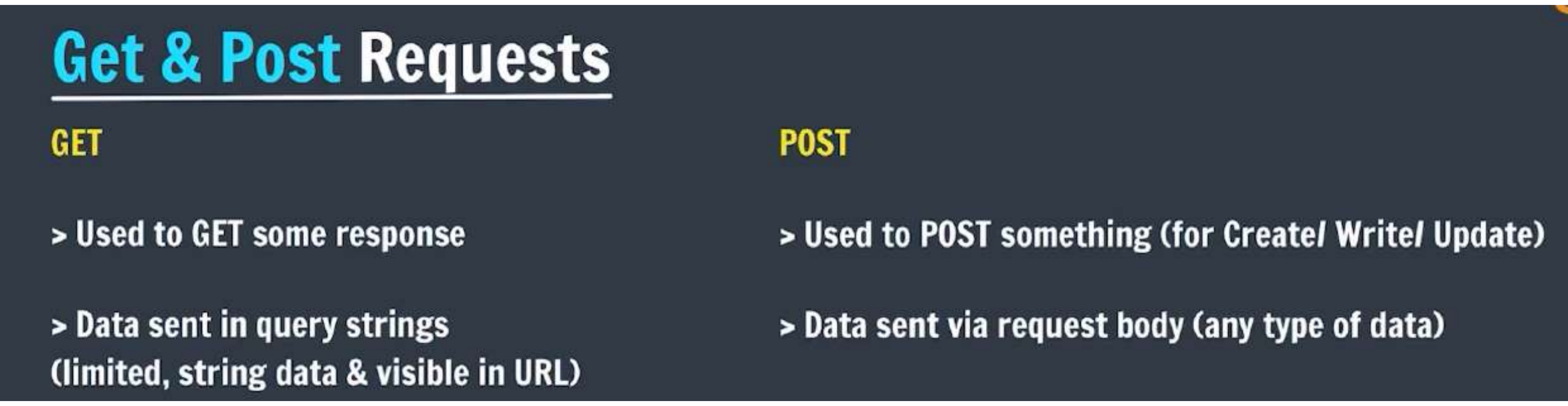
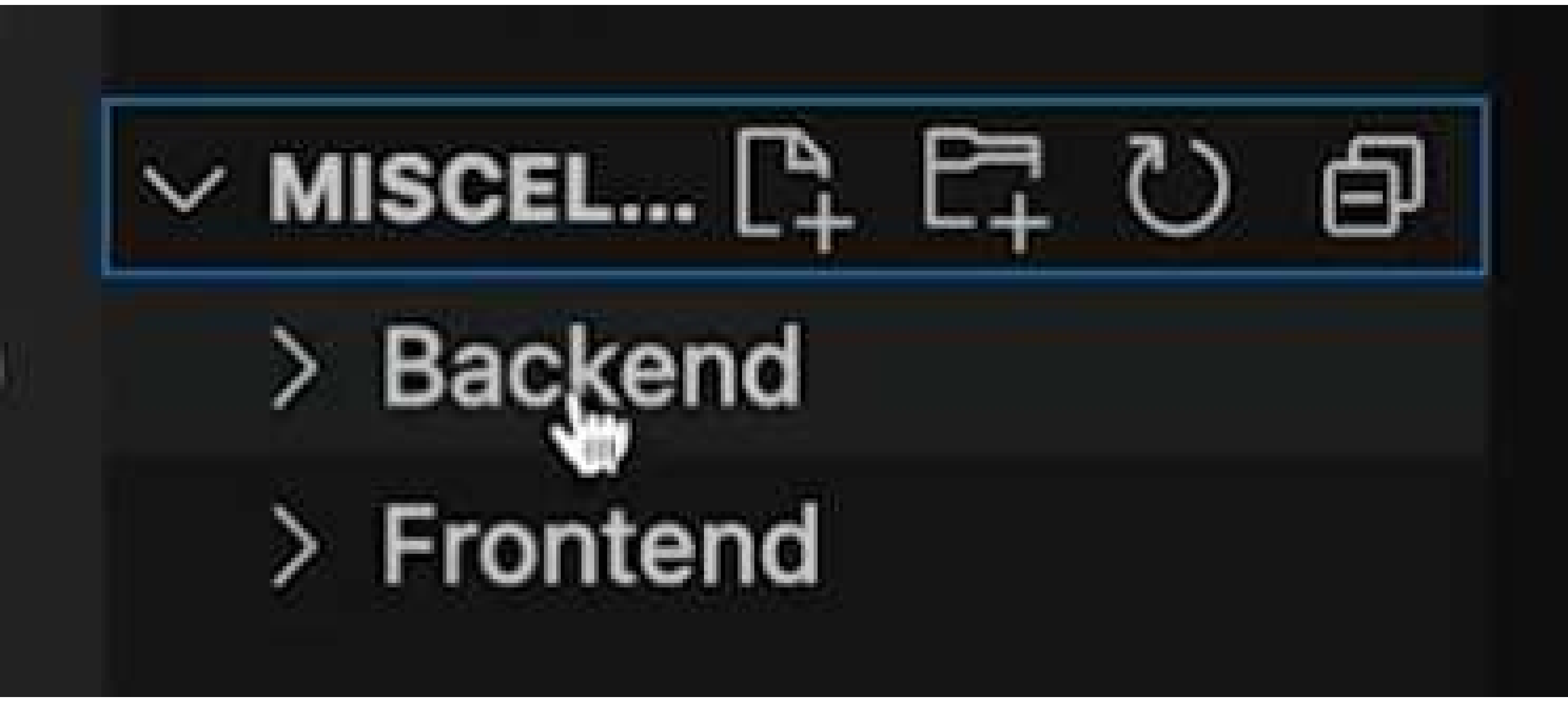
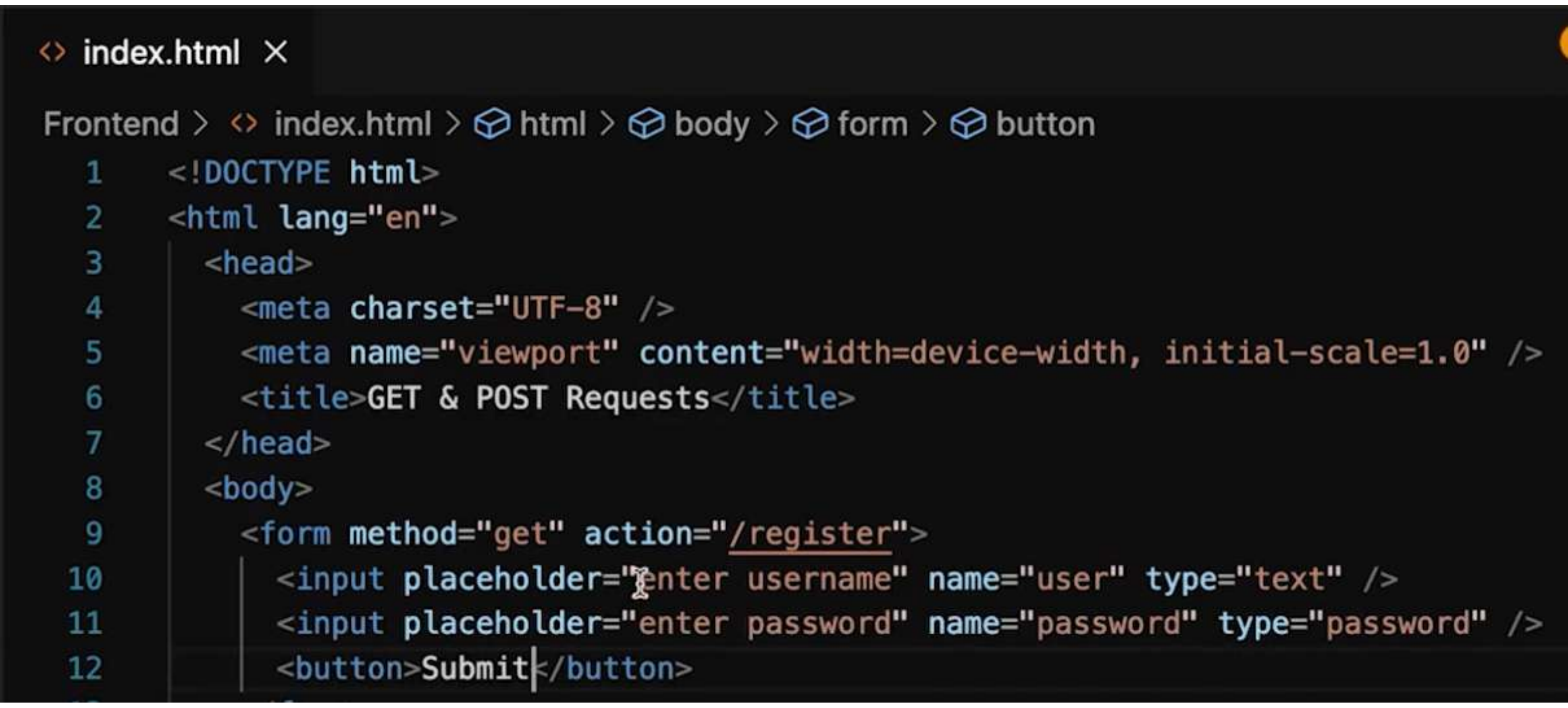


Get & Post Requests

GET	POST
> Used to GET some response	> Used to POST something (for Create/ Write/ Update)
> Data sent in query strings (limited, string data & visible in URL)	> Data sent via request body (any type of data)

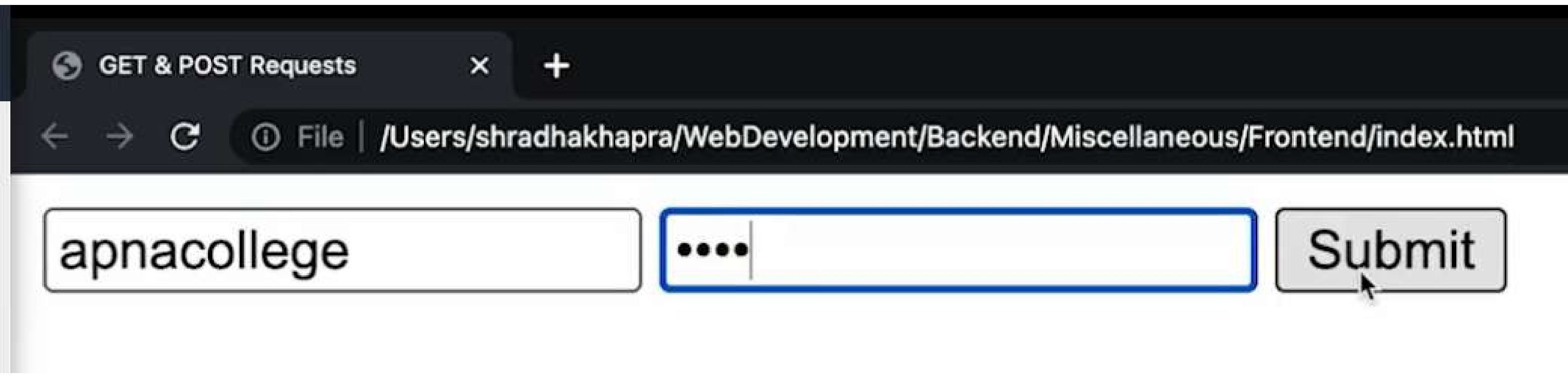

MISCEL...
Backend
Frontend

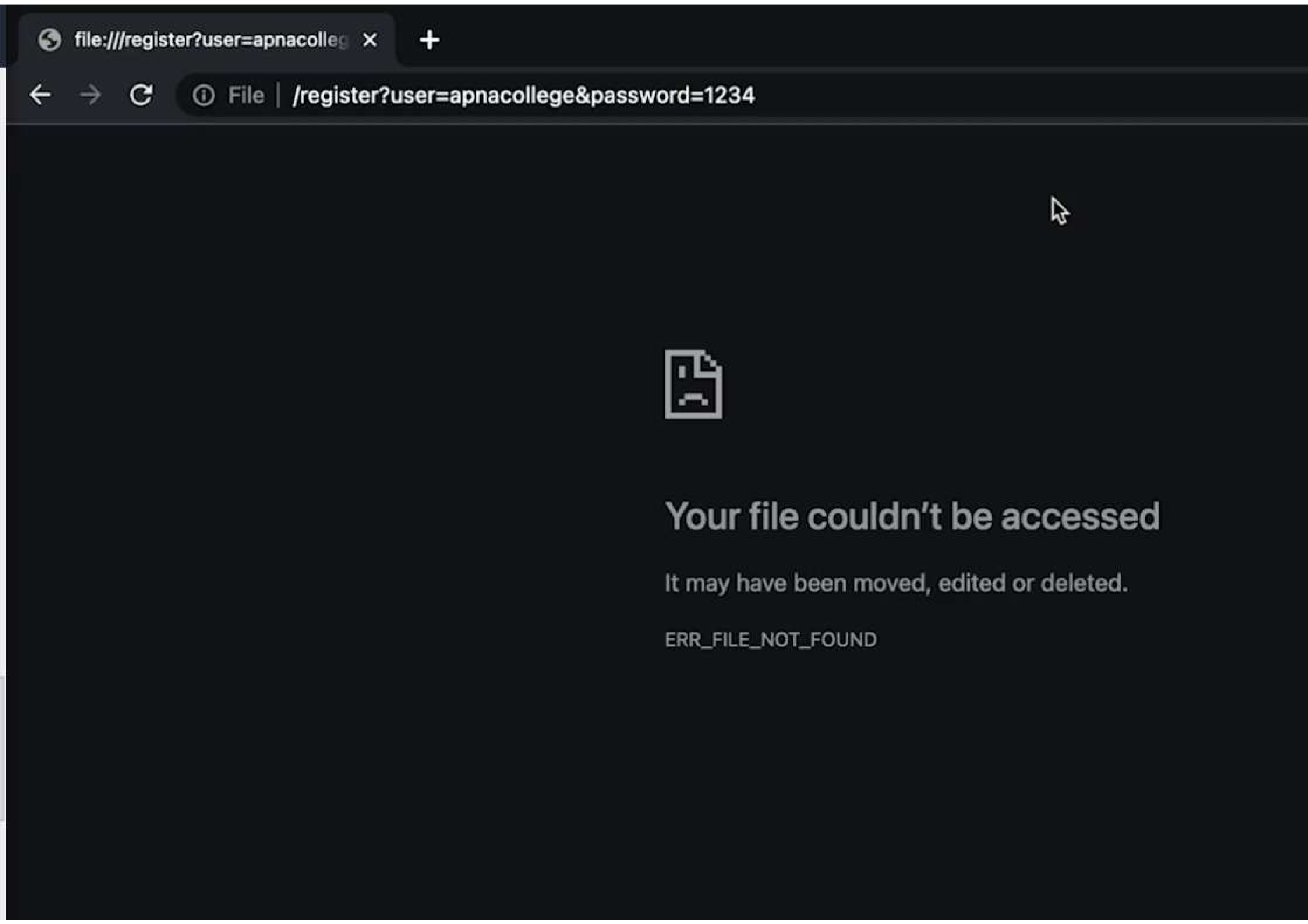


index.html ×

Frontend > index.html > html > body > form > button

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>GET & POST Requests</title>
  </head>
  <body>
    <form method="get" action="/register">
      <input placeholder="enter username" name="user" type="text" />
      <input placeholder="enter password" name="password" type="password" />
      <button>Submit</button>
```

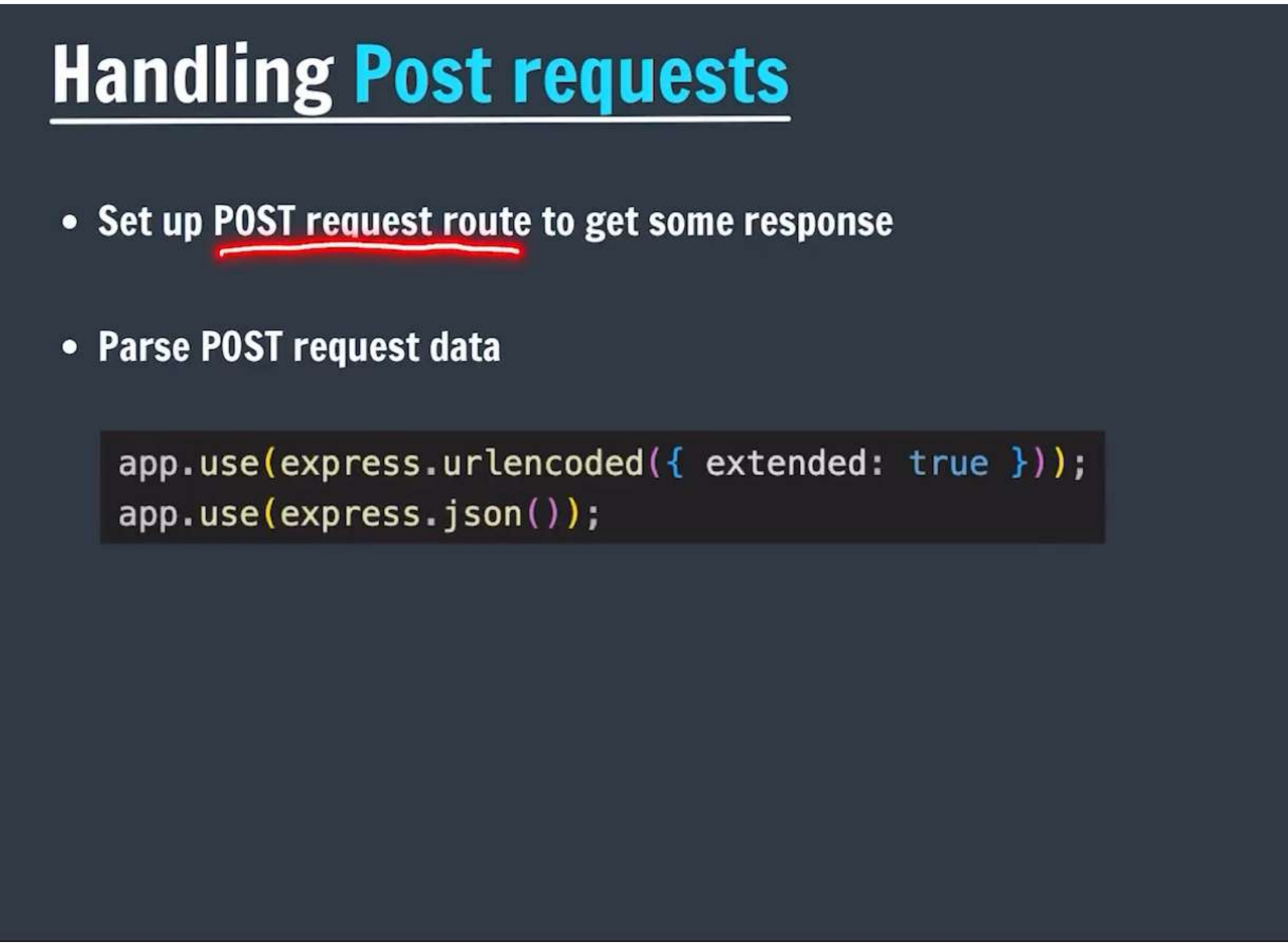

GET & POST Requests
File
/Users/shradhakhapra/WebDevelopment/Backend/Miscellaneous/Frontend/index.html
apnacollege
••••
Submit


file:///register?user=apnacolleg
File
/register?user=apnacollege&password=1234

Your file couldn't be accessed

It may have been moved, edited or deleted.

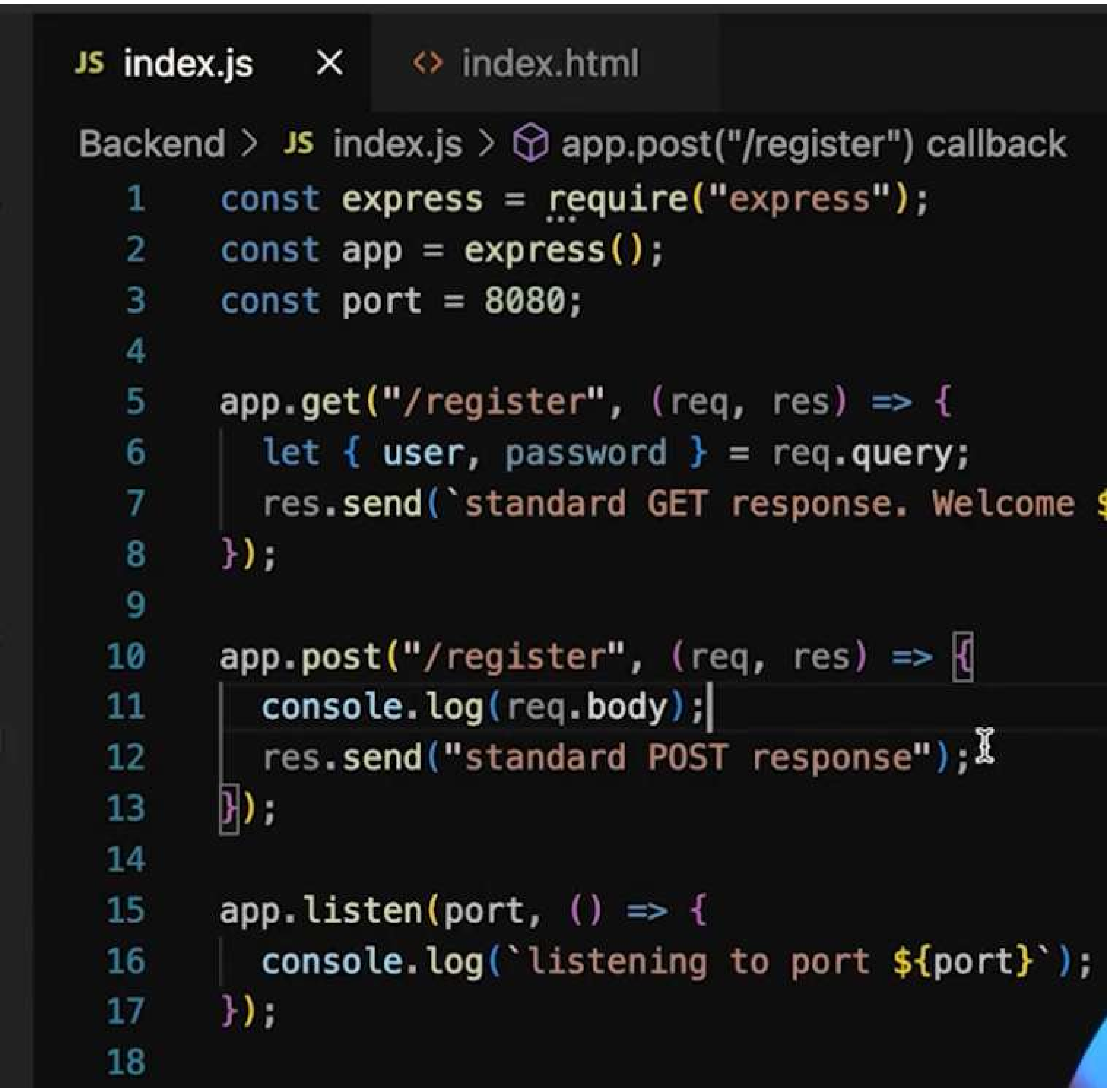
ERR_FILE_NOT_FOUND



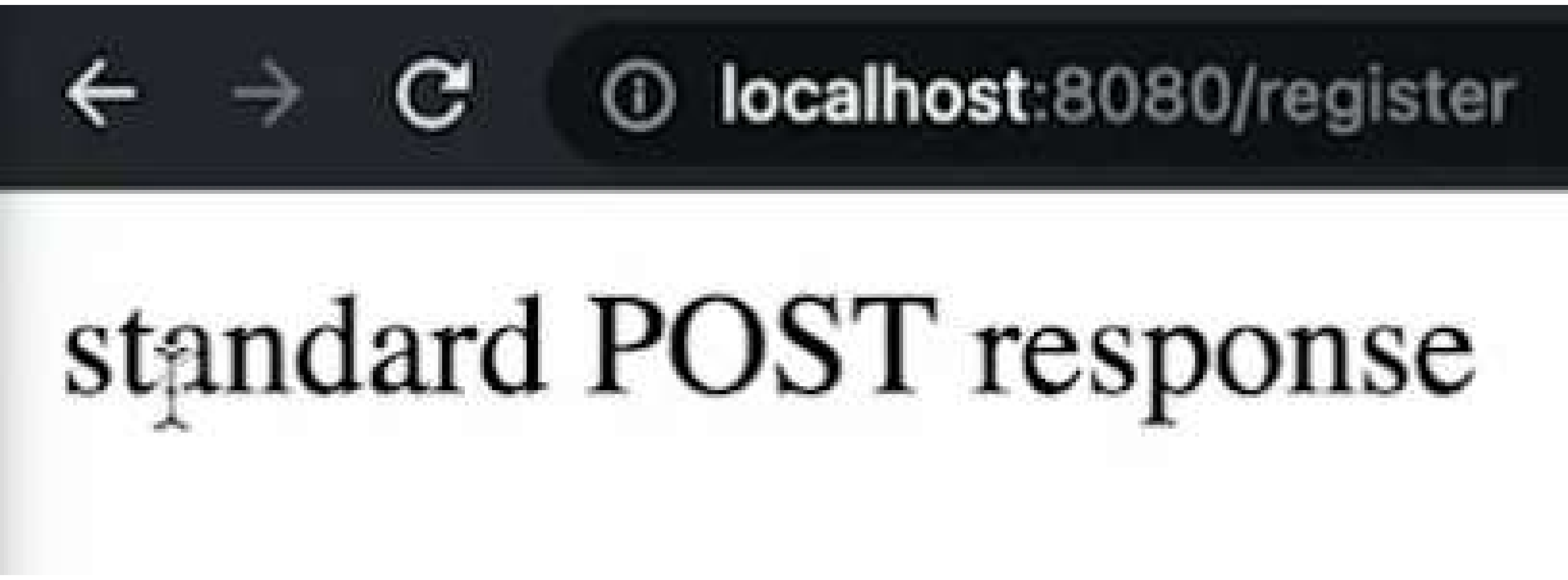
Handling Post requests

- Set up POST request route to get some response
- Parse POST request data

```
app.use(express.urlencoded({ extended: true }));
app.use(express.json());
```

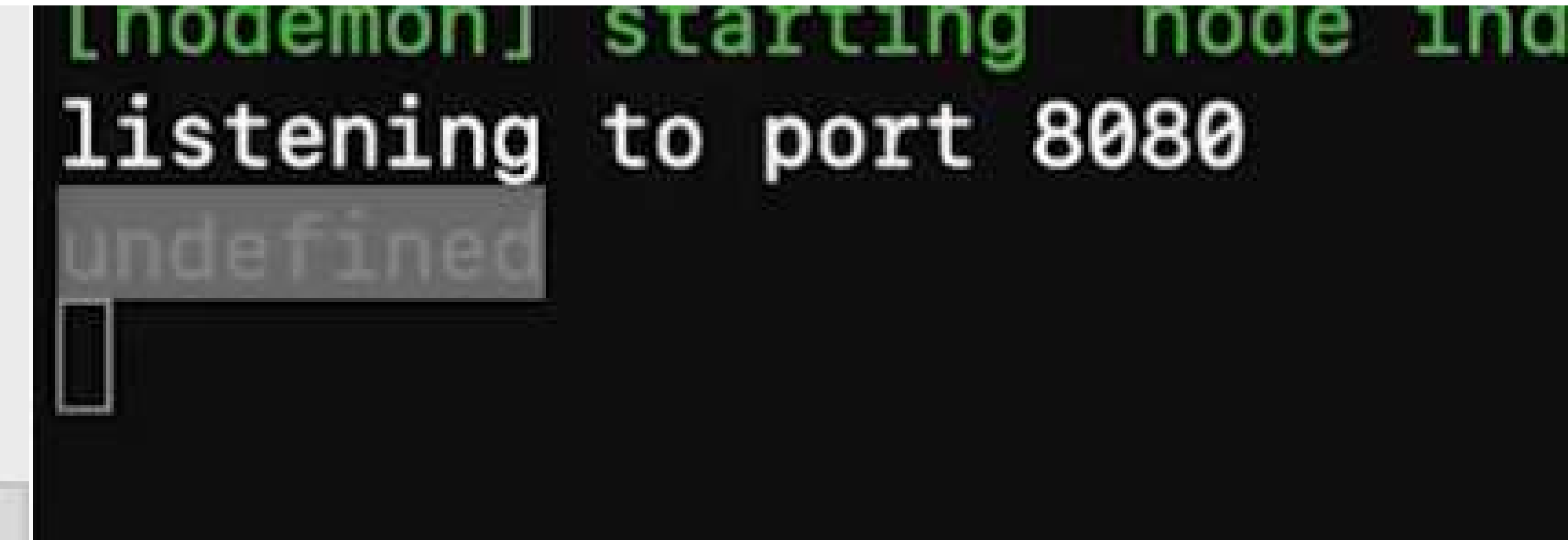


```
const express = require("express");
const app = express();
const port = 8080;

app.get("/register", (req, res) => {
  let { user, password } = req.query;
  res.send(`standard GET response. Welcome $
});

app.post("/register", (req, res) => {
  console.log(req.body);
  res.send("standard POST response");
});

app.listen(port, () => {
  console.log(`listening to port ${port}`);
});

```

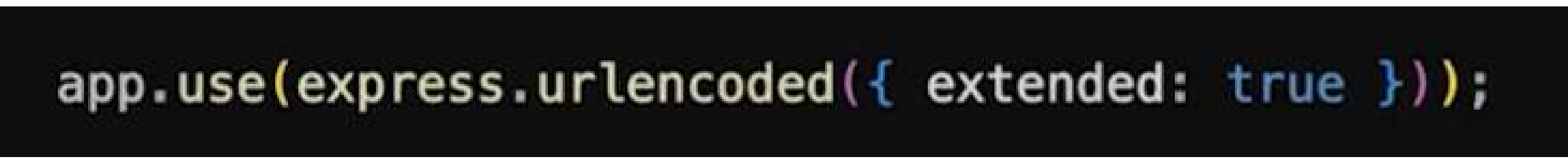


localhost:8080/register

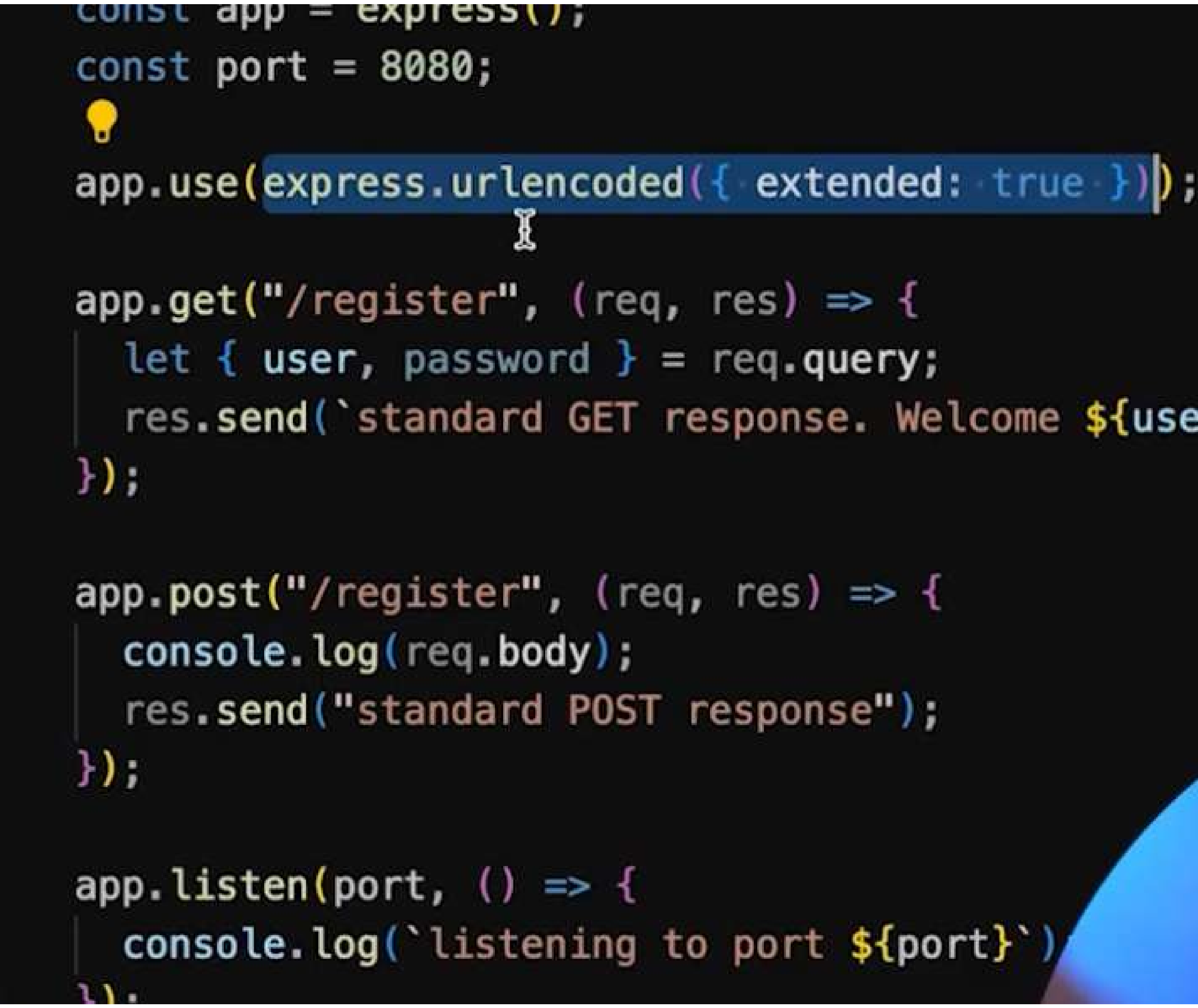
standard POST response



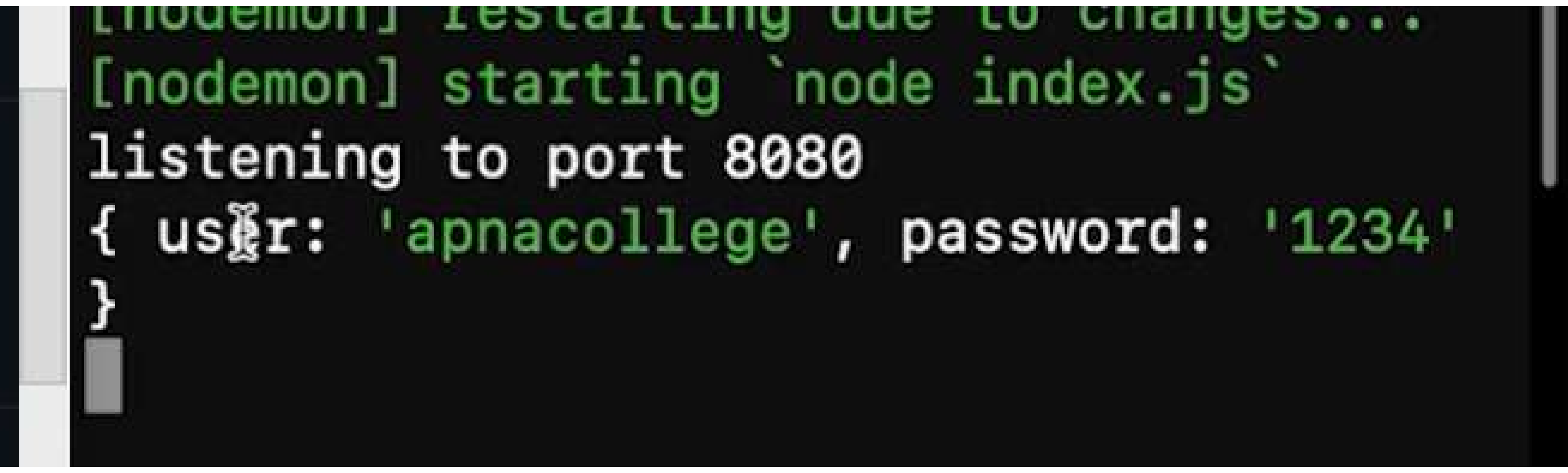
```
[nodemon] starting `node ind
listening to port 8080
undefined
```



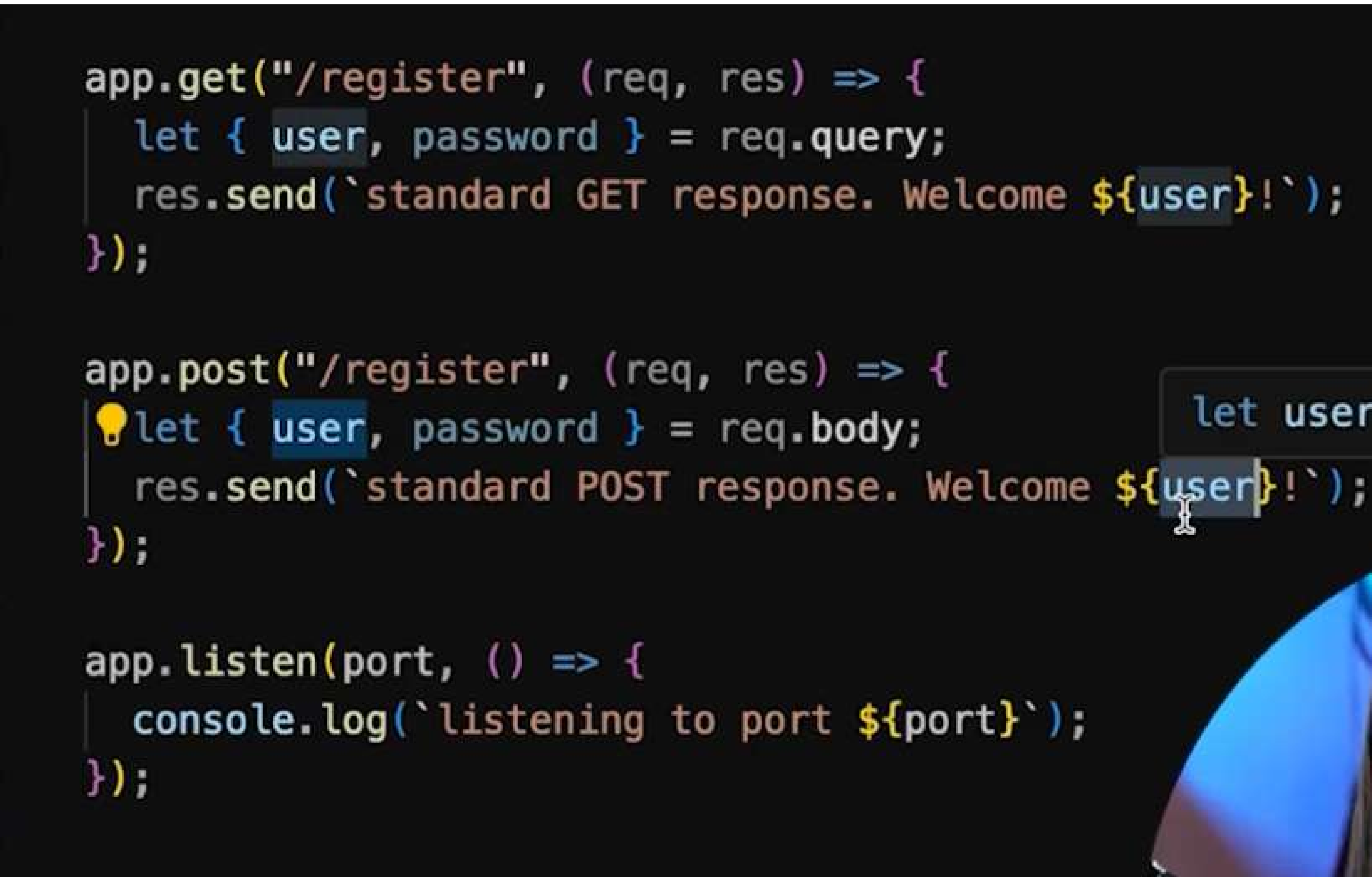
```
app.use(express.urlencoded({ extended: true }));
```



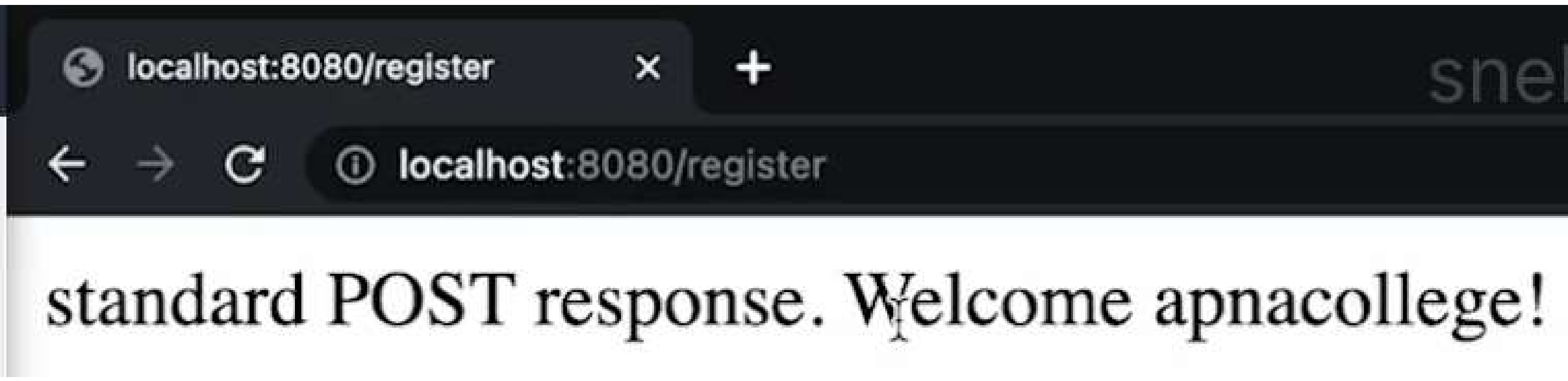
```
const app = express();
const port = 8080;

app.use(express.urlencoded({ extended: true }));

app.get("/register", (req, res) => {
  let { user, password } = req.query;
  res.send(`standard GET response. Welcome ${use
});

app.post("/register", (req, res) => {
  console.log(req.body);
  res.send("standard POST response");
});

app.listen(port, () => {
  console.log(`listening to port ${port}`)
});
```



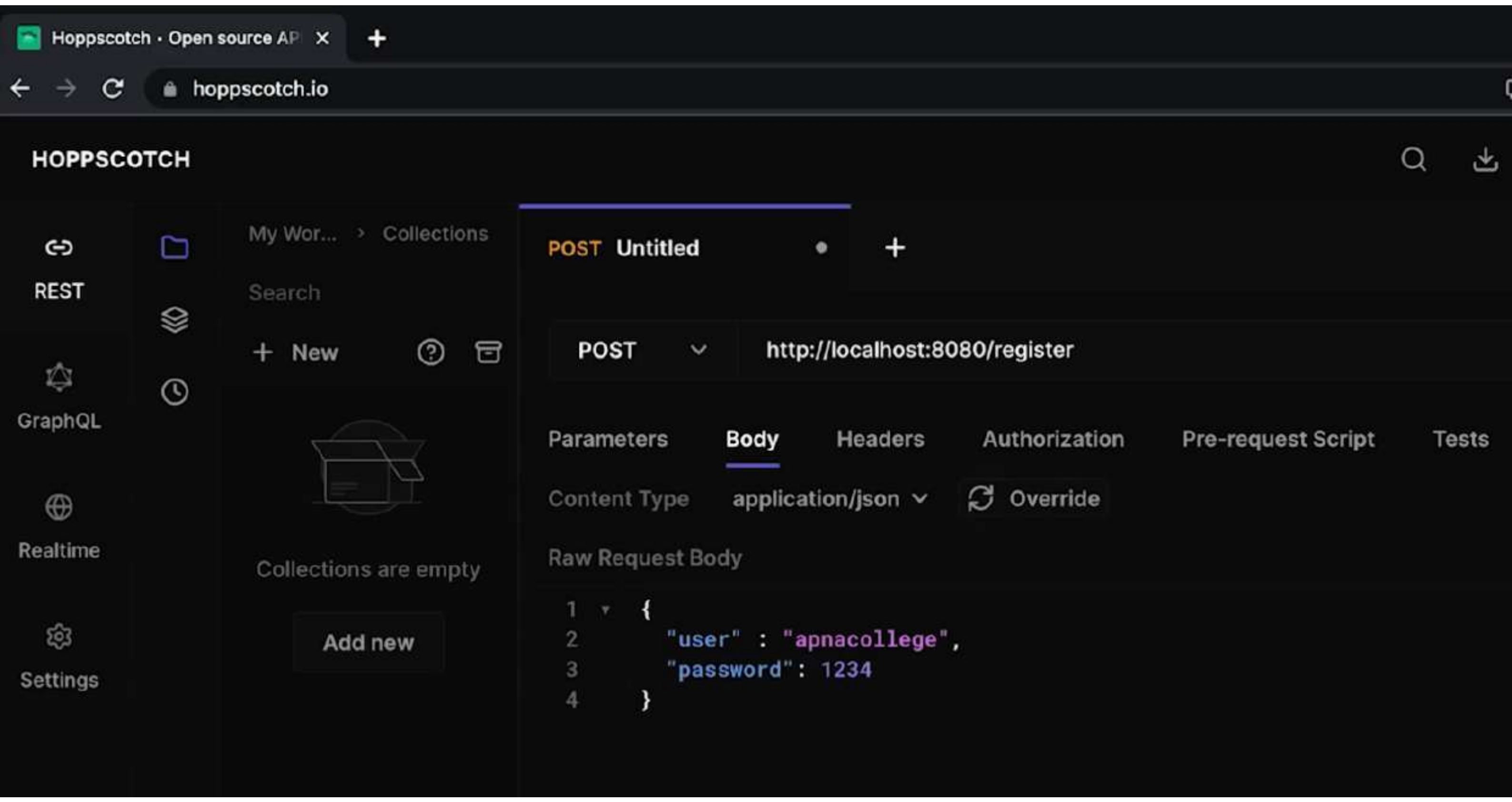
```
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
listening to port 8080
{ user: 'apnacollege', password: '1234'
}
```



```
app.get("/register", (req, res) => {
  let { user, password } = req.query;
  res.send(`standard GET response. Welcome ${user}!`);
});

app.post("/register", (req, res) => {
  let { user, password } = req.body;
  res.send(`standard POST response. Welcome ${user}!`);
});

app.listen(port, () => {
  console.log(`listening to port ${port}`);
});
```



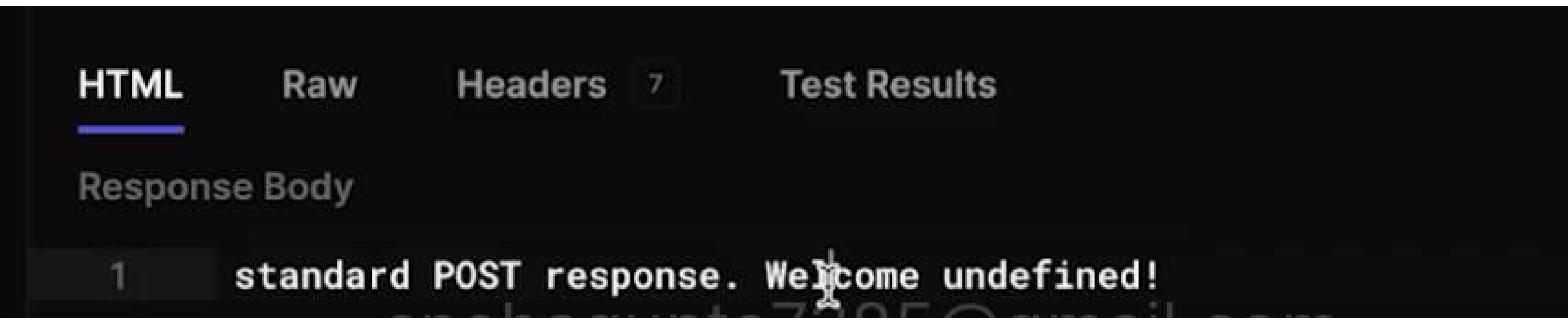
standard POST response. Welcome apnacollege!



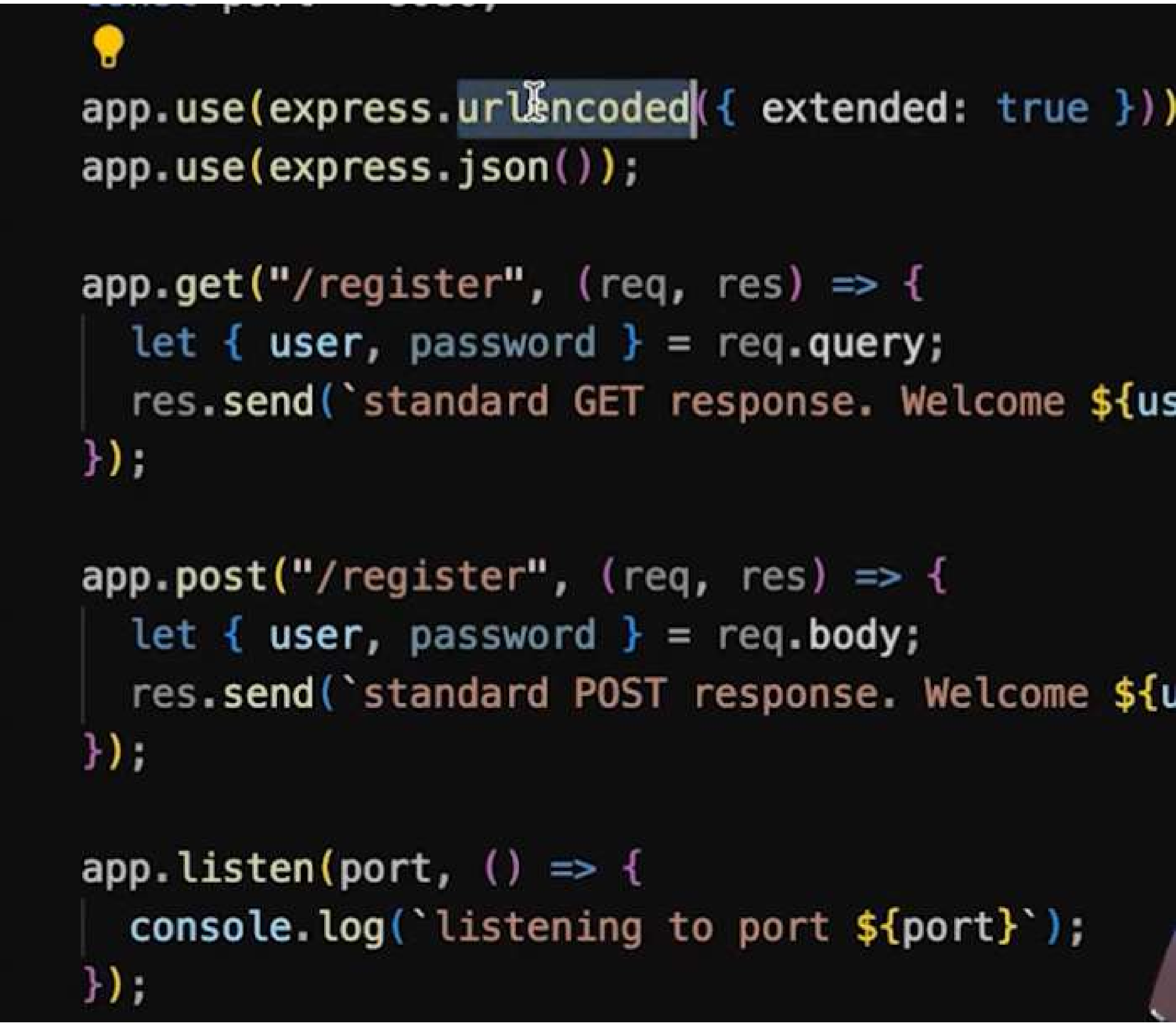
POST http://localhost:8080/register

Content Type application/json

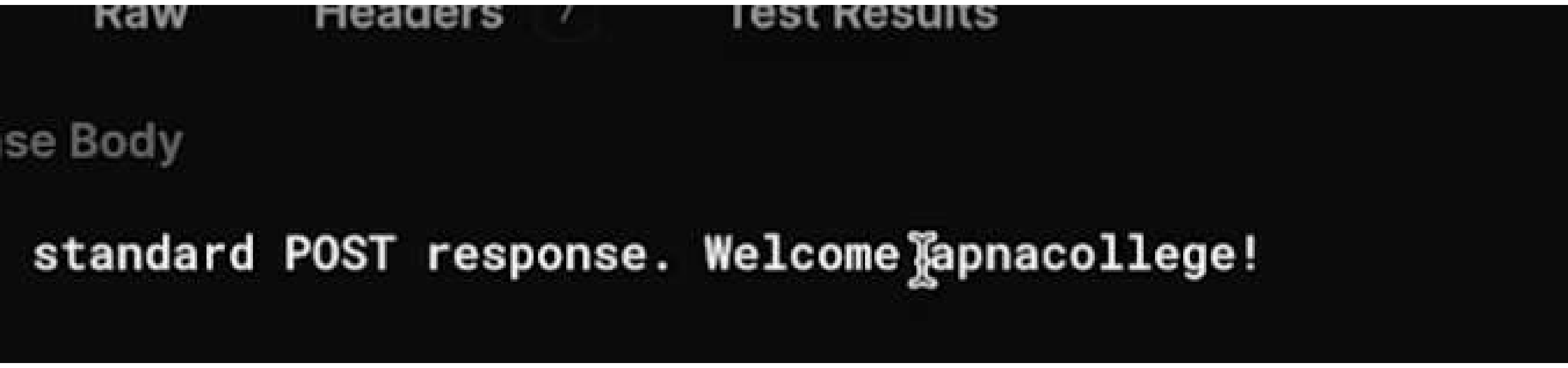
```
{
    "user" : "apnacollege",
    "password": 1234
}
```



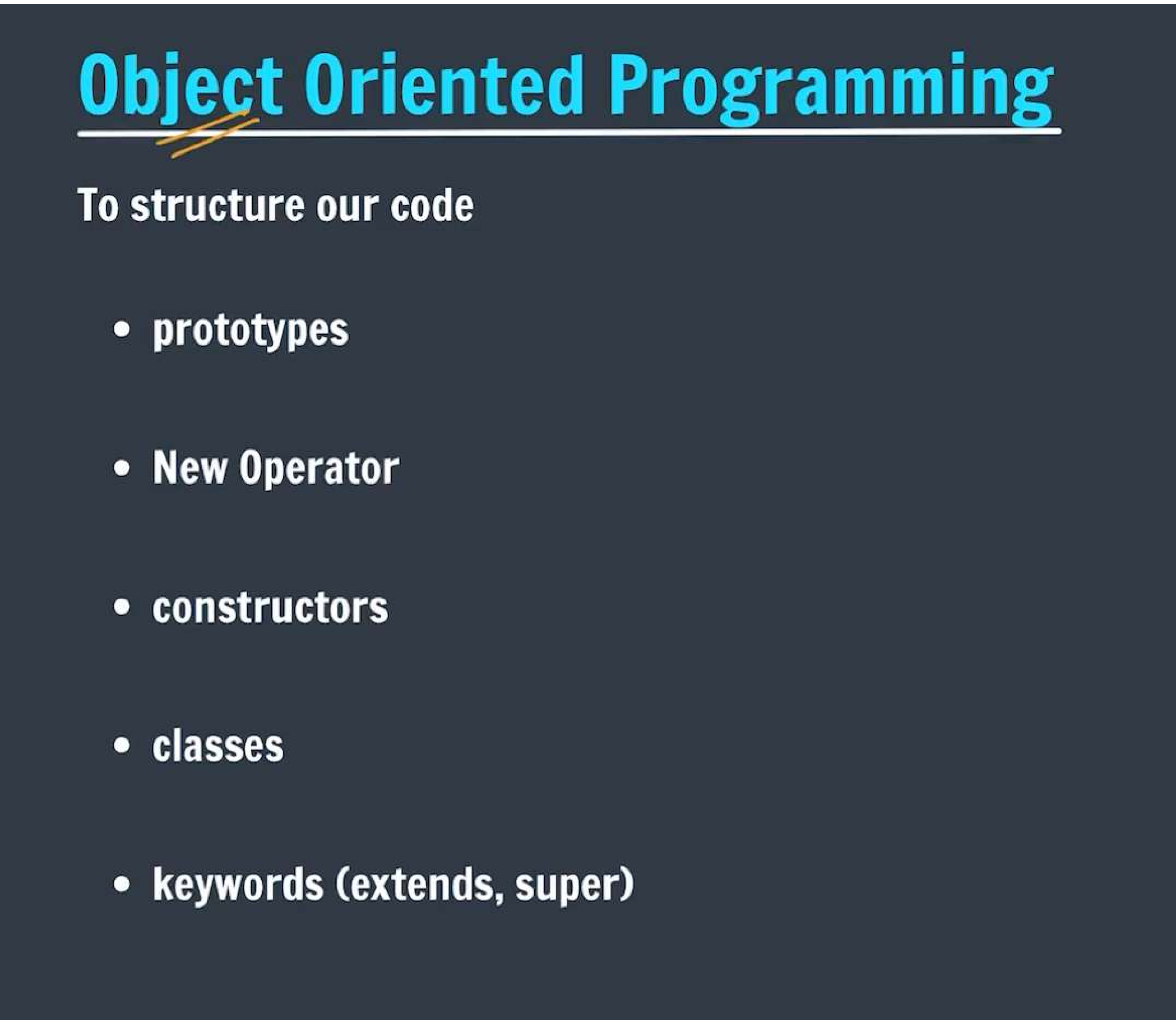
```
standard POST response. Welcome undefined!
```



```
app.use(express.urlencoded({ extended: true }))
app.use(express.json());

app.get("/register", (req, res) => {
  let { user, password } = req.query;
  res.send(`standard GET response. Welcome ${us
});

app.post("/register", (req, res) => {
  let { user, password } = req.body;
  res.send(`standard POST response. Welcome ${u
});

app.listen(port, () => {
  console.log(`listening to port ${port}`);
});
```



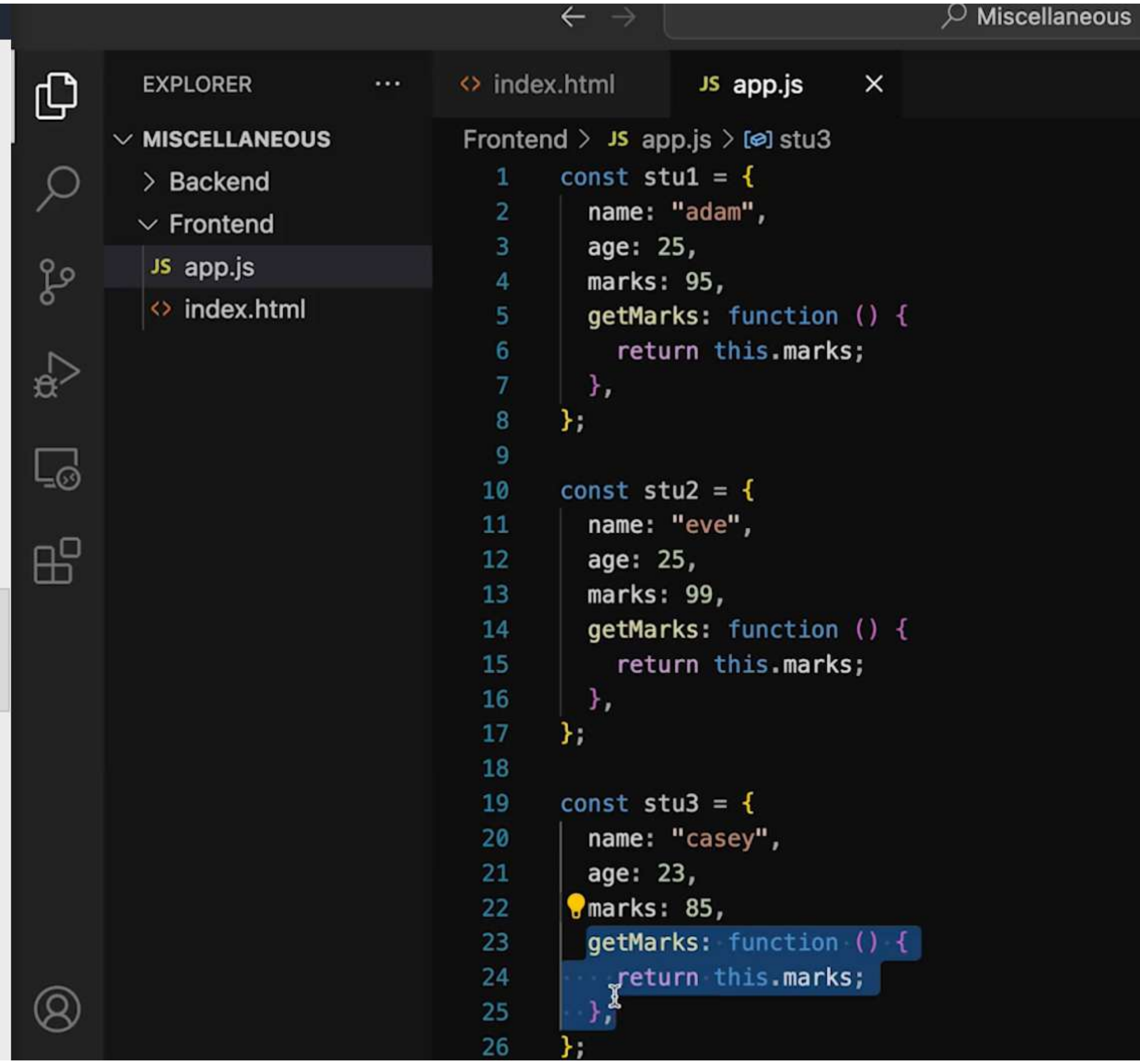
```
standard POST response. Welcome apnacollege!
```



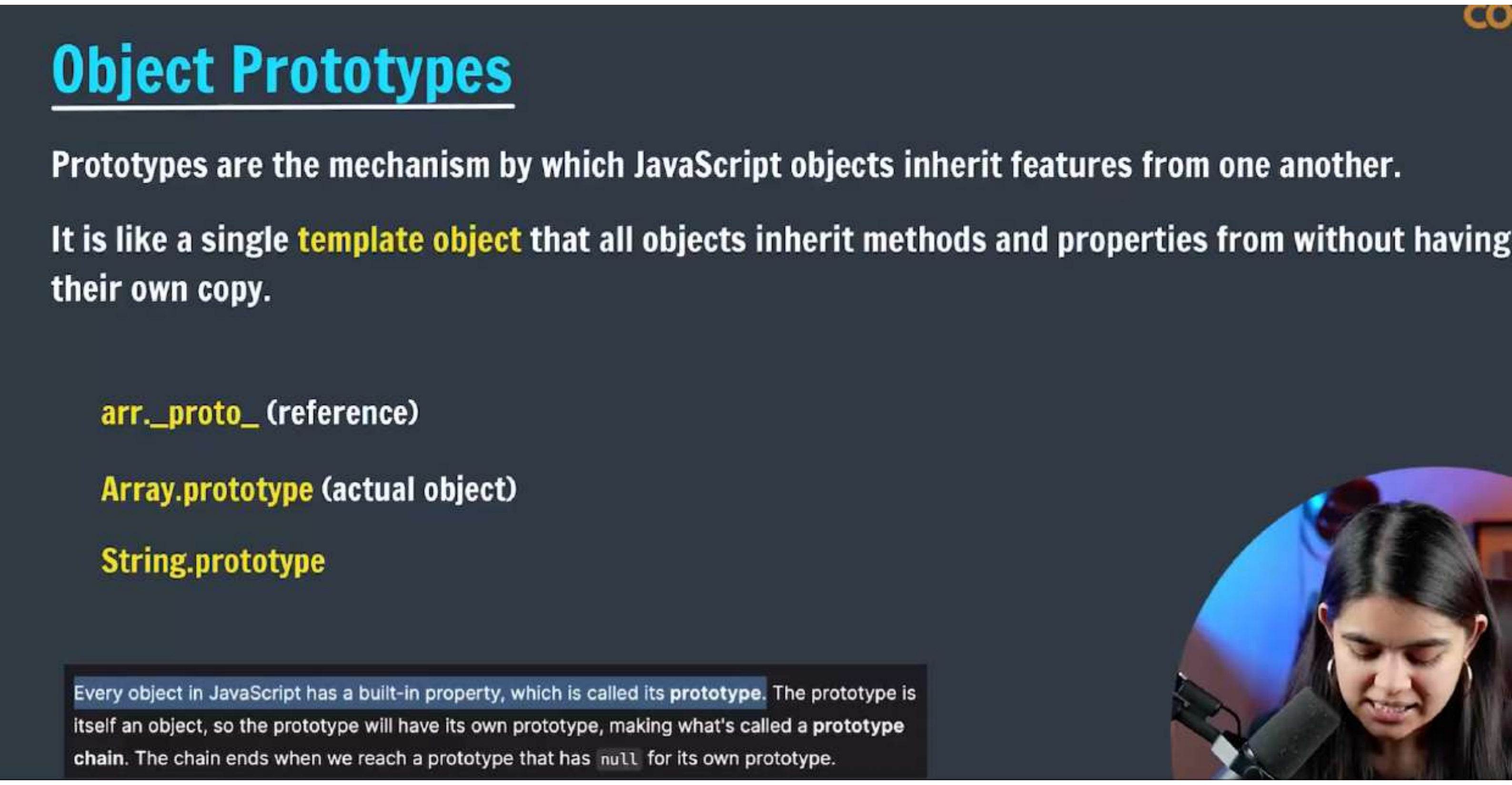
Object Oriented Programming

To structure our code

- prototypes
- New Operator
- constructors
- classes
- keywords (extends, super)



```javascript
const stu1 = {
  name: "adam",
  age: 25,
  marks: 95,
  getMarks: function () {
    return this.marks;
  },
};

const stu2 = {
  name: "eve",
  age: 25,
  marks: 99,
  getMarks: function () {
    return this.marks;
  },
};

const stu3 = {
  name: "casey",
  age: 23,
  marks: 85,
  getMarks: function () {
    return this.marks;
  },
};
```



Object Prototypes

Prototypes are the mechanism by which JavaScript objects inherit features from one another.

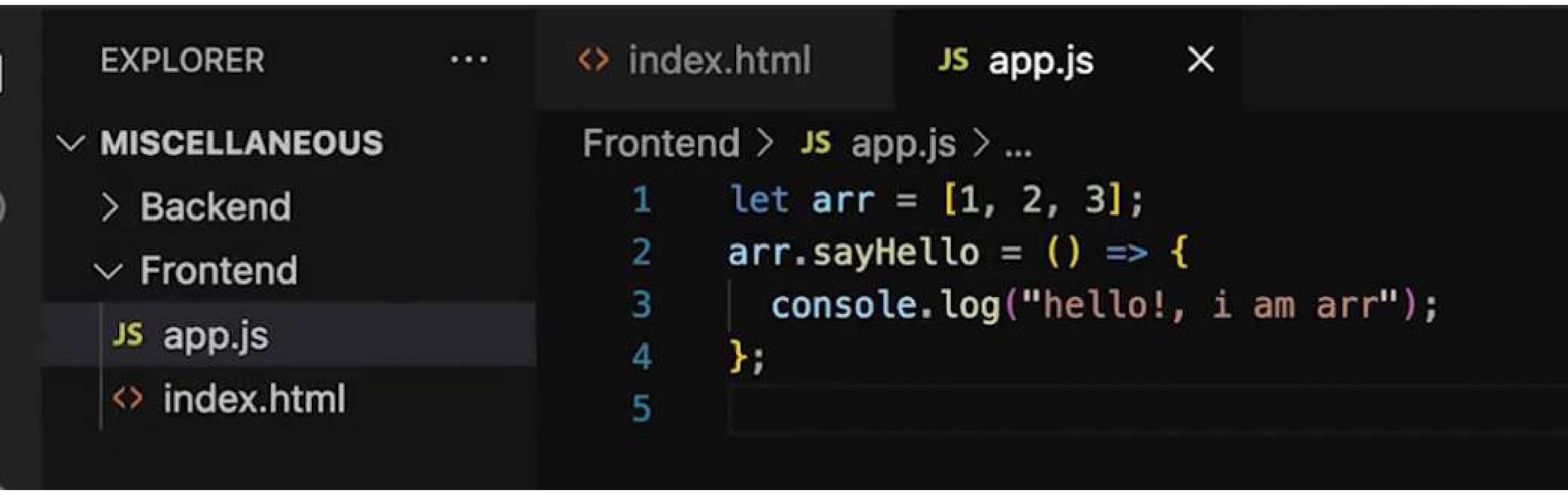
It is like a single template object that all objects inherit methods and properties from without having their own copy.

arr._proto_ (reference)

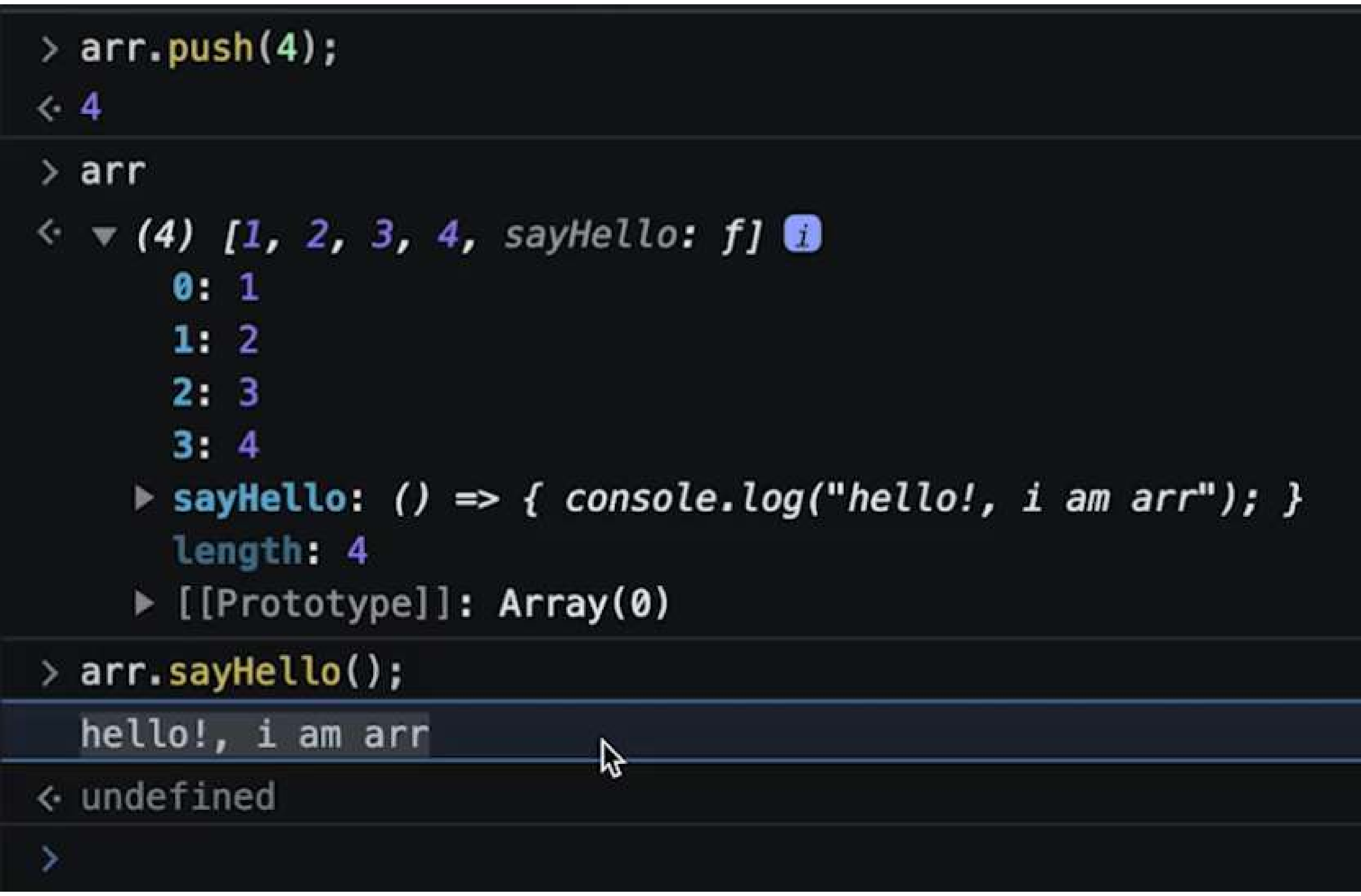
Array.prototype (actual object)

String.prototype

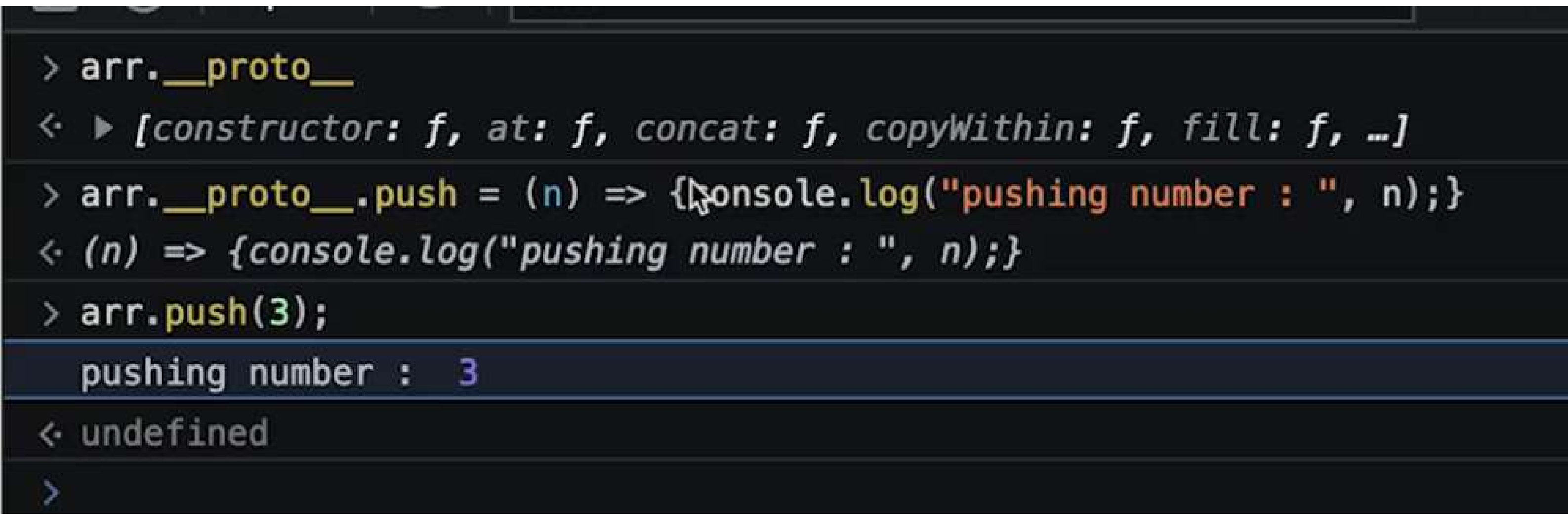
Every object in JavaScript has a built-in property, which is called its **prototype**. The prototype is itself an object, so the prototype will have its own prototype, making what's called a **prototype chain**. The chain ends when we reach a prototype that has `null` for its own prototype.



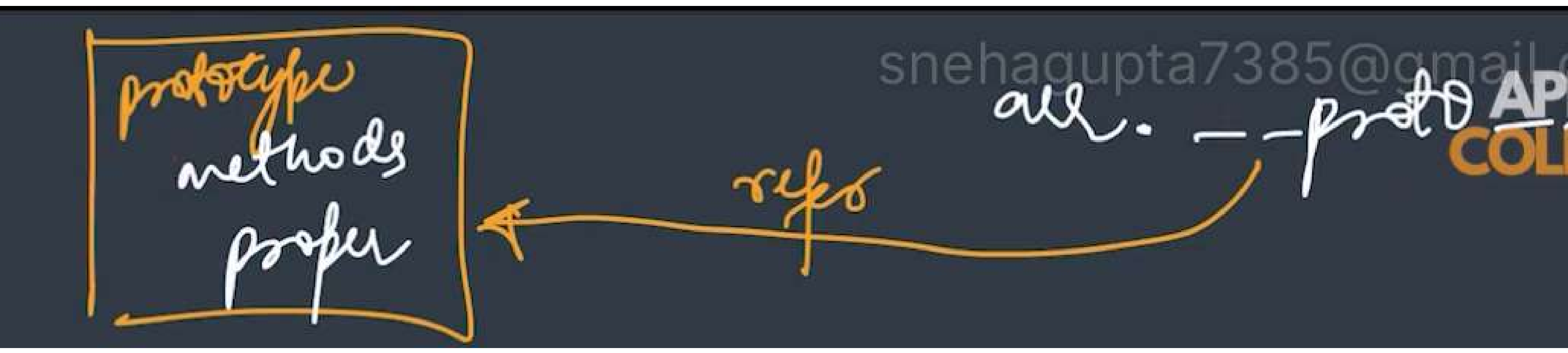
```javascript
let arr = [1, 2, 3];
arr.sayHello = () => {
  console.log("hello!, i am arr");
};
```

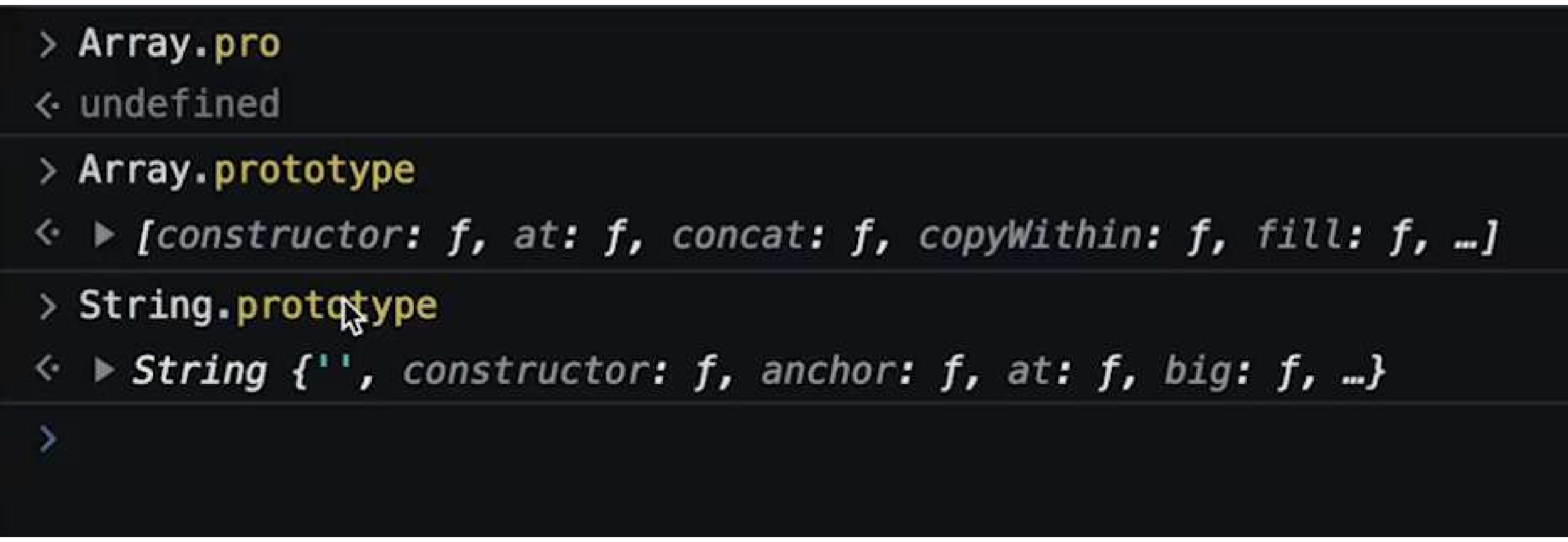


```
> arr.push(4);
<· 4
> arr
<· ▼ (4) [1, 2, 3, 4, sayHello: f] i
       0: 1
       1: 2
       2: 3
       3: 4
     ▶ sayHello: () => { console.log("hello!, i am arr"); }
       length: 4
     ▶ [[Prototype]]: Array(0)
> arr.sayHello();
  hello!, i am arr
<· undefined
>
```

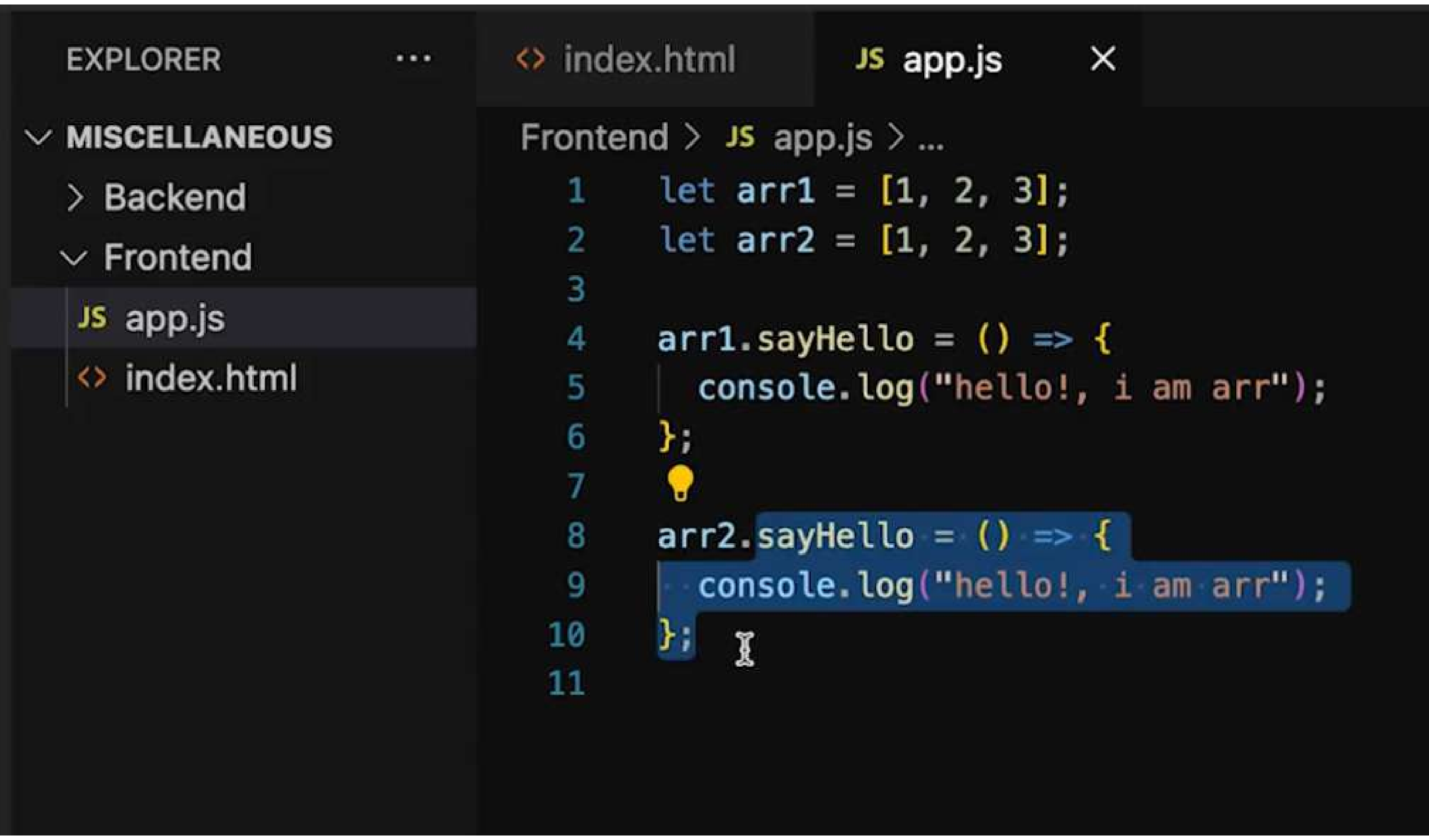


```
> arr.__proto__
< ▶ [constructor: f, at: f, concat: f, copyWithin: f, fill: f, …]
> arr.__proto__.push = (n) => {console.log("pushing number : ", n);}
< (n) => {console.log("pushing number : ", n);}
> arr.push(3);
  pushing number :  3
< undefined
>
```

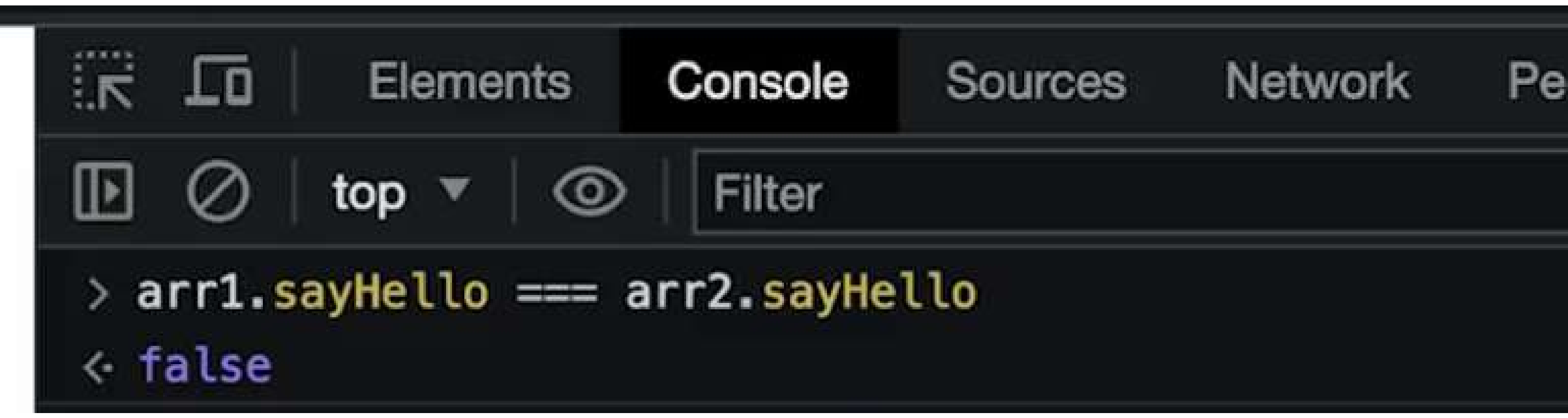

prototype
methods
proper
refer
arr.__proto__



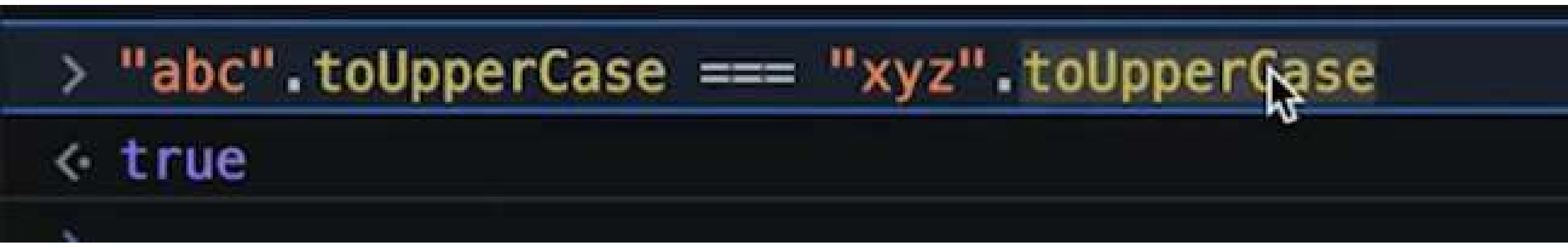
```
> Array.pro
<· undefined
> Array.prototype
<· ▶ [constructor: f, at: f, concat: f, copyWithin: f, fill: f, …]
> String.prototype
<· ▶ String {'', constructor: f, anchor: f, at: f, big: f, …}
>
```



```
let arr1 = [1, 2, 3];
let arr2 = [1, 2, 3];

arr1.sayHello = () => {
  console.log("hello!, i am arr");
};

arr2.sayHello = () => {
  console.log("hello!, i am arr");
};

```



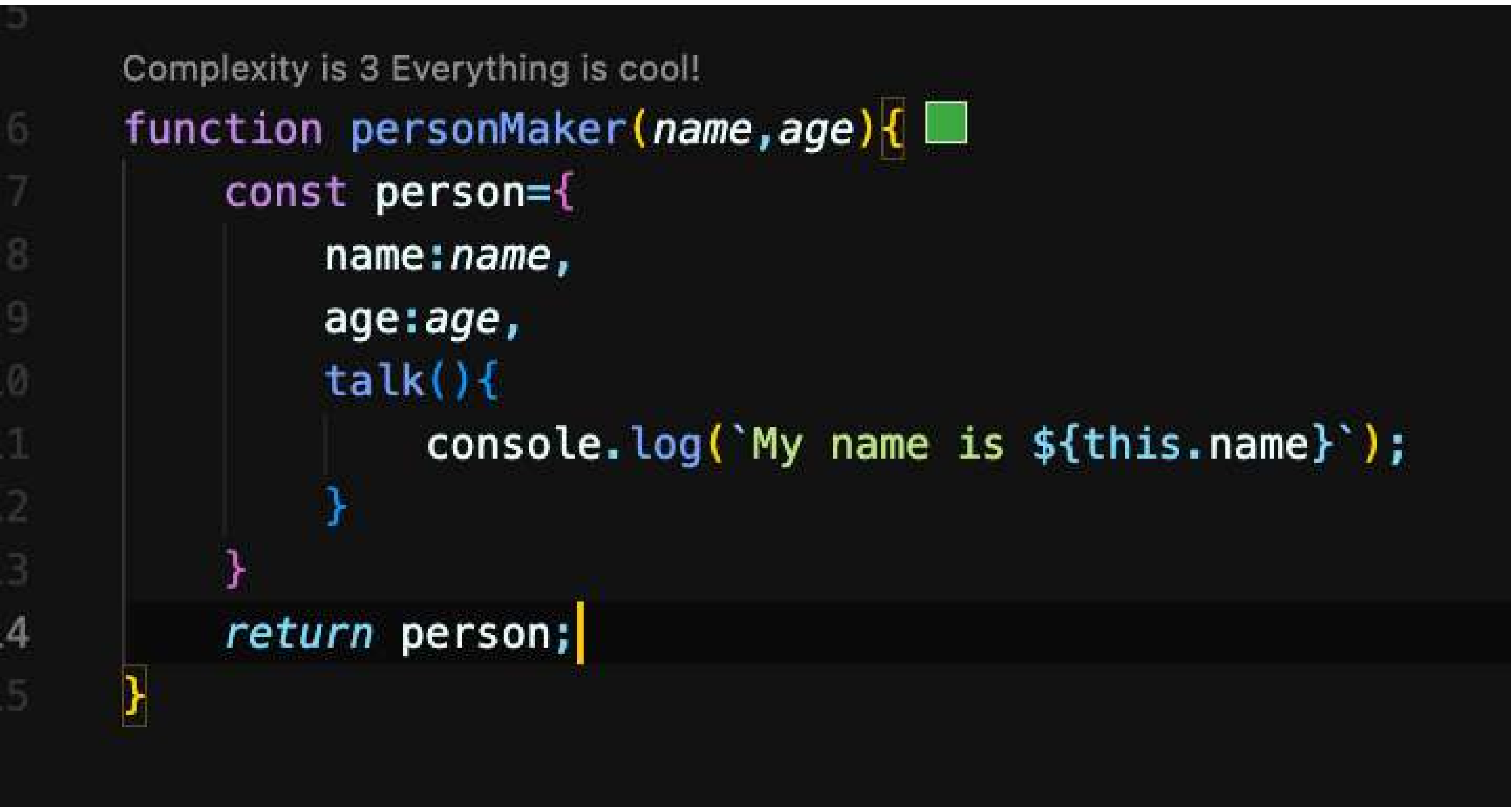
```
> arr1.sayHello === arr2.sayHello
<· false
```



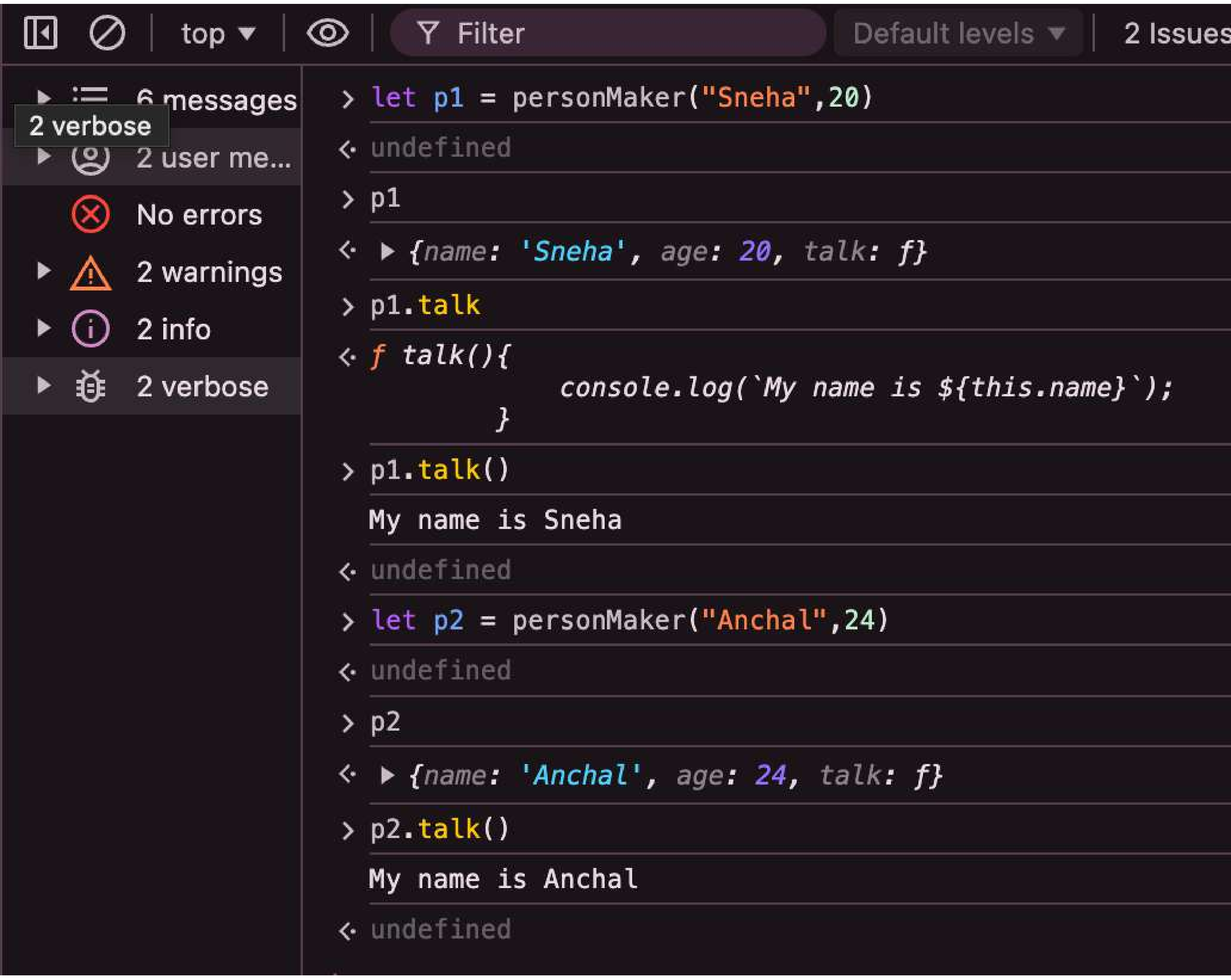
```
> "abc".toUpperCase === "xyz".toUpperCase
<· true
```

Factory Functions

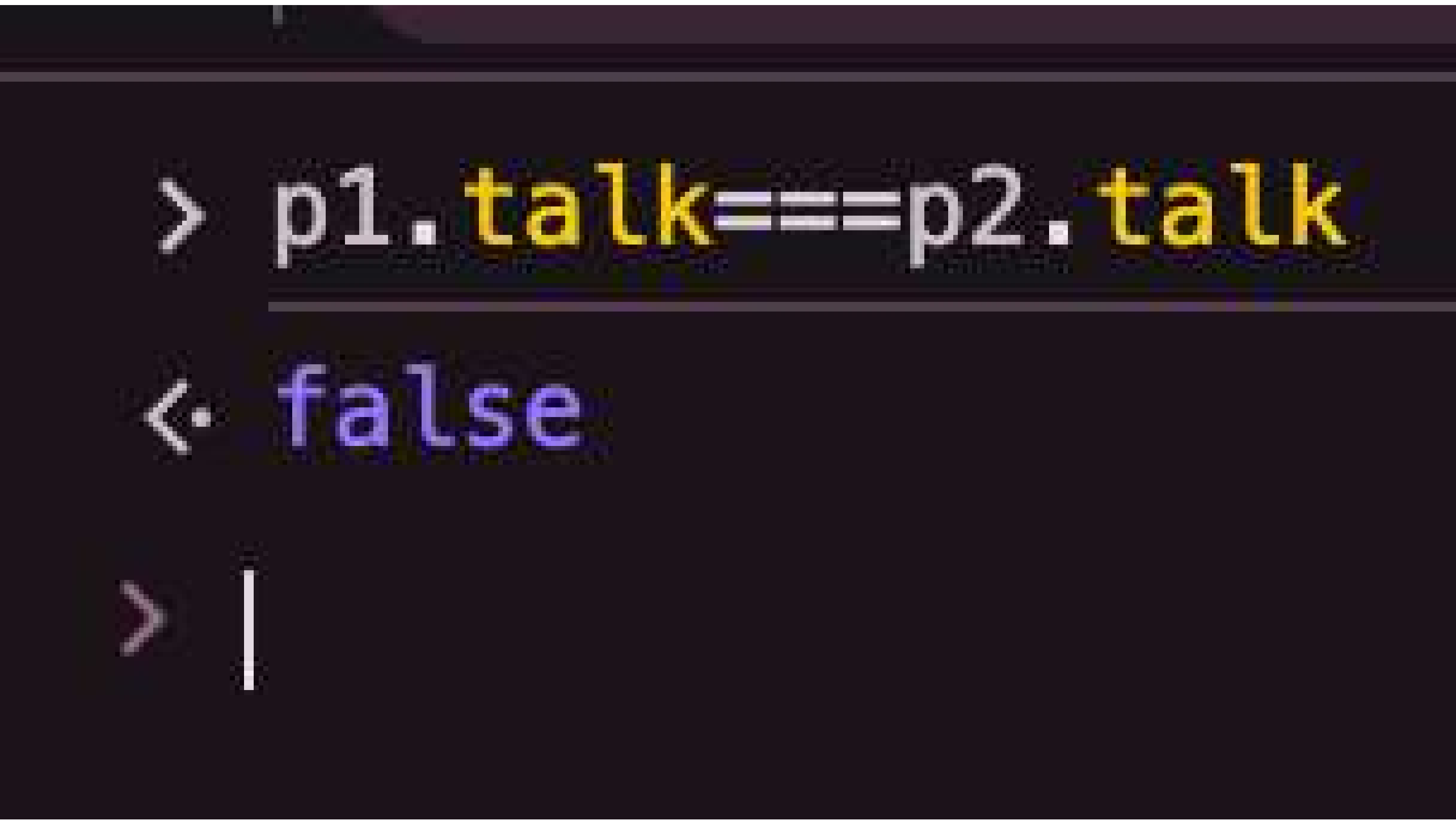
A function that creates objects



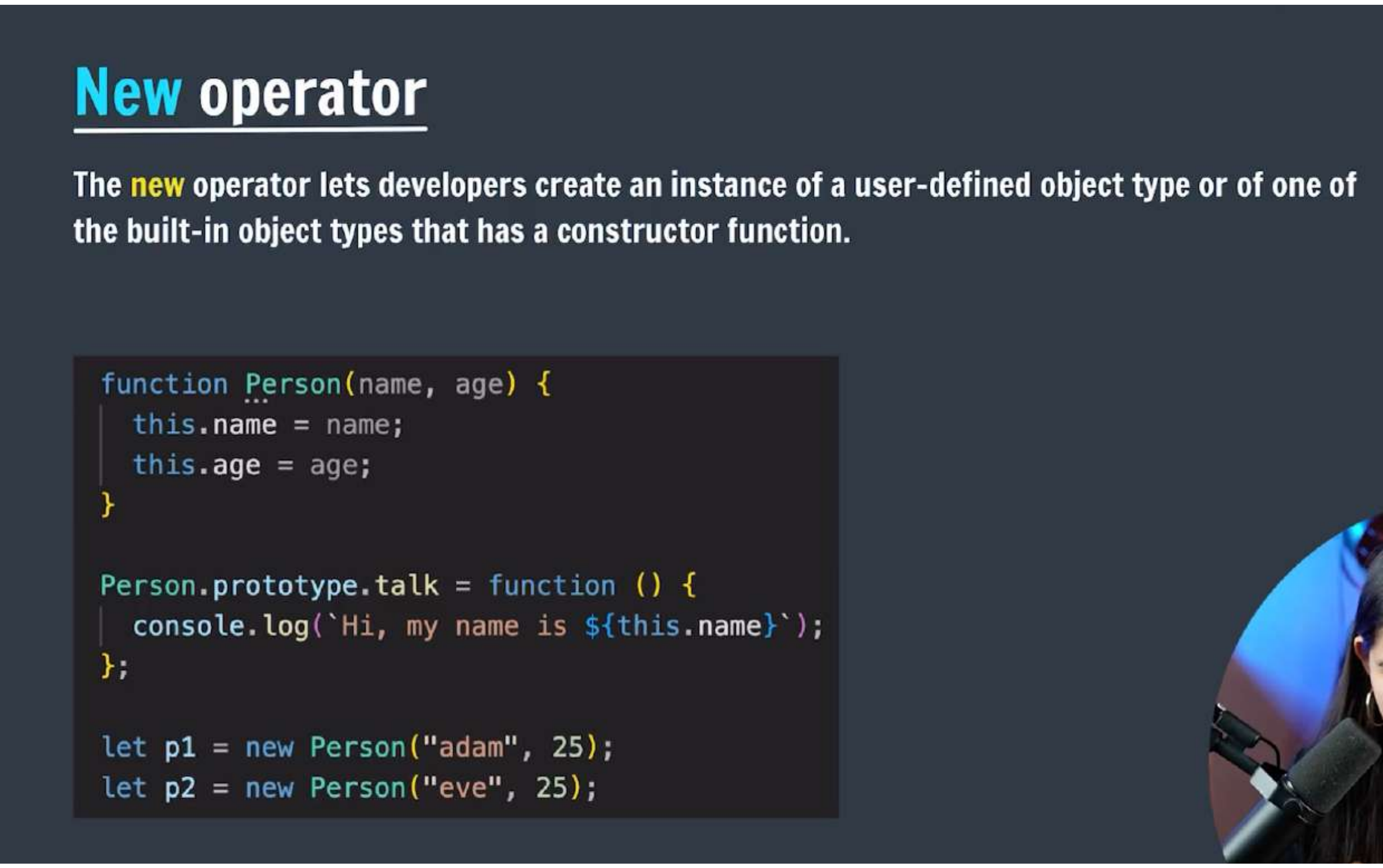
```
Complexity is 3 Everything is cool!
function personMaker(name,age){
    const person={
        name:name,
        age:age,
        talk(){
            console.log(`My name is ${this.name}`);
        }
    }
    return person;
}
```



```
> let p1 = personMaker("Sneha",20)
<· undefined
> p1
<· ▶ {name: 'Sneha', age: 20, talk: f}
> p1.talk
<· f talk(){
            console.log(`My name is ${this.name}`);
        }
> p1.talk()
  My name is Sneha
<· undefined
> let p2 = personMaker("Anchal",24)
<· undefined
> p2
<· ▶ {name: 'Anchal', age: 24, talk: f}
> p2.talk()
  My name is Anchal
<· undefined
```



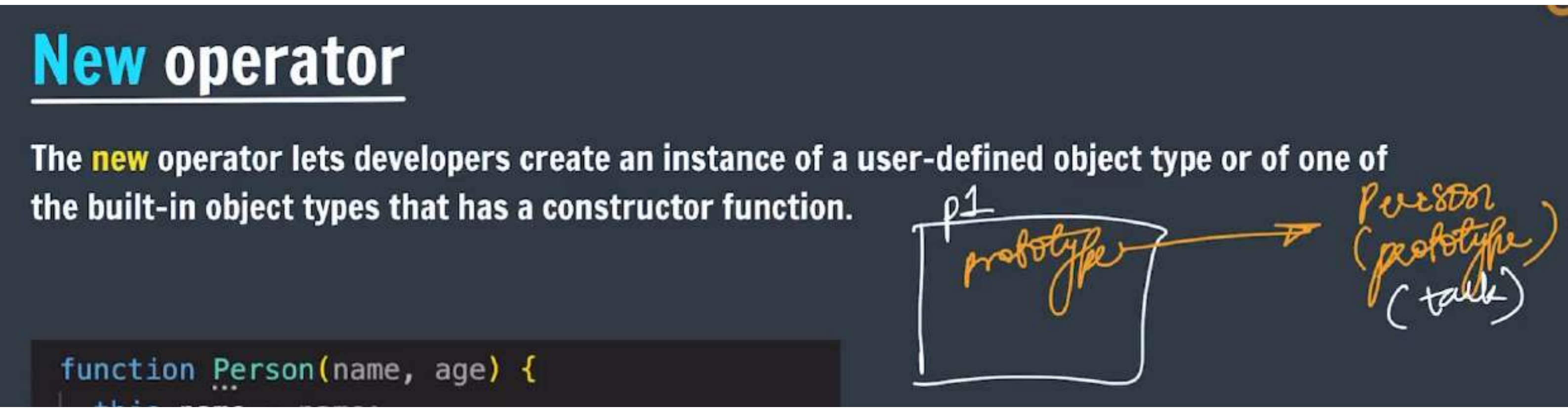
```
> p1.talk===p2.talk
<· false
> 
```



New operator

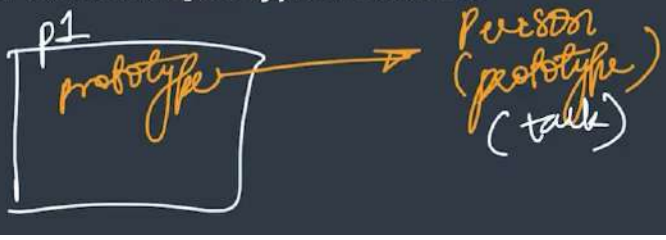
The **new** operator lets developers create an instance of a user-defined object type or of one of the built-in object types that has a constructor function.

```javascript
function Person(name, age) {
  this.name = name;
  this.age = age;
}

Person.prototype.talk = function () {
  console.log(`Hi, my name is ${this.name}`);
};

let p1 = new Person("adam", 25);
let p2 = new Person("eve", 25);
```

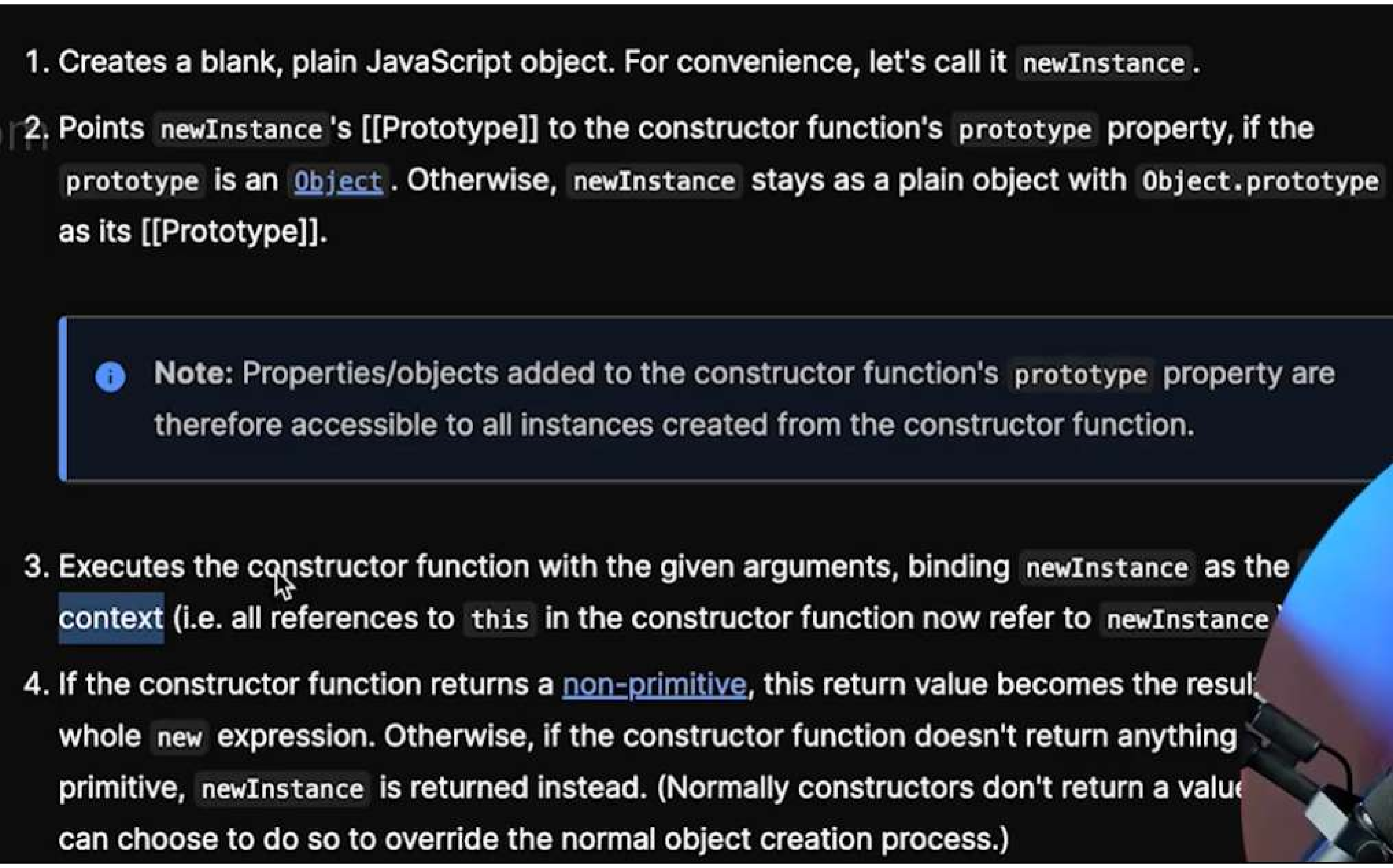


New operator

The **new** operator lets developers create an instance of a user-defined object type or of one of the built-in object types that has a constructor function.



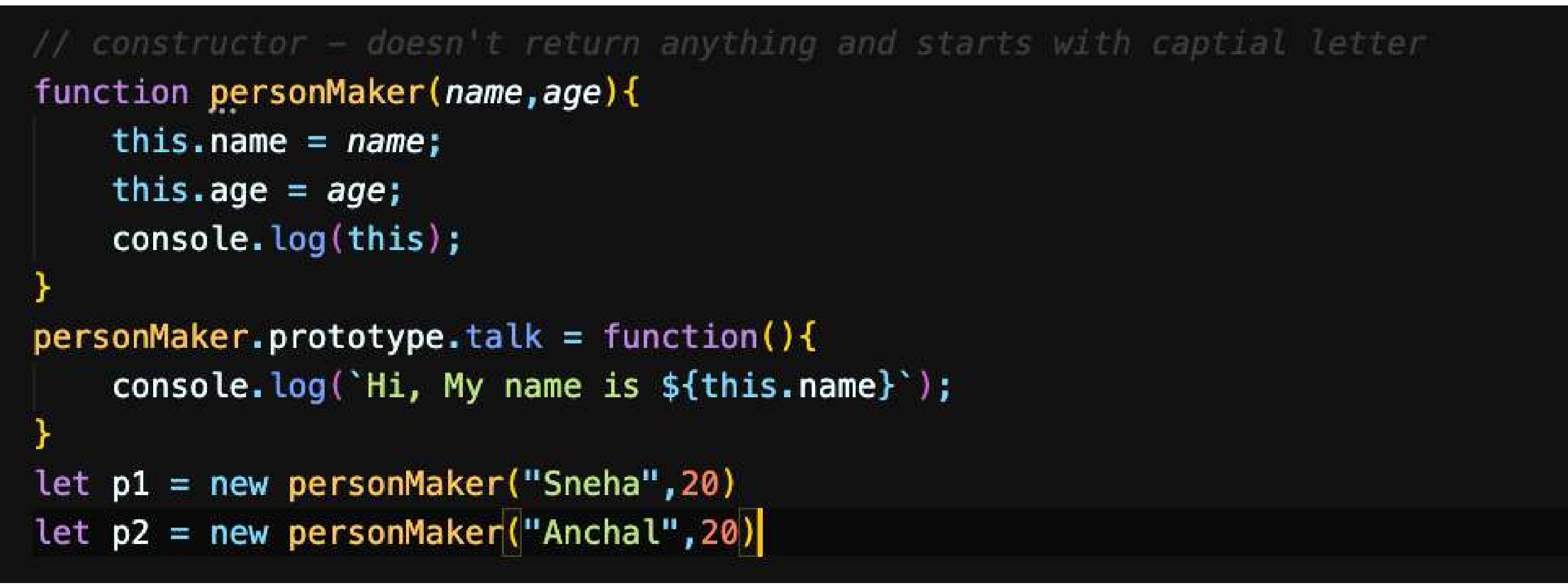
```
function Person(name, age) {
```



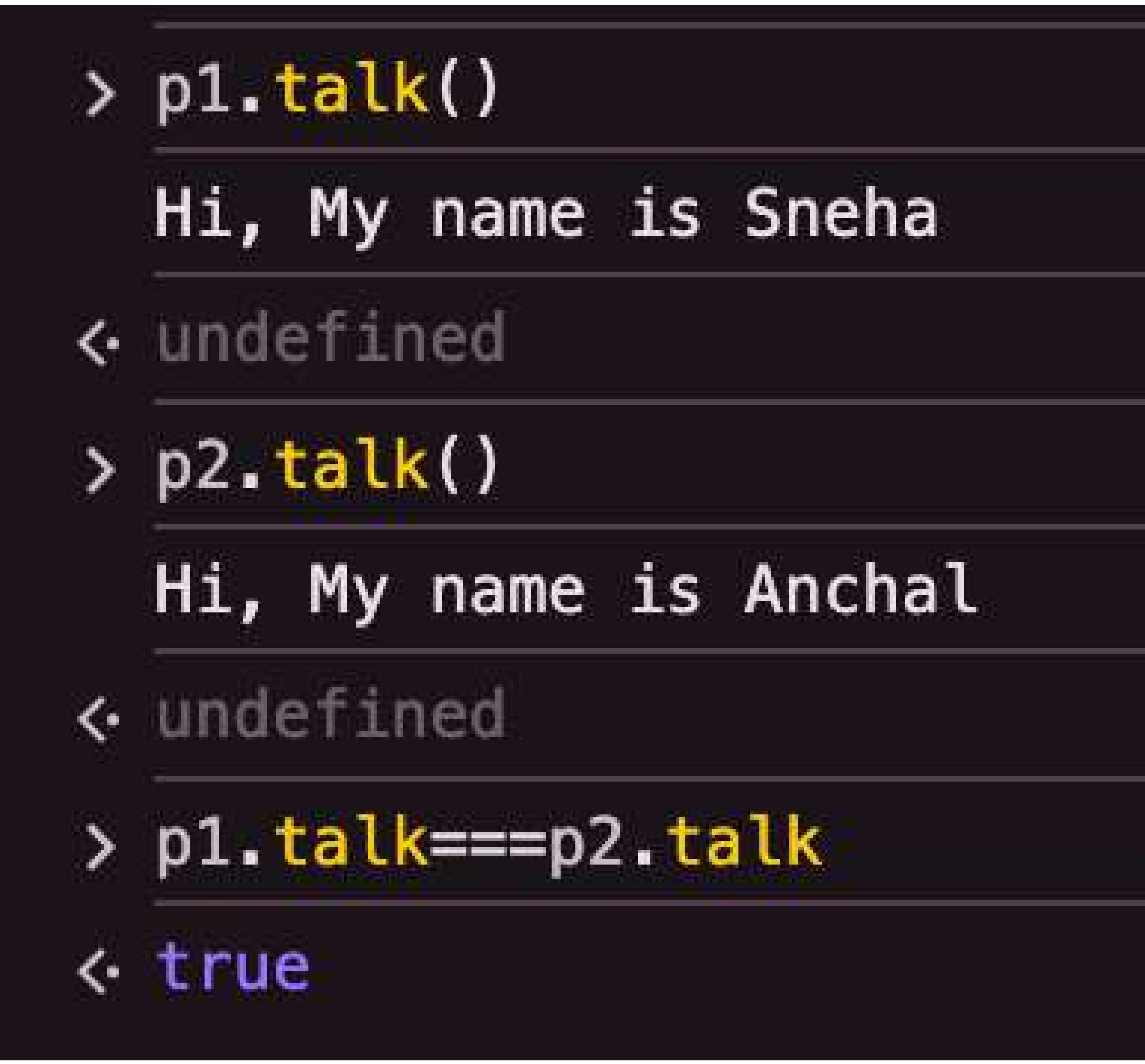
1. Creates a blank, plain JavaScript object. For convenience, let's call it `newInstance`.
2. Points `newInstance`'s [[Prototype]] to the constructor function's `prototype` property, if the `prototype` is an `Object`. Otherwise, `newInstance` stays as a plain object with `Object.prototype` as its [[Prototype]].

   Note: Properties/objects added to the constructor function's `prototype` property are therefore accessible to all instances created from the constructor function.

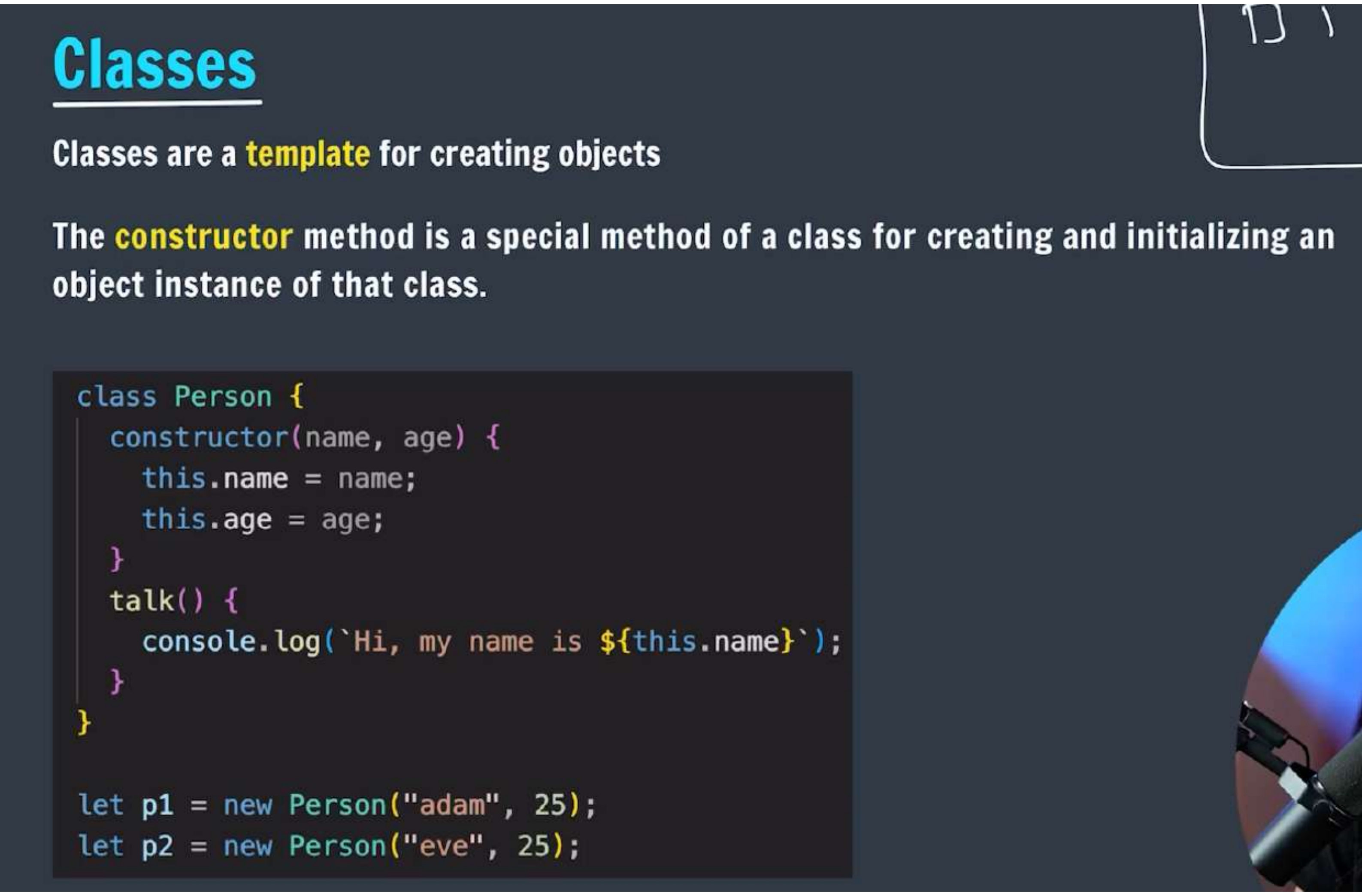
3. Executes the constructor function with the given arguments, binding `newInstance` as the context (i.e. all references to `this` in the constructor function now refer to `newInstance`
4. If the constructor function returns a non-primitive, this return value becomes the resul whole `new` expression. Otherwise, if the constructor function doesn't return anything primitive, `newInstance` is returned instead. (Normally constructors don't return a valu can choose to do so to override the normal object creation process.)



```
// constructor - doesn't return anything and starts with captial letter
function personMaker(name,age){
    this.name = name;
    this.age = age;
    console.log(this);
}
personMaker.prototype.talk = function(){
    console.log(`Hi, My name is ${this.name}`);
}
let p1 = new personMaker("Sneha",20)
let p2 = new personMaker("Anchal",20)
```



```
> p1.talk()
  Hi, My name is Sneha
<· undefined
> p2.talk()
  Hi, My name is Anchal
<· undefined
> p1.talk===p2.talk
<· true
```

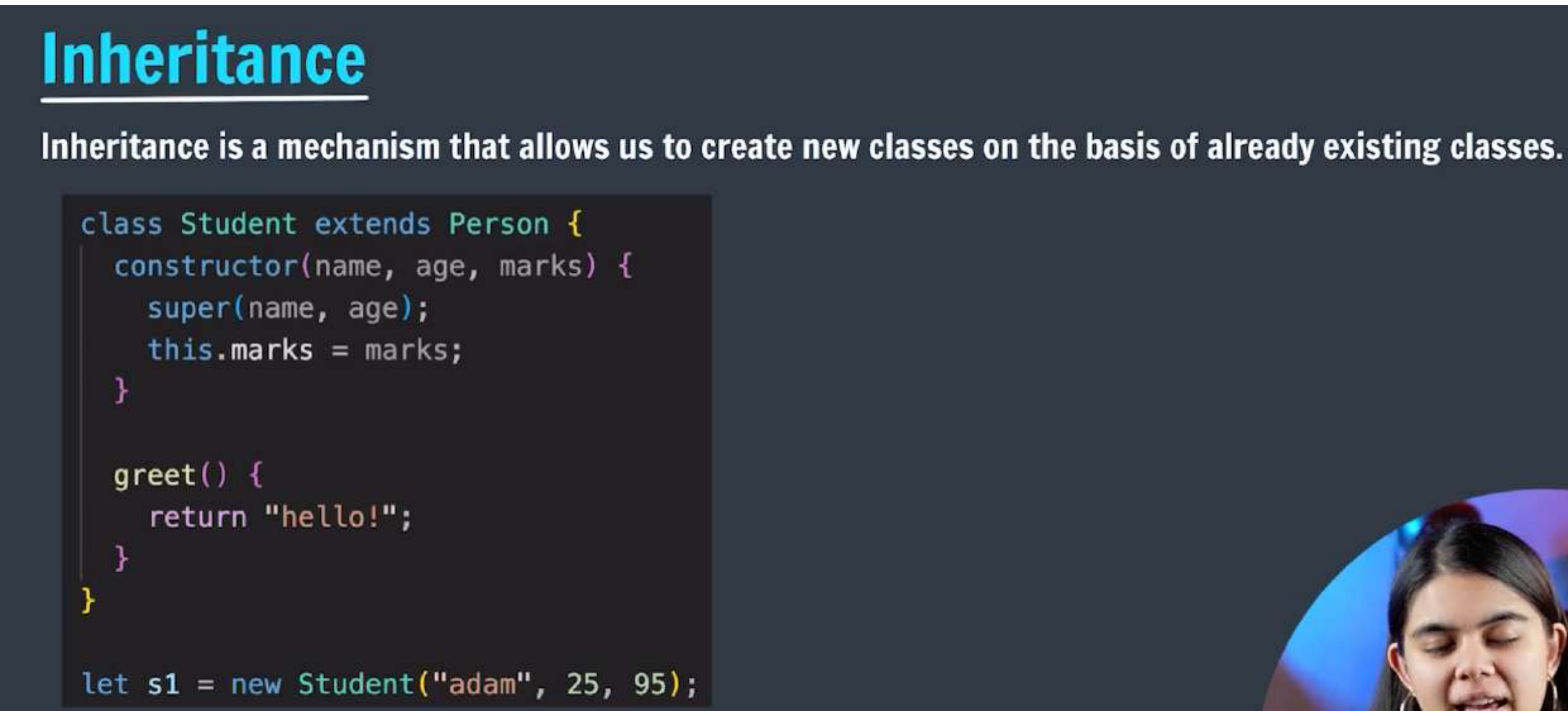


Classes

Classes are a template for creating objects

The constructor method is a special method of a class for creating and initializing an object instance of that class.

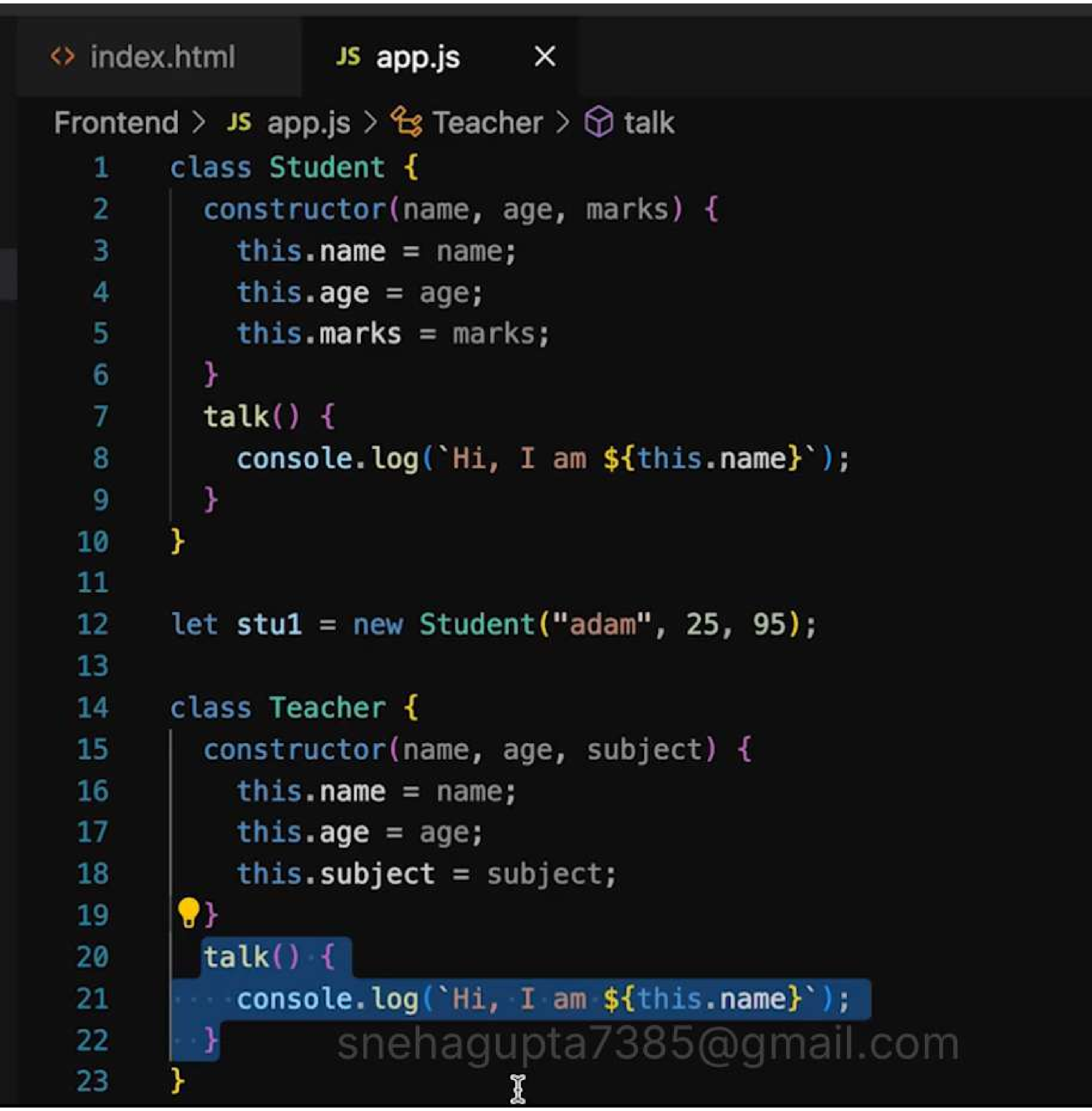
```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  talk() {
    console.log(`Hi, my name is ${this.name}`);
  }
}

let p1 = new Person("adam", 25);
let p2 = new Person("eve", 25);
```



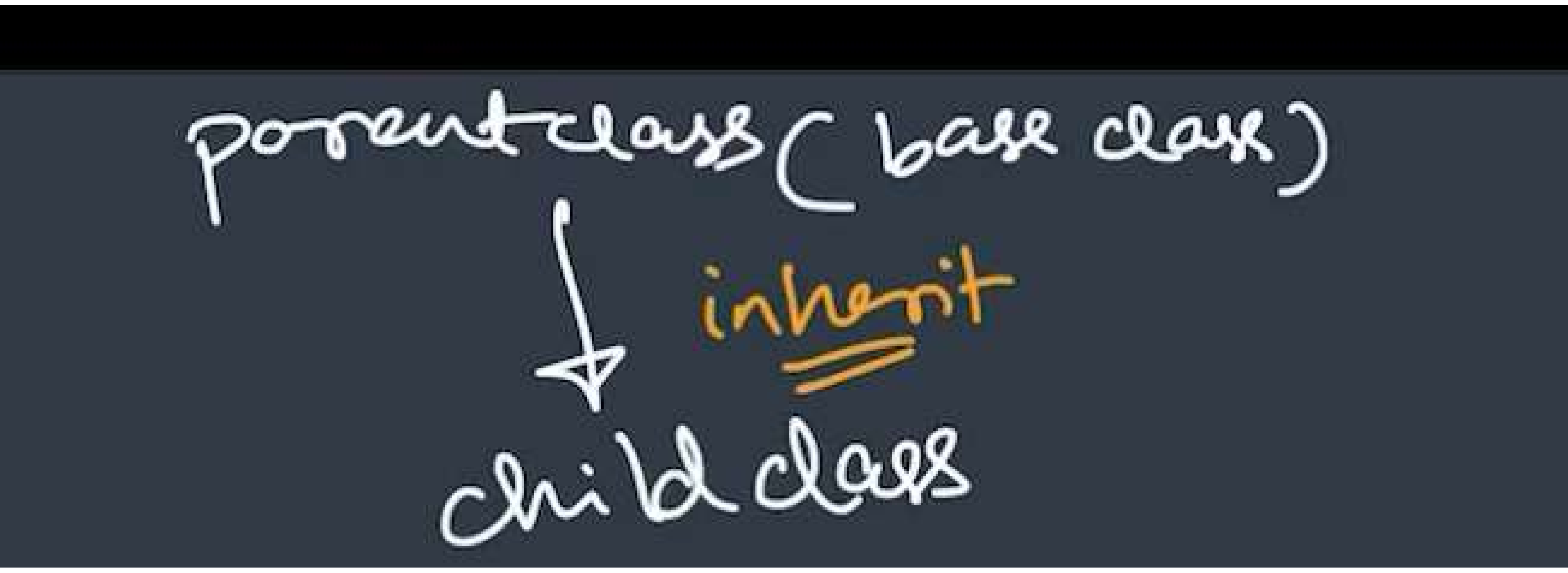
Inheritance

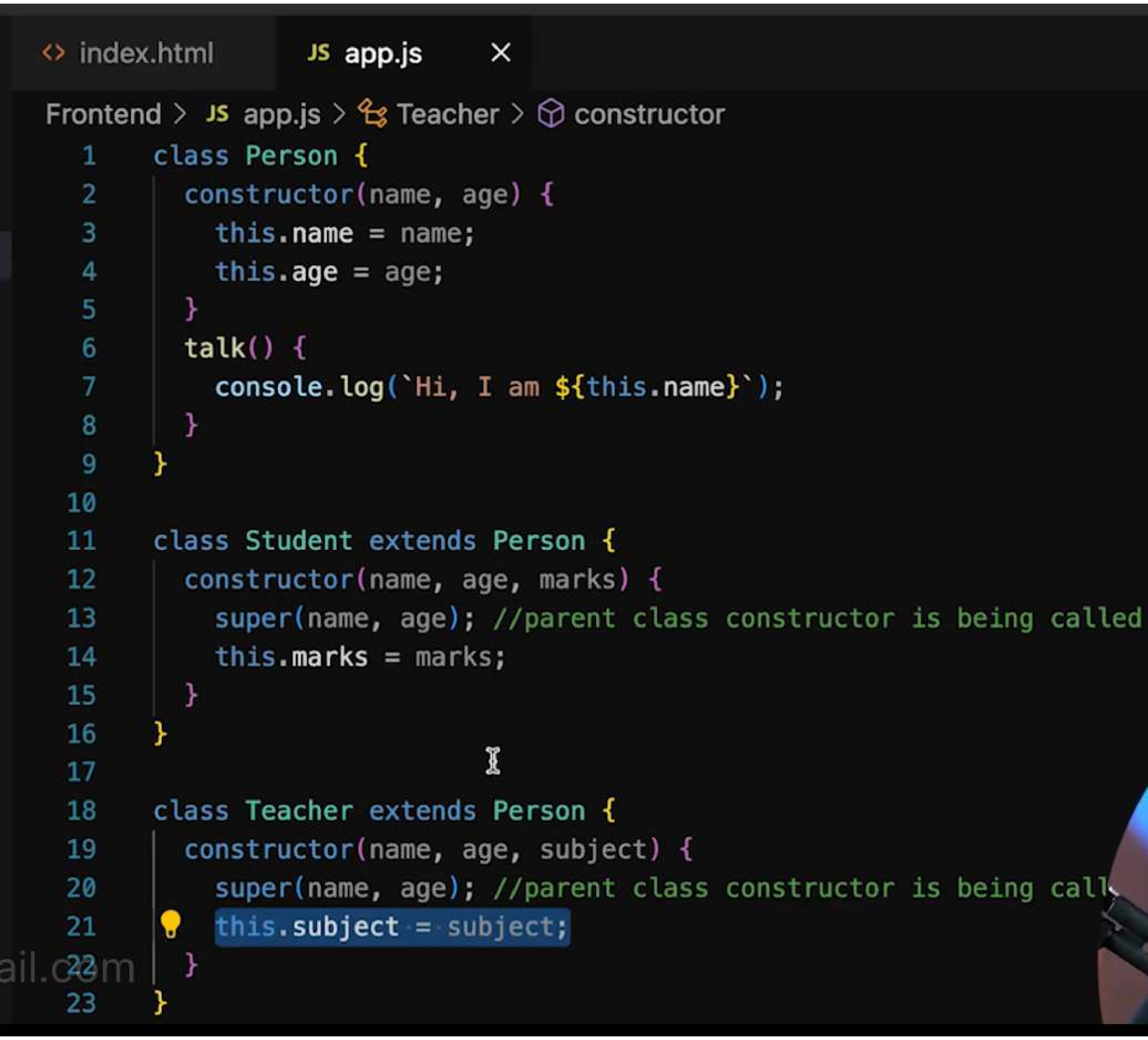
Inheritance is a mechanism that allows us to create new classes on the basis of already existing classes.

```
class Student extends Person {
  constructor(name, age, marks) {
    super(name, age);
    this.marks = marks;
  }

  greet() {
    return "hello!";
  }
}

let s1 = new Student("adam", 25, 95);
```

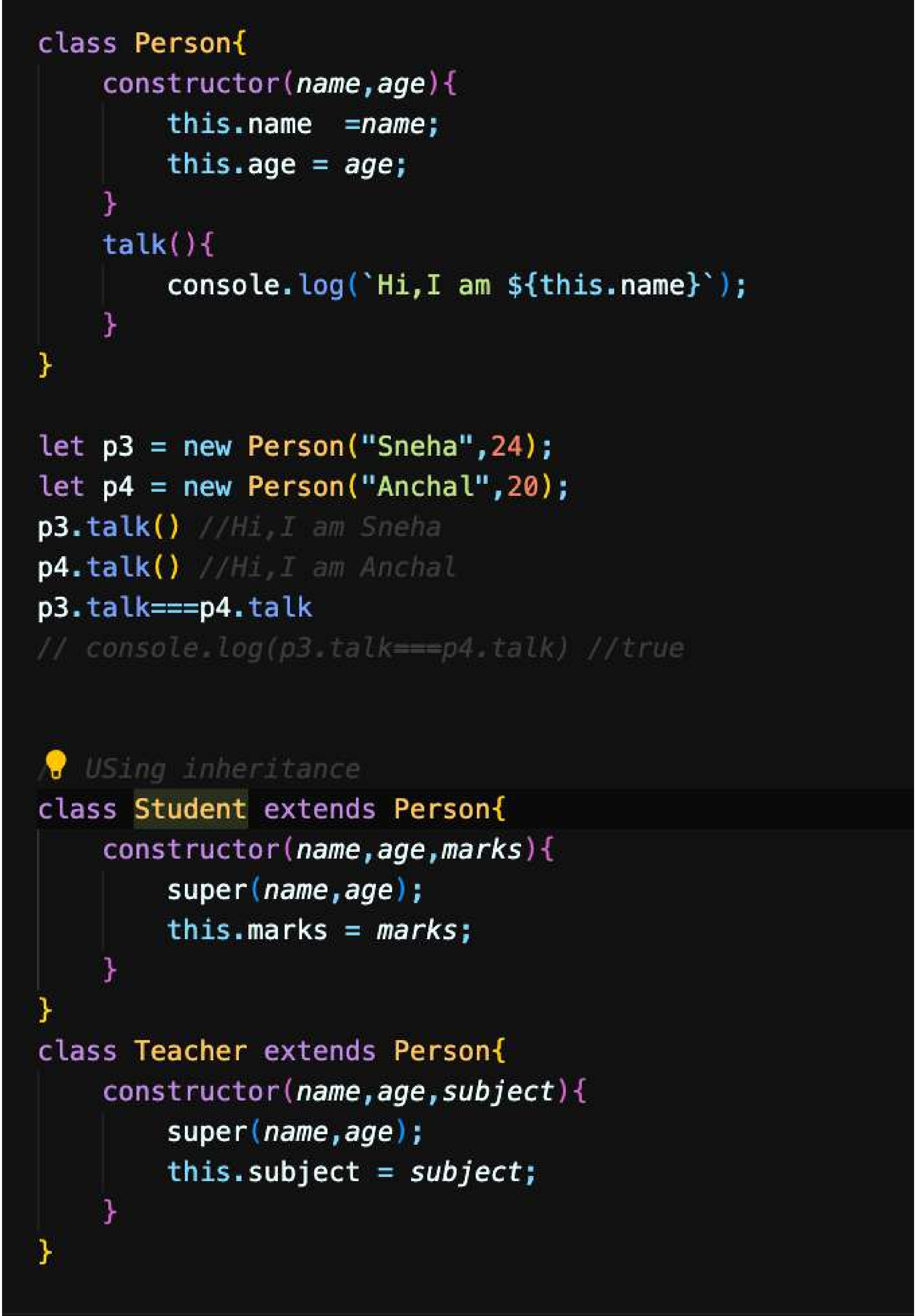


```javascript
class Student {
  constructor(name, age, marks) {
    this.name = name;
    this.age = age;
    this.marks = marks;
  }
  talk() {
    console.log(`Hi, I am ${this.name}`);
  }
}

let stu1 = new Student("adam", 25, 95);

class Teacher {
  constructor(name, age, subject) {
    this.name = name;
    this.age = age;
    this.subject = subject;
  }
  talk() {
    console.log(`Hi, I am ${this.name}`);
  }
}
```

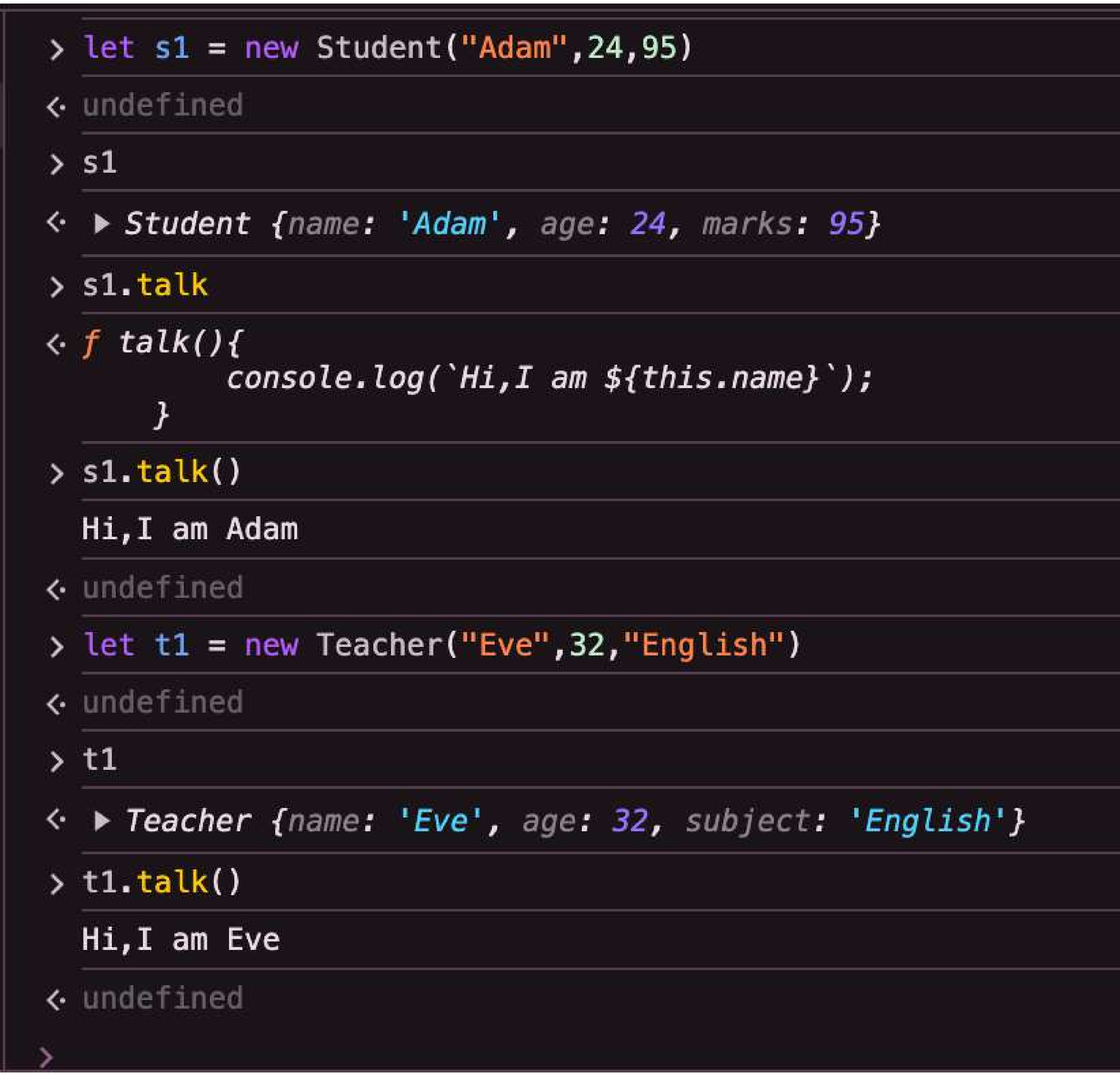

parent class (base class)
inherit
child class



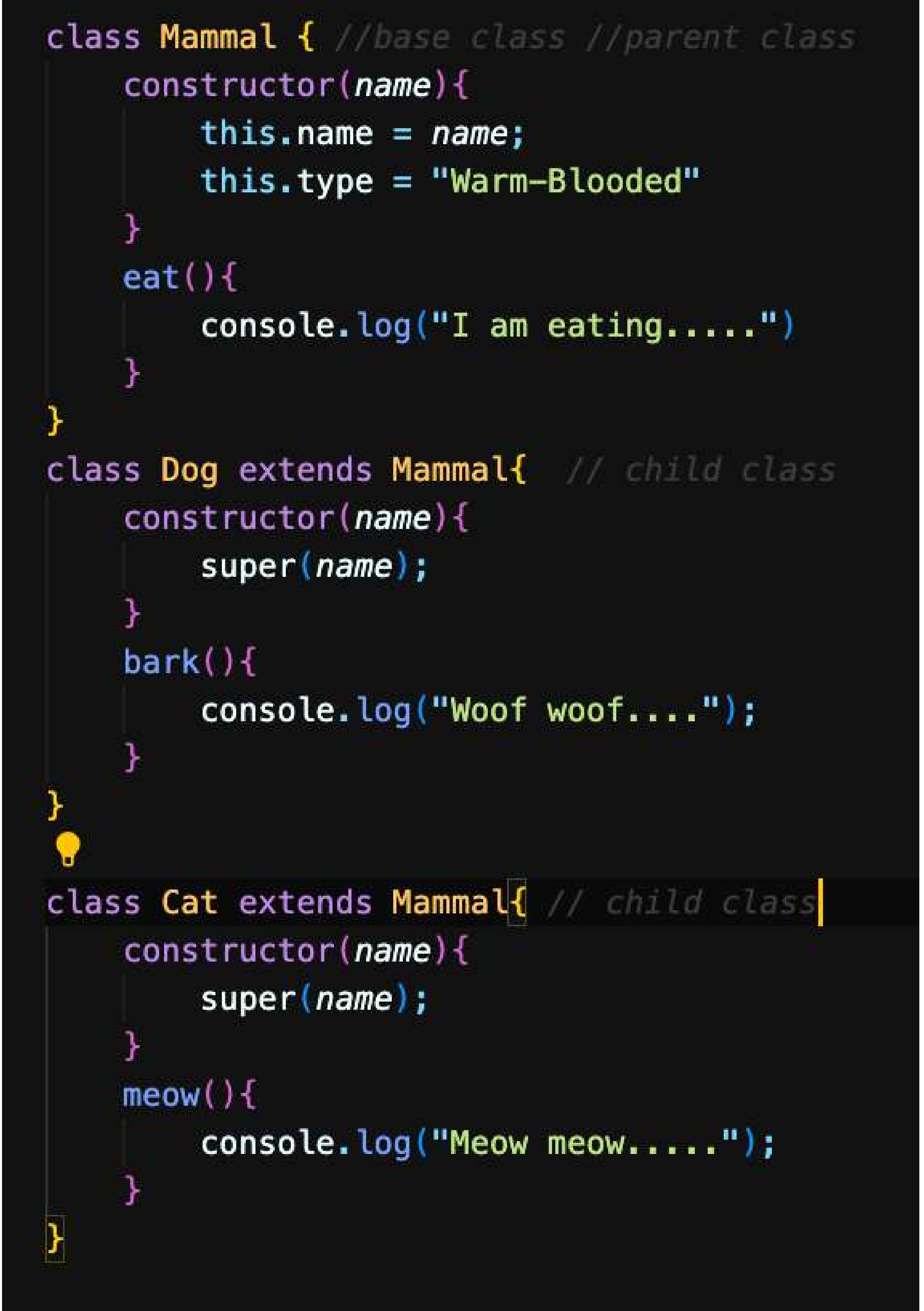
```javascript
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  talk() {
    console.log(`Hi, I am ${this.name}`);
  }
}

class Student extends Person {
  constructor(name, age, marks) {
    super(name, age); //parent class constructor is being called
    this.marks = marks;
  }
}

class Teacher extends Person {
  constructor(name, age, subject) {
    super(name, age); //parent class constructor is being call
    this.subject = subject;
  }
}
```



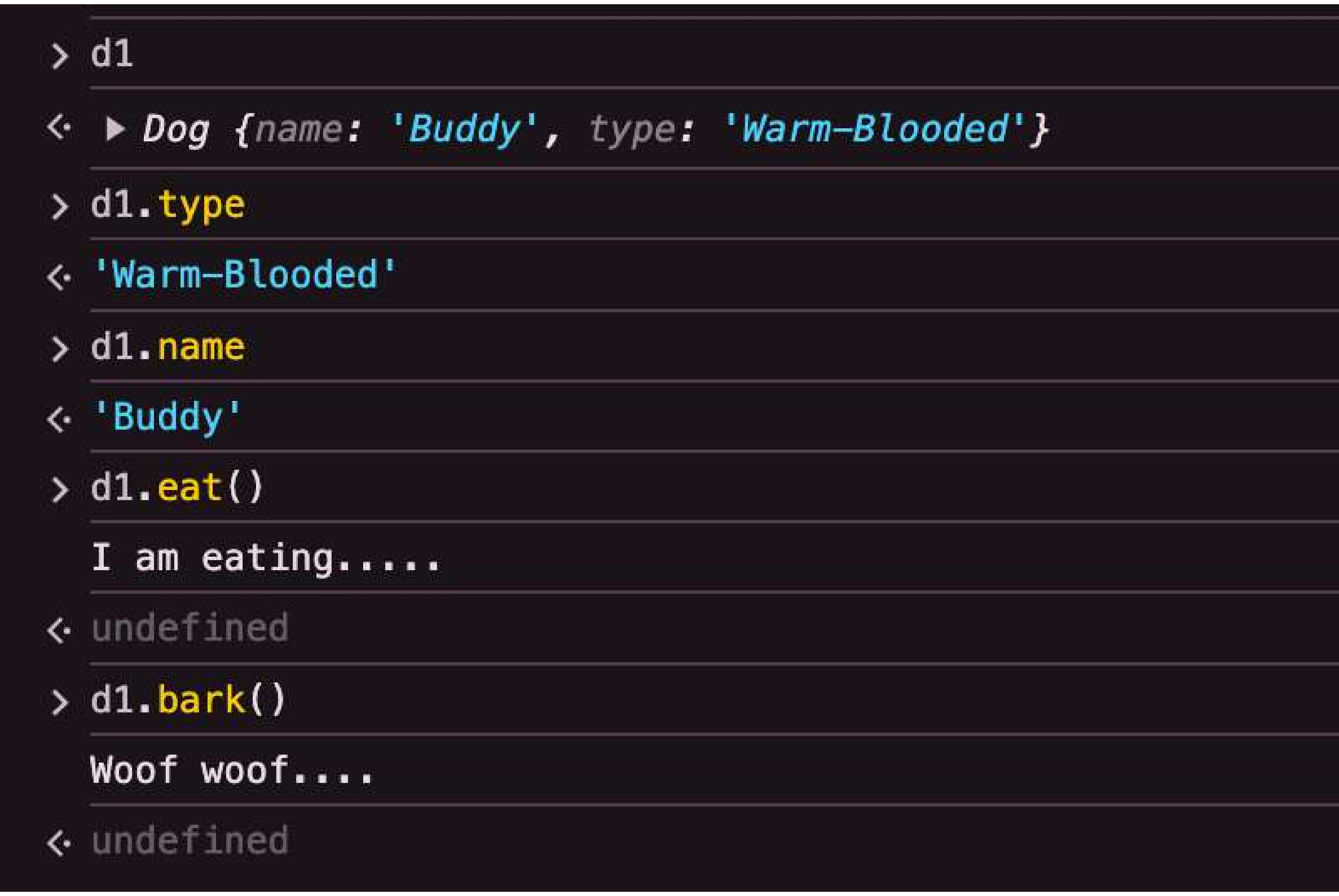
```
class Person{
    constructor(name,age){
        this.name  =name;
        this.age = age;
    }
    talk(){
        console.log(`Hi,I am ${this.name}`);
    }
}

let p3 = new Person("Sneha",24);
let p4 = new Person("Anchal",20);
p3.talk() //Hi,I am Sneha
p4.talk() //Hi,I am Anchal
p3.talk===p4.talk
// console.log(p3.talk===p4.talk) //true


// USing inheritance
class Student extends Person{
    constructor(name,age,marks){
        super(name,age);
        this.marks = marks;
    }
}
class Teacher extends Person{
    constructor(name,age,subject){
        super(name,age);
        this.subject = subject;
    }
}
```



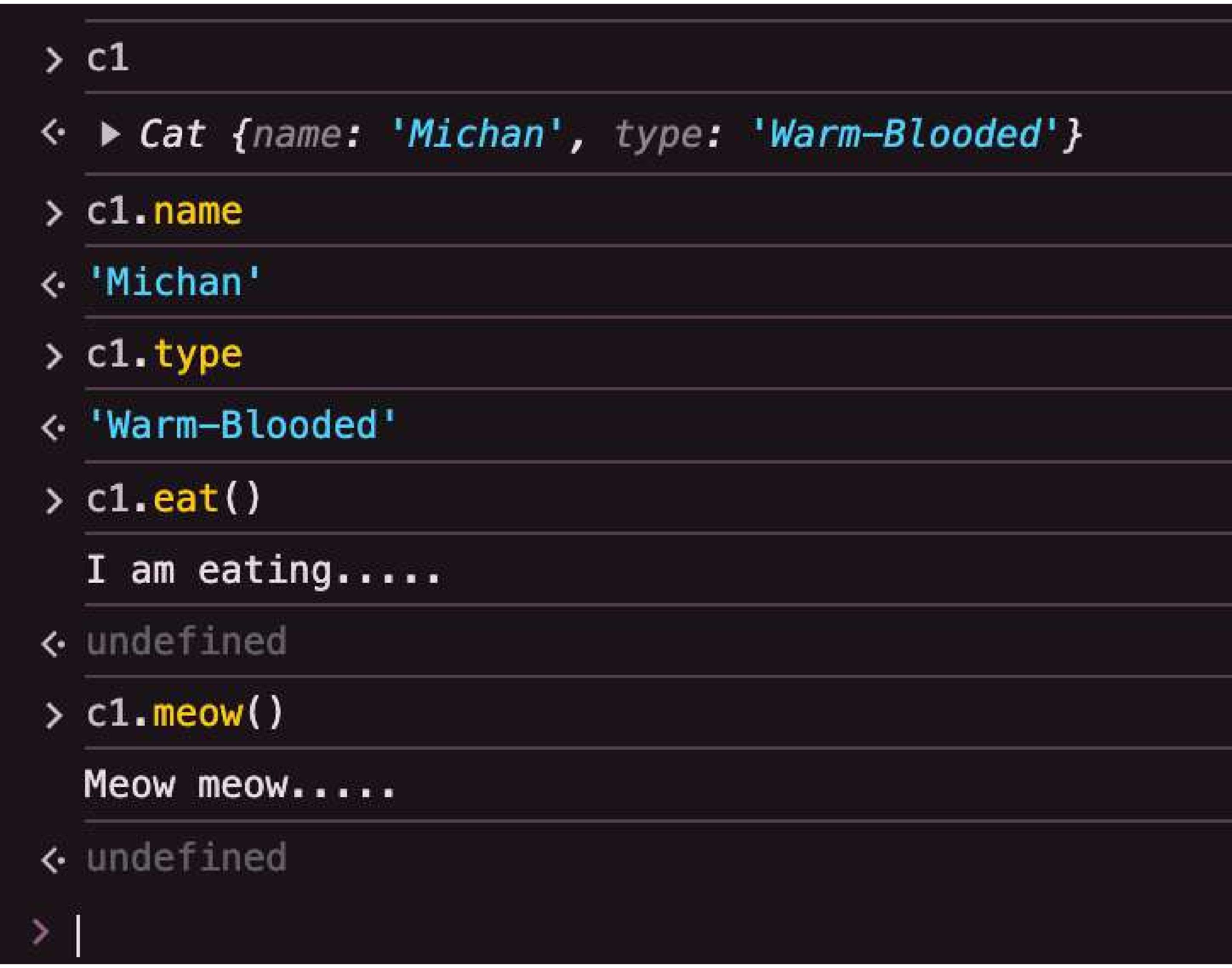
```
> let s1 = new Student("Adam",24,95)
<· undefined
> s1
<· ▶ Student {name: 'Adam', age: 24, marks: 95}
> s1.talk
<· ƒ talk(){
          console.log(`Hi,I am ${this.name}`);
     }
> s1.talk()
  Hi,I am Adam
<· undefined
> let t1 = new Teacher("Eve",32,"English")
<· undefined
> t1
<· ▶ Teacher {name: 'Eve', age: 32, subject: 'English'}
> t1.talk()
  Hi,I am Eve
<· undefined
>
```



```
class Mammal { //base class //parent class
    constructor(name){
        this.name = name;
        this.type = "Warm-Blooded"
    }
    eat(){
        console.log("I am eating.....")
    }
}
class Dog extends Mammal{  // child class
    constructor(name){
        super(name);
    }
    bark(){
        console.log("Woof woof....");
    }
}

class Cat extends Mammal{ // child class
    constructor(name){
        super(name);
    }
    meow(){
        console.log("Meow meow.....");
    }
}
```



```
> d1
<· ▶ Dog {name: 'Buddy', type: 'Warm-Blooded'}
> d1.type
<· 'Warm-Blooded'
> d1.name
<· 'Buddy'
> d1.eat()
  I am eating.....
<· undefined
> d1.bark()
  Woof woof....
<· undefined
```



```
> c1
<· ▶ Cat {name: 'Michan', type: 'Warm-Blooded'}
> c1.name
<· 'Michan'
> c1.type
<· 'Warm-Blooded'
> c1.eat()
  I am eating.....
<· undefined
> c1.meow()
  Meow meow.....
<· undefined
> |
```

APNA COLLEGE

JS (OOP)

Summary Sheet

Qs1. What is Object Oriented Programming (OOP)?
<u>Ans.</u> Object-Oriented Programming (OOP) is a programming paradigm in computer science that relies on the concept of classes and objects. It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects.

Qs2. What are some benefits of using OOP in JavaScript?
<u>Ans.</u> Some benefits of using OOP in JavaScript includes:

a. Improved code organization (structure of code)
b. Reusability of code
c. Better maintainability of code
d. Closeness to real-world objects

Qs3. What is the difference between an object and a class in JavaScript?
<u>Ans.</u> Objects in JS is a standalone entity, with properties, methods and a type. It can be created directly from functions or through constructor functions.
Class in JS acts as a blueprint for creating objects.

Qs4. What is a constructor function in JS?
<u>Ans.</u> constructor function is a special function that is used to create & initialize objects in JS. When a new object is created using a constructor function, it is automatically assigned a set of properties and methods that are defined within the function.

Qs5. What is a prototype chain in JavaScript?
<u>Ans.</u> Every object in JavaScript has a built-in property, which is called its prototype. The prototype is itself an object, so the prototype will have its own prototype, making

what's called a prototype chain. The chain ends when we reach a prototype that has null for its own prototype.

Qs6. What is the difference between a constructor and a class in JavaScript?
Ans. A constructor is a function that creates an object, while a class is a blueprint for creating objects. Classes define the framework whereas, constructor actually creates the objects & initializes them.
(In JavaScript, classes are syntactic sugar over constructor functions.)

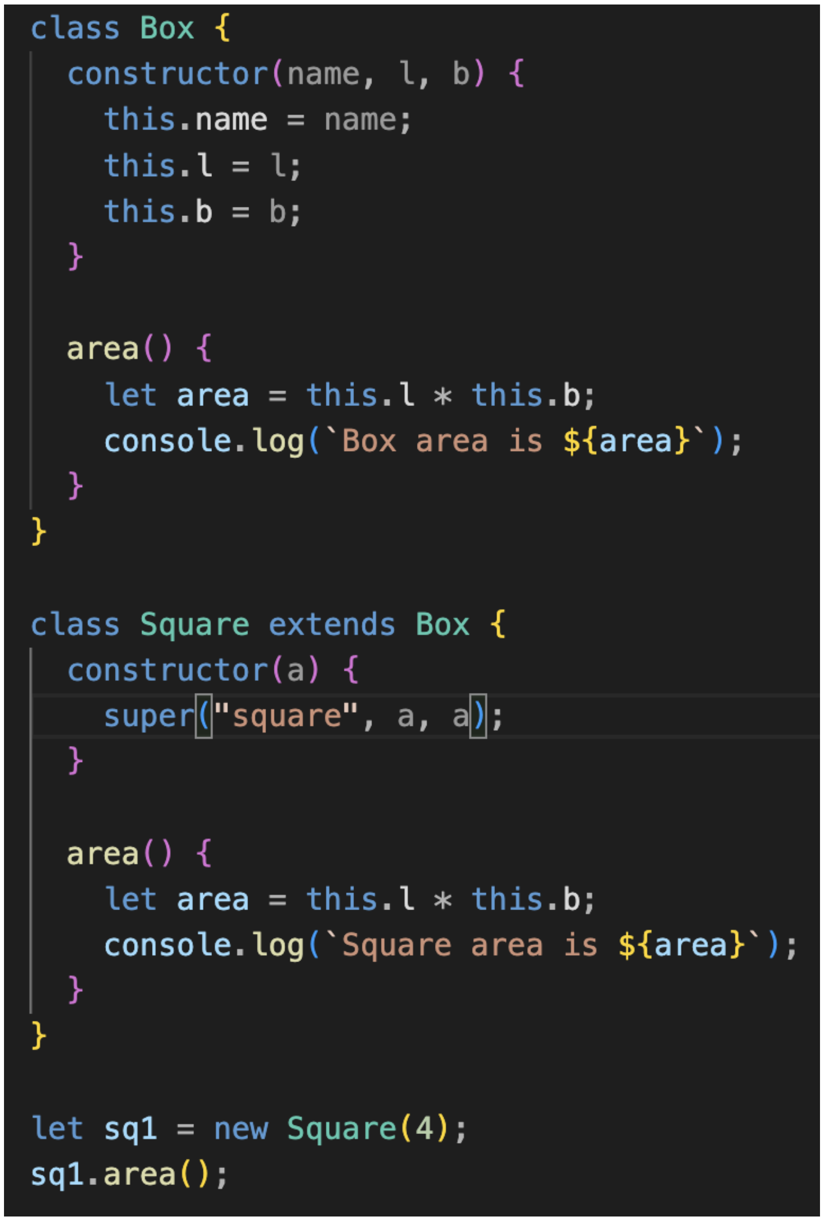
Qs7. Why is the “new” keyword used in JavaScript?
Ans. The 'new' keyword is used to create an instance of an object. When used with a constructor function, it creates a new object and sets the constructor function's 'this' keyword to point to the new object.

Qs8. What is Inheritance in OOP?
Ans. Inheritance in OOP is defined as the ability of a class to derive properties and characteristics from another class while having its own properties as well.

Qs9. What is the “super” keyword in JS?
Ans. The super keyword in JavaScript acts as a reference variable to the parent class. It is mainly used when we want to access a variable, method, or constructor in the base class from the derived class.

Qs10. What will be the output for the following code:

APNA COLLEGE



```
class Box {
  constructor(name, l, b) {
    this.name = name;
    this.l = l;
    this.b = b;
  }

  area() {
    let area = this.l * this.b;
    console.log(`Box area is ${area}`);
  }
}

class Square extends Box {
  constructor(a) {
    super("square", a, a);
  }

  area() {
    let area = this.l * this.b;
    console.log(`Square area is ${area}`);
  }
}

let sq1 = new Square(4);
sq1.area();
```

Ans. The output will be “Square area is 16” as the child class (Square) implementation of area() function will override parent class (Box) implementation of the function with the same name.