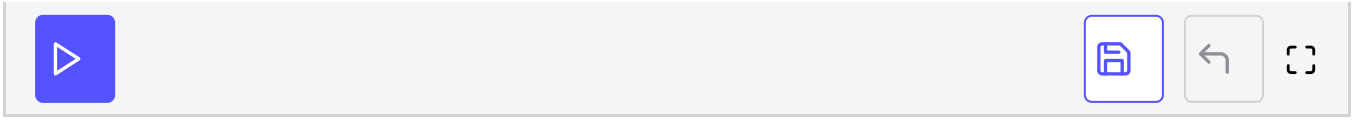
Symbol

symbol - a new ES6 primitive type and its use cases

ES6 introduces a new primitive type for JavaScript: Symbols. The global `Symbol()` function creates a JavaScript symbol. Each time the `Symbol()` function is called, a new unique symbol is returned.

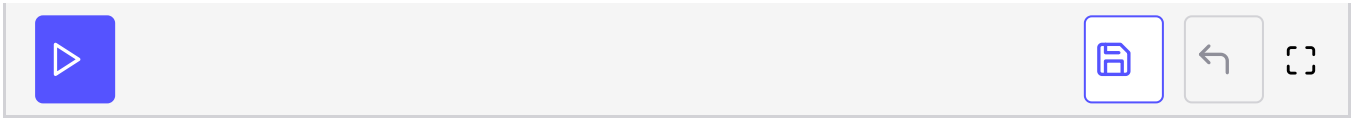
```
let symbol1 = Symbol();
let symbol2 = Symbol();

console.log( symbol1 === symbol2 );
//> false
```



Symbols don't have a literal value. All you should know about the value of a symbol is that each symbol is treated as a unique value. In other words, no two symbols are equal.

A symbol is a new *type* in JavaScript.

```
console.log( typeof symbol1 );
//> "symbol"
```



Symbols are useful, because they act as unique object keys.

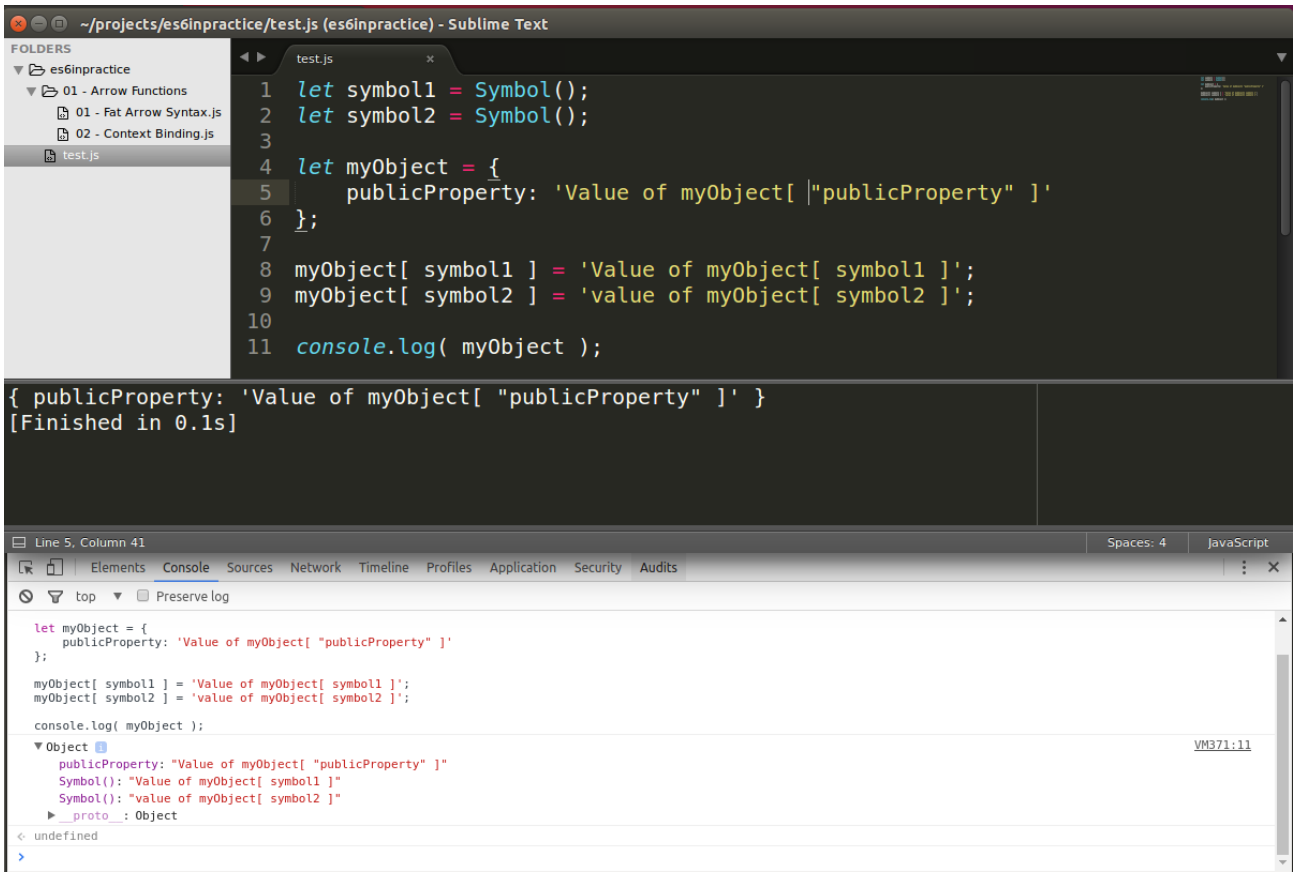
```
let myObject = {
    publicProperty: 'Value of myObject ["publicProperty"]'
};

myObject[ symbol1 ] = 'Value of myObject [symbol1]';
myObject[ symbol2 ] = 'value of myObject [symbol2]';

console.log( myObject );
//> Object
```

```
//> Object
//>    publicProperty: "Value of myObject[ "publicProperty" ]"
//>    Symbol(): "Value of myObject[ symbol1 ]"
//>    Symbol(): "value of myObject[ symbol2 ]"
//>    __proto__: Object

console.log( myObject[ symbol1 ] );
//> Value of myObject[ symbol1 ]
```

When console logging `myObject` inside the *Chrome Developer Tools*, you can see that both symbol properties are stored in the object. The literal `"Symbol()"` is the return value of the `toString()` method called on the symbol. This value denotes the presence of a symbol key in the console. We can retrieve the corresponding values if we have access to the right symbol.

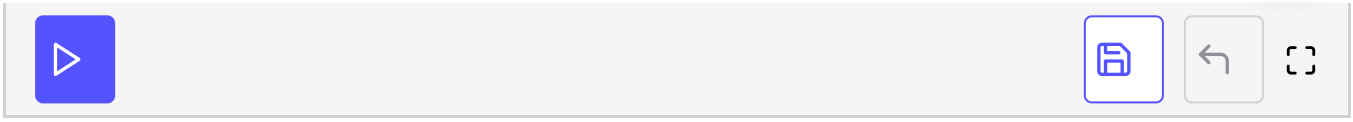
In some environments such as node and Microsoft Edge console, Symbol keys are not logged. In case you don't see the two Symbol properties, the problem is not with your copy-pasting or typing skills.

Symbol properties do not appear in the console of NodeJs (bottom part of the code editor), but they appear in the Google Chrome devtools(below the code editor).



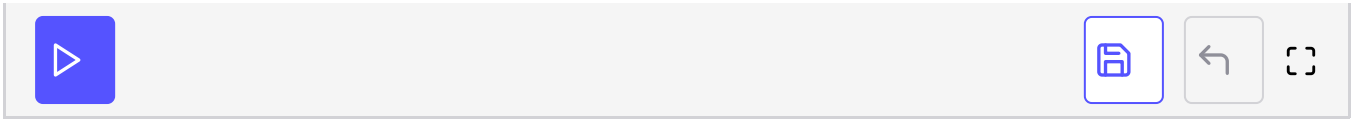
Following this node issue, using `console.dir`, with `showHidden` flag set to `true`, reveals the Symbol keys.

```
console.dir( myObject, {showHidden: true} )
```



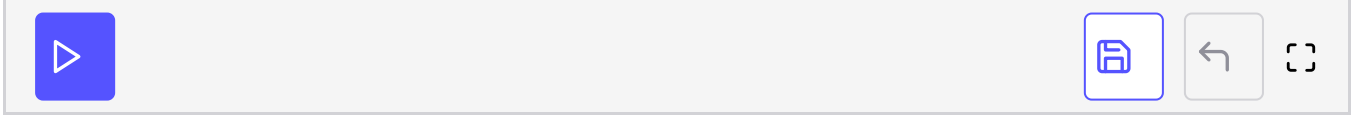
Properties with a symbol key don't appear in the JSON representation of your object. Not even the for-in loop or `Object.keys` can enumerate them:

```
JSON.stringify( myObject )
//> "{"publicProperty":"Value of myObject[ \"publicProperty\" ] "}"

for( var prop in myObject ) {
    console.log( prop, myObject[prop] );
}
//> publicProperty Value of myObject[ "publicProperty" ]

console.log( Object.keys( myObject ) );
//> ["publicProperty"]
```



Even though properties with Symbol keys don't appear in the above cases, these properties are not fully private in a strict sense. `Object.getOwnPropertySymbols` provides a way to retrieve the symbol keys of your objects:

```
console.log(Object.getOwnPropertySymbols(myObject));
//> [Symbol(), Symbol()]

console.log(myObject[ Object.getOwnPropertySymbols(myObject)[0] ]);
//> "Value of myObject[ symbol1 ]"
```



> If you choose to represent private variables with Symbol keys, make sure you don't use `Object.getOwnPropertySymbols` to retrieve properties that are intended to be private. In this case, the only use cases for

> `Object.getOwnPropertySymbols` are testing and debugging.

As long as you respect the above rule, your object keys will be private from the perspective of developing your code. In practice, however, be aware that others will be able to access your private values.

Even though symbol keys are not enumerated by `for...of`, the spread operator, or `Object.keys`, they still make it to shallow copies of our objects:

```
clonedObject = Object.assign( {}, myObject );

console.log( clonedObject );
//> Object
//>    publicProperty: "Value of myObject[ "publicProperty" ]"
//>    Symbol(): "Value of myObject[ symbol1 ]"
//>    Symbol(): "value of myObject[ symbol2 ]"
//>    __proto__: Object
```

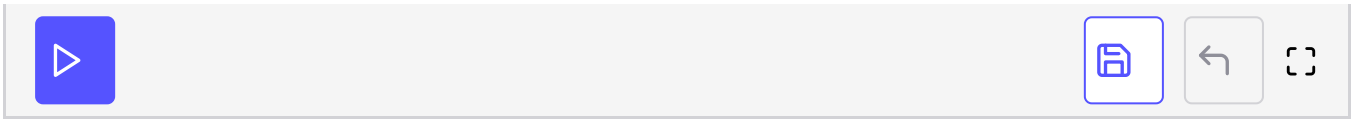
Naming your symbols properly is essential in indicating what your symbol is used for. If you need additional semantic guidance, it is also possible to attach a description to your symbol. The description of the symbol appears in the string value of the symbol.

```
let leftNode = Symbol( 'Binary tree node' );
let rightNode = Symbol( 'Binary tree node' );

console.log( leftNode )
//> Symbol(Binary tree node)
```

> Always provide a description for your symbols, and make your descriptions unique. If you use symbols for accessing private properties, treat their descriptions as if they were variable names.

Even if you pass the same description to two symbols, their value will still differ. Knowing the description does not make it possible for you to create the same symbol.

```
let leftNode = Symbol( 'Binary tree node' );
let rightNode = Symbol( 'Binary tree node' );
console.log( leftNode === rightNode );
//> false
```



In the next lesson, we will talk about the global resource for creating symbols.