Day 40

"Web Development + Security"

Classes & Objects - Object Oriented Programming in JavaScript:

What is Object-Oriented Programming (OOP)?

OOP is a programming style that organizes code into objects — reusable, logical units that hold data (properties) and behaviors (methods).

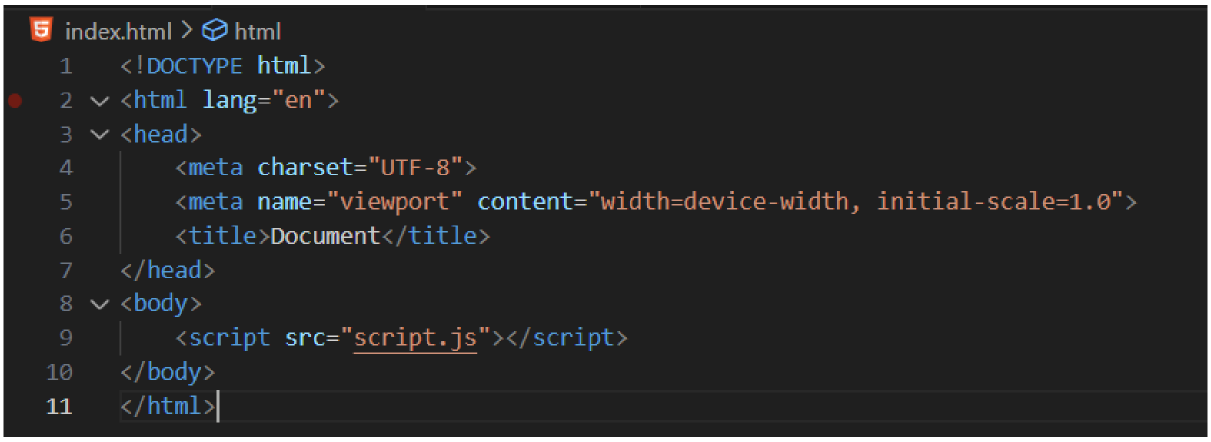
In short:
Object = data + actions
Example: A car object can have properties like color, brand, and methods like start() or stop().

What is class?

A class is a blueprint for creating objects.

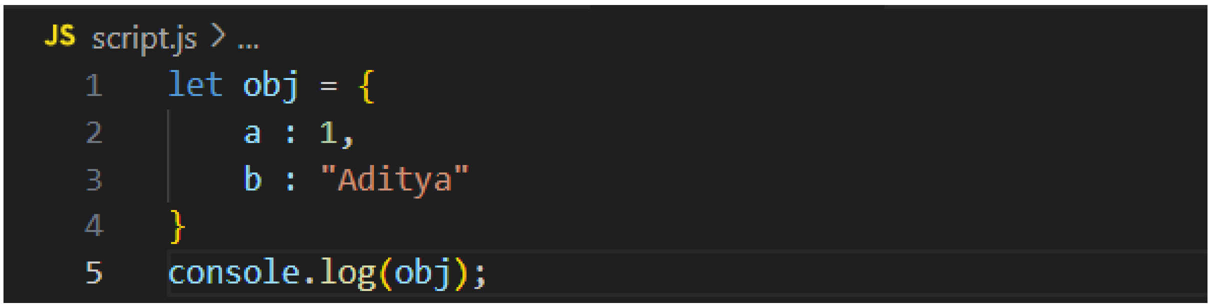
A basic example of creating an object:

Index.html:



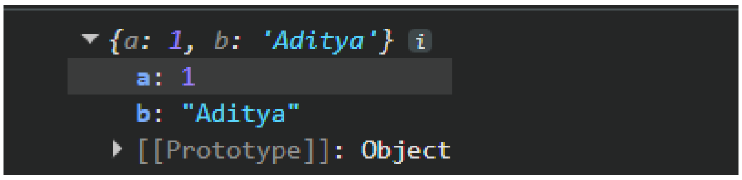
```
index.html > html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script src="script.js"></script>
</body>
</html>
```

Script.js:



```
script.js > ...
let obj = {
    a : 1,
    b : "Aditya"
}
console.log(obj);
```

Console:



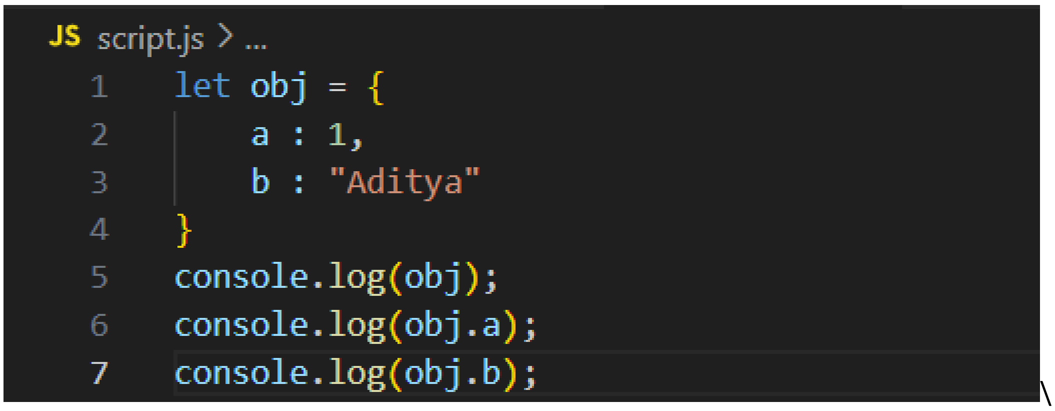
```
▾ {a: 1, b: 'Aditya'} i
    a: 1
    b: "Aditya"
  ▸ [[Prototype]]: Object
```

This is a JavaScript object — {a: 1, b: 'Aditya'} — which stores data in key–value pairs. Here, the key a has the value 1, and b has the value 'Aditya'. The part [[Prototype]]: Object is shown in the browser console (like Chrome DevTools) and means this object automatically inherits properties and methods from JavaScript's built-in Object prototype (for example, methods like toString() or hasOwnProperty()).

Accessing the object: we will be using the "." As obj.a and obj.b as shown below.

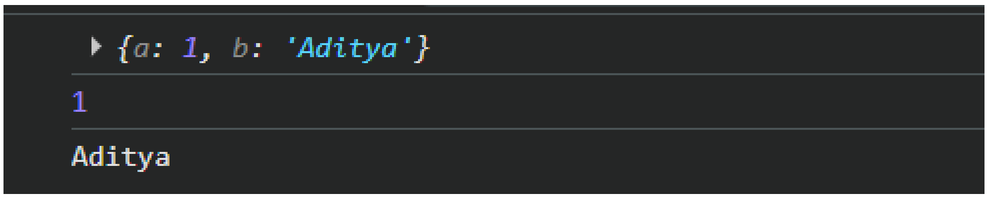
Index.html: same as above

Script.js:



```
JS script.js > ...
let obj = {
    a : 1,
    b : "Aditya"
}
console.log(obj);
console.log(obj.a);
console.log(obj.b);
```

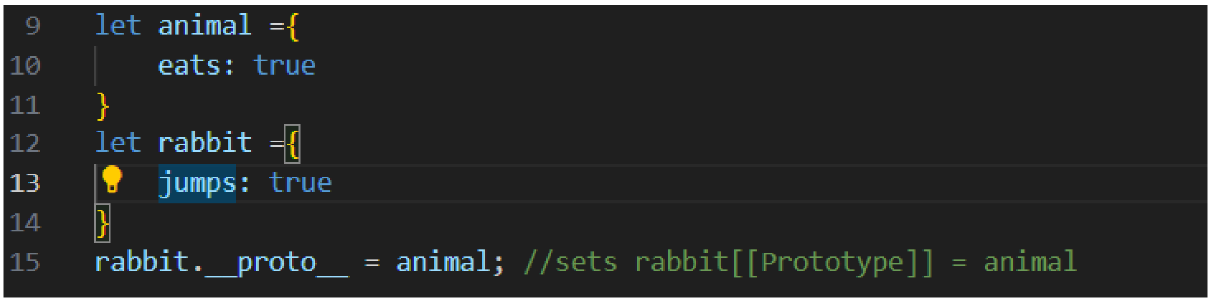
Console:



```
▸ {a: 1, b: 'Aditya'}
1
Aditya
```

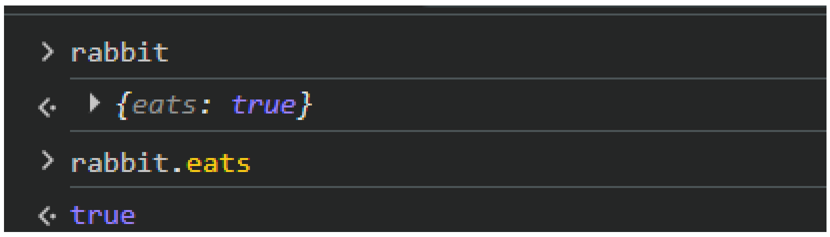
Also, we can use the prototype to assign the properties of one object to the another:

Script.js:



```
let animal ={
    eats: true
}
let rabbit ={
    jumps: true
}
rabbit.__proto__ = animal; //sets rabbit[[Prototype]] = animal
```

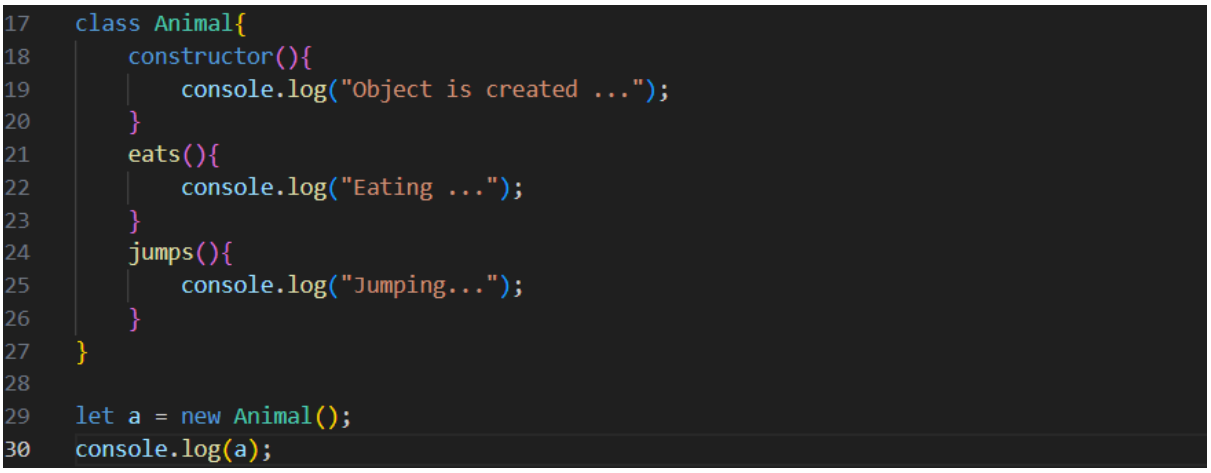
Console:



```
> rabbit
<· ▸ {eats: true}
> rabbit.eats
<· true
```

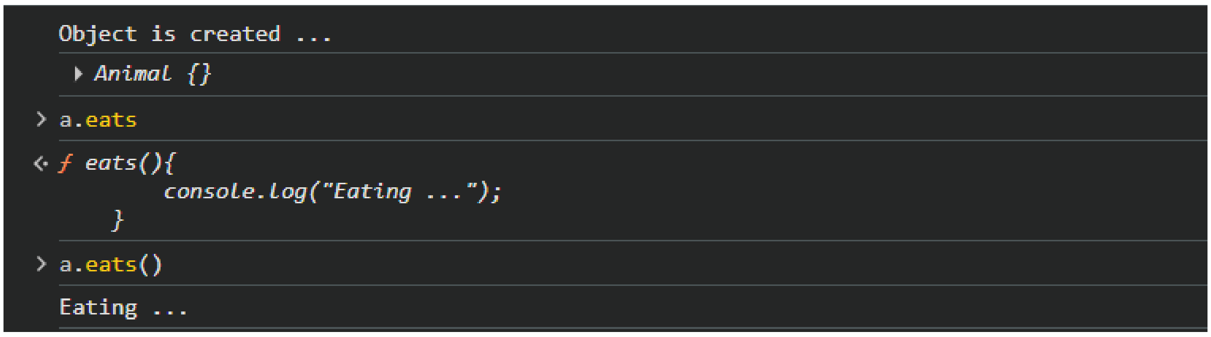
Now, creating the object using the class:

Script.js:



```javascript
class Animal{
    constructor(){
        console.log("Object is created ...");
    }
    eats(){
        console.log("Eating ...");
    }
    jumps(){
        console.log("Jumping...");
    }
}

let a = new Animal();
console.log(a);
```

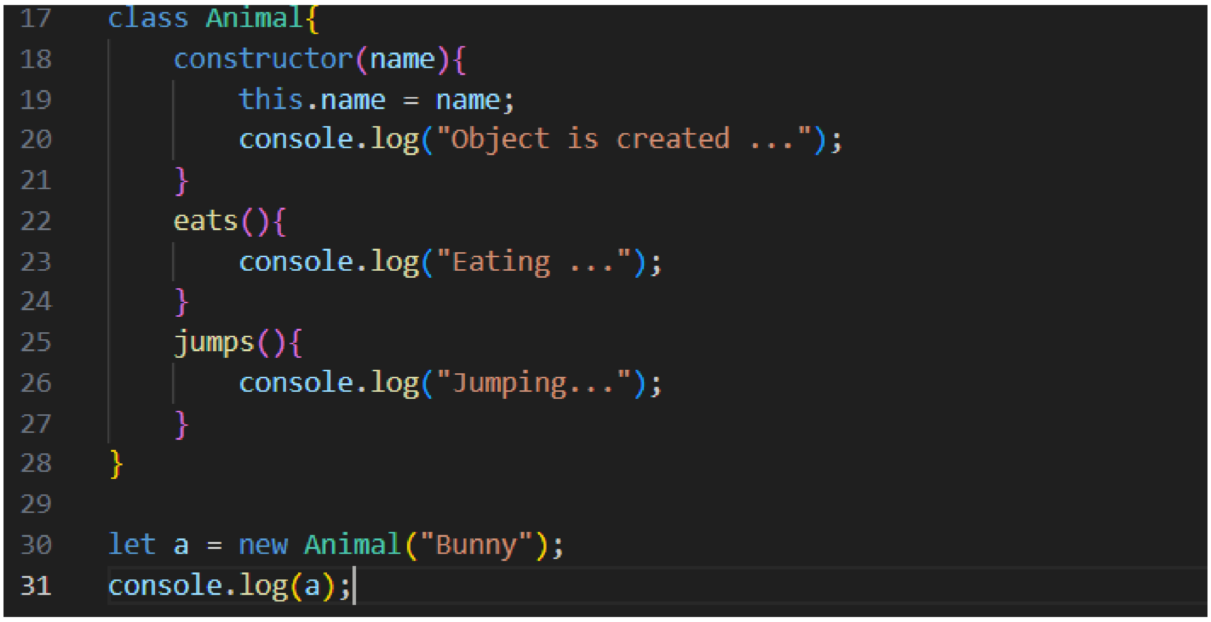
Console:



```
  Object is created ...
   ▸ Animal {}
> a.eats
<· ƒ eats(){
          console.log("Eating ...");
      }
> a.eats()
  Eating ...
```

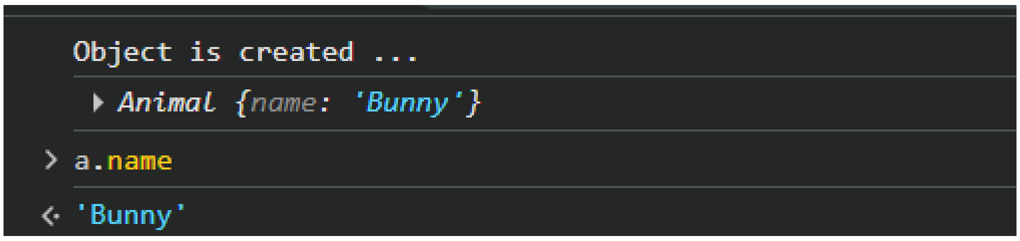
Now, creating properties in an object: we will use “this” keyword.

Script.js:



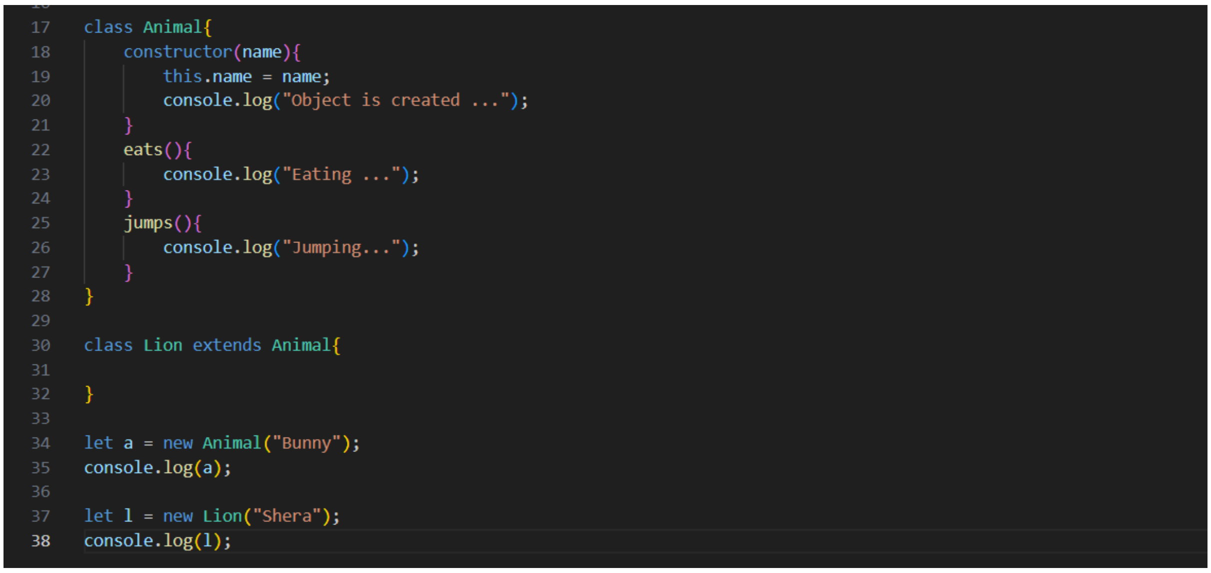
```javascript
class Animal{
    constructor(name){
        this.name = name;
        console.log("Object is created ...");
    }
    eats(){
        console.log("Eating ...");
    }
    jumps(){
        console.log("Jumping...");
    }
}

let a = new Animal("Bunny");
console.log(a);
```

Console:

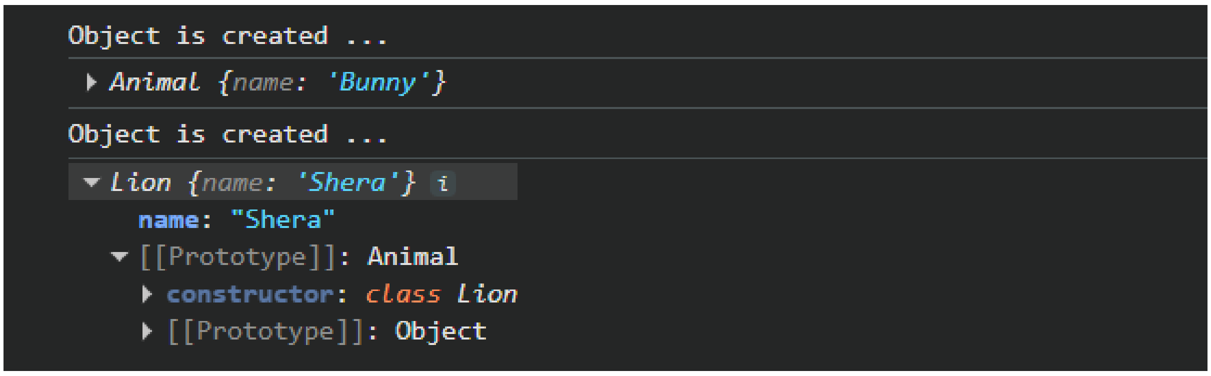

```
Object is created ...
▸ Animal {name: 'Bunny'}
> a.name
<· 'Bunny'
```

Now, using the “extend” keyword to pass the properties of Animal to the other class:

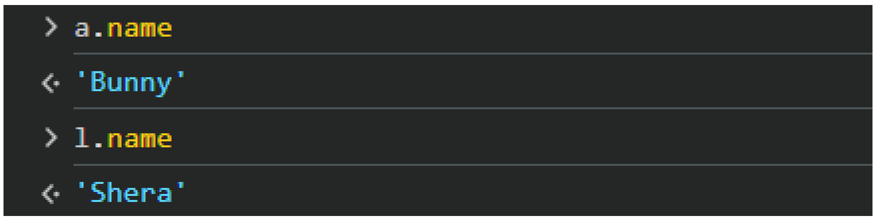
Script.js:


```
class Animal{
    constructor(name){
        this.name = name;
        console.log("Object is created ...");
    }
    eats(){
        console.log("Eating ...");
    }
    jumps(){
        console.log("Jumping...");
    }
}

class Lion extends Animal{

}

let a = new Animal("Bunny");
console.log(a);

let l = new Lion("Shera");
console.log(l);
```

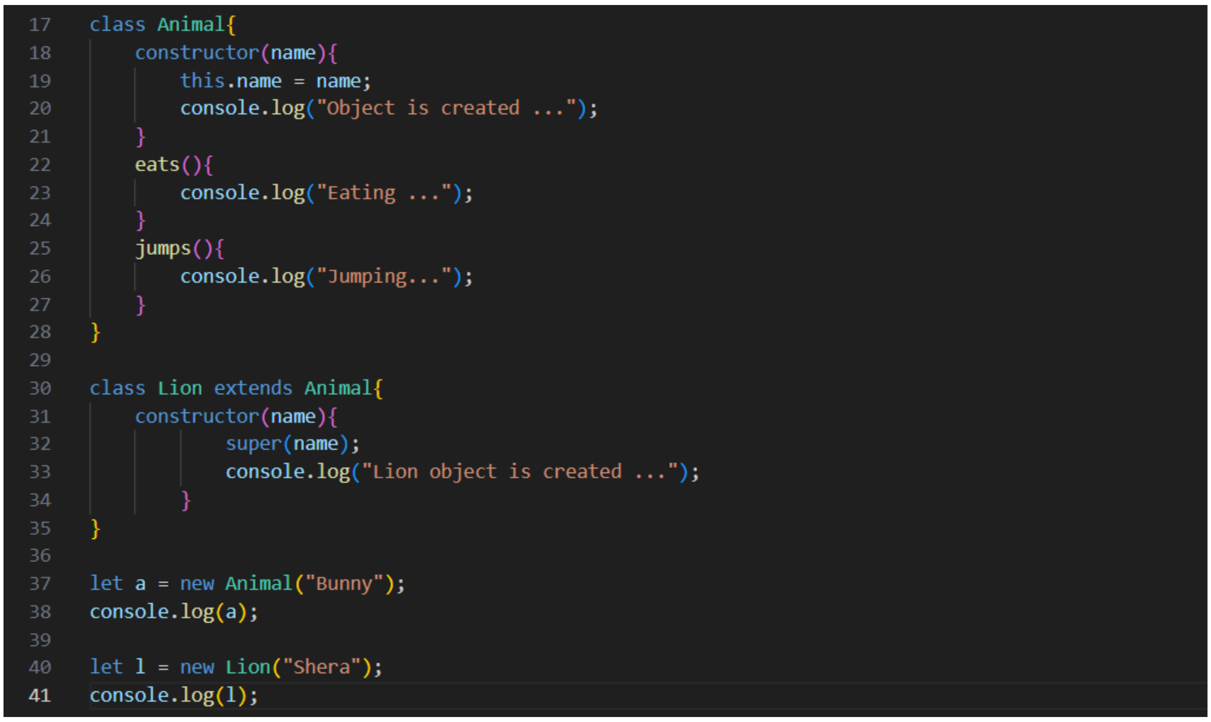
Console: clearly, prototype of Lion says it belongs to Animal.


```
Object is created ...
▸ Animal {name: 'Bunny'}
Object is created ...
▾ Lion {name: 'Shera'} i
    name: "Shera"
  ▾ [[Prototype]]: Animal
    ▸ constructor: class Lion
    ▸ [[Prototype]]: Object
```

For above:

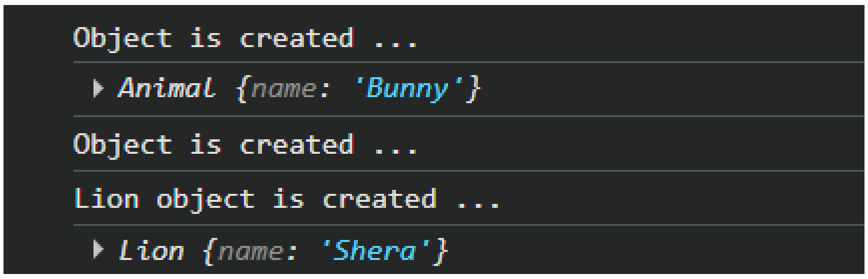

```
> a.name
<· 'Bunny'
> l.name
<· 'Shera'
```

Now, giving properties to the Lion class: we will use super() keyword to achieve so.



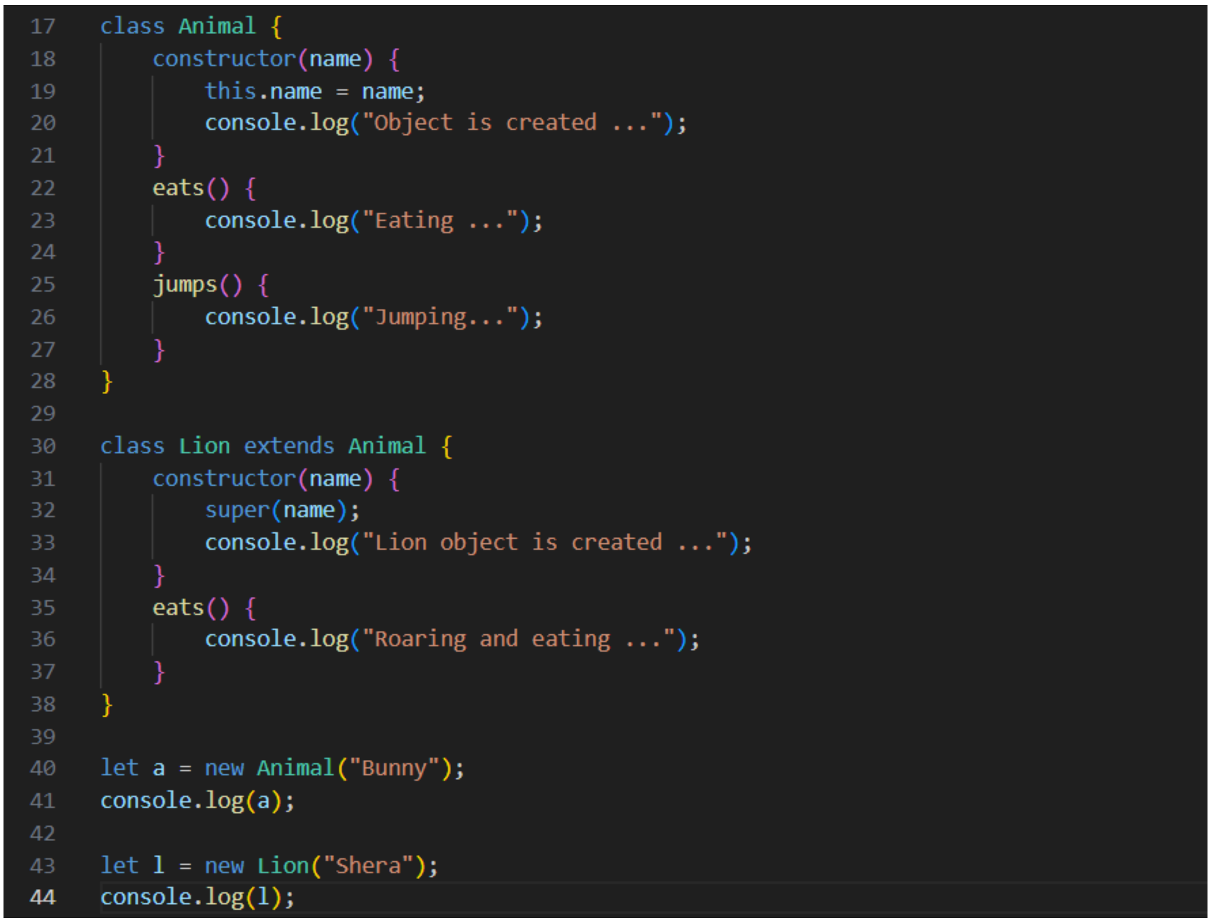
```javascript
class Animal{
    constructor(name){
        this.name = name;
        console.log("Object is created ...");
    }
    eats(){
        console.log("Eating ...");
    }
    jumps(){
        console.log("Jumping...");
    }
}

class Lion extends Animal{
    constructor(name){
            super(name);
            console.log("Lion object is created ...");
        }
}

let a = new Animal("Bunny");
console.log(a);

let l = new Lion("Shera");
console.log(l);
```

Console:



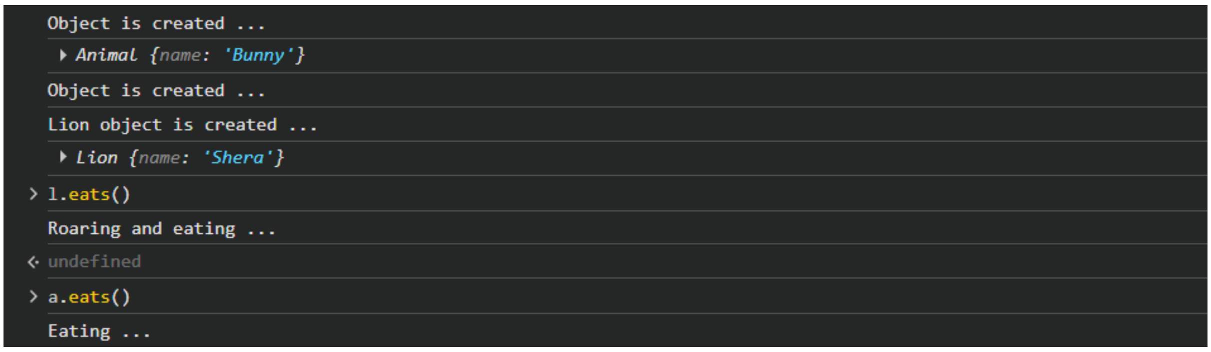
```
Object is created ...
 ▸ Animal {name: 'Bunny'}
Object is created ...
Lion object is created ...
 ▸ Lion {name: 'Shera'}
```

Now, overriding:



```javascript
class Animal {
    constructor(name) {
        this.name = name;
        console.log("Object is created ...");
    }
    eats() {
        console.log("Eating ...");
    }
    jumps() {
        console.log("Jumping...");
    }
}

class Lion extends Animal {
    constructor(name) {
        super(name);
        console.log("Lion object is created ...");
    }
    eats() {
        console.log("Roaring and eating ...");
    }
}

let a = new Animal("Bunny");
console.log(a);

let l = new Lion("Shera");
console.log(l);
```

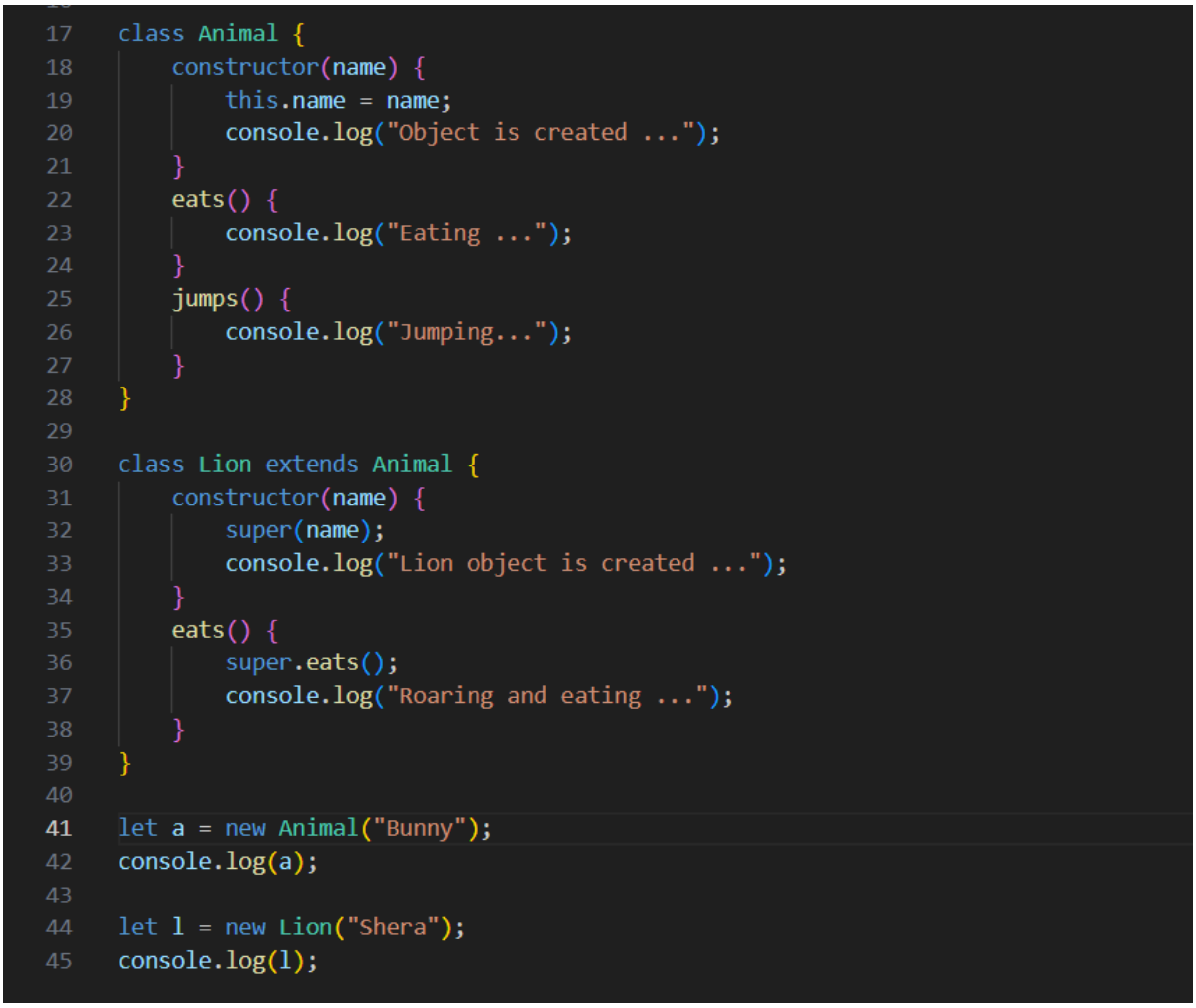
Console:



```
  Object is created ...
   ▸ Animal {name: 'Bunny'}
  Object is created ...
  Lion object is created ...
   ▸ Lion {name: 'Shera'}
> l.eats()
  Roaring and eating ...
< undefined
> a.eats()
  Eating ...
```

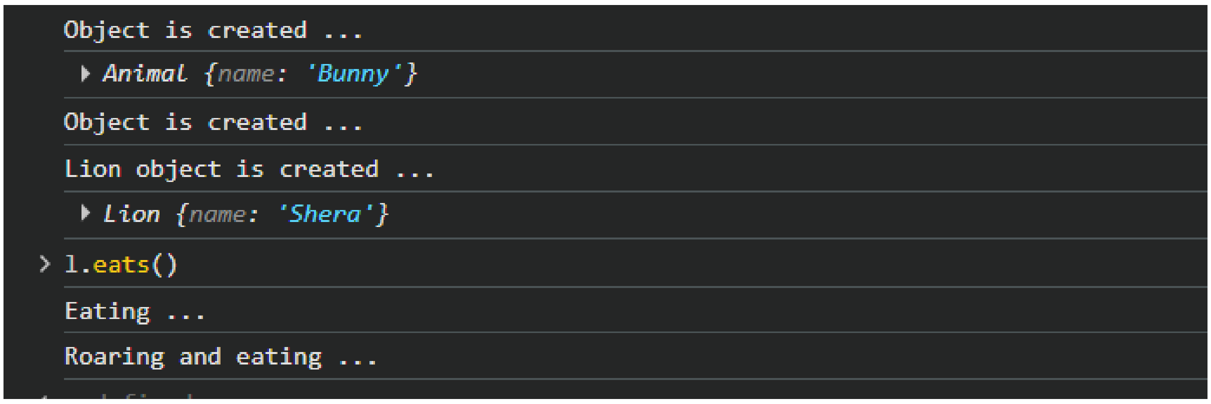
Now, what if we want the eats() of the parent class to run too? We will use the super keyword along with the method name:

Script.js:



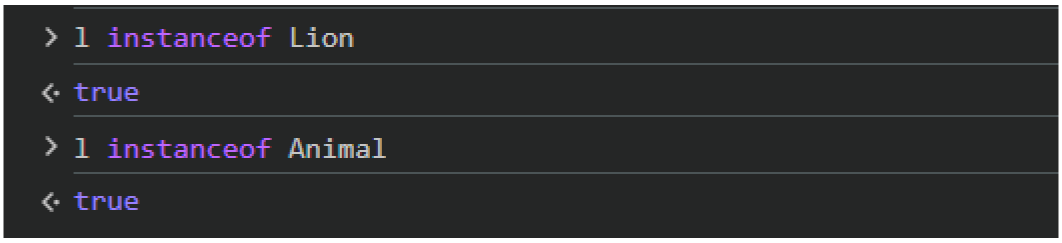
```
class Animal {
    constructor(name) {
        this.name = name;
        console.log("Object is created ...");
    }
    eats() {
        console.log("Eating ...");
    }
    jumps() {
        console.log("Jumping...");
    }
}

class Lion extends Animal {
    constructor(name) {
        super(name);
        console.log("Lion object is created ...");
    }
    eats() {
        super.eats();
        console.log("Roaring and eating ...");
    }
}

let a = new Animal("Bunny");
console.log(a);

let l = new Lion("Shera");
console.log(l);
```

Console:



```
Object is created ...
▸ Animal {name: 'Bunny'}
Object is created ...
Lion object is created ...
▸ Lion {name: 'Shera'}
> l.eats()
Eating ...
Roaring and eating ...
```

Now, we have “instanceof”:



```
> l instanceof Lion
<· true
> l instanceof Animal
<· true
```

--The End--