Object.preventExtensions, seal, and freeze

Learn how Object.freeze and Object.seal are similar to Object.assign and help us implement functional programming. ES2015 adds the ability to truly freeze an object and make it immutable.

In this lesson, we'll cover three functions that allow us to enforce functional programming. Each of these functions makes our object inflexible to varying degrees. We can prevent property addition, or we can freeze an object entirely.

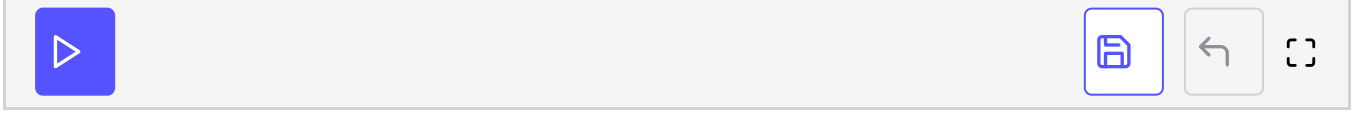
`Object.preventExtensions`

Preventing Extensions

This is a function that helps us approach immutability. Once it's called on an object, we can't add anything to it.

Deletions and changes are still allowed. Only property addition is forbidden.

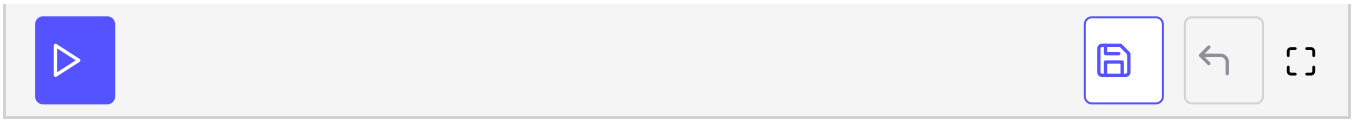
```
const obj = {
    prop: 'value',
};

Object.preventExtensions(obj);

obj.nextProp = 8; // fails
console.log(obj); // -> { prop: 'value' }

obj.prop = 4; // succeeds
console.log(obj); // -> { prop: 4 }

delete obj.prop; // succeeds
console.log(obj); // -> {}

obj.prop = 17; // since the property no longer exists, fails
console.log(obj); // -> {}
```



Checking Extensibility

We have a utility method that tells us if an object is extensible or not: `Object.isExtensible`.

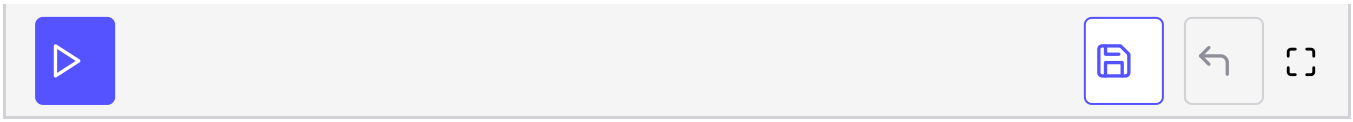
```
const obj = {};

console.log(Object.isExtensible(obj)); // -> true
Object.preventExtensions(obj);
console.log(Object.isExtensible(obj)); // -> false
```



Locked Prototype

`Object.preventExtensions` permanently locks an object's prototype. Attempts to use `Object.setPrototypeOf` or to change the `__proto__` property directly result in an error.

```
const obj = {};
Object.preventExtensions(obj);
Object.setPrototypeOf(obj, {});
//-> TypeError: #<Object> is not extensible
```

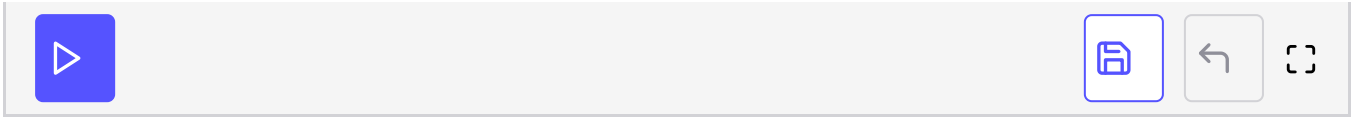


Configuration

Property attributes (configurable, writable, enumerable) can still be changed through `Object.defineProperty`.

```
const obj = {
    prop: 49,
};

Object.preventExtensions(obj);

for(let i in obj) {
    console.log(i); // -> prop
}

Object.defineProperty(obj, 'prop', {
    value: 17,
    enumerable: false,
    configurable: false,
    writable: false
}); // succeeds

obj.prop = 9; // fails
console.log(obj.prop); // -> 17
```

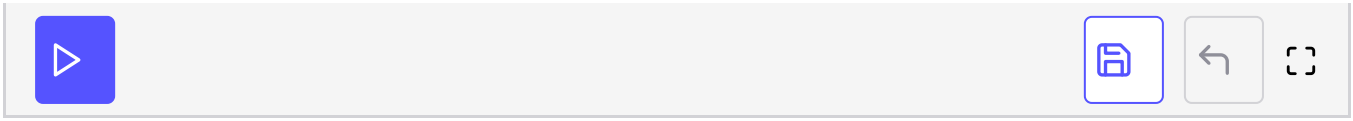
```
for(let i in obj) {
    console.log(i); // -> ∅
}
```



Shallow Change

The effect is shallow. It does not go into nested objects and prevent their extensions. This means that an object containing references to a non-frozen object, array, or function is not truly non-extensible. The inner object is unaffected.

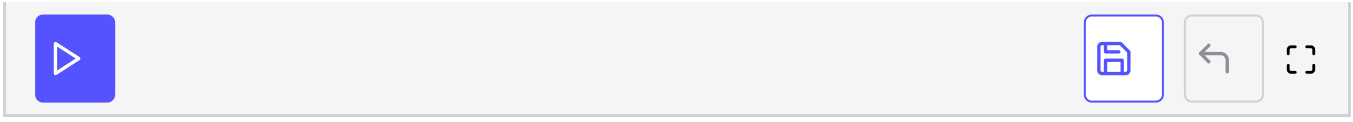
```
const obj = {
    innerObj: {}
};

Object.preventExtensions(obj);

obj.innerObj.prop = 8; // succeeds
console.log(obj); // -> { innerObj: { prop: 8 } }
```



Arrays

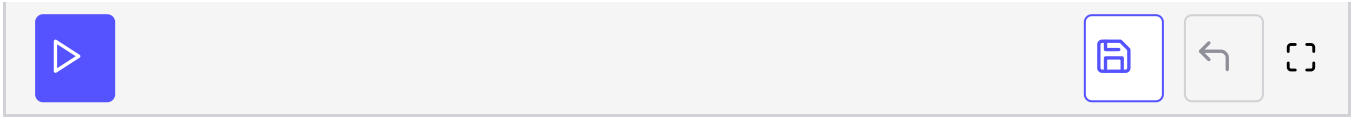
`Object.preventExtensions` will work on arrays the same as it does on objects. We will not be able to add anything to an array that is not extensible. Attempting to do so through assignment will fail silently.

```
const arr = ['abc', 'def'];

Object.preventExtensions(arr);

arr[1] = 24; // succeeds
arr[2] = 88; // fails
console.log(arr); // -> [ 'abc', 24 ]
```



Instead of failing silently, attempts to `push`, `pop`, `shift`, and `unshift` will throw an error.

```
const arr = [ 'abc', 'def' ];
Object.preventExtensions(arr);


arr.push('ghi');
// -> TypeError: Cannot add property 2, object is not extensible
```



Functions

The same goes for functions. Functions are first-class objects and can have properties placed on them. `Object.preventExtensions` will do all of the same things to a function that it does to objects and arrays.

`Object.seal`

Similarities

This function is similar to `Object.preventExtensions`, but goes a bit further. All of the same limitations of `Object.preventExtensions` apply.

- We can't add properties to or change the prototype of sealed objects.
- Properties retain their `writable` and `enumerable` status. They can still be deleted.
- `Object.seal` performs a shallow seal.
- It works on objects, arrays, and functions.

Configuration

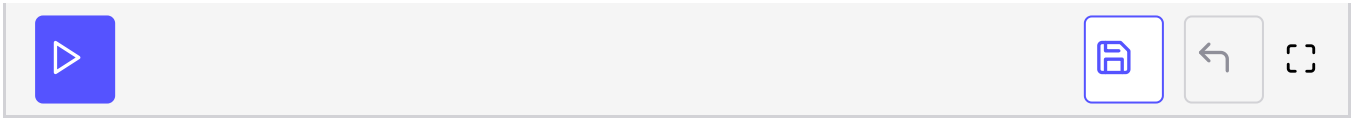
When an object is sealed, however, all existing properties have their `configurable` status set to `false`. This means attempts to use `Object.defineProperty` will all result in an error.

Checking the Seal

A utility method to check whether an object is sealed is `Object.isSealed`. It will return `true` for a sealed object. `Object.isExtensible` will return `false` for a sealed object.

```
const obj = {
    prop: 49,
};
```

```
Object.seal(obj);

console.log(Object.isSealed(obj)); // -> true
console.log(Object.isExtensible(obj)); // -> false

obj.prop = 17; // works
console.log(obj); // -> { prop: 17 }

Object.defineProperty(obj, 'prop', {
    value: 24,
    enumerable: true,
    configurable: true,
    writable: true
}); // -> TypeError: Cannot redefine property: prop
```



`Object.freeze`

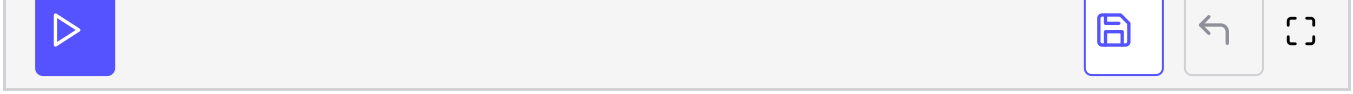
Immutability

This function allows us to truly freeze an object and make it immutable. When an object is frozen, we can't add, remove, or change properties. `writable` and `configurable` are set to false for every existing property. The prototype is locked. Again, it's a shallow freeze.

Checking Temperature

A utility method to check whether an object is frozen is `Object.isFrozen`. `Object.isFrozen` and `Object.isSealed` will return `true` for a frozen object, and `Object.isExtensible` will return `false`.

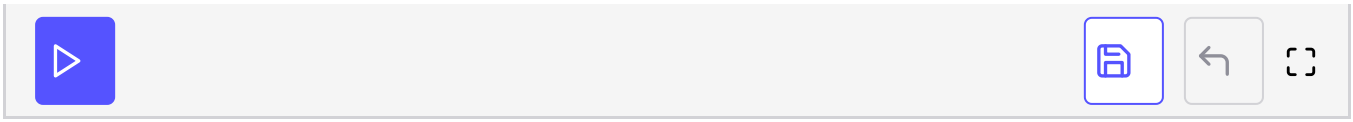
```
const obj = {
    prop: 'value'
};

Object.freeze(obj);
console.log(Object.isFrozen(obj)); // -> true
console.log(Object.isSealed(obj)); // -> true
console.log(Object.isExtensible(obj)); // -> false

obj.prop = 4; // fails
console.log(obj); // -> { prop: 'value' }

delete obj.prop; // fails
console.log(obj); // -> { prop: 'value' }

obj.nextProp = 8; // fails
console.log(obj); // -> { prop: 'value' }
```



Accessors

Any attempt to change the internal state of an object will fail. This means a setter function will not work as expected and will fail silently.

```
const obj = {
    _val: 'value',

    get value() {
        return this._val;
    },

    set value(val) {
        this._val = val;
    }
};

Object.freeze(obj);

obj.value = 4; // fails
console.log(obj.value); // -> value
```



Summary

`preventExtensions`

This function, when called on an object:

- Prevents addition of properties
- Locks prototype
- Works on objects, arrays, and functions
- Performs a shallow change
- Makes `Object.isExtensible` return `false`

`seal`

In addition to all of the above, this function, when called on an object:

- Makes every existing property non-configurable
- Makes `Object.isSealed` return `true`

`freeze`

In addition to all of the above, this function, when called on an object:

- Makes every existing property non-writable
- Makes `Object.isFrozen` return `true`

Together, these functions provide yet another way to implement functional programming. They enforce it. Freezing an object is the most powerful way to ensure immutability. Sealing and declaring non-extensible are lesser actions.