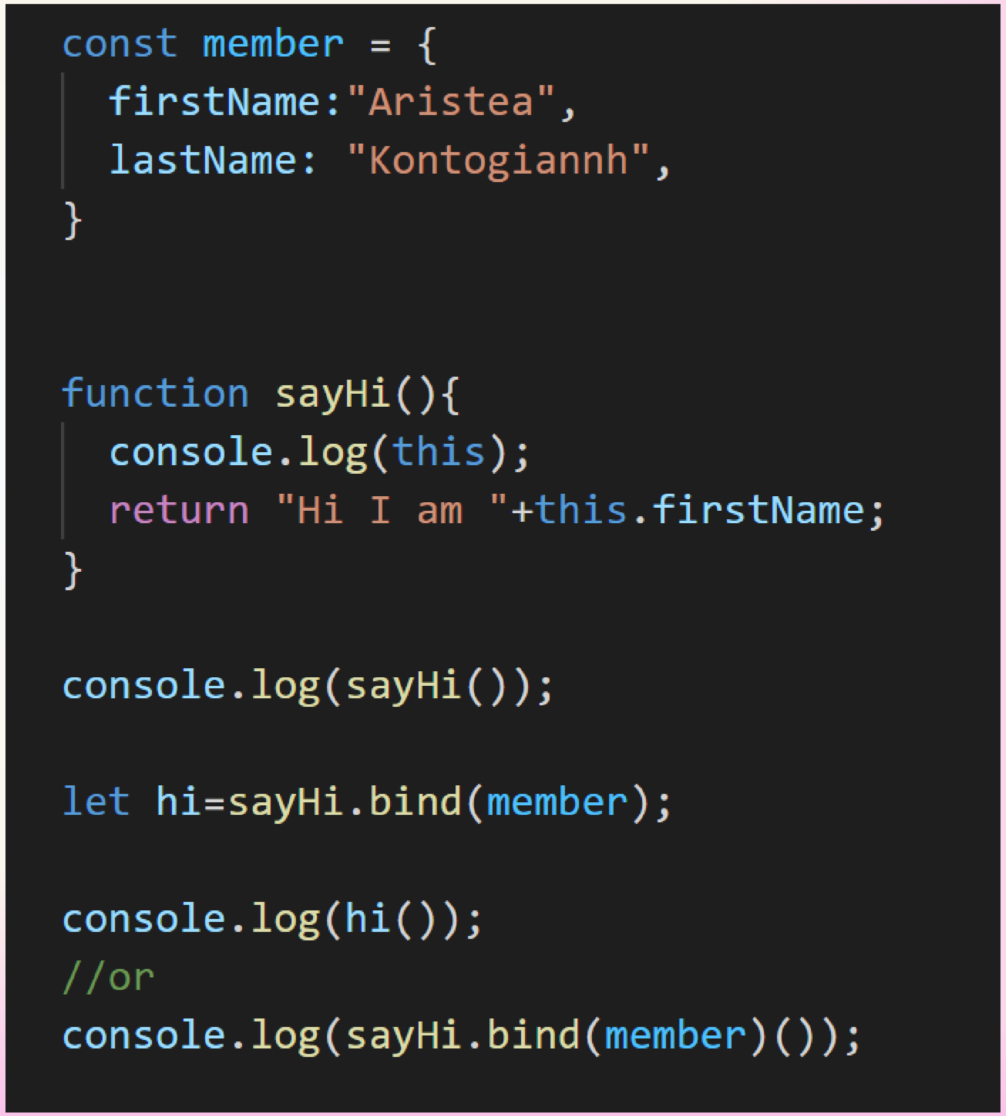
This, objects & more…
Part 2

Bind method

- Η μέθοδος bind(): **δημιουργεί** μια **νέα συνάρτηση** που, όταν καλείται, “δένει” τη λέξη-κλειδί this με μια τιμή που ορίζουμε εμείς

Σύνταξη:

- bind(thisArg)
- bind(thisArg, arg1)
- bind(thisArg, arg1, arg2)
- bind(thisArg, arg1, arg2, /* …, */ argN)



```
const member = {
   firstName:"Aristea",
   lastName: "Kontogiannh",
}


function sayHi(){
   console.log(this);
   return "Hi I am "+this.firstName;
}

console.log(sayHi());

let hi=sayHi.bind(member);

console.log(hi());
//or
console.log(sayHi.bind(member)());
```

Bind method

- Ας δούμε πως θα μπορούσαμε να χρησιμοποιήσουμε τη μέθοδο bind() με μια μέθοδο ενός αντικειμένου.

Call vs Apply vs Bind

call: **binds** the **this** value, **calls** the function, and accepts a list of arguments

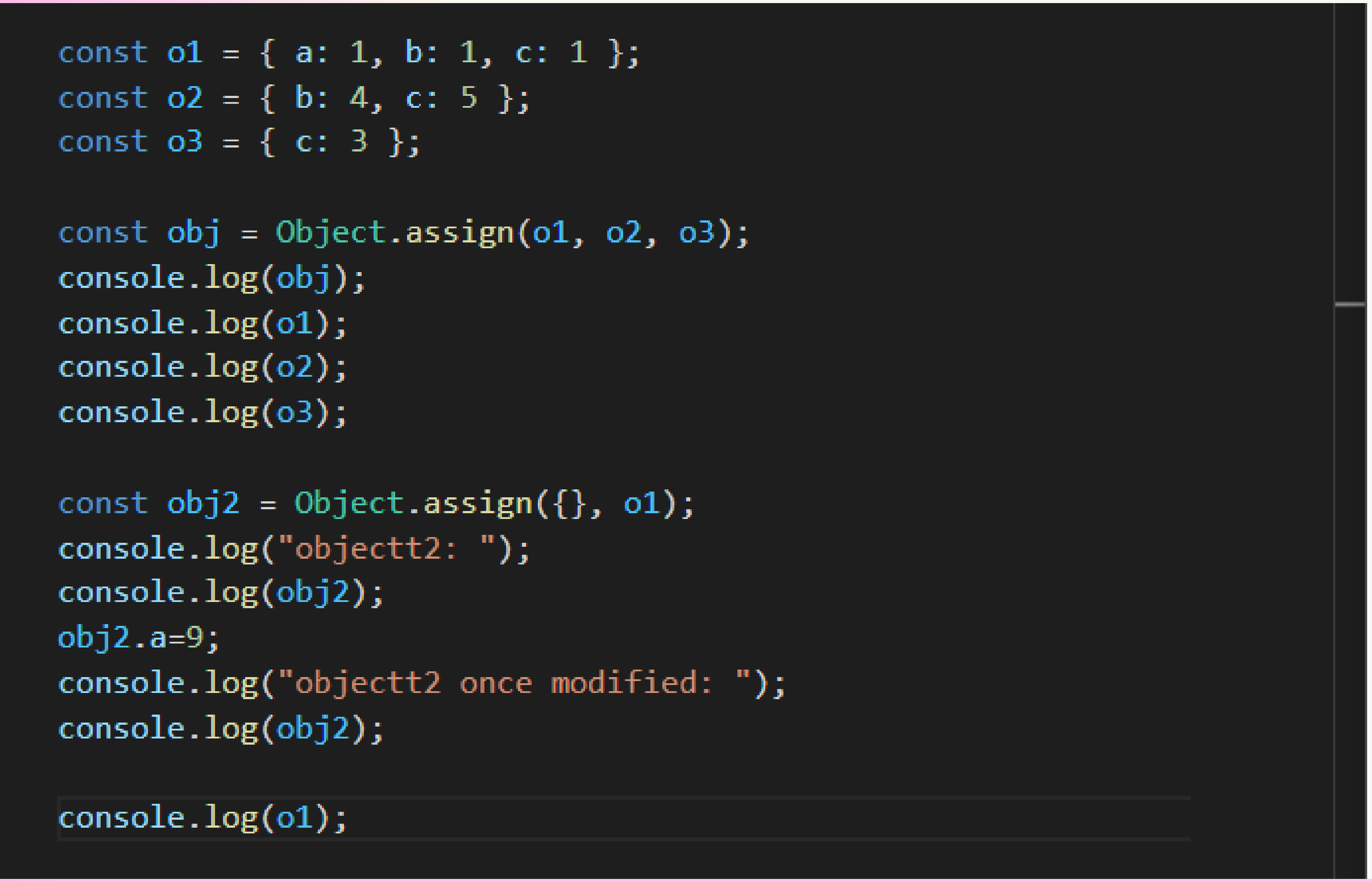
apply: **binds** the **this** value, **calls** the function, and accepts arguments as an **array**

bind: **binds** the **this** value, **returns** a new function (we still need to separately invoke the returned function), and accepts a list of arguments.

Object.assign()

- Object.assign() method-> is used to copy the values and properties from one or more source objects to a target object
- Object.assign() is used for cloning an object.
- Object.assign() is used to merge object with same properties.

Object.assign()

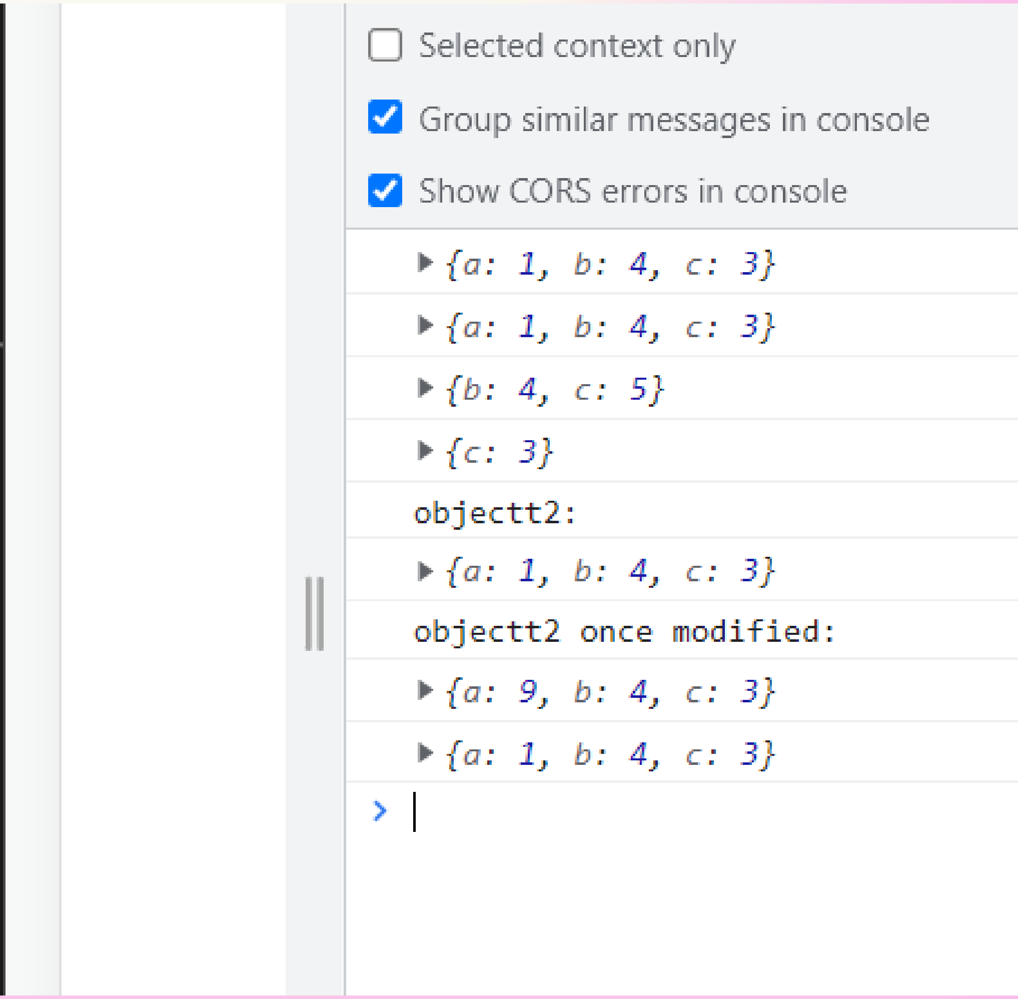


```javascript
const o1 = { a: 1, b: 1, c: 1 };
const o2 = { b: 4, c: 5 };
const o3 = { c: 3 };

const obj = Object.assign(o1, o2, o3);
console.log(obj);
console.log(o1);
console.log(o2);
console.log(o3);

const obj2 = Object.assign({}, o1);
console.log("objectt2: ");
console.log(obj2);
obj2.a=9;
console.log("objectt2 once modified: ");
console.log(obj2);

console.log(o1);
```



☐ Selected context only
☑ Group similar messages in console
☑ Show CORS errors in console

```
▶ {a: 1, b: 4, c: 3}
▶ {a: 1, b: 4, c: 3}
▶ {b: 4, c: 5}
▶ {c: 3}
objectt2:
▶ {a: 1, b: 4, c: 3}
objectt2 once modified:
▶ {a: 9, b: 4, c: 3}
▶ {a: 1, b: 4, c: 3}
>
```

Exception handling

```
try {
  iDontExist();
}
catch(e){
  //process error here
  out(e);
}
finally {
  //do some work here
}
```

```
>> ReferenceError: iDontExist is not defined
```

Exception handling

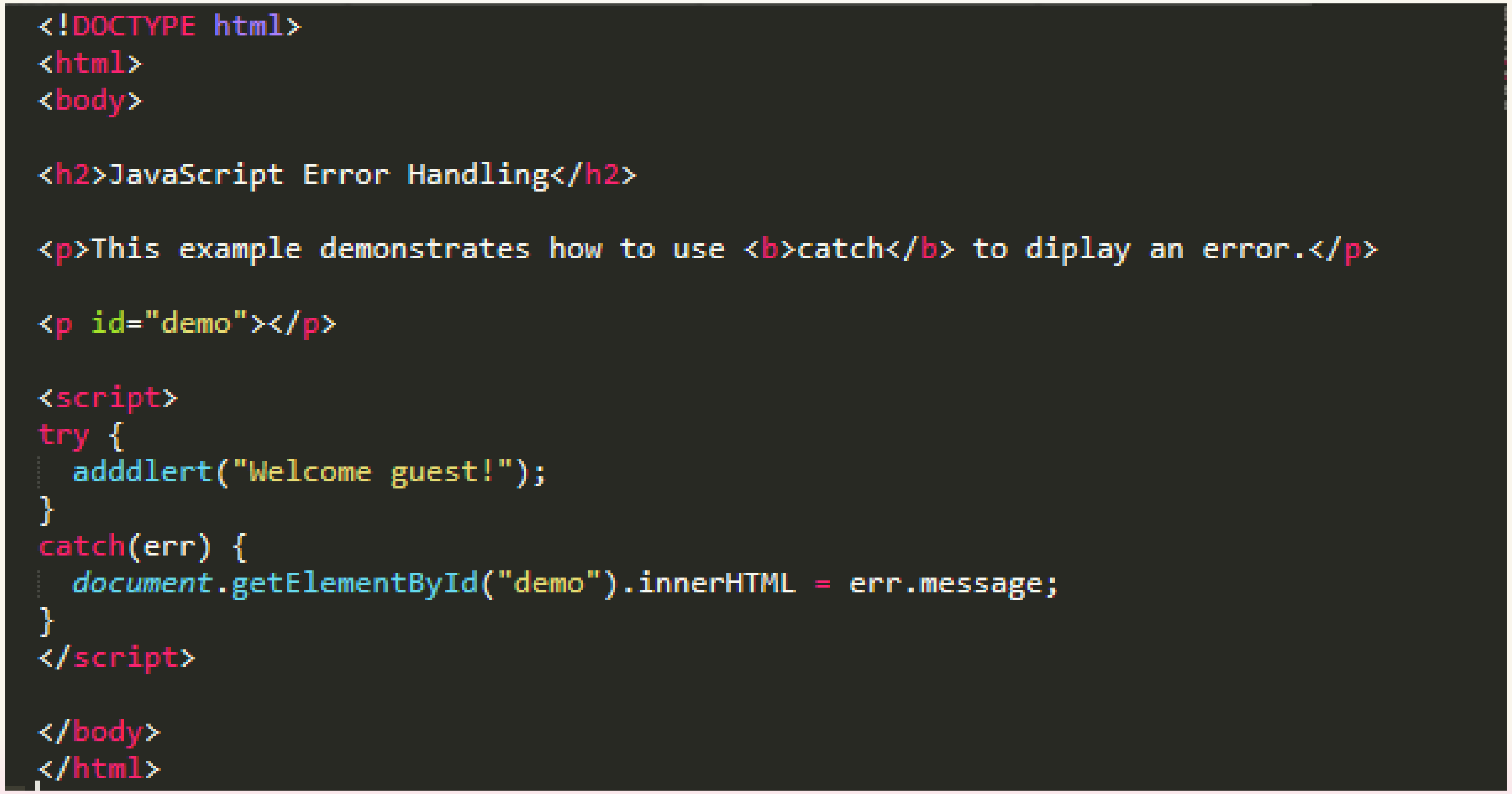
- The **try** statement lets you test a block of code for errors.
- The **catch** statement lets you handle the error.
- The **throw** statement lets you create custom errors.
- The **finally** statement lets you execute code, after try and catch, regardless of the result.

Errors happen for a plethora of reasons!

- To handle them we may use:
- **try statement ->** define a block of code to be tested for errors while it is being executed.
- **catch statement ->** define a block of code to be executed, if an error occurs in the try block.

- Note: use try/catch block when the normal path through the code should proceed without error unless there are truly some exceptional conditions

Example



```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Error Handling</h2>

<p>This example demonstrates how to use <b>catch</b> to diplay an error.</p>

<p id="demo"></p>

<script>
try {
  adddlert("Welcome guest!");
}
catch(err) {
  document.getElementById("demo").innerHTML = err.message;
}
</script>

</body>
</html>
```

Destructuring assignment

- **destructuring assignment** syntax is a **JavaScript expression** that: makes it possible to **unpack values** from arrays, or properties from objects, into distinct variables

- Basic variable assignment with more elements

- const test = ['one', 'two'];

- const [red, yellow, green, blue] = test;
- console.log(red); // "one"
- console.log(yellow); // "two"
- console.log(green); // undefined
- console.log(blue); //undefined

Destructuring assignment

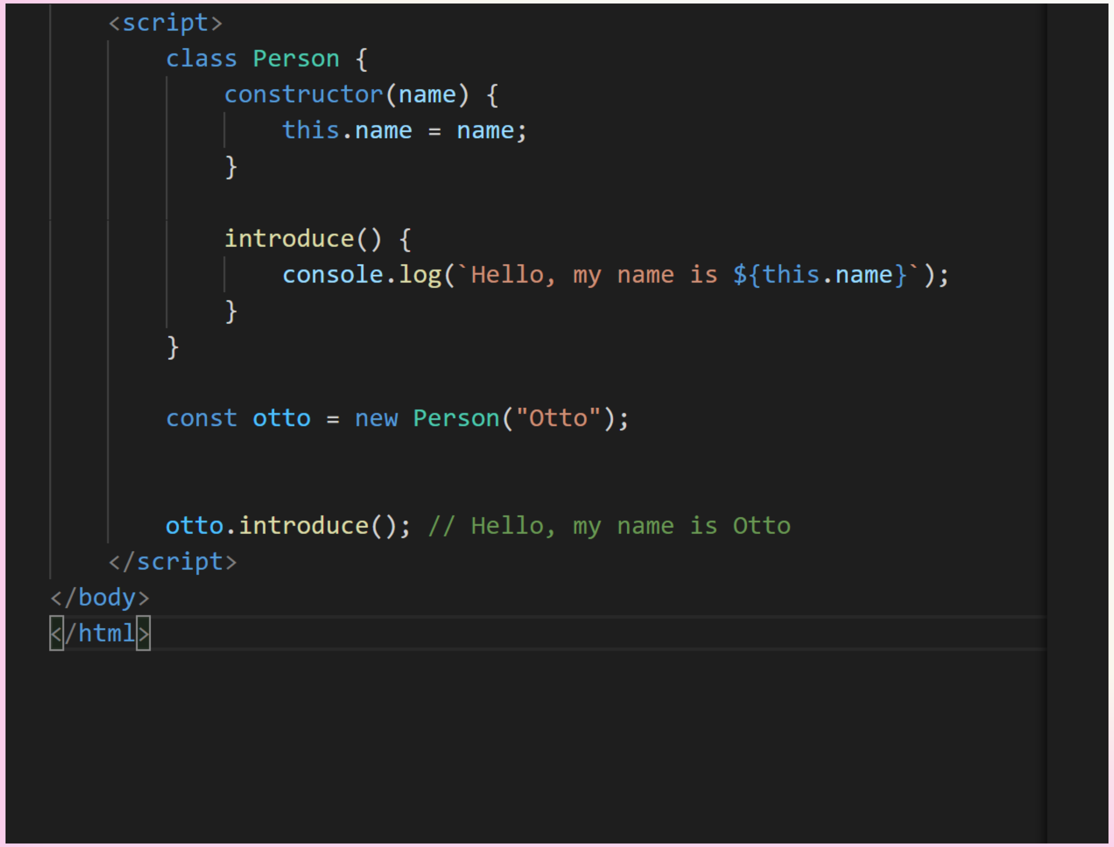
- Check for more

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

Classes

- Οι κλάσεις αποτελούν ένα πρότυπο (template) για τη δημιουργία αντικειμένων (objects).

Class declarations



```
    <script>
        class Person {
            constructor(name) {
                this.name = name;
            }

            introduce() {
                console.log(`Hello, my name is ${this.name}`);
            }
        }

        const otto = new Person("Otto");


        otto.introduce(); // Hello, my name is Otto
    </script>
</body>
</html>
```

constructor =>enables us to provide any custom initialization that must be done before any other methods can be called on an instantiated object.

If we don't provide your own constructor=> then a default constructor will be supplied

We could say that classes are an easier way to write constructors…

More about classes

- https://www.w3schools.com/js/js_class_intro.asp
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this#class_context
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes