Exercise on Symbols

The exercises below will give you a deeper understanding of symbol along with the pros and cons of using them.

Exercise 1:

What are the pros and cons of using an underscore prefix for expressing our intention that a field is private? Compare this approach with symbols!

```
let mySquare = {
    _width: 5,
    getWidth() { return this._width; }
}
```

Solution 1:

Pros:

- Notation and developer experience is simple, provided that your team spreads this practice
- It does not result in a hard-to-read code structure, all you need is one more character

Cons:

- The properties are not private in practice, they are just denoted as private, which opens up a possibility of hacking quick and dirty solutions
- Unlike symbols, there is no clear separation between public and private properties. Private properties appear in the public interface of an object and they are enumerated in for...of loops, using the spread operator, and Object.keys

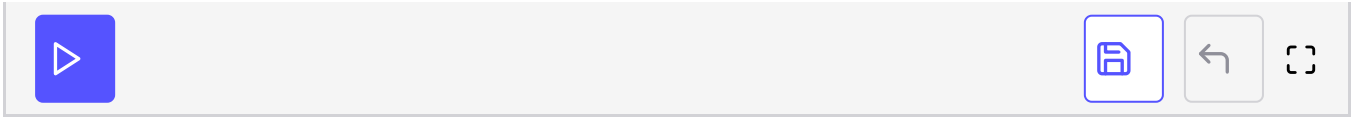
Exercise 2:

Find a way to simulate truly private fields in JavaScript!

Solution 2:

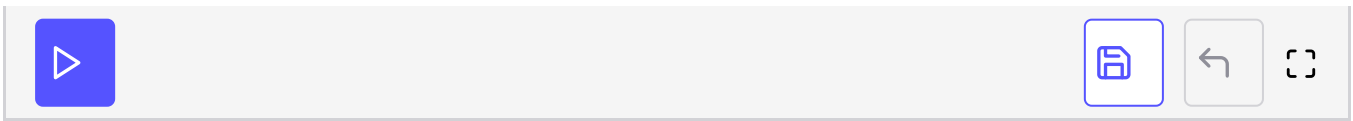
When it comes to constructor functions, private members can be declared inside a constructor function using var, let, or const. As in the following,

```
function F() {
  let privateProperty = 'b';
  this.publicProperty = 'a';
}

let f = new F();
console.log(f.publicProperty);  // returns 'a'
console.log(f.privateProperty); // returns undefined
```



In order to use the same idea for classes, we have to place the method definitions that use private properties in the constructor method in a scope where the private properties are accessible. We will use Object.assign to accomplish this goal. This solution was inspired by an article I read on this topic by Dr. Axel Rauschmayer on Managing private data of ES6 classes11.

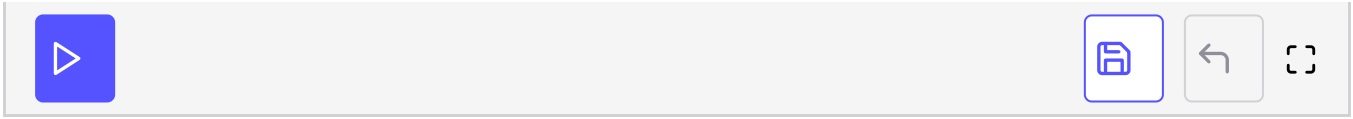
```
class C
{
  constructor()
  {
    let privateProperty = 'a';
     Object.assign( this,
     {
       logPrivateProperty()
       {
         console.log( privateProperty );
       }
     } );
  }
}

let c = new C();
c.logPrivateProperty();
```



The field privateProperty is not accessible in the c object. The solution also

The field privateProperty is not accessible in the C object. The solution also works when we extend the C class.

```
// Class C from the above is prepended to keep the code short
class D extends C { constructor() {
        super();
        console.log( 'Constructor of D' );
    }
}

let d = new D()
d.logPrivateProperty()
```



For the sake of completeness, there are two other ways for creating private variables:

- Weak maps: we will introduce it in a later section. We can achieve true privacy with it, at the expense of writing less elegant code,
- TypeScript: introduces compile time checks whether our code treats private variables as private.