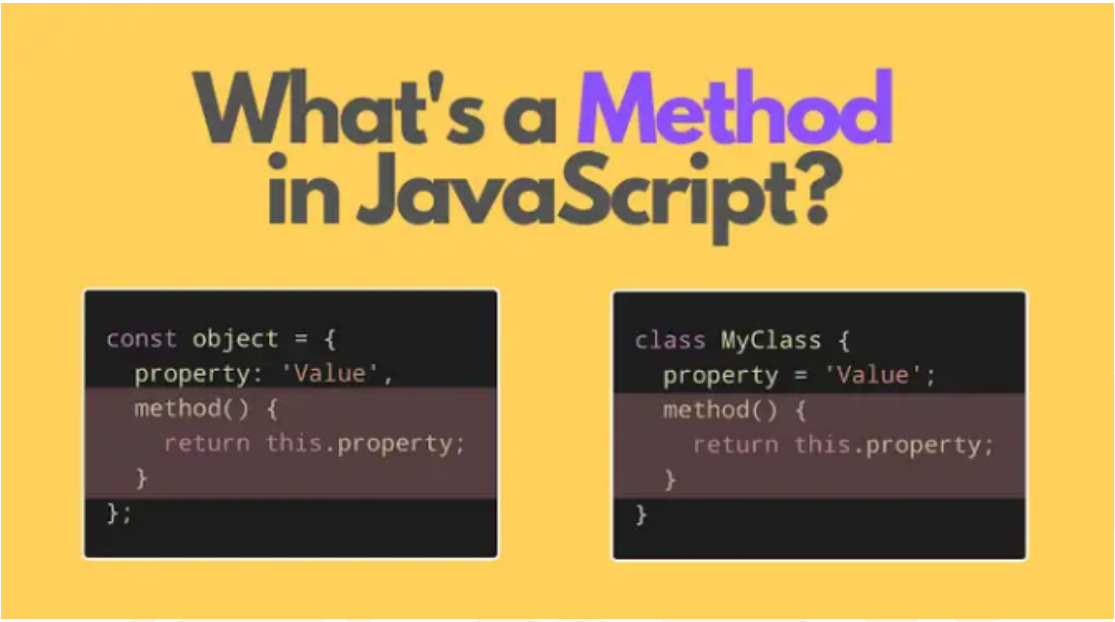
Lesson:

Object methods and enumerating properties

Topics

- Object Methods
- Different types of Object Methods with examples.
- Enumerating properties

Object Methods



In JavaScript, object methods are functions that are associated with an object. These methods can be defined as properties of an object and can be invoked to perform specific actions or operations related to that object. Actions on objects are carried out using methods. An object Property that includes a function declaration is known as an object method.

Note - Objects are mutable by default i.e. we can freely change, add, and remove the properties of objects.

Different types of Object Methods with examples.

The different types of object methods include-

1. Object.keys(),
2. Object.values(),
3. Object.entries(),
4. Object.assign(),
5. Object.freeze() & Object.isfrozen(),
6. Object.seal() & Object.isSeal(),
7. Object.fromEntries(),
8. Object.create(),
9. Object.hasOwn(),
10. Object.getOwnPropertyNames(),,

11. Object.getOwnPropertyDescriptors(),
12. Object.defineProperty(),
13. Object.defineProperties(),
14. Object.isExtensible(),
15. Object.PreventExtension(),

It is a method that returns an array of an object's own property names.

Example -

```
JavaScript
var emp = {
  name: "Alex",
  age: 27,
  salary: 10000,
};
var keys = Object.keys(emp);
console.log(keys); // output - [ 'name', 'age', 'salary' ]var
emp = {
  name: "Alex",
  age: 27,
  salary: 10000,
};
 var keys = Object.keys(emp);
 console.log(keys); // output - [ 'name', 'age', 'salary' ]
```

Object.values()

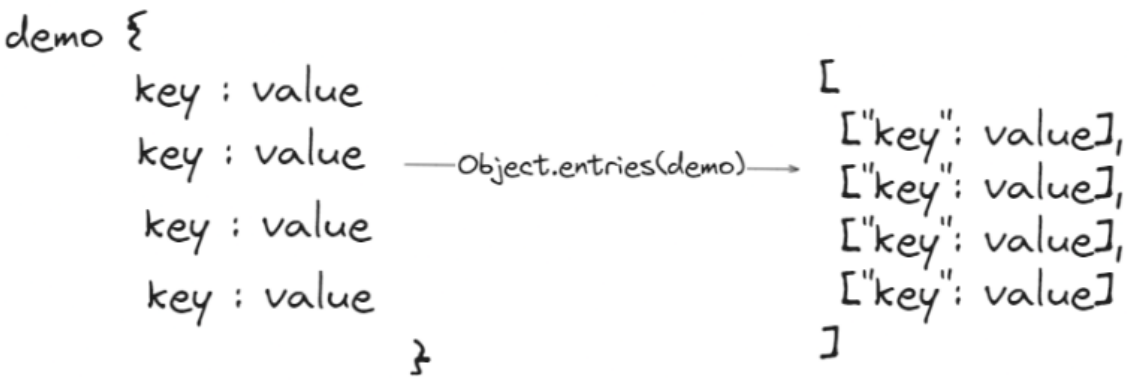
Object.values() is a method that returns an array of an object's own property values.

Example -

```
JavaScript
var emp = {
  name: "Alex",
  age: 27,
  salary: 10000,
};
var rec = Object.values(emp);
console.log(rec); // output - [ 'Alex', 27, 10000 ]
```

Object.entries()

This method is used to return an array of enumerable property [key, value] pairs of the object passed as the parameter.



Example -

```
JavaScript
const emp = {
  name: "alex",
  age: 24,
  salary: 100000,
};
```

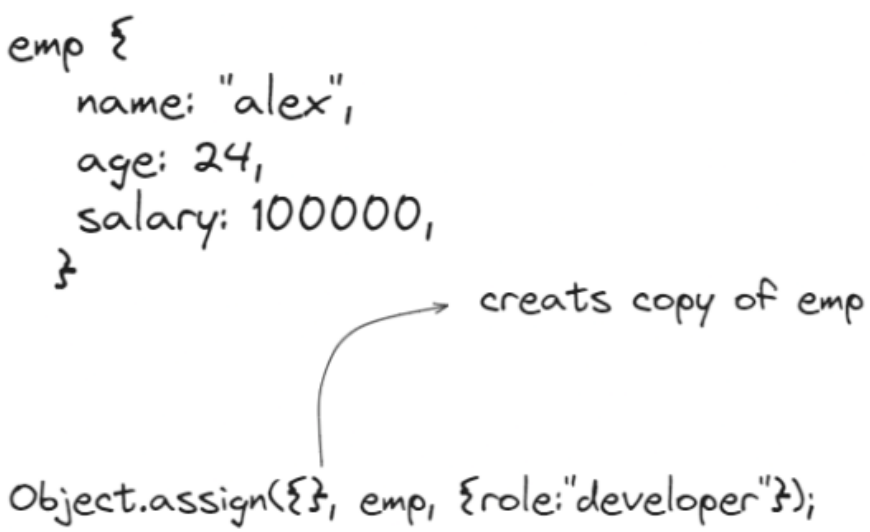
```
console.log(Object.entries(emp));
//output -  [ [ 'name', 'alex' ], [ 'age', 24 ], [ 'salary',
100000 ] ]

console.log(Object.entries(emp)[1]);
//  output - [ 'age', 24 ]

for (const key in emp) {
  console.log(`${key} : ${emp[key]}`);
}
// output -
// name: alex;
// age: 24;
// salary: 100000;
```

Object.assign()

The values and properties of one or more source objects are copied to a destination object using the Object.assign() function. It is used for cloning an object.



Example -

```
JavaScript
//syntax - Object.assign(target, source1, source2, ...)
let emp = {
  name: "Alex",
  age: 27,
  salary: 10000,
};
```

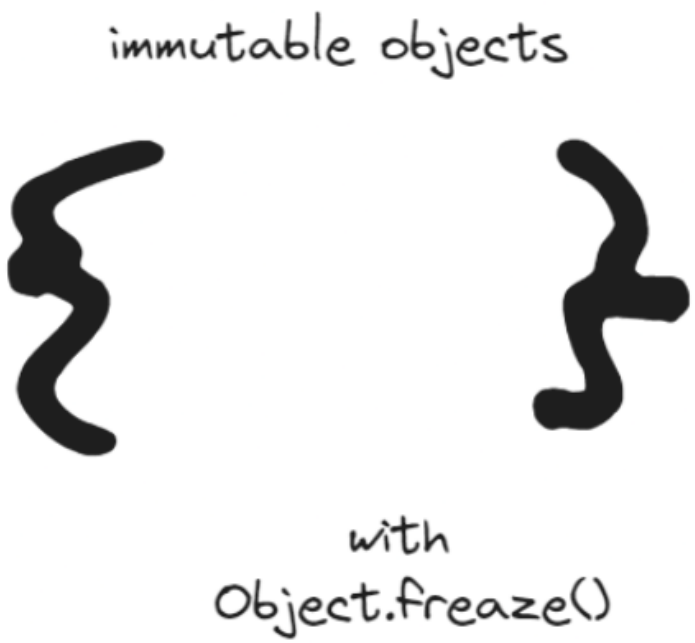
```
const newObject = Object.assign({}, emp,{role: “developer”});
console.log(newObject);

// output: { name: 'Alex', age: 27, salary: 10000 , role:
“developer” }
```

Object.freeze() & Object.isFrozen()

Object.freeze() - An object is frozen using this method.
Changing a frozen object is impossible. It prevents the addition and deletion of properties. Additionally, it prevents changes to property values from occurring unless an object is involved.

Object.isFrozen() - is to check whether the object is frozen.



Example -

```
JavaScript
var emp = {
  name: "Alex",
  age: 27,
  salary: 10000,
};
```

```
Object.freeze(emp);
emp.name = "John"; // not applied since its freeze
console.log(Object.isFrozen(emp)); // true
console.log(emp);
// output -- { name: 'Alex', age: 27, salary: 10000 }
```

Object.seal() & isSealed()

Object.seal() - It is a method identical to Object.freeze(). You cannot add or remove an object's properties, but you can edit the value of an existing property.
Object.isSealed() - is used to check whether the object is sealed. It returns true if the object is sealed and returns false if the object is not sealed.

Example -

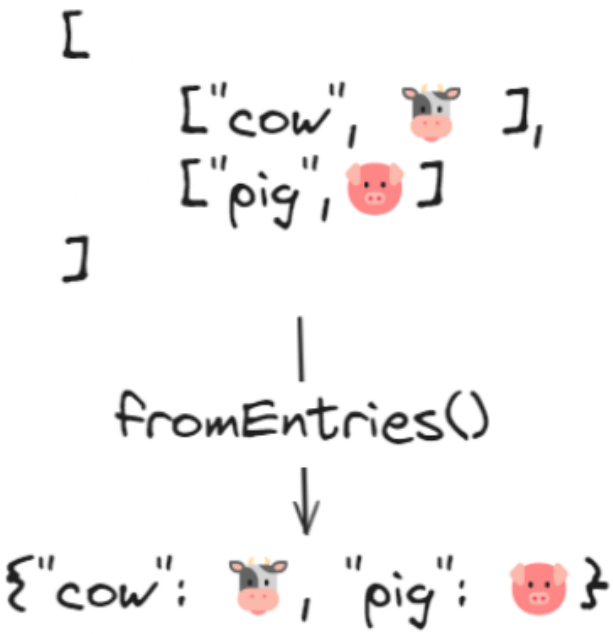
JavaScript

```
var emp = {
  name: "Alex",
  age: 27,
  salary: 10000,
};
emp.depart = "web dev"; // can be added since run before
Object.seal() method
Object.seal(emp);
emp.id = 1001; // not added since run after Object.seal()
method
console.log(Object.isSealed(emp));

console.log(emp);
// Output --
// true
// { name: 'Alex', age: 27, salary: 10000, depart: 'web dev' }
```

Object.fromEntries()

Object.fromEntires() creates an object from an iterable over entries. Each entry is a two-element Array with a property key and a property value.



Example -

```
JavaScript
const data = [
  ["id", "123445"],
  ["username", "dummy"],
  ["email", "dummy@gmail.com"],
];
const dataObj = Object.fromEntries(data);
console.log(dataObj);
// output -  { id: '123445', username: 'dummy', email:
'dummy@gmail.com' }
```

Object.create()

It is the method in JavaScript that is used to create a new object with an existing specified object and properties.

Example -

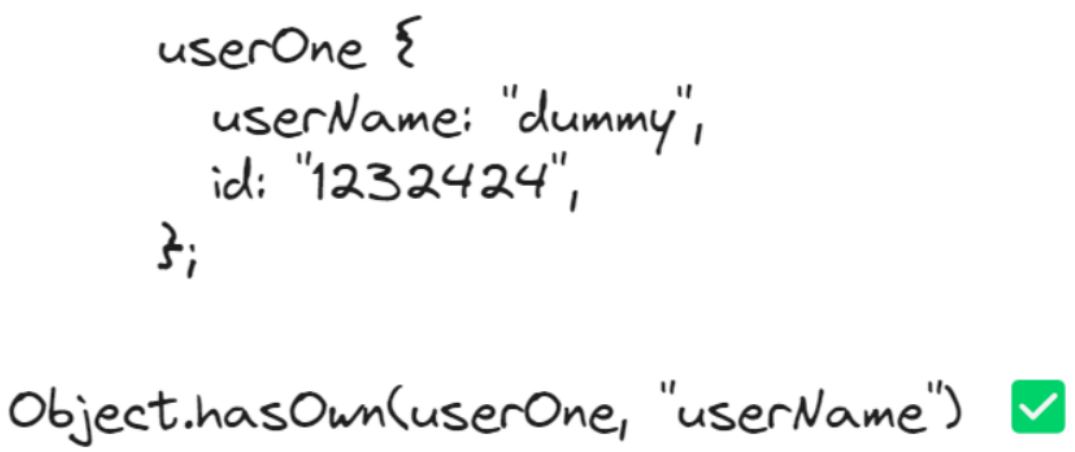
```
JavaScript
// syntax -
// Object.create(object)
const userOne = {
  userName: "dummy",
```

```
  id: "1232424",
};
const userTwo = Object.create(userOne);
userTwo.dept = "web dev";
userTwo.userName = "alex";
userTwo.id = "233444343";
console.log(userTwo);

// output - { dept: 'web dev', userName: 'alex', id:
'233444343' }
```

Object.hasOwn()

This method in JavaScript returns a boolean value indicating whether the object has the specified property as its own property. It returns true if present else false. It is intended as a replacement for Object.hasOwnProperty()



Example -

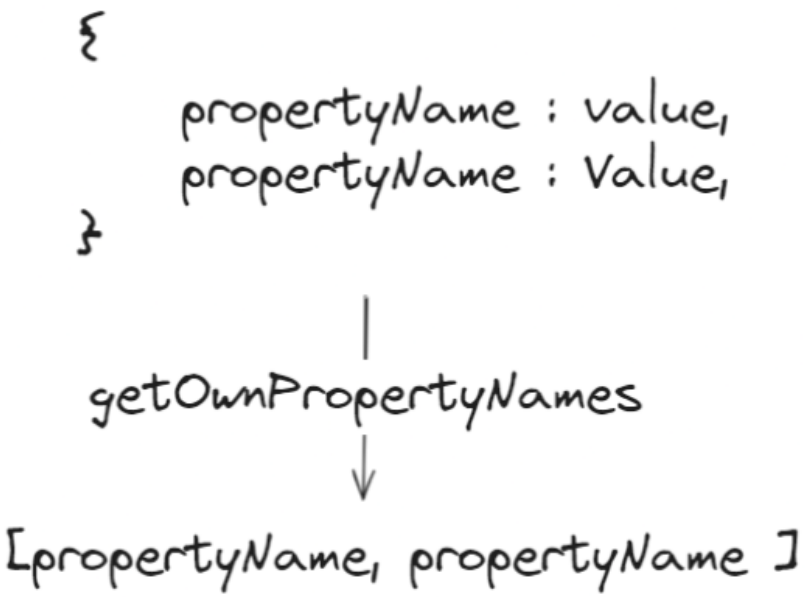
JavaScript

```
// syntax - Object.hasOwn(object, properties)
const userOne = {

  userName: "dummy",
  id: "1232424",
};
console.log(Object.hasOwn(userOne, "userName"));
// output - true
```

Object.getOwnPropertyNames()

It is the method in JavaScript that return an array of all properties found directly in a given object.



Example -

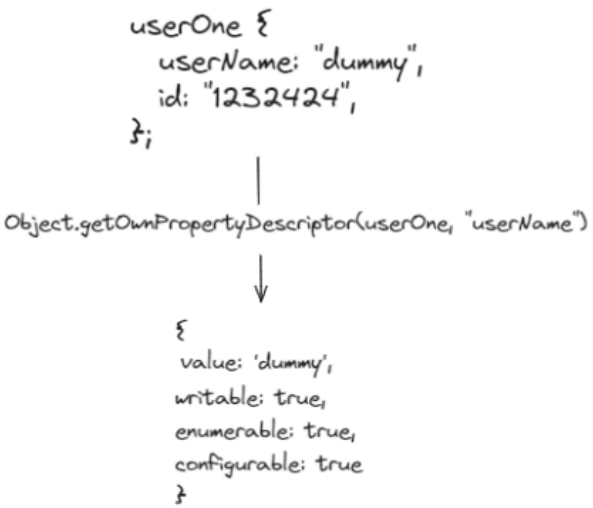
```JavaScript
/**
 ********** Object.getOwnPropertyNames() ************
 * syntax - Object.getOwnPropertyNames(obj)
 */
const userOne = {
  userName: "dummy",
  id: "1232424",

};

console.log(Object.getOwnPropertyNames(userOne));
// output - [ 'userName', 'id' ]
```

Object.getOwnPropertyDescriptor()

This method retrieves the descriptor of a single property on an object. It returns an object that contains the attributes of the specified property.



Example -

```
JavaScript
/**
 ********** Object.getOwnPropertyDescriptor() *******
 * syntax - Object.getOwnPropertyDescriptor(object, property)
 */
const userOne = {
  userName: "dummy",
  id: "1232424",
};

const des1 = Object.getOwnPropertyDescriptor(userOne,
"userName");

console.log(des1.configurable);
// output - true

console.log(des1.value);
// output - dummy
```

Object.getOwnPropertyDescriptors()

This method is used to retrieve descriptors of all properties on an object. It returns an object that contains descriptors for all own properties of the object.

Example -

JavaScript

```
/**
 ******* Object.getOwnPropertyDescriptors() ******
 * syntax - Object.getOwnPropertyDescriptors(obj)
 */
const userOne = {
  userName: "dummy",
  id: "1232424",
};

console.log(Object.getOwnPropertyDescriptors(userOne));
// output -
// {
```

```
//     userName: {
//       value: 'dummy',
//       writable: true,
//       enumerable: true,
//       configurable: true
//     },
//     id: {
//       value: '1232424',
//       writable: true,
//       enumerable: true,
//       configurable: true
//     }
//   }
console.log(Object.getOwnPropertyDescriptors(userOne).userName
.configurable);
// output - true
```

Object.defineProperty()

The Object.defineProperty() method is used to define or modify a single property on an object. It allows you to specify the attributes of the property individually.
Example -

JavaScript

```
const user = {}
Object.defineProperty(user, "name", {
  value: "Alex",
  writable: false, // false -> value can be change, true ->
//value can be change
});

console.log(user.name);
// Output - Alex
```

Object.defineProperties()

The Object.defineProperties() method is used to define or modify multiple properties on an object at once. It allows you to specify the attributes of multiple properties using a single descriptor object.
Example -

```javascript
/**
 ****** Object.defineProperties() *******
 *syntax- Object.defineProperties(object, properties)
 */
const data = {};

Object.defineProperties(data, {
  username: {
    value: "Alex",
    writable: true,
  },
  email: {
    value: "alex@gmail.com",
    writable: "true",
  },
});

console.log(data.username);// output - Alex
console.log(data.email); // output - alex@gmail.com
```

Object.isExtensible()

The Object.isExtensible() method in JavaScript is used to determine if an object is extensible i.e. whether it can have new properties added to it. Return boolean value, true if an object can have new properties else false if the object cant have new properties added to it.
Example

```javascript
/**
 ******* Object.isExtensible() *********
 *syntax -  Object.isExtensible(obj)
 */
const data = {
  username: "alex",
  email: "alex@gmail.com",
};
console.log(Object.isExtensible(data));
//output - true
```

Object.preventExtension()

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

Example

```javascript
/**
 ******* Object.preventExtension() *********
 *syntax -  Object.preventExtension(obj)
 */
const data = {
  username: "alex",
  email: "alex@gmail.com",
};
console.log(Object.isExtensible(data)); // output -  true

Object.preventExtensions(data);
console.log(Object.isExtensible(data)); // output -  false
```

Enumerating Properties

In JavaScript, the property of every object can be classified by three factors:

1. **Enumerable or non-enumerable**
2. **String or symbol**
3. **Own property or inherited property from the prototype chain.**

Enumerable properties are the objects with properties of the internal enumerable flag set to true which is the default property and allow us to loop over the object using for...in or Object.keys().

Enumerable properties can be set to false manually with the help of Object.defineProperty() or Object.defineProperties(). As a result, the object will not be possible to loop or iterate.

Example -

```javascript
const data = {};

Object.defineProperties(data, {
  userName: {
    value: "Alex",
    enumerable: true,
  },
  email: {
    value: "alex@gmail.com",
    enumerable: true,
```

```
  },
  phone: {
    value: "1414426267251",
    enumerable: false,
  },
});

for (const key in data) {
  console.log(`${key} : ${data[key]}`);
}
// output -
//userName : Alex
//email : alex@gmail.com
```

From the above example, only the user and email can be loop since their enumerable property is set to true, while phone property is not looped out as its enumerable property is set to false.

Interview Points

1. **Explain the purpose and usage of Object.defineProperty() in JavaScript. Provide an example where Object.defineProperty() is used to create a read-only property and clarify why this might be useful.**

Ans:
Object.defineProperty() is a method in JavaScript that allows fine-grained control over the behavior of object properties. It can be used to add a new property or modify an existing property on an object.
Example

JavaScript

```
"use strict";

const car = {};

// Using Object.defineProperty() to create a read-only
property 'make'
Object.defineProperty(car, 'make', {
  value: 'Toyota',
  writable: false,
  enumerable: true,
  configurable: false
});

console.log(car.make); // Output: Toyota
```

```
console.log(car.make); // Output: Toyota

// Attempting to modify the 'make' property will throw a
TypeError in strict mode
car.make = 'Honda'; // Throws TypeError: Cannot assign to
read-only property 'make' of object '#<Object>'
```

In this example, with strict mode enabled, attempting to modify the read-only property make will result in a TypeError, providing a clearer indication of the intention to prevent modification.

2. Explain difference between Object.assign() and Object.create() methods.

Ans:
The Object.assign method is used for copying the values of all enumerable properties from one or more source objects to a target object.

Example:

JavaScript
```
const target = {};
const source1 = { a: 1 };
const source2 = { b: 2 };

Object.assign(target, source1, source2);

console.log(target); // Output: { a: 1, b: 2 }
```

And **Object.create** method is used to create a new object with the specified prototype object and properties.

Example:

JavaScript
```
const sourceObject = { x: 1, y: 2 };
const newObject = Object.create(sourceObject);

console.log(newObject.x); // Output: 1
```

3. Write a JavaScript function that takes an object representing student scores and prints them in the following format:

JavaScript

```
function printStudentScores(scores) {
  // Your code here
}

// Example Usage:
const studentScores = {
  Alice: 85,
  Bob: 92,
  Carol: 78,
};

printStudentScores(studentScores);
```

```
Output //

Alice: 85
Bob: 92
Carol: 78
```

Answer

JavaScript

```
function printStudentScores(scores) {
  // Your code here
  for (const [key, value] of Object.entries(studentScores)) {
  console.log(`${key}: ${value}`);
}
}

// Example Usage:
const studentScores = {
  Alice: 85,
  Bob: 92,
  Carol: 78,
};
```

```
printStudentScores(studentScores);

Output //
Alice: 85
Bob: 92
Carol: 78
```