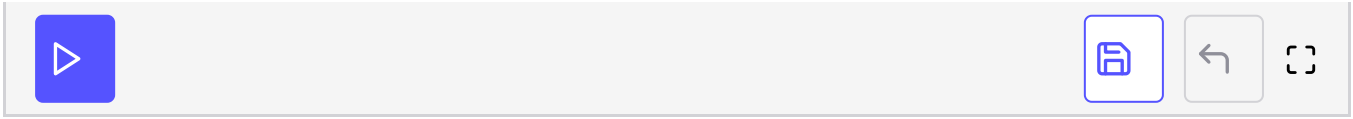
> Note that the concise method notation is mandatory for us to use the `super` keyword. Otherwise, a JavaScript error is thrown.

```
let proto = {
    whoami() { console.log('I am proto'); }
};

let obj = {
    whoami: function() {
        super.whoami();
        console.log('I am obj');
    }
};

Object.setPrototypeOf( obj, proto );

obj.whoami();
```



Uncaught SyntaxError: 'super' keyword unexpected here

Now, let’s solve some exercises before learning new concepts.