Symbols

WE'LL COVER THE FOLLOWING

Symbols are a new primitive data type in JavaScript, they can to create anonymous keys for object properties, among other things. One key feature of these properties is that they do not show up in a `for..in` loop. Symbols are created using the `Symbol` function and passed a description.

```
const mySym = Symbol('name');
```

No two symbols are the same, if we created another `Symbol('name')` it would be completely different.

Symbols for private keys

Because of a Symbols ability to be unique and excluded from enumeration, it is common to use them to create private properties. Because JavaScript has no `private` keyword this is a great use case for the `Symbol`.

Let's assume we made a `Superhero` class in a module, we can use a `Symbol` to create a `_secretIdenity` property to be used in the class.

```
//Super hero modules
const _secretIdenity = Symbol('_secert');

exports default class Superhero {
    constructor(identity) {
        this[_secretIdenity] = identity;
        this.power = 100;
    }
```

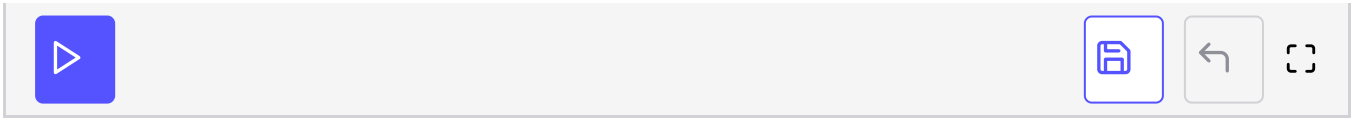
```
    getIdentity() {
      return this[_secretIdenity];

    }
}
```

In our main app we import this class and create a new `Superhero`.

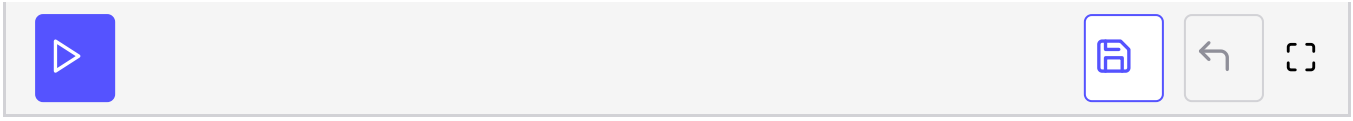
```
//Main application
//import Superhero from 'superhero.js';

const wonderWoman = new Superhero("Diana Prince");
console.log(wonderWoman.getIdentity());
```



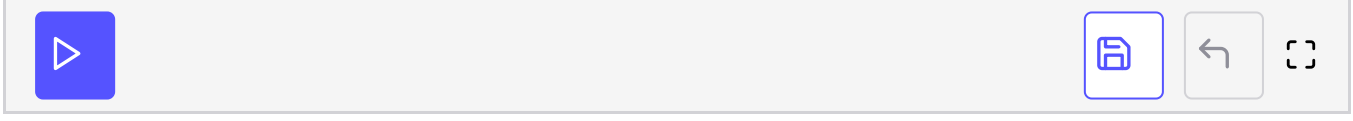
We are not able to access the property like we normally would, however using our `getIdentity` method we can get access to the property! If we run this new object through a `for...in` loop you will notice we don't see the Symbol property at all!

```
for(let key in wonderWoman){
  console.log(key);
}
//power
```



This is not perfect however, because we can us the `Object.getOwnPropertySymbols()` method to get a list of the symbols.

```
const theSymbol = Object.getOwnPropertySymbols(wonderWoman);
console.log(wonderWoman[theSymbol[0]]); //Diana Prince
```



But is starts to get us there.

Well known symbols

Symbols can also be used to change how we an iterate through an array. Take

this `cities` array for example, we can use the `Symbol.iterator` property of

our array to create a new Iterator Object, remember we talked about Iterators in the Generators Chapter.

```
const cities = ['Toronto', 'Montreal', 'Boston', 'San Fransisco'];
let cityIterator = cities[Symbol.iterator]();

console.log(cityIterator.next());
/*
    { value: "Toronto", done: false }
*/
console.log(cityIterator.next());
/*
    { value: "Montreal", done: false }
*/
console.log(cityIterator.next());
/*
    { value: "Boston", done: false }
*/
console.log(cityIterator.next());
/*
    { value: "San Fransisco", done: false }
*/
console.log(cityIterator.next());
/*
    { value: undefined, done: true }
*/
```

This now gives us a new way to take an existing array and iterate through it. Because of this, we can use this well known Symbol to create an iterator object out of a HTMLCollection.

```
let listItems = document.querySelectorAll('li');
let listIterator = listItems[Symbol.iterator]();

listIterator.next();
```

I am very interested to see how these will be used going forward, for more information about Symbols, please check out the additional resources below.

Additional Resources

- https://developer.mozilla.org/en-US/docs/Glossary/Symbol
- https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Symbol