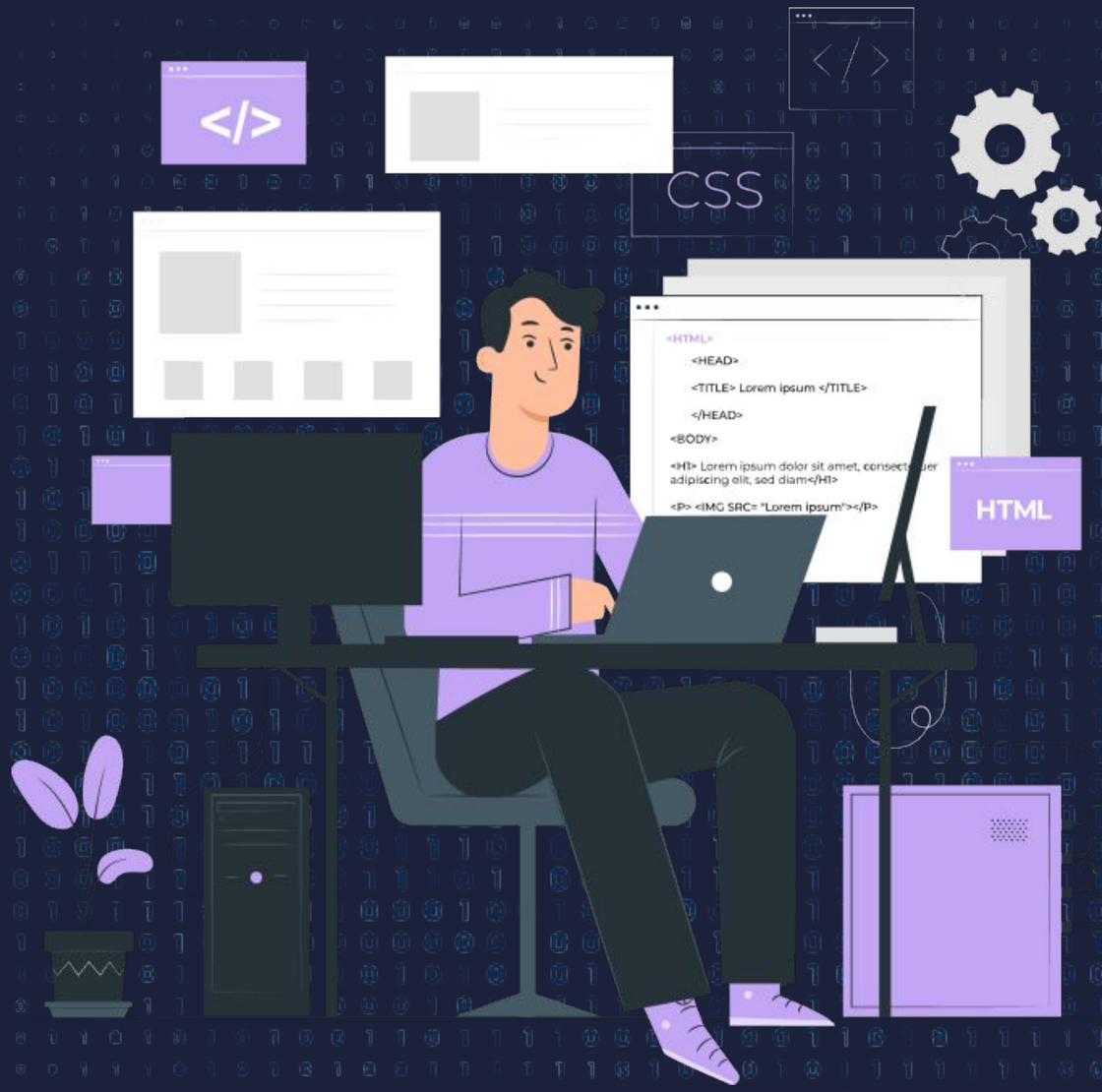
PW SKILLS

Object methods and enumerating properties



Lecture CheckList

- 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.

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

Object.keys()

```
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()

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

Object.entries()

```
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()

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

const newObject = Object.assign({}, emp);
console.log(newObject);
```

Object.freeze() & Object.isFrozen()

```
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.fromEntries()

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

Object.seal() & isSealed()

```
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.create()

```
// 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()

```
// syntax - Object.hasOwn(object, properties)
const userOne = {
  userName: "dummy",
  id: "1232424",
};
console.log(Object.hasOwn(userOne, "userName"));
// output - true
```

Object.getOwnPropertyNames()

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

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

Object.getOwnPropertyDescriptors()

```
/**
 ********** 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.defineProperty()

```
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.getOwnPropertyDescriptor()

```
/**
 ******* 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.isExtensible()

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

Object.preventExtension()

```
/**
 ******* 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

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 property 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 -

```
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 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.


SKILLS
THANK YOU