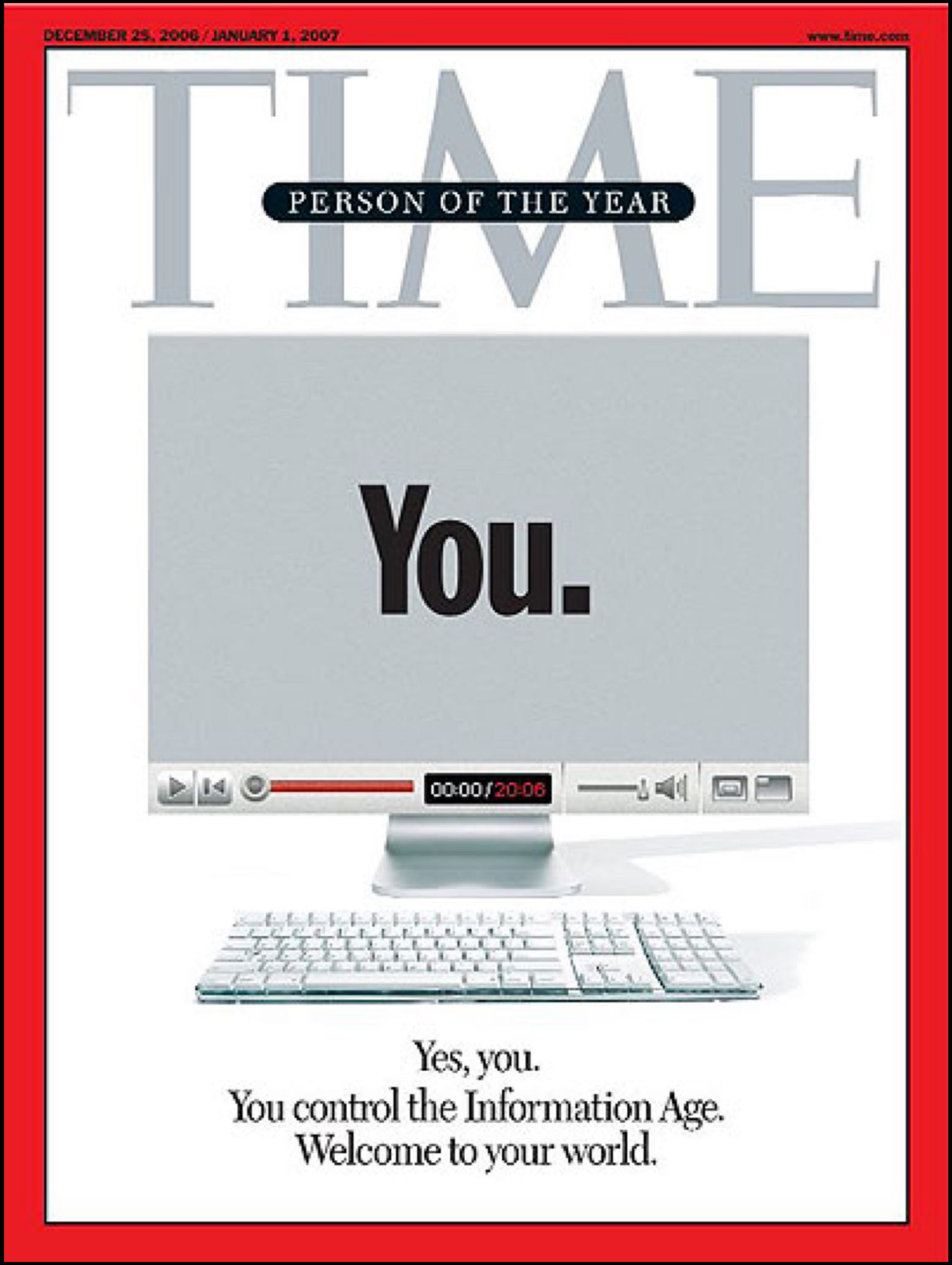
Social Media
Snapchat or Instagram Stories

Jennifer Ringley
Jennicam, 1996 - 2003



Youtube, February 2005

You: Person of the Year
Time Magazine, December 2006

What is Web 2.0

"the network as platform"

- Services, not packaged software, with cost-effective scalability
- Control over unique, hard-to-recreate data sources that get richer as more people use them
- Trusting users as co-developers
- Harnessing collective intelligence
- Leveraging the long tail through customer self-service
- Software above the level of a single device
- Lightweight user interfaces, development models, AND business models

— **Tim O'Reilly**, 2005

Web Squared

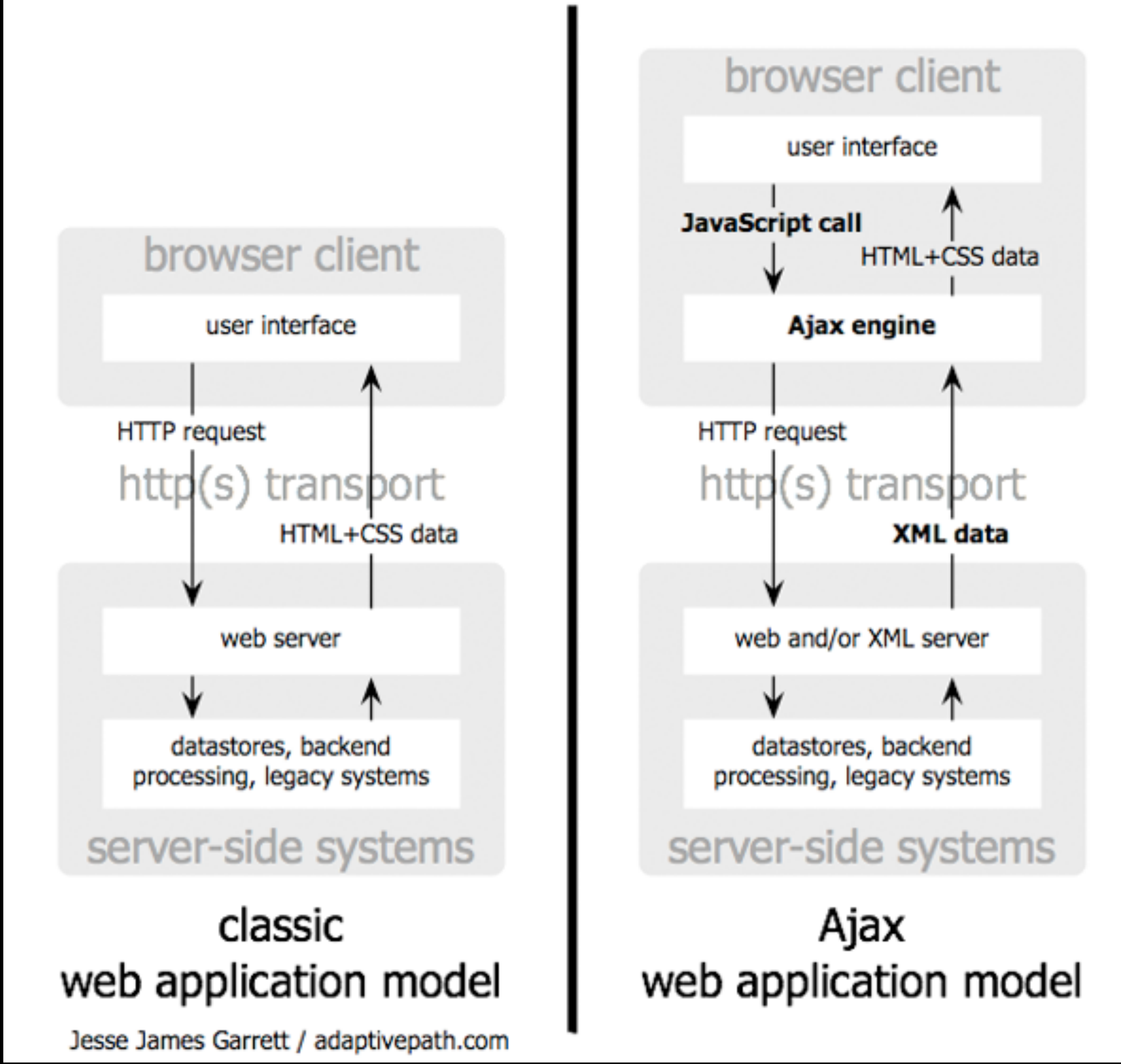
“search as vote”

Modern search engines now use complex algorithms and hundreds of different ranking criteria to produce their results. Among the data sources is the feedback loop generated by the frequency of search terms, the number of user clicks on search results, and our own personal search and browsing history. For example, if a majority of users start clicking on the fifth item on a particular search results page more often than the first, Google’s algorithms take this as a signal that the fifth result may well be better than the first, and eventually adjust the results accordingly.

The collection of technologies used by Google was christened **AJAX**, in a seminal essay by Jesse James Garrett of web design firm Adaptive Path.

"**Ajax** isn't a technology. It's really several technologies, each flourishing in its own right, coming together in powerful new ways.

Ajax incorporates:
- standards-based presentation using **XHTML** and **CSS**;
- dynamic display and interaction using the Document Object Model (DOM);
- data interchange and manipulation using XML and XSLT;
- asynchronous data retrieval using XMLHttpRequest;
- + **JavaScript** binding everything together."



Jesse James Garrett, 2005

(To view this link you'll need the Wayback Machine)

Server Side Programming

Unitl now, we have been learning about *client side* web development. The client side, in this case, is the web browser and the technologies that go into it (HTML, CSS, JavaScript).

Static vs. Dynamic Web Pages

Server side programming languages (e.x. JavaScript, PHP, .NET, Python, Perl) make creating dynamic web pages possible. Using HTML and CSS we have been creating static web pages in this class. Static web pages will always contain the same content or information in them no matter what, until you change the source code of the page. Dynamic web pages are capable of producing different content for a web page or web site using the same source code, based on the application logic written by the developer. (RSS feeds were one of the first instances of this w/ **Web 2.0**).

Dynamic web pages will often be powered by a server side programming language and a database (e.x. JavaScript, MySQL, SQL Server, Oracle). Together, these technologies can determine what information should be returned to the browser and construct dynamic content for the client side to present to the user.

Objects group together a set of variables and functions to create a model of something you would recognize from the real world.

Object properties + methods

In an object:

1. **Variables** become known as **properties**
2. **Functions** become known as **methods**

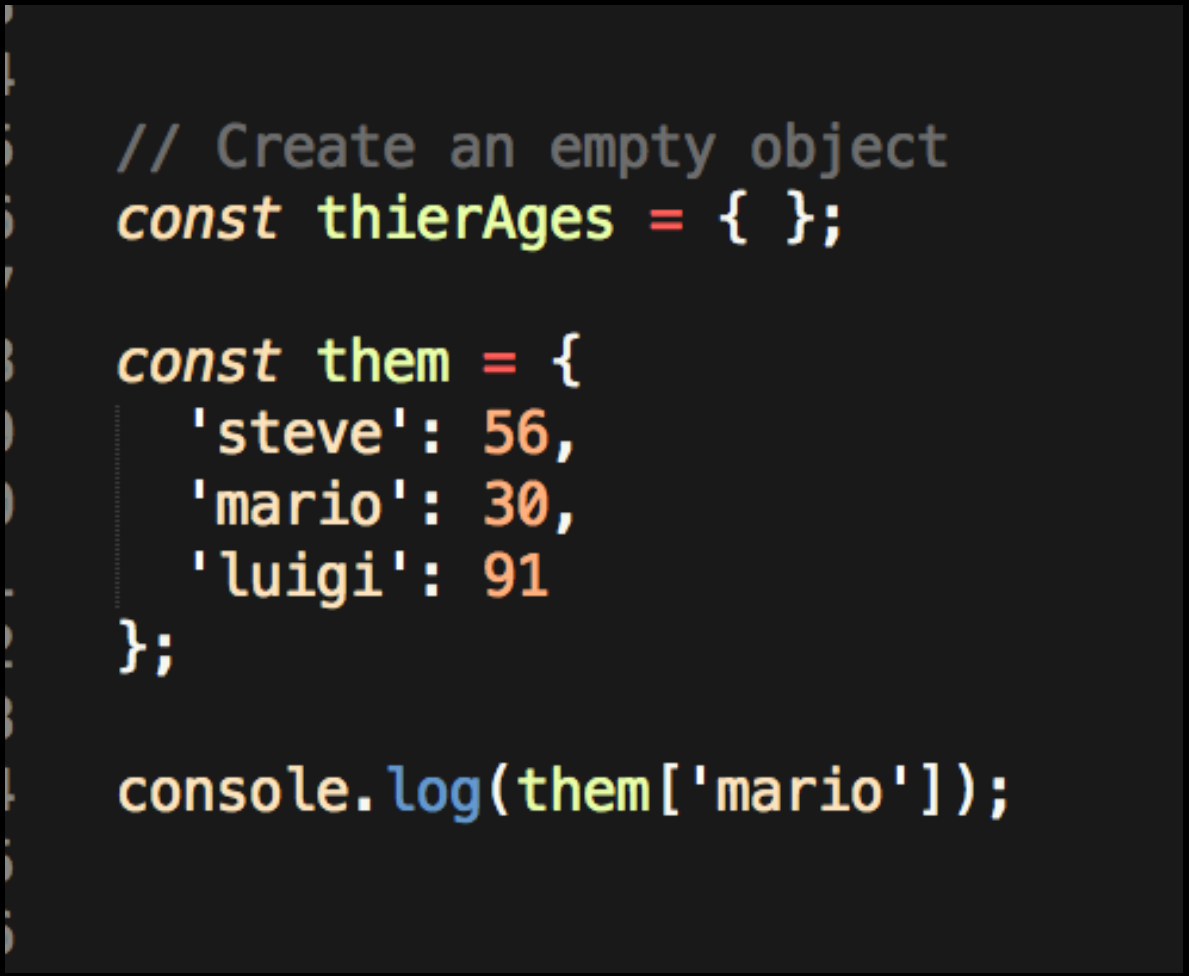
Properties tell us about the object, such as the height of a chair and how many chairs there are

Methods represent tasks that are associated with the object, e.g. count the height of all chairs by adding all heights together

Maps through objects

Every JavaScript object is a collection of property-value pairs. (We'll talk about this more later.)

Therefore you can define maps by creating Objects:

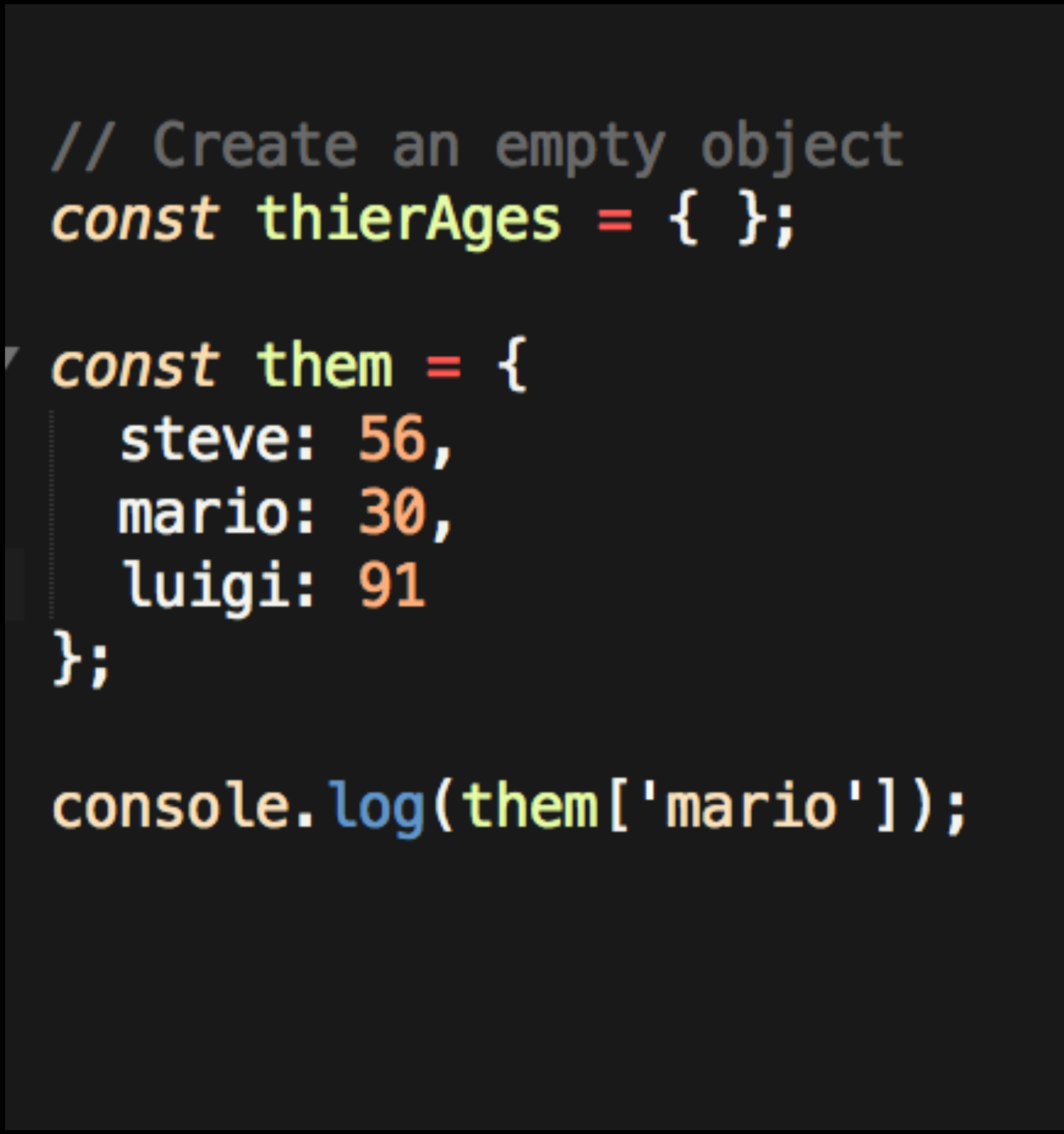


```
// Create an empty object
const thierAges = { };

const them = {
  'steve': 56,
  'mario': 30,
  'luigi': 91
};

console.log(them['mario']);
```

Maps through objects

string keys do not need quotes around them. Without the quotes, the keys are still of type string.

This is the same as the previous slide.



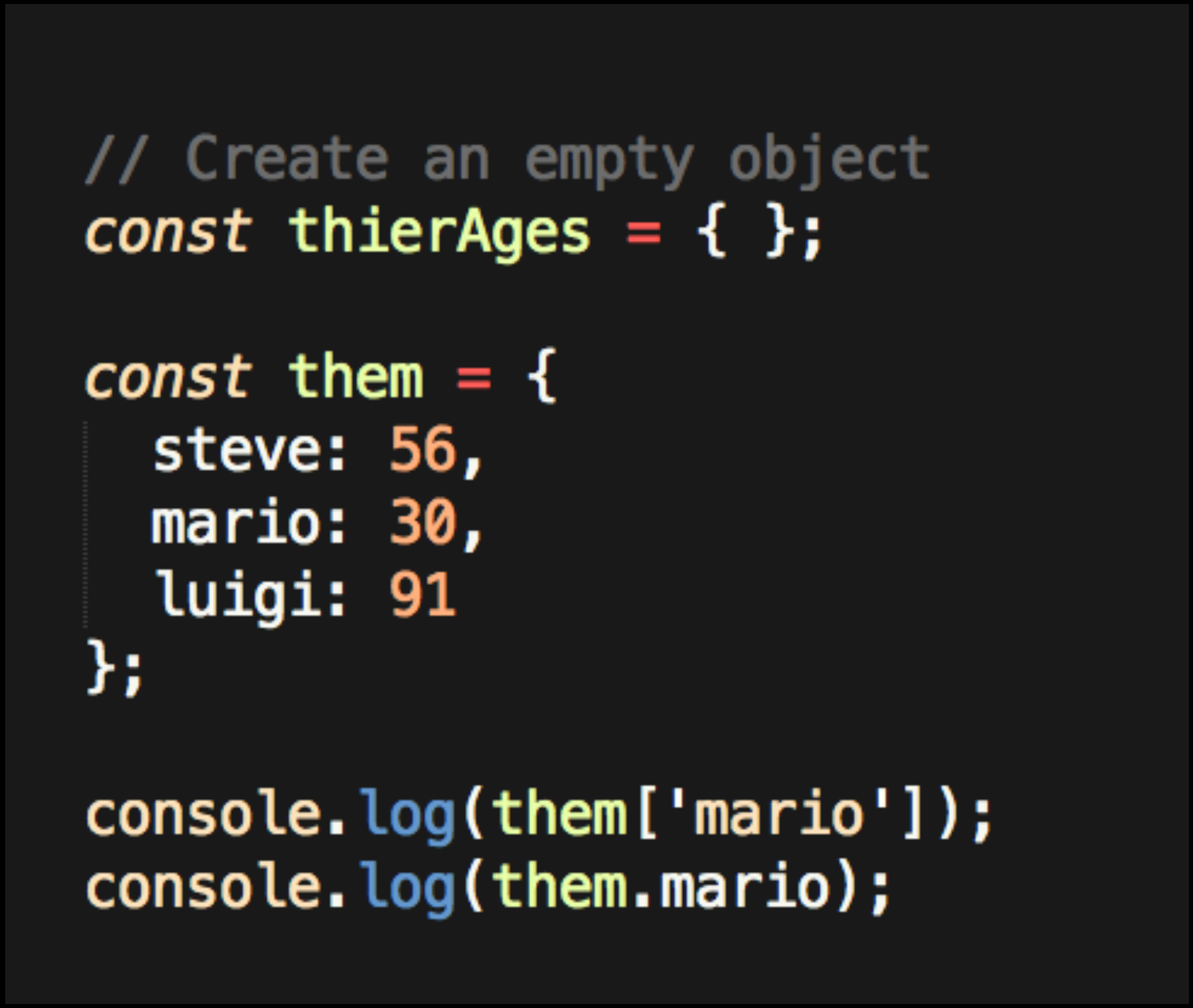
```
// Create an empty object
const thierAges = { };

const them = {
  steve: 56,
  mario: 30,
  luigi: 91
};

console.log(them['mario']);
```

Maps through objects

There are two ways to access the value of a property:

1. objectName[property]
2. objectName.property

(2 only works for string keys.)



```
// Create an empty object
const thierAges = { };

const them = {
  steve: 56,
  mario: 30,
  luigi: 91
};

console.log(them['mario']);
console.log(them.mario);
```

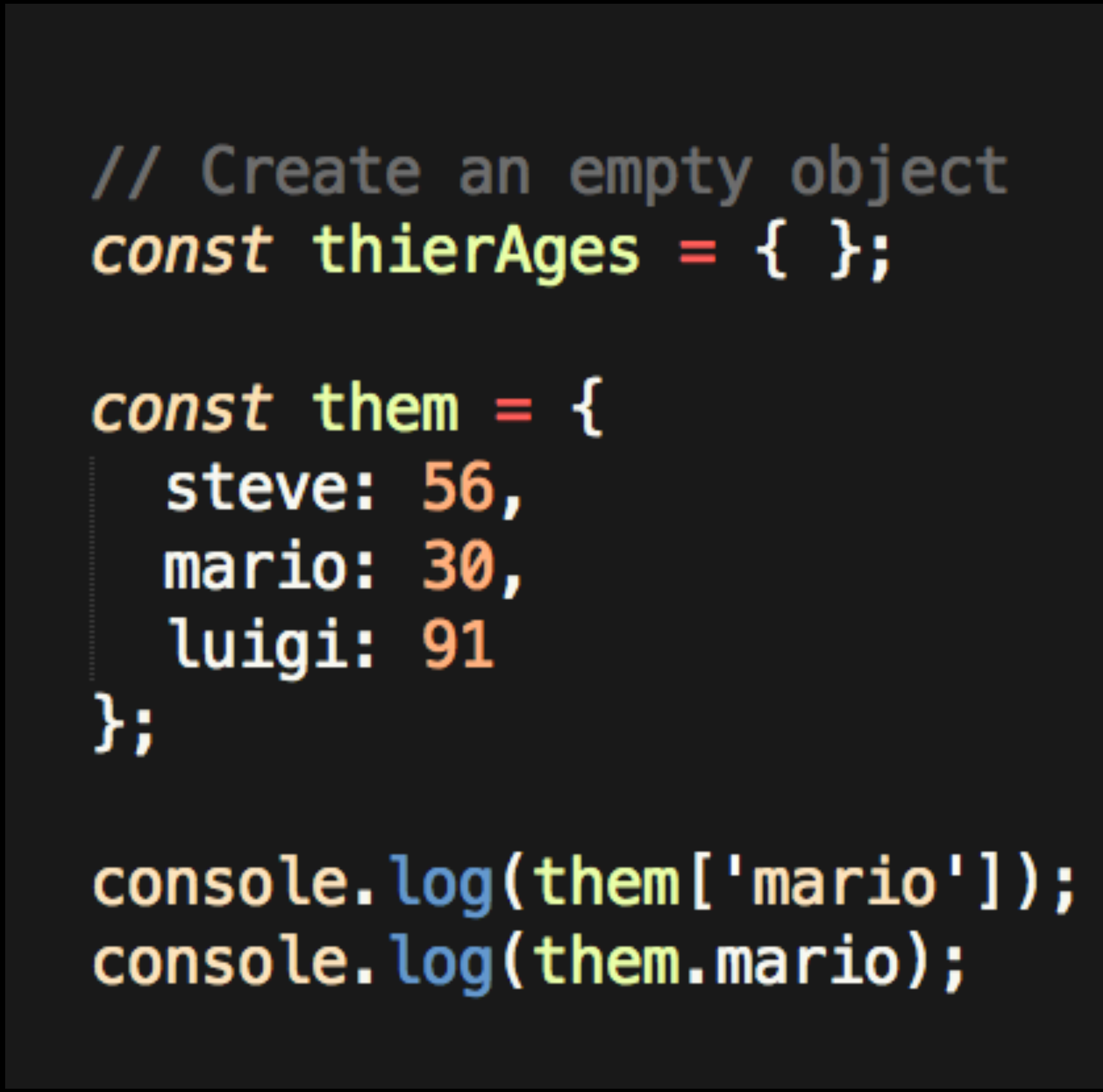
Maps through objects

There are two ways to access the value of a property:

1. objectName[property]
2. objectName.property

(2 only works for string keys.)

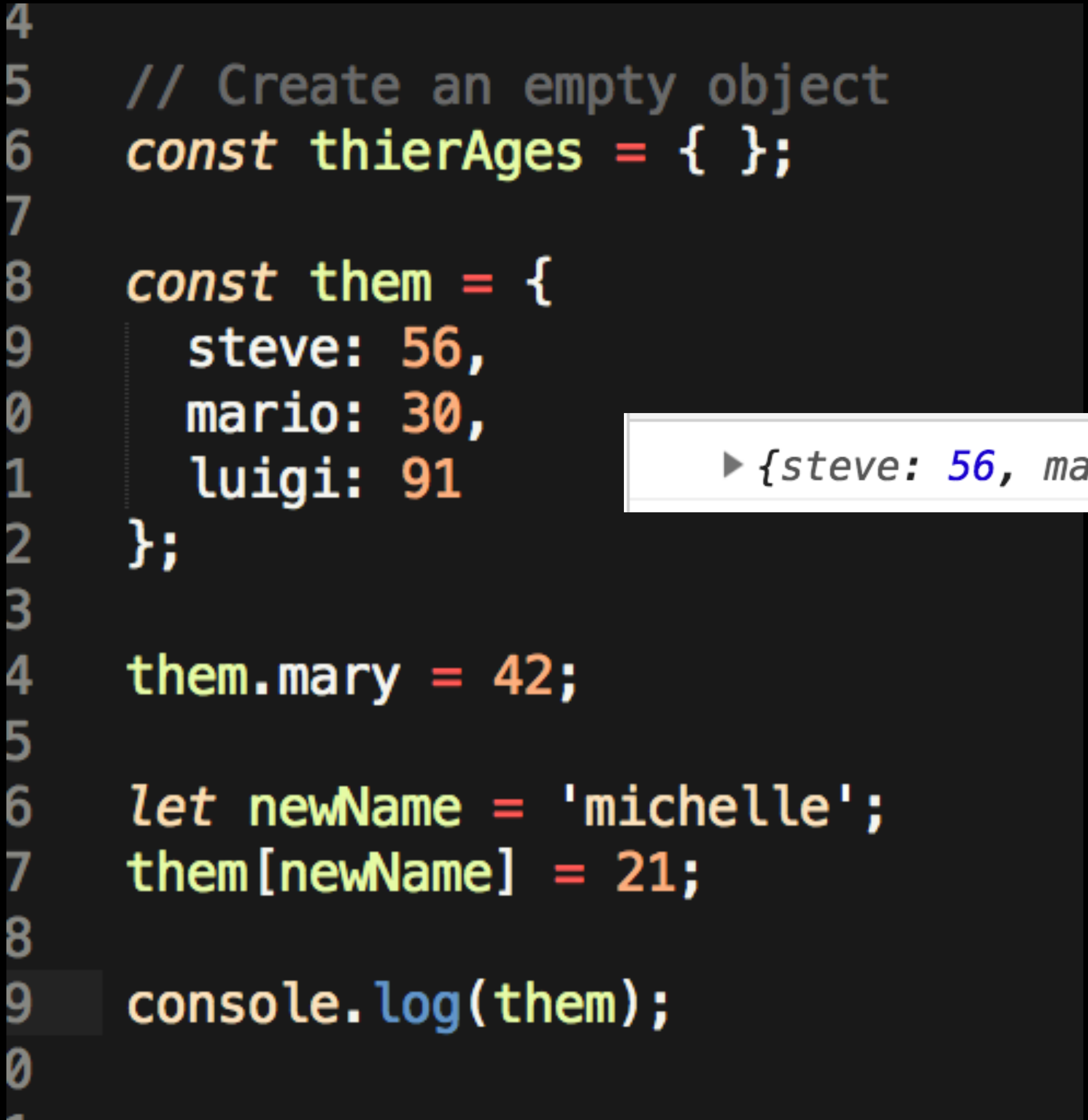
Generally prefer style (2), unless the property is stored in a variable, or if the property is not a string.



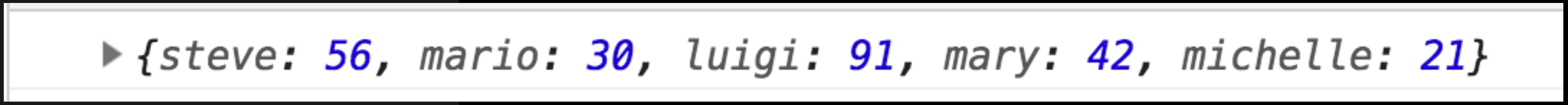
```
// Create an empty object
const thierAges = { };

const them = {
  steve: 56,
  mario: 30,
  luigi: 91
};

console.log(them['mario']);
console.log(them.mario);
```

Maps through objects

To add a property to an object, name the property and give it a value:



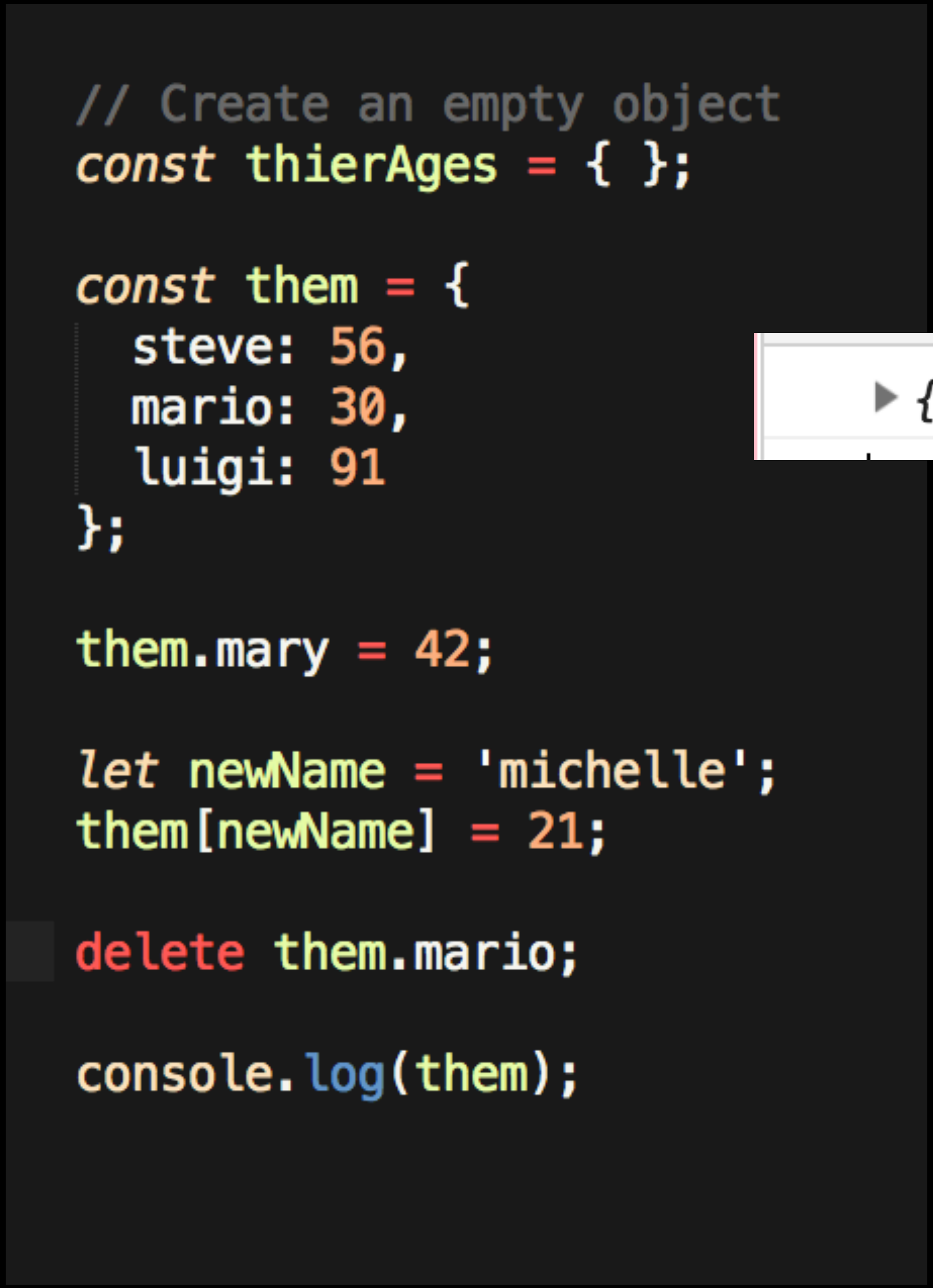
```
// Create an empty object
const thierAges = { };

const them = {
  steve: 56,
  mario: 30,
  luigi: 91
};

them.mary = 42;

let newName = 'michelle';
them[newName] = 21;

console.log(them);
```



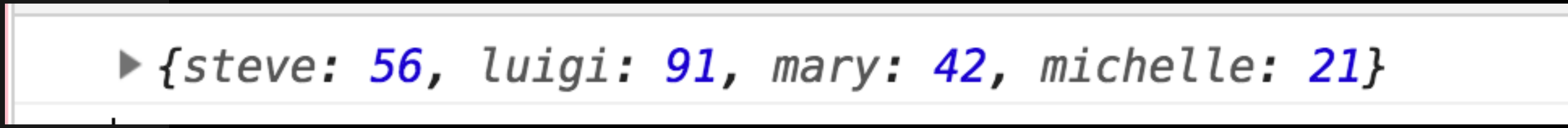
```
▶{steve: 56, mario: 30, luigi: 91, mary: 42, michelle: 21}
```

Maps through objects

To remove a property to an object, use **delete**:



```
// Create an empty object
const thierAges = { };

const them = {
  steve: 56,
  mario: 30,
  luigi: 91
};

them.mary = 42;

let newName = 'michelle';
them[newName] = 21;

delete them.mario;

console.log(them);
```



```
▶ {steve: 56, luigi: 91, mary: 42, michelle: 21}
```

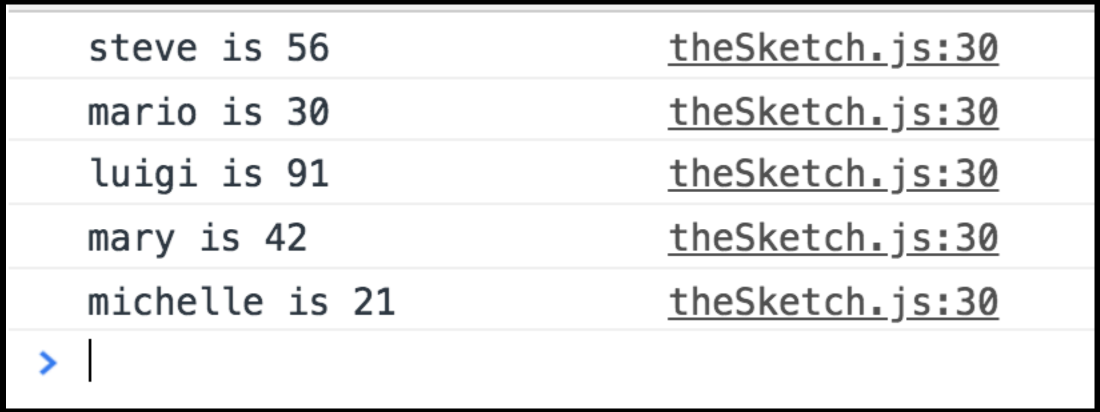
Iterating through Map

Iterate through a map using a for...in loop (mdn): (intuition: for each key in the object) :

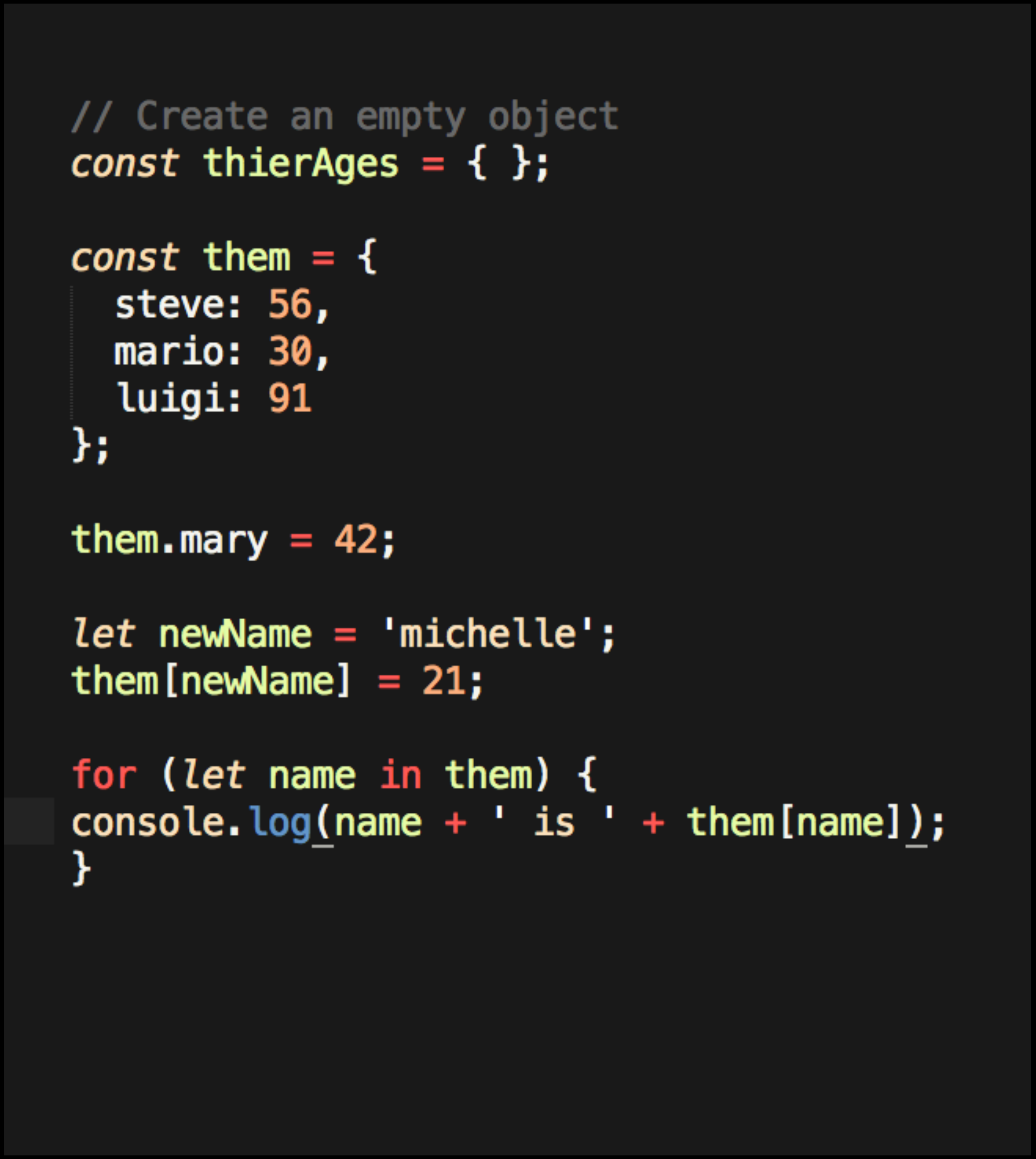
```
for (key in object) {
    // ... do something with object[key]
}
```

- You can't use for...in on lists; only on object types
- You can't use for...of on objects; only on list types

Iterating through Map



```
steve is 56          theSketch.js:30
mario is 30          theSketch.js:30
luigi is 91          theSketch.js:30
mary is 42           theSketch.js:30
michelle is 21       theSketch.js:30
>
```

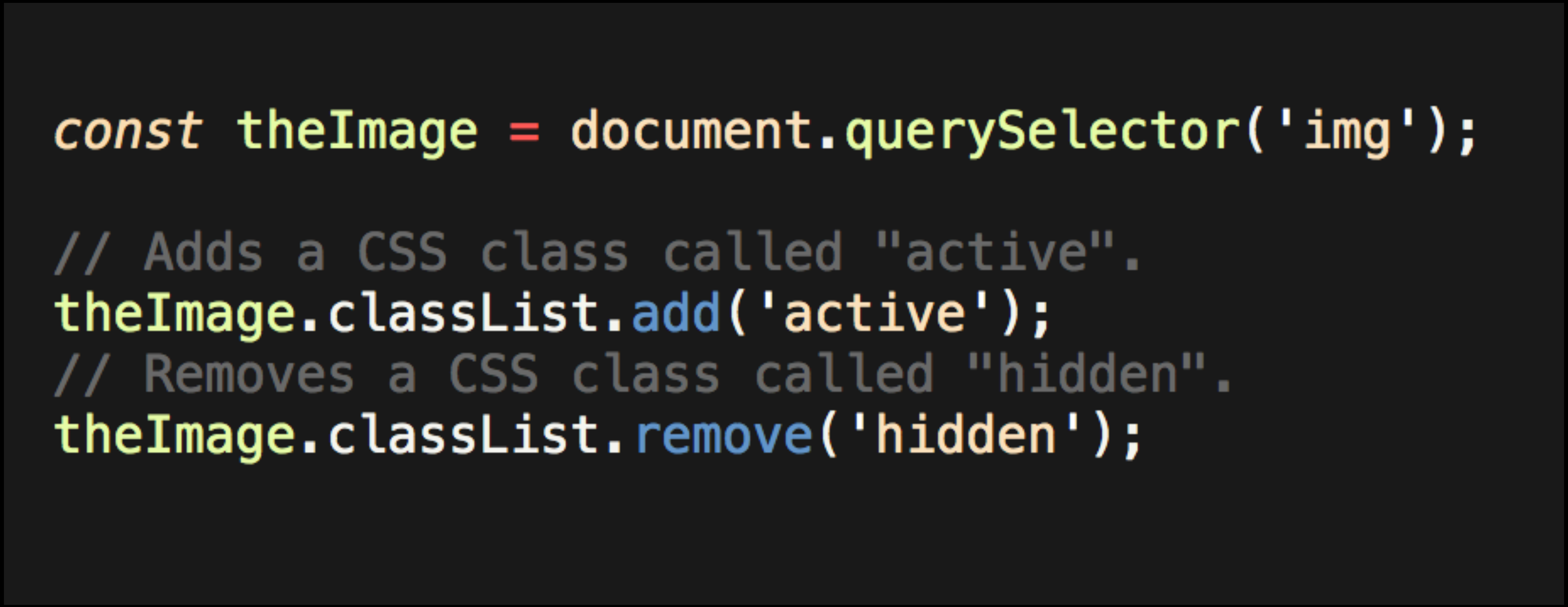


```
// Create an empty object
const thierAges = { };

const them = {
  steve: 56,
  mario: 30,
  luigi: 91
};

them.mary = 42;

let newName = 'michelle';
them[newName] = 21;

for (let name in them) {
console.log(name + ' is ' + them[name]);
}
```

- You can't use **for...in** on lists; only on **object types**
- You can't use **for...of** on objects; only on **list types**

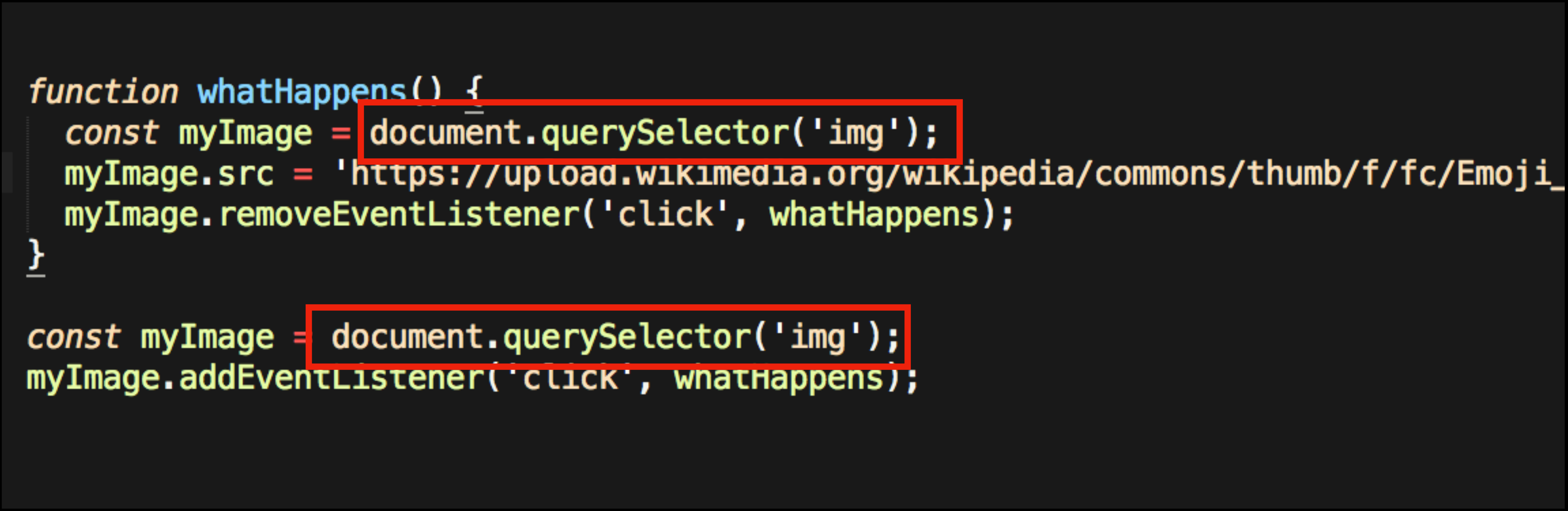
Adding + Removing Classes

You can can control classes applied to an HTML element via **classList.add** and **classList.remove**:



```
const theImage = document.querySelector('img');

// Adds a CSS class called "active".
theImage.classList.add('active');
// Removes a CSS class called "hidden".
theImage.classList.remove('hidden');
```

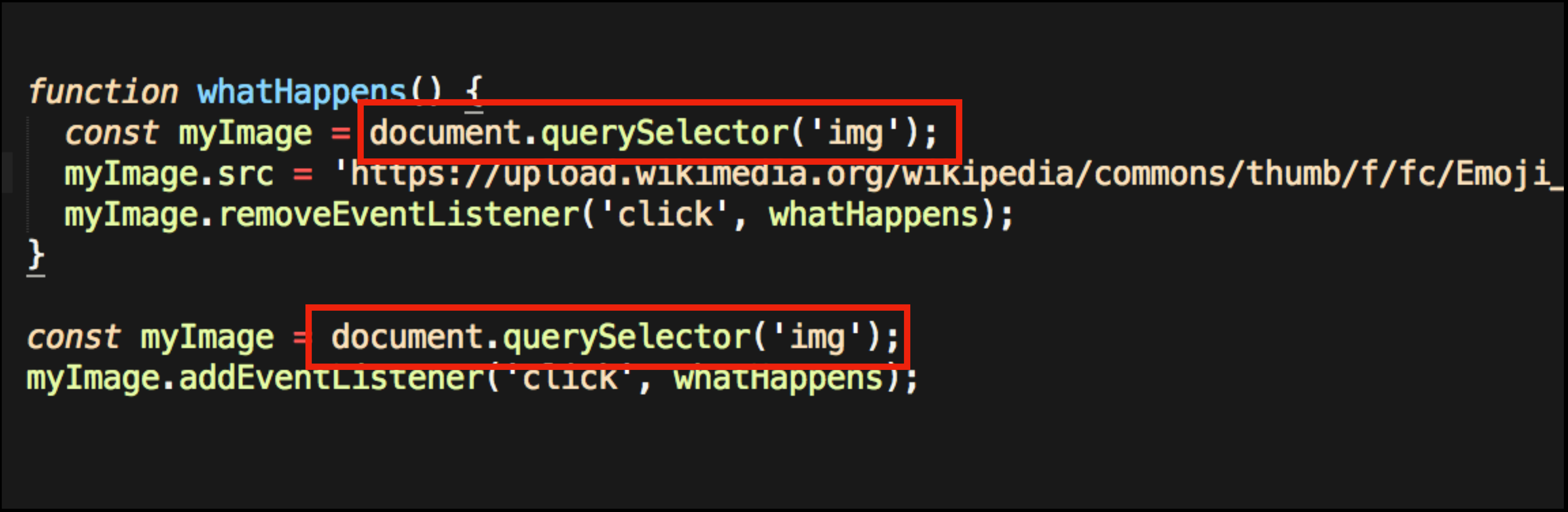
finding the element twice…



```
function whatHappens() {
  const myImage = document.querySelector('img');
  myImage.src = 'https://upload.wikimedia.org/wikipedia/commons/thumb/f/fc/Emoji_
  myImage.removeEventListener('click', whatHappens);
}

const myImage = document.querySelector('img');
myImage.addEventListener('click', whatHappens);
```

This repetition is inelegant.

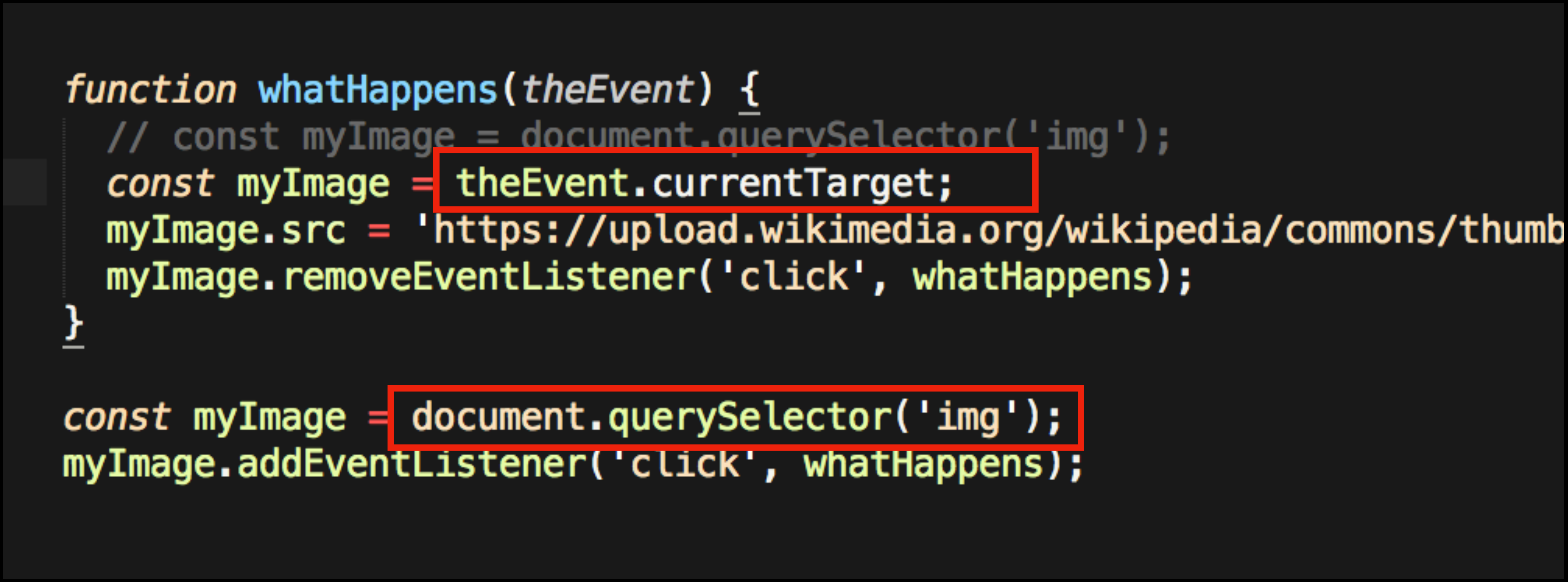
finding the element twice...



```
function whatHappens() {
  const myImage = document.querySelector('img');
  myImage.src = 'https://upload.wikimedia.org/wikipedia/commons/thumb/f/fc/Emoji_
  myImage.removeEventListener('click', whatHappens);
}

const myImage = document.querySelector('img');
myImage.addEventListener('click', whatHappens);
```

This repetition is inelegant.

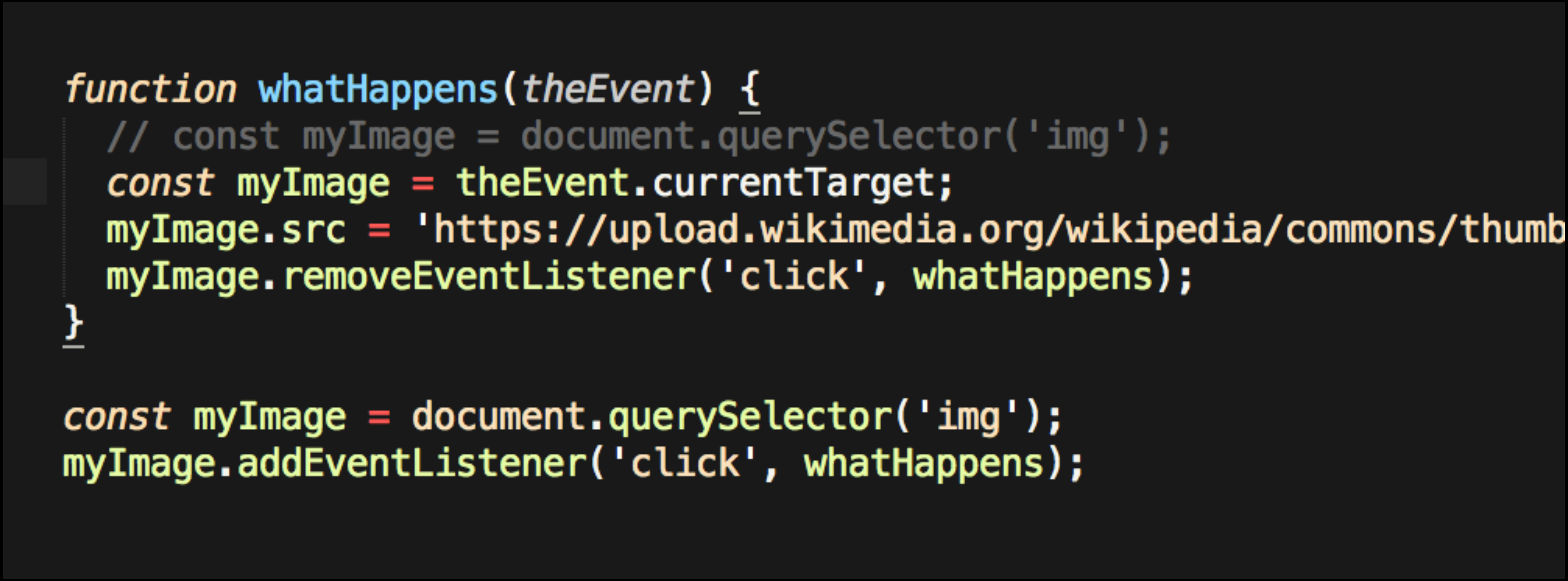
Q: is there a way to fix?


```
function whatHappens(theEvent) {
  // const myImage = document.querySelector('img');
  const myImage = theEvent.currentTarget;
  myImage.src = 'https://upload.wikimedia.org/wikipedia/commons/thumb
  myImage.removeEventListener('click', whatHappens);
}

const myImage = document.querySelector('img');
myImage.addEventListener('click', whatHappens);
```

An **Event** element is passed to the listener as a parameter:

Event.currentTarget

An Event element is passed to the listener as a parameter:



```
function whatHappens(theEvent) {
  // const myImage = document.querySelector('img');
  const myImage = theEvent.currentTarget;
  myImage.src = 'https://upload.wikimedia.org/wikipedia/commons/thumb
  myImage.removeEventListener('click', whatHappens);
}

const myImage = document.querySelector('img');
myImage.addEventListener('click', whatHappens);
```

The event's currentTarget property is a reference to the object that we attached to the event, in this case the <img>'s Element to which we added the listener.

Not to be confused with Event.target

Note: Event has both:

theEvent.**target**:

the element that was clicked / "dispatched the event" (might be a child of the target)

theEvent.currentTarget:

the element that the original event handler was attached to)

Some properties of Element objects

Property	Description
id	The value of the id attribute of the element, as a string
innerHTML	The raw HTML between the starting and ending tags of an element, as a string
textContent	The text content of a node and its descendants. (This property is inherited from Node)
classList	An object containing the classes applied to the element