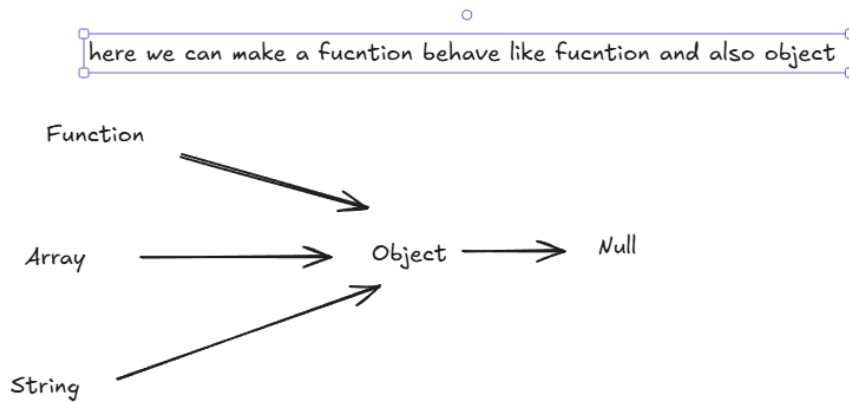


OOPs in JS

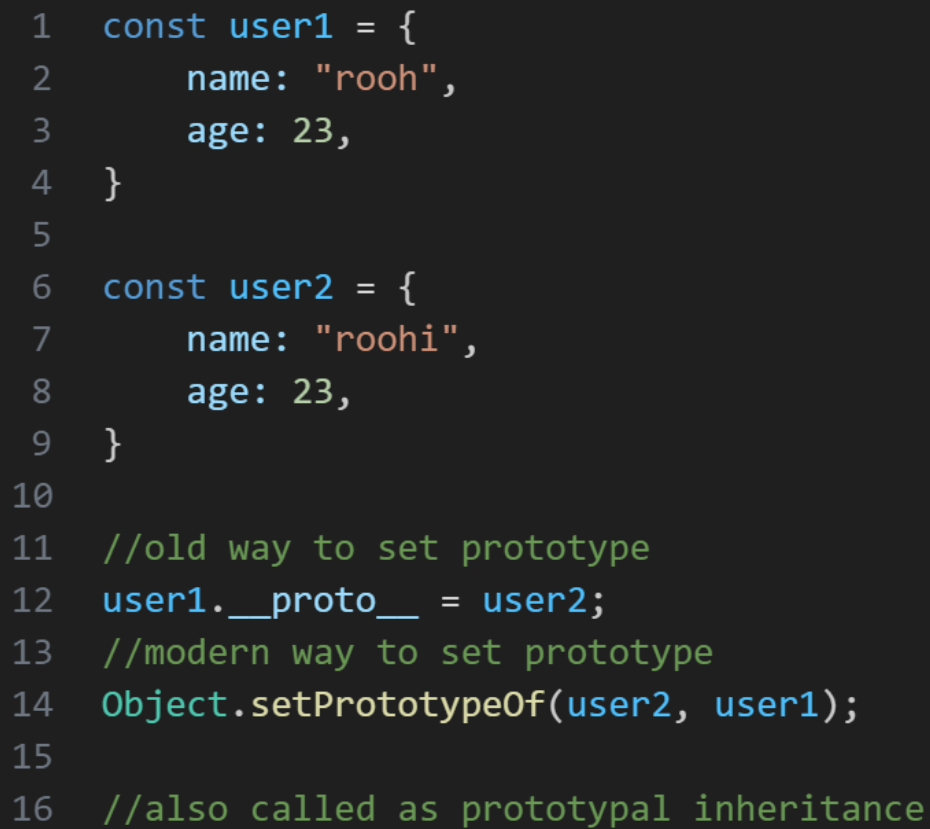
Prototypal behavior of js :

It doesn't give up like if javascript can't find something so it goes to the upper layer and then upper layer like parents grand parents and so on until it finds null. We also can call this inheritance prototypal inheritance.



Prototypal inheritance:

We can also access the key properties of other objects using the `__proto__` old syntax and in new syntax `setPrototypeOf` this is called prototypal inheritance



```
1  const user1 = {
2      name: "rooh",
3      age: 23,
4  }
5
6  const user2 = {
7      name: "roohi",
8      age: 23,
9  }
10
11  //old way to set prototype
12  user1.__proto__ = user2;
13  //modern way to set prototype
14  Object.setPrototypeOf(user2, user1);
15
16  //also called as prototypal inheritance
```



```
1  const userName = "rooh      ";
2
3  String.prototype.trueLength = function(){
4      console.log(`${this}`);
5      console.log(`true length is: ${this.trim().length}`);
6  }
7
8  userName.trueLength();
9
10
11
12  "rooohullah khan    ".trueLength()
```



```
1  function multiplyBy5(num){  
2      return num * 5;  
3  }  
4  
5  multiplyBy5.power= 5;  
6  
7  console.log(multiplyBy5(10));  
8  console.log(multiplyBy5.power);
```

In this code we are using the dot with the function so what does it mean it means that we can make the function act like an object as well.

Prototype in js:

In Js every function has a special object that is called prototype.

In simple words it is an object that can hold shared methods and properties.

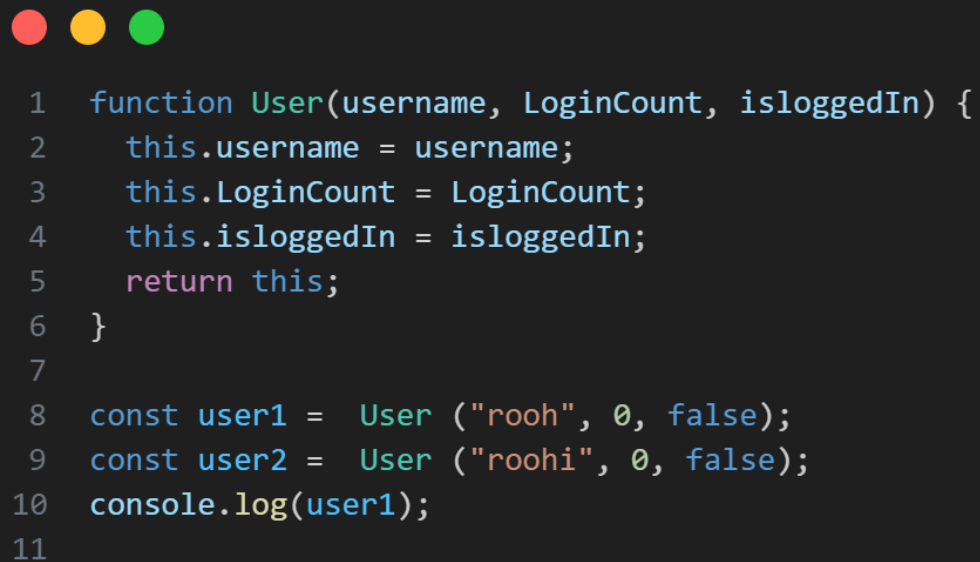
Constructor function:

Constructor function gives us a new instance a new copy use to create and initialize

Simple words a function becomes a constructor when it is called with the new

Create new object set this to that object link the prototype return the object

Magic of new key word :



```
1  function User(username, LoginCount, isloggedIn) {  
2    this.username = username;  
3    this.LoginCount = LoginCount;  
4    this.isloggedIn = isloggedIn;  
5    return this;  
6  }  
7  
8  const user1 = User ("rooh", 0, false);  
9  const user2 = User ("roohi", 0, false);  
10 console.log(user1);  
11
```

Here we are calling to the normal function like this will point to the global object in node so user 1 and user 2 to point to the same thing the global object



```
1  function User(username, LoginCount, isloggedIn) {  
2    this.username = username;  
3    this.LoginCount = LoginCount;  
4    this.isloggedIn = isloggedIn;  
5    return this;  
6  }  
7  
8  const user1 = new User ("rooh", 0, false);  
9  const user2 = User ("roohi", 0, false);  
10 console.log(user1);
```

Here when we are using the new so we are creating a new fresh object where this refer to that object and we are getting separate instances.

Here we have abstraction to like user don't know how the new is working internally so it is abstraction

Also encapsulation is there like we are encapsulation every thing in user 1 and the user don't know where it is getting stored



```
1  function createUser(name, price){
2      this.name = name;
3      this.price = price;
4  }
5
6  createUser.prototype.incrementPrice = function(){
7      this.price ++;
8  }
9  createUser.prototype.printPrice = function(){
10     console.log(`Price of ${this.name} is ${this.price}`);
11 }
12
13 const chai = new createUser("chai", 10);
14 const tea = createUser("tea", 250);
15
16
17
18 chai.printPrice();
```