Getters and Setters

introduction to getters and setters, and their advantages

Getters and setters are used to create computed properties.

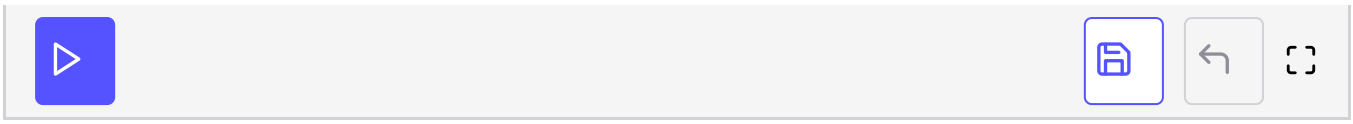
```
class Square {
    constructor( width ) { this.width = width; }
    get area() {
        console.log('get area');
        return this.width * this.width;
    }
}

let square = new Square( 5 );

console.log(square.area)
```

Note that `area` only has a getter. Setting `area` does not change anything, as `area` is a computed property that depends on the width of the square.

For the sake of demonstrating setters, let's define a `height` computed property.

```
class Square {
    constructor( width ) { this.width = width; }
    get height() {
        console.log( 'get height' );
        return this.width;
    }
    set height( h ) {
        console.log( 'set height', h );
        this.width = h;
    }
    get area() {
        console.log( 'get area' );
        return this.width * this.height;
    }
}

let square = new Square( 5 );
let print;
```

```
let print;

print=square.width;
console.log(print+'\n')
//> 5

print=square.height
console.log(print+'\n')
//> get height
//> 5

print=square.height = 6
console.log(print+'\n')
//> set height 6
//> 6

print=square.width
console.log(print+'\n')
//> 6

print=square.area
console.log(print+'\n')
//> get area
//> get height
//> 36

print=square.width = 4
console.log(print+'\n')
//> 4

print=square.height
console.log(print+'\n')
//> get height
//> 4
```



Width and height can be used as regular properties of a `Square` object, and the two values are kept in sync using the height getter and setter.

Advantages of getters and setters:

- **Elimination of redundancy**: Computed fields can be derived using an algorithm depending on other properties.
- **Information hiding**: It allows you to hide properties that are retrievable or settable through getters or setters.
- **Encapsulation**: You can couple other functionality with getting/setting a value.
- **Defining a public interface**: It is possible to keep these definitions constant and reliable, while you are free to change the internal

representation used for computing these fields. This comes in handy e.g. when dealing with a DOM structure, where the template may change.

- **Easier debugging**: Just add debugging commands or breakpoints to a setter, and you will know what caused a value to change.

Now, let's move on to static methods.