Object Methods

Object.assign()

The Object.assign() method copies all enumerable own properties from one or more source objects to a target object. It returns the modified target object.

JavaScript Demo: Object.assign()

```
const target = { a: 1, b: 2 };
const source = { b: 4, c: 5 };

const returnedTarget = Object.assign(target, source);

console.log(target);
// expected output: Object { a: 1, b: 4, c: 5 }

console.log(returnedTarget === target);
// expected output: true

```

Object.create()

The Object.create() method creates a new object, using an existing object as the prototype of the newly created object.

JavaScript Demo: Object.create()

```
const person = {
  isHuman: false,
  printIntroduction: function() {
    console.log(`My name is ${this.name}. Am I human? ${this.isHuman}`);
  }
};

const me = Object.create(person);

me.name = 'Matthew'; // "name" is a property set on "me", but not on "person"
me.isHuman = true; // inherited properties can be overwritten

me.printIntroduction();
// expected output: "My name is Matthew. Am I human? true"
```

Object.entries()

The Object.entries() method returns an array of a given object's own enumerable string-keyed property [key, value] pairs. This is the same as iterating with a for...in loop, except that a for...in loop enumerates properties in the prototype chain as well.

JavaScript Demo: Object.entries()

```
const object1 = {
  a: 'somestring',
  b: 42
};

for (const [key, value] of Object.entries(object1)) {
  console.log(`${key}: ${value}`);
}

// expected output:
// "a: somestring"
// "b: 42"

```

Object.freeze()

The Object.freeze() method *freezes* an object. Freezing an object prevents extensions and makes existing properties non-writable and non-configurable. A frozen object can no longer be changed: new properties cannot be added, existing properties cannot be removed, their enumerability, configurability, writability, or value cannot be changed, and the object's prototype cannot be re-assigned. freeze() returns the same object that was passed in.

JavaScript Demo: Object.freeze()

```
const obj = {
  prop: 42
};

Object.freeze(obj);

obj.prop = 33;
// Throws an error in strict mode

console.log(obj.prop);
// expected output: 42

```

Object.hasOwn()

The Object.hasOwn() static method returns true if the specified object has the indicated property as its *own* property. If the property is inherited, or does not exist, the method returns false.

JavaScript Demo: Object.hasOwn()

```
const object1 = {
  prop: 'exists'
};

console.log(Object.hasOwn(object1, 'prop'));
// expected output: true

console.log(Object.hasOwn(object1, 'toString'));
// expected output: false

console.log(Object.hasOwn(object1, 'undeclaredPropertyValue'));
// expected output: false

```

Object.hasOwnProperty()

The hasOwnProperty() method returns a boolean indicating whether the object has the specified property as its own property (as opposed to inheriting it).

JavaScript Demo: Object.prototype.hasOwnProperty()

```
const object1 = {
    property1: 42
};

console.log(object1.hasOwnProperty('property1'));
// expected output: true

console.log(object1.hasOwnProperty('toString'));
// expected output: false

console.log(object1.hasOwnProperty('hasOwnProperty'));
// expected output: false

```

Object.keys()

The Object.keys() method returns an array of a given object's own enumerable property **names**, iterated in the same order that a normal loop would.

JavaScript Demo: Object.keys()

```
1  const object1 = {
2    a: 'somestring',
3    b: 42,
4    c: false
5  };
6
7  console.log(Object.keys(object1));
8  // expected output: Array ["a", "b", "c"]
9
```

Object.isExtensible()

The Object.isExtensible() method determines if an object is extensible (whether it can have new properties added to it).

JavaScript Demo: Object.isExtensible()

```
1  const object1 = {};
2
3  console.log(Object.isExtensible(object1));
4  // expected output: true
5
6  Object.preventExtensions(object1);
7
8  console.log(Object.isExtensible(object1));
9  // expected output: false
10
```

Object.preventExtensions()

The Object.preventExtensions() method prevents new properties from ever being added to an object (i.e. prevents future extensions to the object). It also prevents the object's prototype from being re-assigned.

JavaScript Demo: Object.preventExtensions()

```
const object1 = {};

Object.preventExtensions(object1);

try {
  Object.defineProperty(object1, 'property1', {
    value: 42
  });
} catch (e) {
  console.log(e);
  // expected output: TypeError: Cannot define property property1, object is not extensible
}

```

Object.prototype.propertyIsEnumerable()

The propertyIsEnumerable() method returns a Boolean indicating whether the specified property is enumerable and is the object's own property.

JavaScript Demo: Object.prototype.propertyIsEnumerable()

```
const object1 = {};
const array1 = [];
object1.property1 = 42;
array1[0] = 42;

console.log(object1.propertyIsEnumerable('property1'));
// expected output: true

console.log(array1.propertyIsEnumerable(0));
// expected output: true

console.log(array1.propertyIsEnumerable('length'));
// expected output: false

```