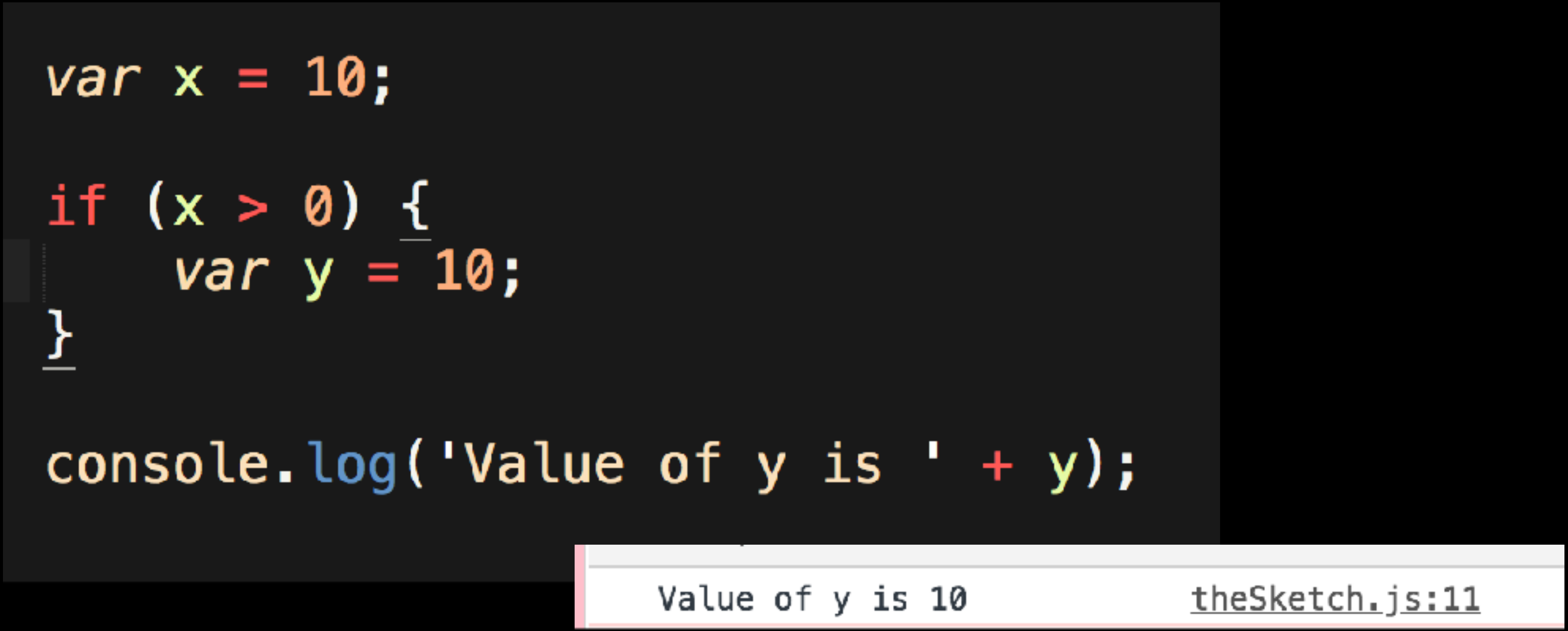
function scope w/ var



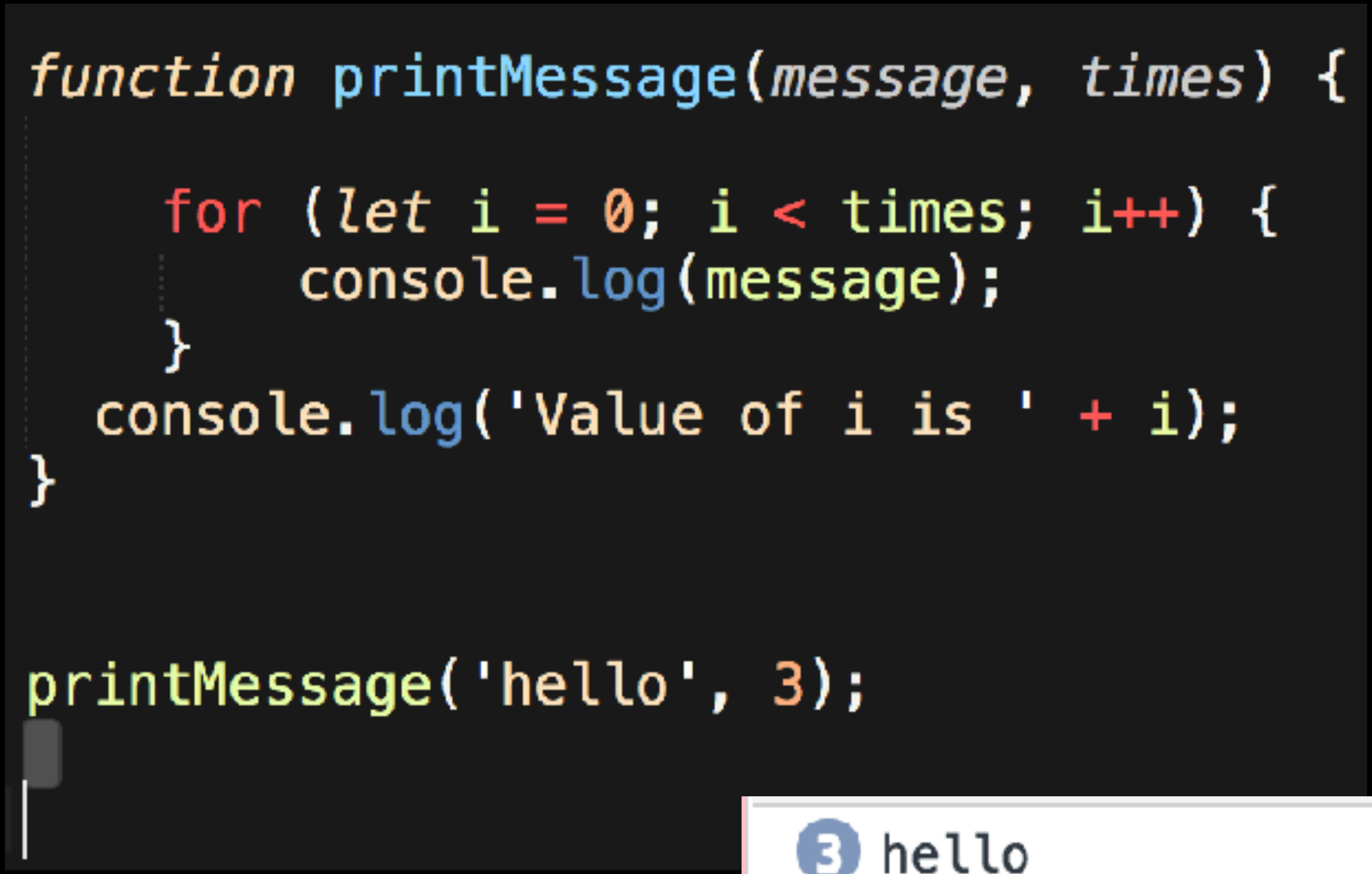
```
var x = 10;

if (x > 0) {
    var y = 10;
}

console.log('Value of y is ' + y);
```

```
Value of y is 10                theSketch.js:11
```

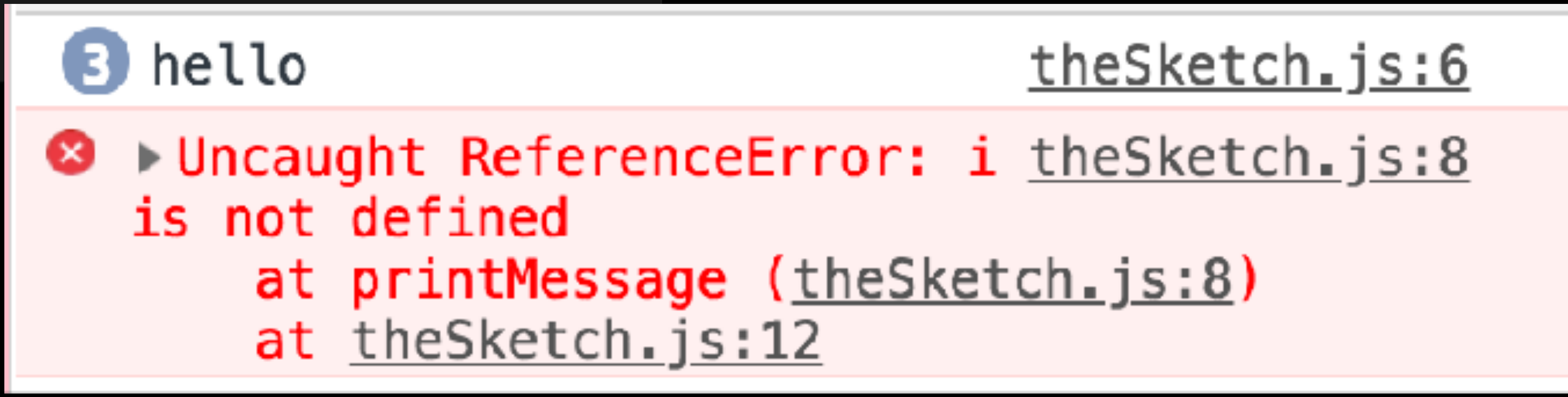
Variables declared with var have function-level scope and do not go out of scope at the end of blocks; only at the end of functions

Therefore you can refer to the same variable after the block has ended (e.g. after the loop or if-statement in which they are declared)

scope w/ let



```
function printMessage(message, times) {

    for (let i = 0; i < times; i++) {
        console.log(message);
    }
  console.log('Value of i is ' + i);
}


printMessage('hello', 3);
```

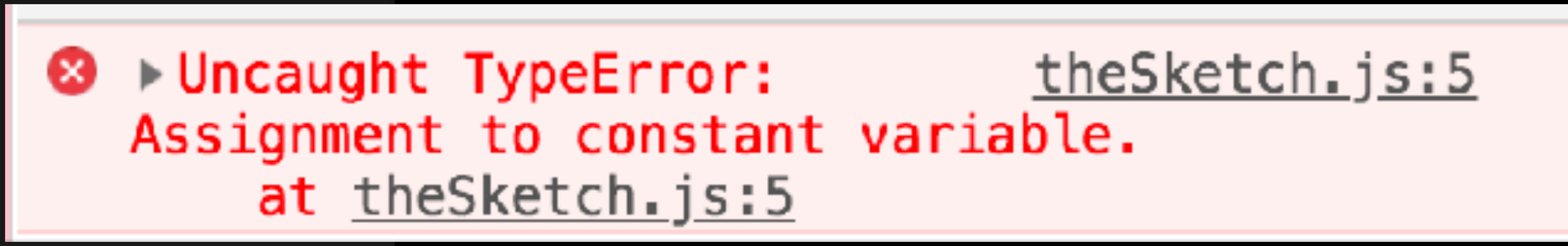


```
3 hello                                theSketch.js:6
✖ ▸Uncaught ReferenceError: i          theSketch.js:8
  is not defined
      at printMessage (theSketch.js:8)
      at theSketch.js:12
```

let has block-scope so this results in an error

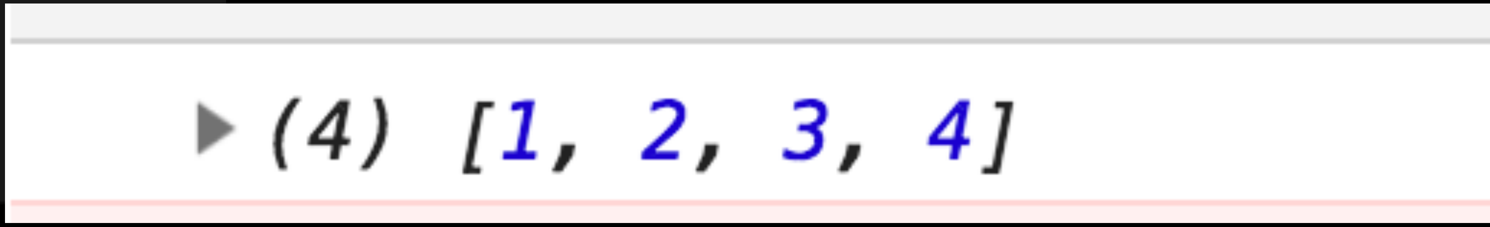
const

```
const y = 10;
y = 0; //error
y++; //error
```



```
▶Uncaught TypeError:          theSketch.js:5
Assignment to constant variable.
    at theSketch.js:5
```

```
const myList = [1, 2, 3];
myList.push(4); //okay

console.log(myList);
```



```
▶(4) [1, 2, 3, 4]
```

const declared variables cannot be reassigned.

However, it doesn't provide true const correctness, so you can still modify the underlying object

- (In other words, it behaves like Java's final keyword and not C++'s const keyword)

let can be reassigned, which is the difference between const and let

JS Syntax - Variables best practices

- Use **const** whenever possible.

- If you need a variable to be reassignable, use **let**.

- Don't use **var.**

  You will see a ton of example code on the internet with var since const and let are relatively new.

  However, **const** and **let** are well-supported, so there's no reason not to use them. (This is also what the Google and AirBnB JavaScript Style Guides recommend.)

You do not declare the datatype of the variable before using it ("dynamically typed")

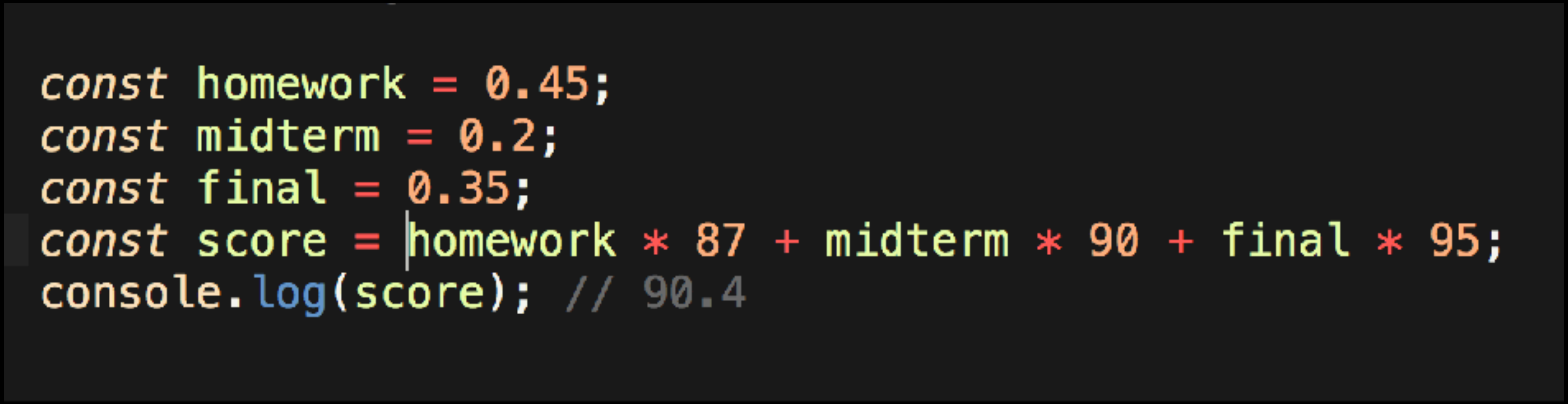
JS Variables do not have types, but the values do.

There are six primitive types (mdn):

- **Boolean** : true and false
- **Number** : everything is a double (no integers)
- **String**: in 'single' or "double-quotes"
- **Symbol:** (skipping this today)
- **Null:** null: a value meaning "this has no value"
- **Undefined:** the value of a variable with no value assigned

There are also Object types, including Array, Date, String (the object wrapper for the primitiv
type), etc.

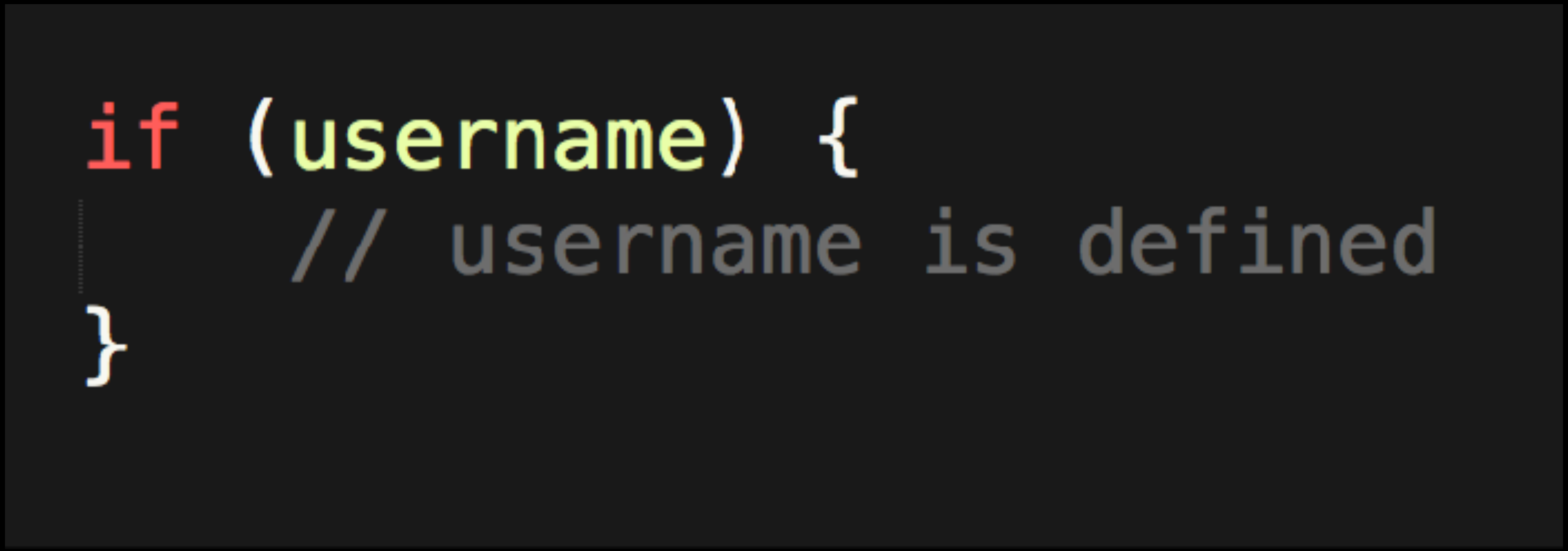
Data Type: Numbers



```
const homework = 0.45;
const midterm = 0.2;
const final = 0.35;
const score = homework * 87 + midterm * 90 + final * 95;
console.log(score); // 90.4
```

- All numbers are floating point real numbers. No integer type.
- Operators are like Java or C++.
- Precedence like Java or C++.
- A few special values: **NaN** (not-a-number), **+Infinity**, **-Infinity**
- There's a Math class: **Math**.floor, **Math**.ceil, etc.

Data Type: Boolean



```
if (username) {
    // username is defined
}
```

Non-boolean values can be used in control statements, which get converted to their "**truthy**" or "**falsy**" value:

- null, undefined, 0, NaN, '', " " evaluate to false
- Everything else evaluates to true

Equality

== (is equal to)

Compares two values to see if they are the same

!= (is not equal to)

Compares two values to see if they are not the same

=== (strict equal to)

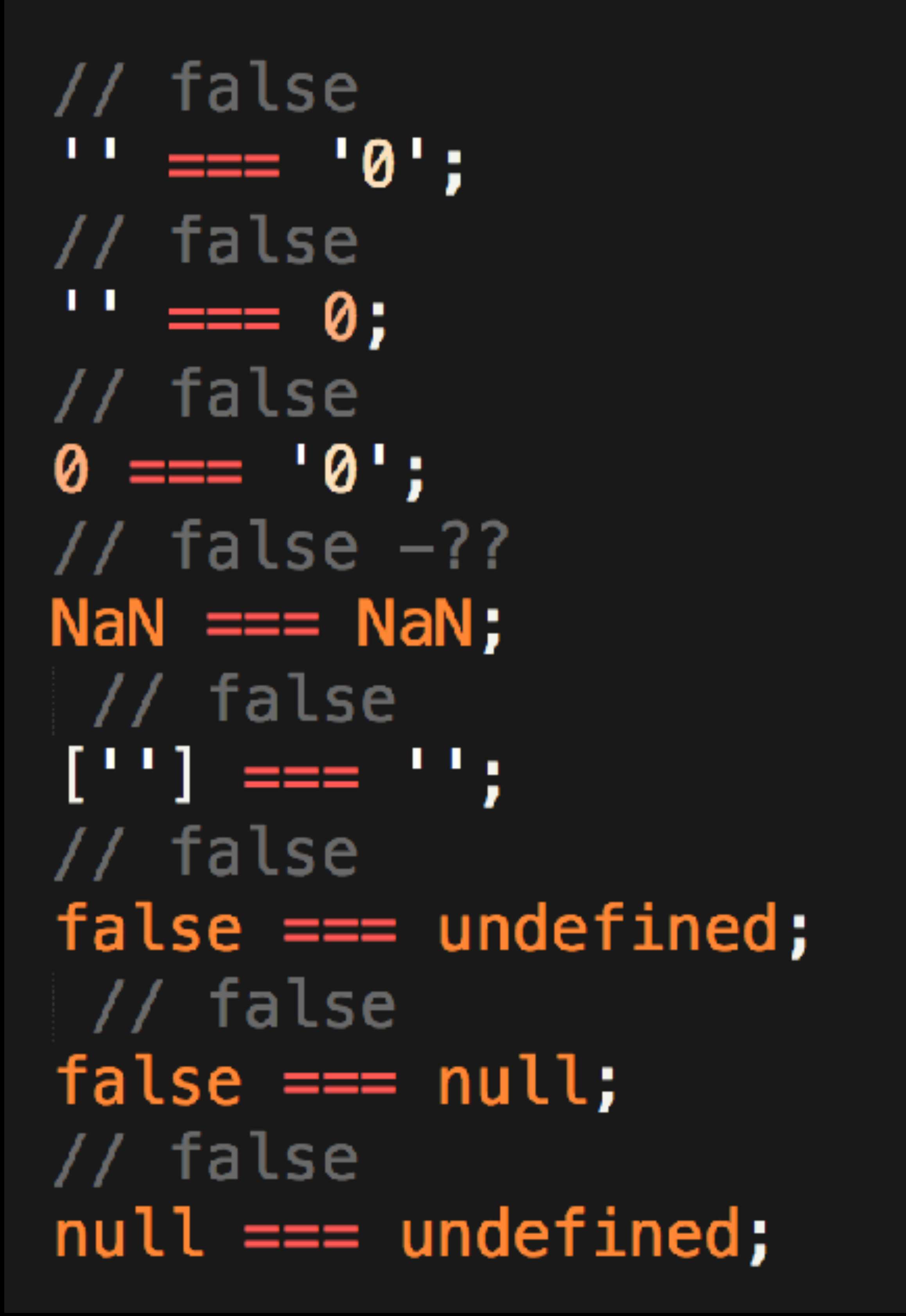
Compares two values to check that both the data and value are the same

!== (strict not equal to)

Compares two values to check that both the data and value are not the same

Equality

Instead d of fixing **==** and **!=** , the ECMAScript standard kept the existing behavior but added **===** and **!==**



```
// false
'' === '0';
// false
'' === 0;
// false
0 === '0';
// false -??
NaN === NaN;
 // false
[''] === '';
// false
false === undefined;
 // false
false === null;
// false
null === undefined;
```

Best practice: always use **===** and **!==** and don't use **==** or **!=**

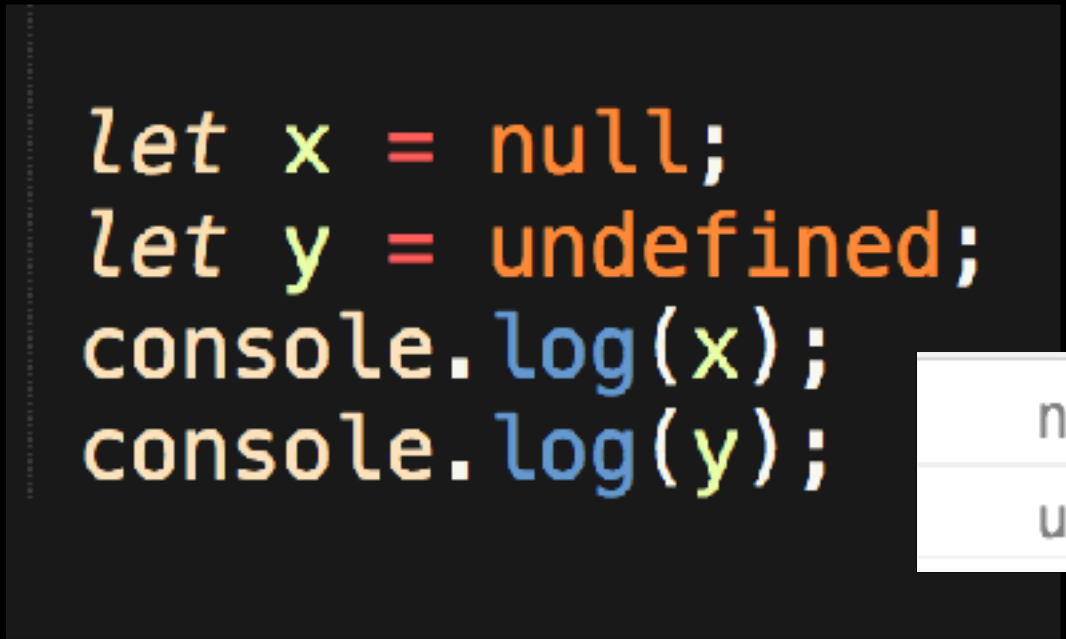
Null + Undefined

What's the difference?

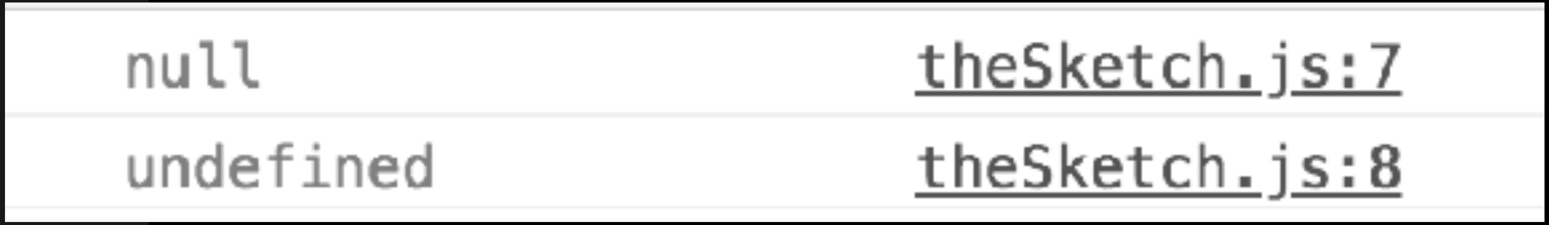
null is a value representing the absence of a value, similar to null in Java and nullptr in C++.

undefined is the value given to a variable that has not been a value.

... however, you can also set a variable's value to undefined bc … javascript.



```
let x = null;
let y = undefined;
console.log(x);
console.log(y);
```



```
null                              theSketch.js:7
undefined                         theSketch.js:8
```

Object properties + methods

In an object:

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

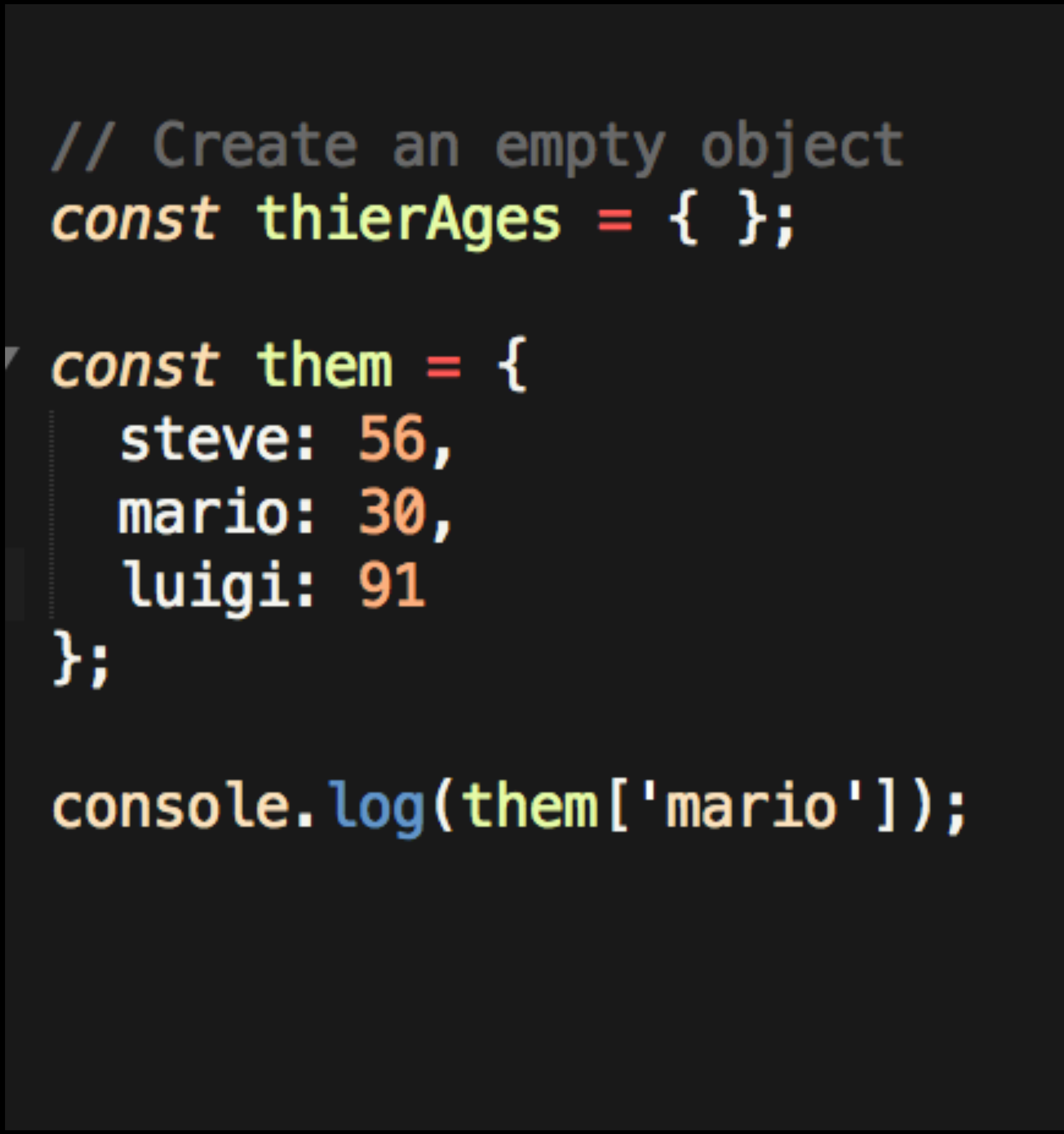
Properties tell us about the object, such as the height of a chair and how many chairs there are

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

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']);
```

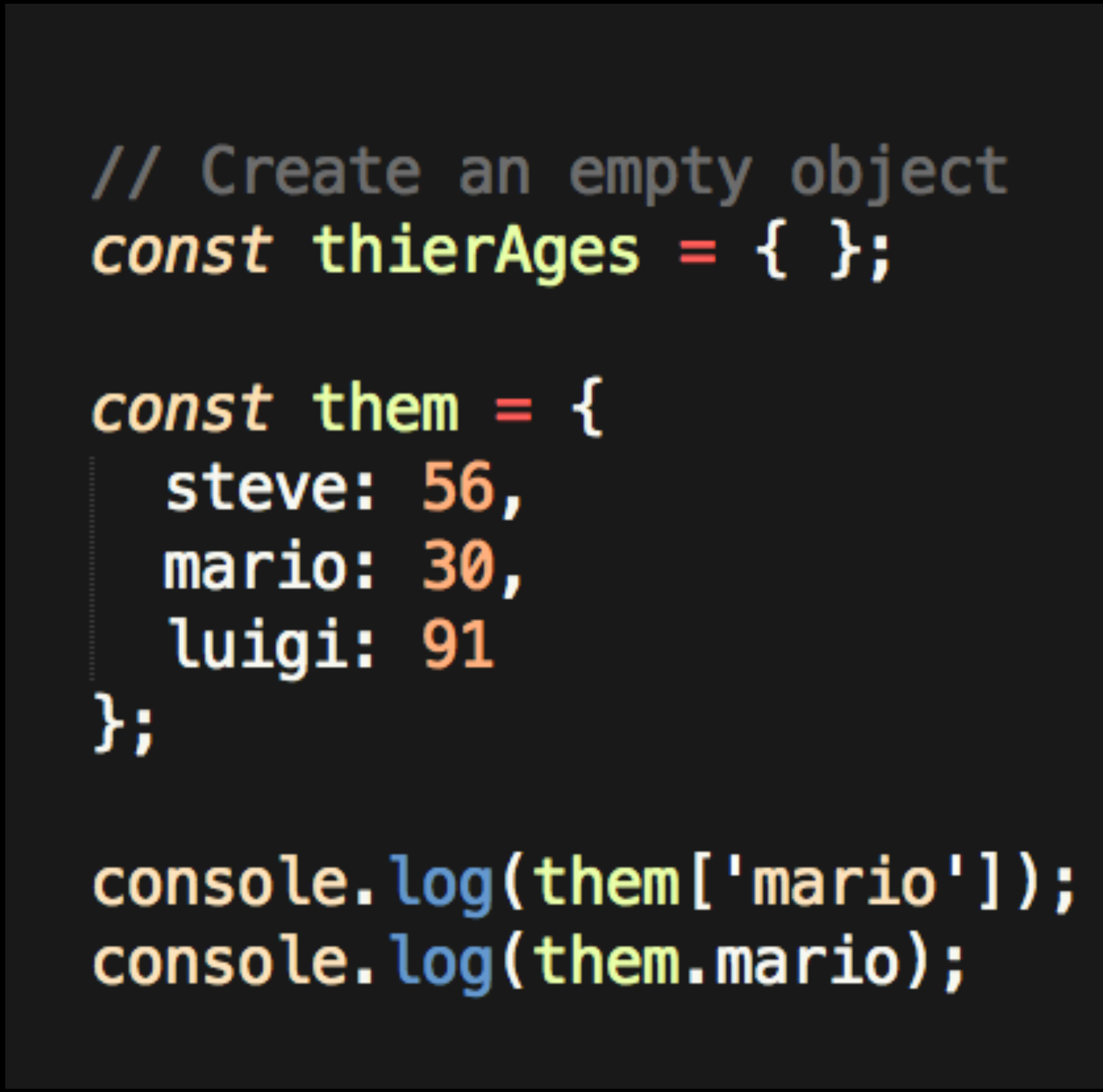
Maps through objects

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

1. objectName[property]
2. objectName.property

(2 only works for string keys.)

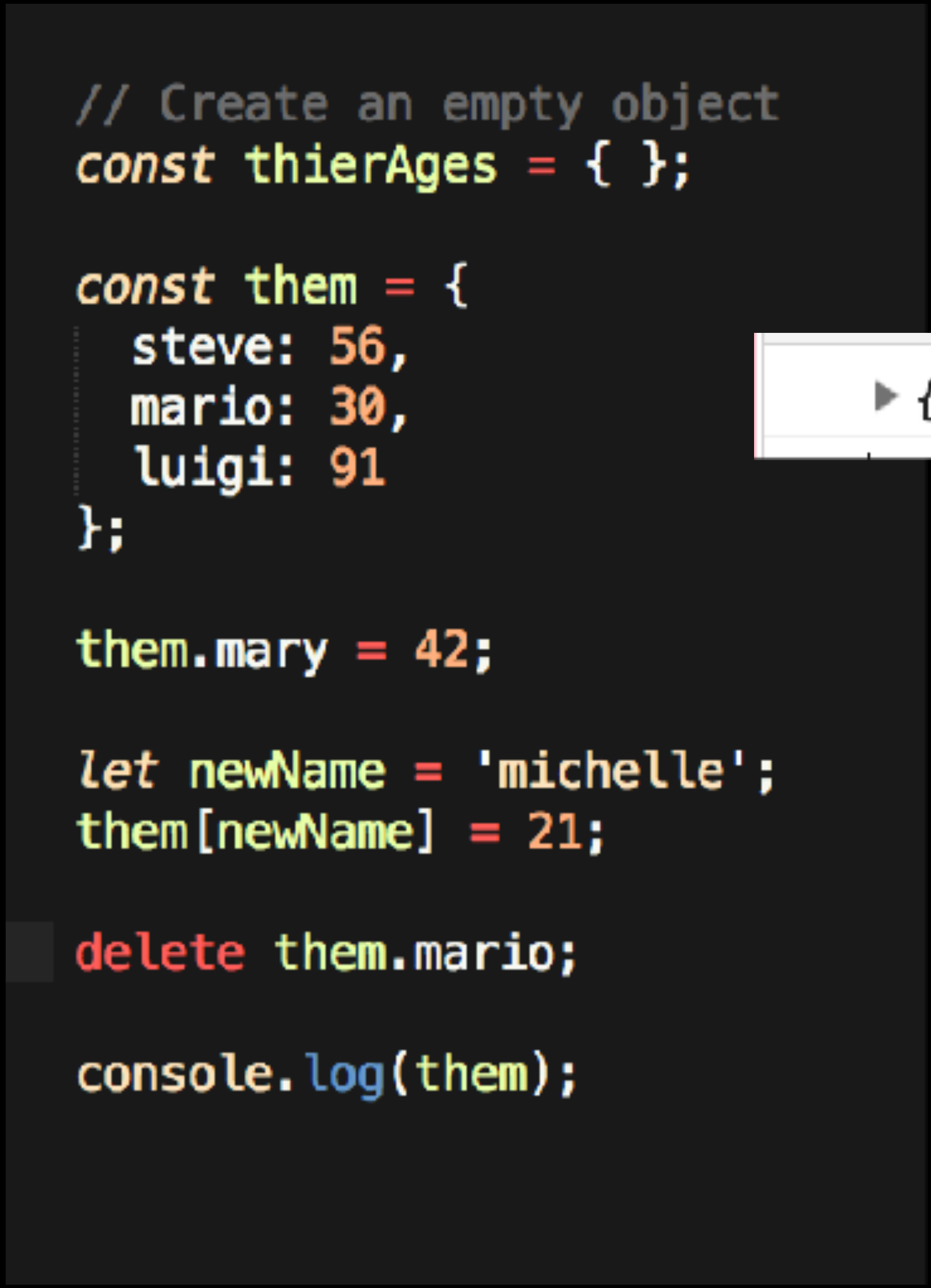
Generally prefer style (2), unless the property is stored in a variable, or if the property is not a string.



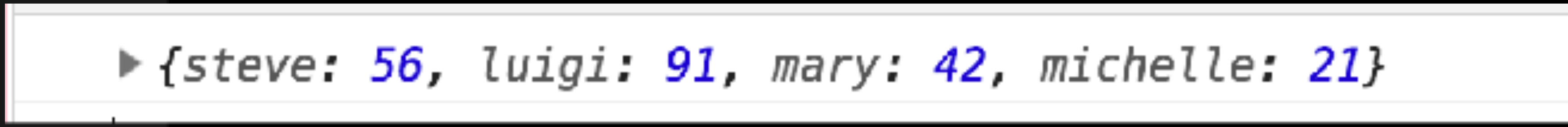
```
// Create an empty object
const thierAges = { };

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

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

Maps through objects

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

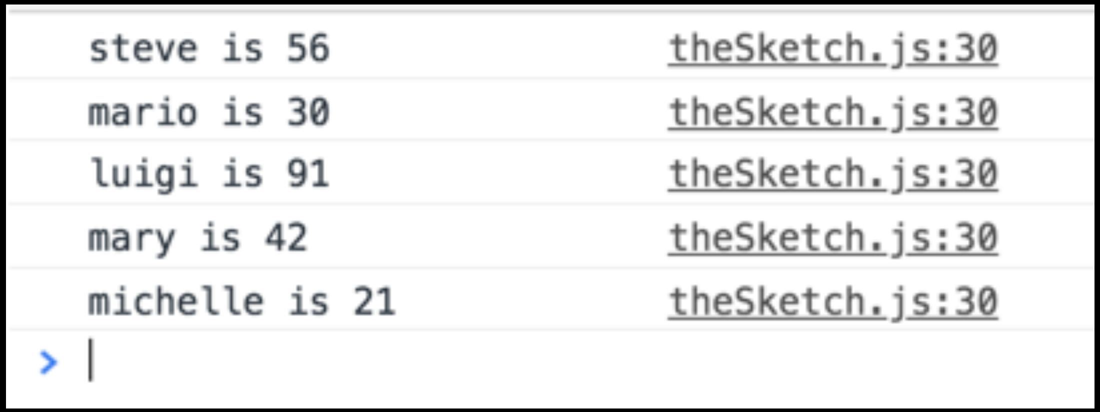


```
// Create an empty object
const thierAges = { };

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

them.mary = 42;

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

delete them.mario;

console.log(them);
```

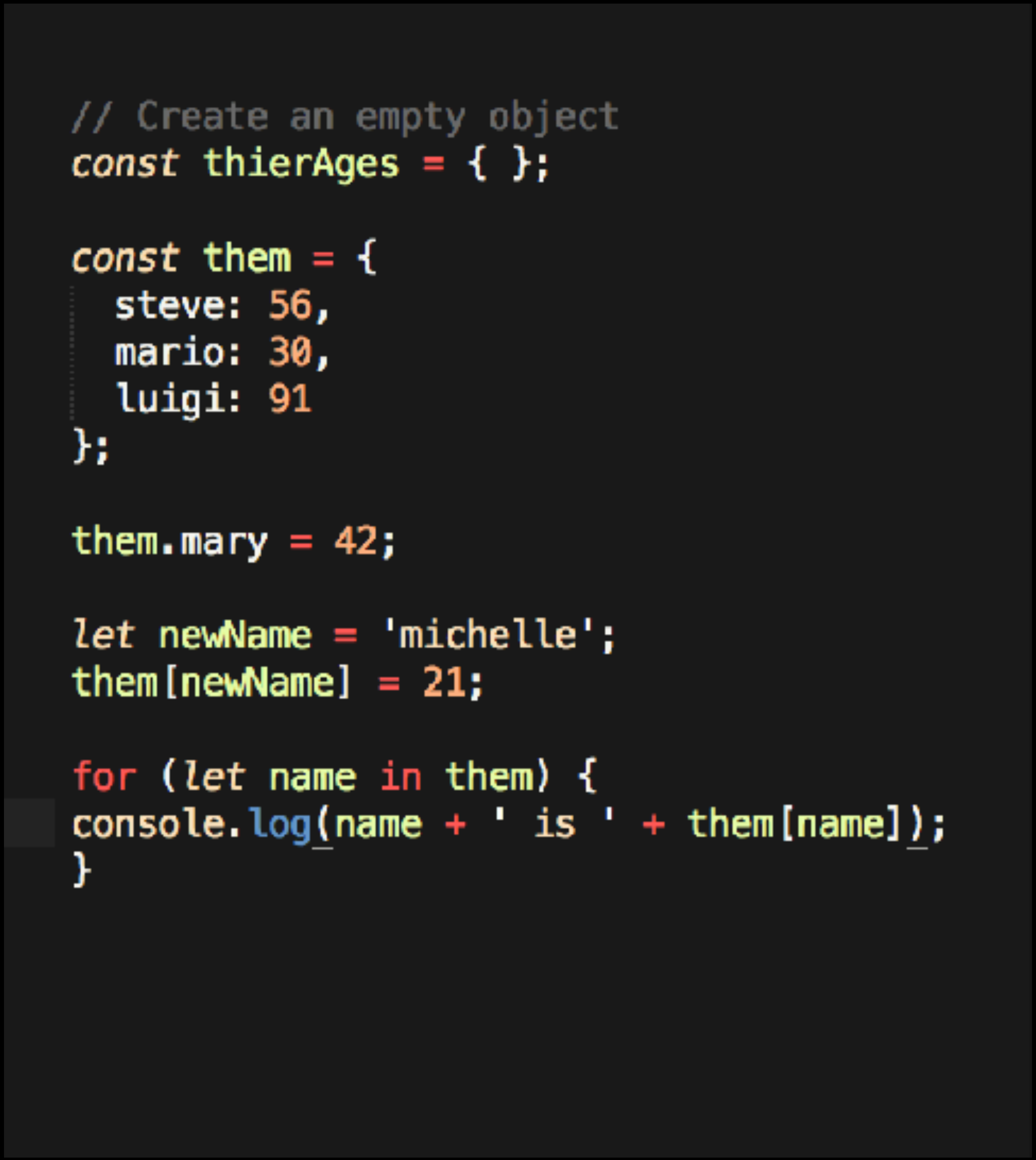


```
▸ {steve: 56, luigi: 91, mary: 42, michelle: 21}
```

Iterating through Map



```
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**

Events

Getting DOM objects

```
let element = document.querySelector('#button');
```

Returns the DOM object for the HTML element with id="button", or null if none exists.

```
let elementList = document.querySelectorAll('.quote, .comment');
```

Returns a list of DOM objects containing all elements that have a "quote" class AND all elements that have a "comment" class.

Adding Event Listeners

+ Removing Event Listeners

Each DOM object has the following function:

```
addEventListener(event name, function name);
removeEventListener(event name, function name);
```

event name is the string name of the JavaScript event you want to listen or stop listening to (Common ones: click, focus, blur, etc)

function name is the name of the JavaScript function you want to execute or no longer execute when the event fires

defer

You can add the defer attribute onto the script tag so that the JavaScript doesn't execute until after the DOM is loaded (mdn):

```
<script src="script.js" defer></script>
```

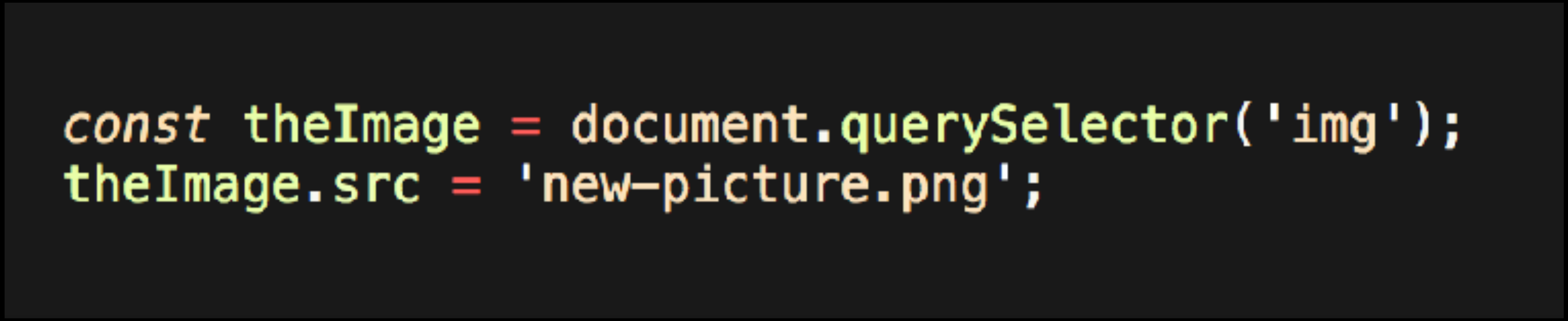
Other old school ways of doing this **(not best practice - don't do!)**:

- Put the <script> tag at the bottom of the page
- Listen for the "load" event on the window object

You will see tons of examples on the internet that do this. They are out of date. defer is wide
supported and better

DOM object properties

You can access attributes of an HTML element via a property (field) of the DOM object



```
const theImage = document.querySelector('img');
theImage.src = 'new-picture.png';
```

Exceptions:
Notably, you can't access the class attribute via object.class

tech details

The DOM objects that we retrieve from querySelector and querySelectorAll have types:

- Every DOM node is of general type Node (an interface)
- Element implements the Node interface

(FYI: This has nothing to do with NodeJS, if you've heard of that)

- Each HTML element has a specific Element derived class, like HTMLImageElement

Attributes and DOM properties

Roughly every **attribute** on an HTML element is a **property** on its respective DOM object…

HTML

```
<img src =“tree.png” />
```

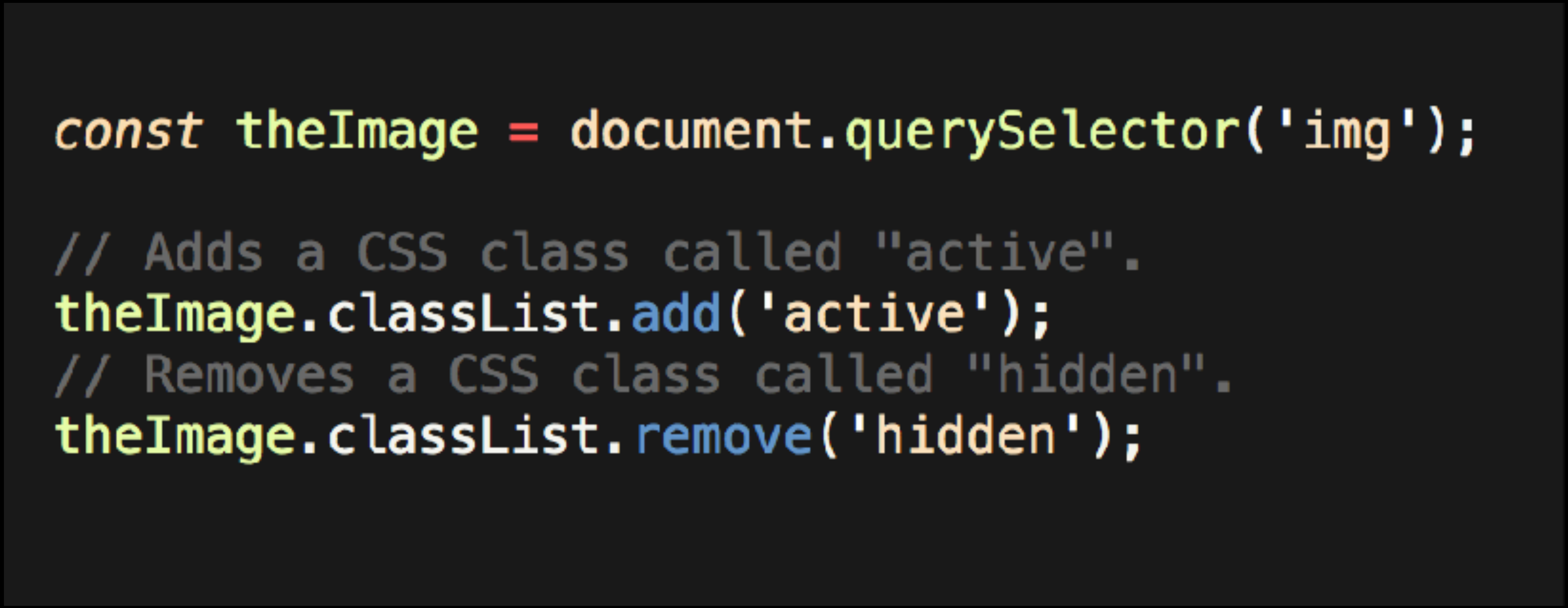
JavaScript

```
const myElement = document.querySelector('img');
myElement.src = ‘bird.png';
```

(But you should always check the JavaScript spec to be sure. In this case, check the HTMLImageElement.)

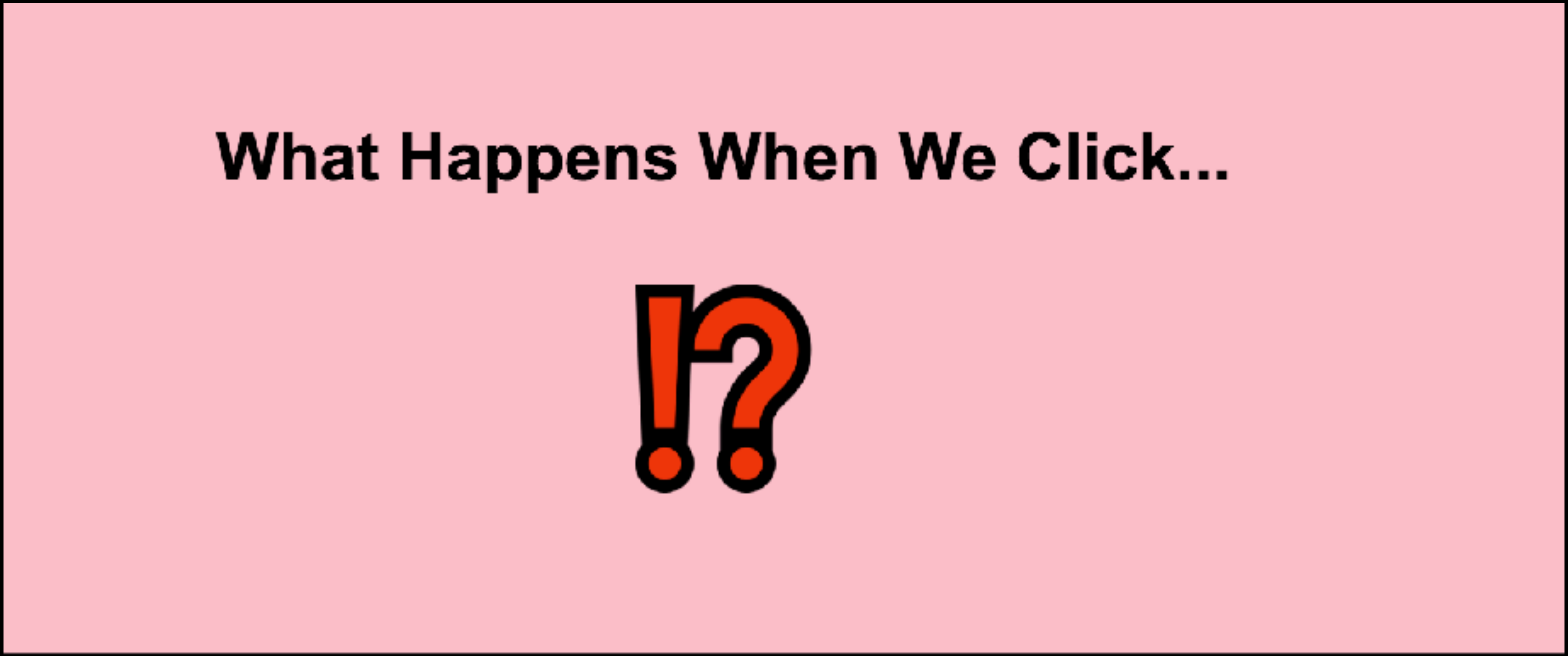
Adding + Removing Classes

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

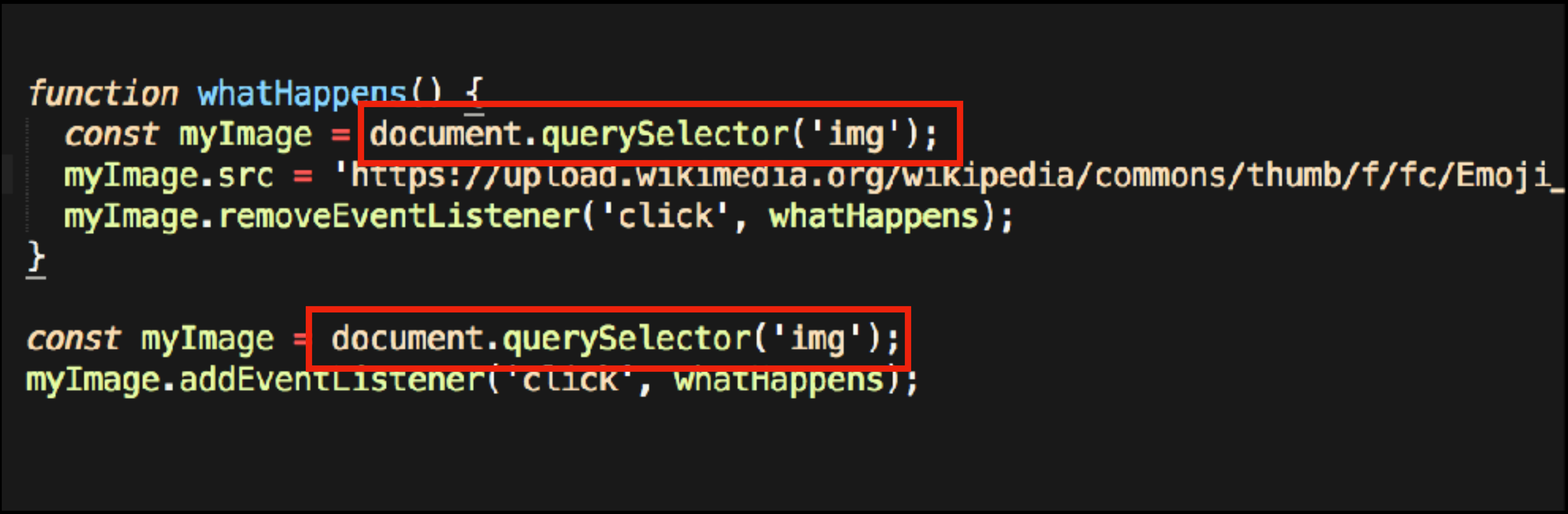


```
const theImage = document.querySelector('img');

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

example code


What Happens When We Click...

example code

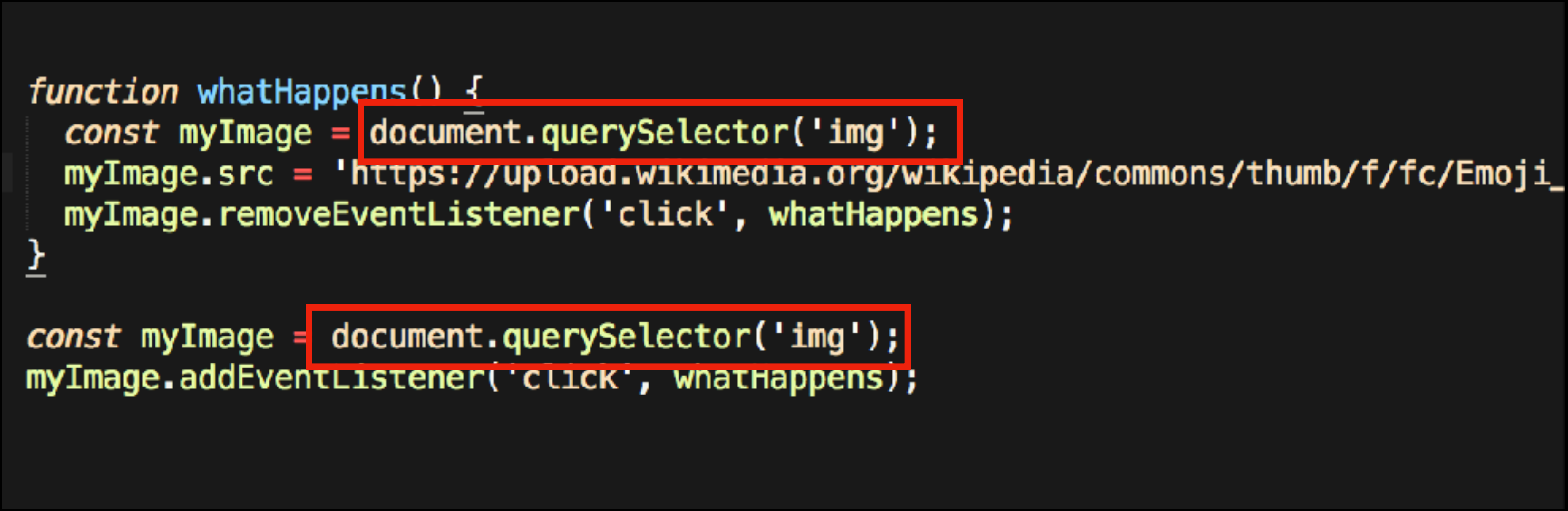


```
function whatHappens() {
  const myImage = document.querySelector('img');
  myImage.src = 'https://upload.wikimedia.org/wikipedia/commons/thumb/f/fc/Emoji_
  myImage.removeEventListener('click', whatHappens);
}

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

This repetition is inelegant.

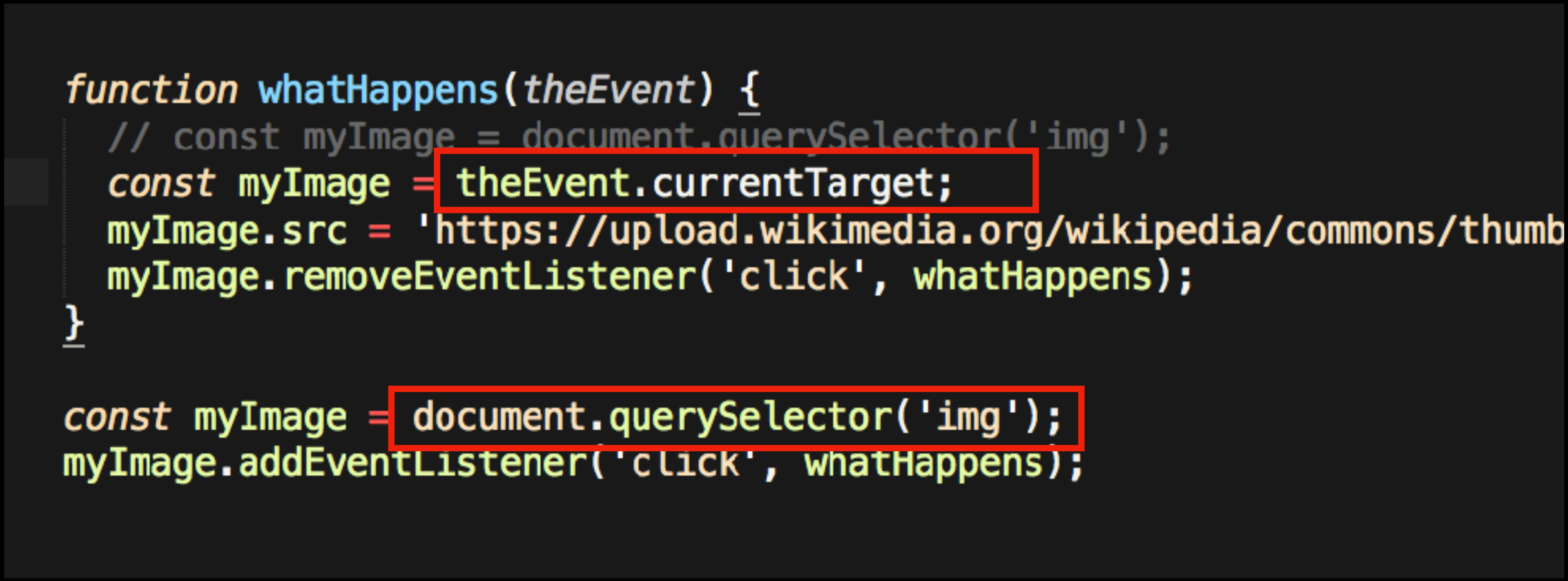
Q: is there a way to fix?

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.

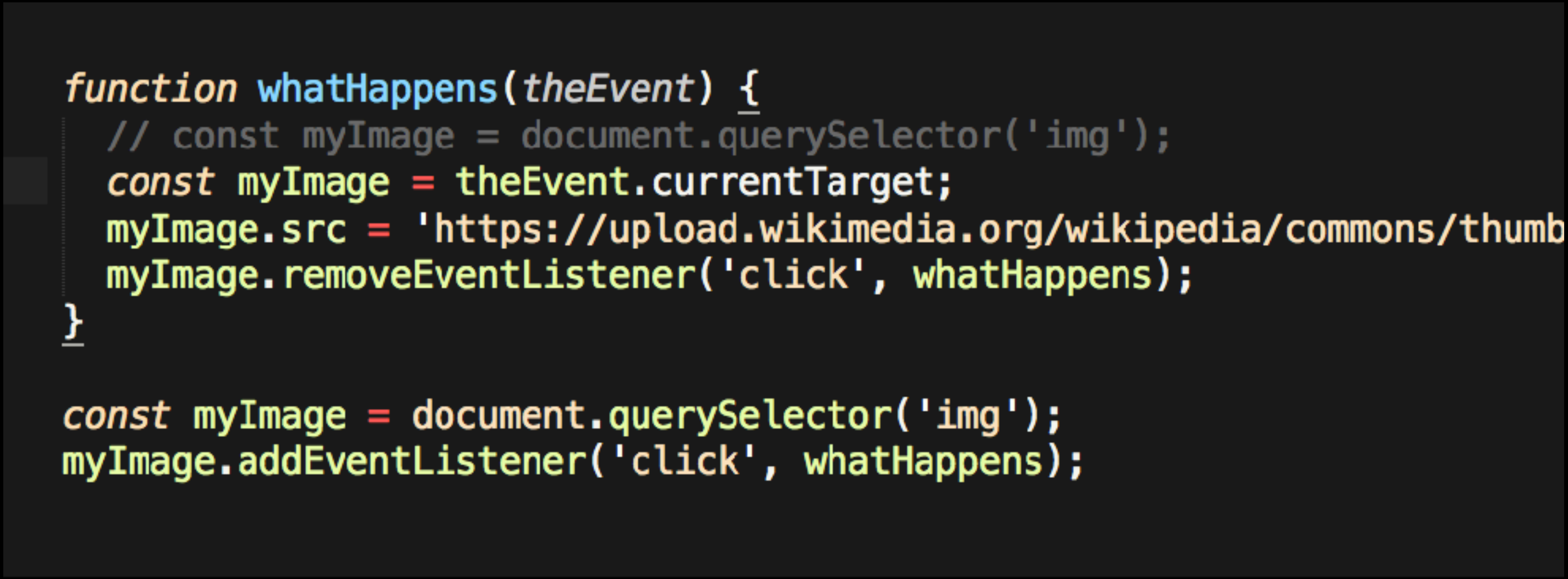


```
function whatHappens(theEvent) {
  // const myImage = document.querySelector('img');
  const myImage = theEvent.currentTarget;
  myImage.src = 'https://upload.wikimedia.org/wikipedia/commons/thumb
  myImage.removeEventListener('click', whatHappens);
}

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

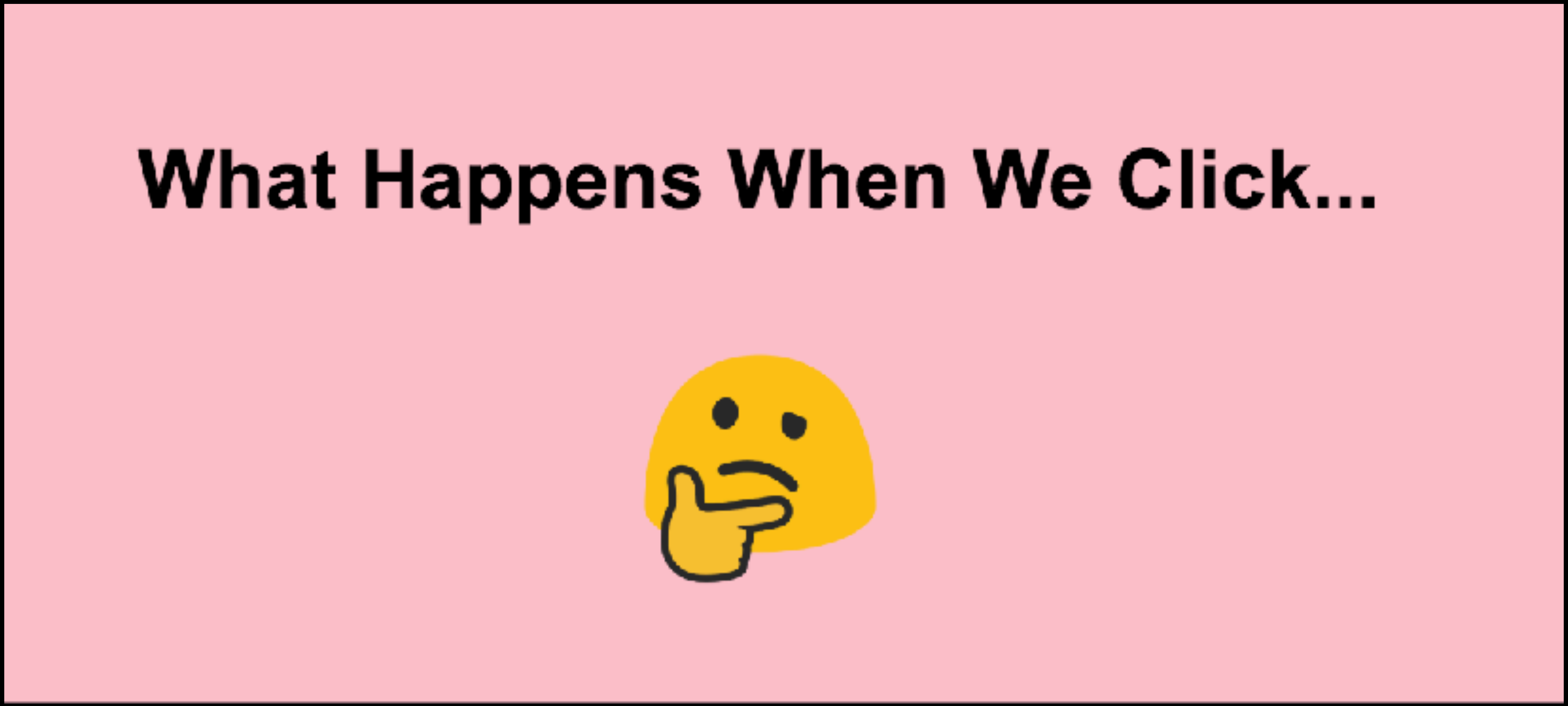
Note: Event has both:

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)

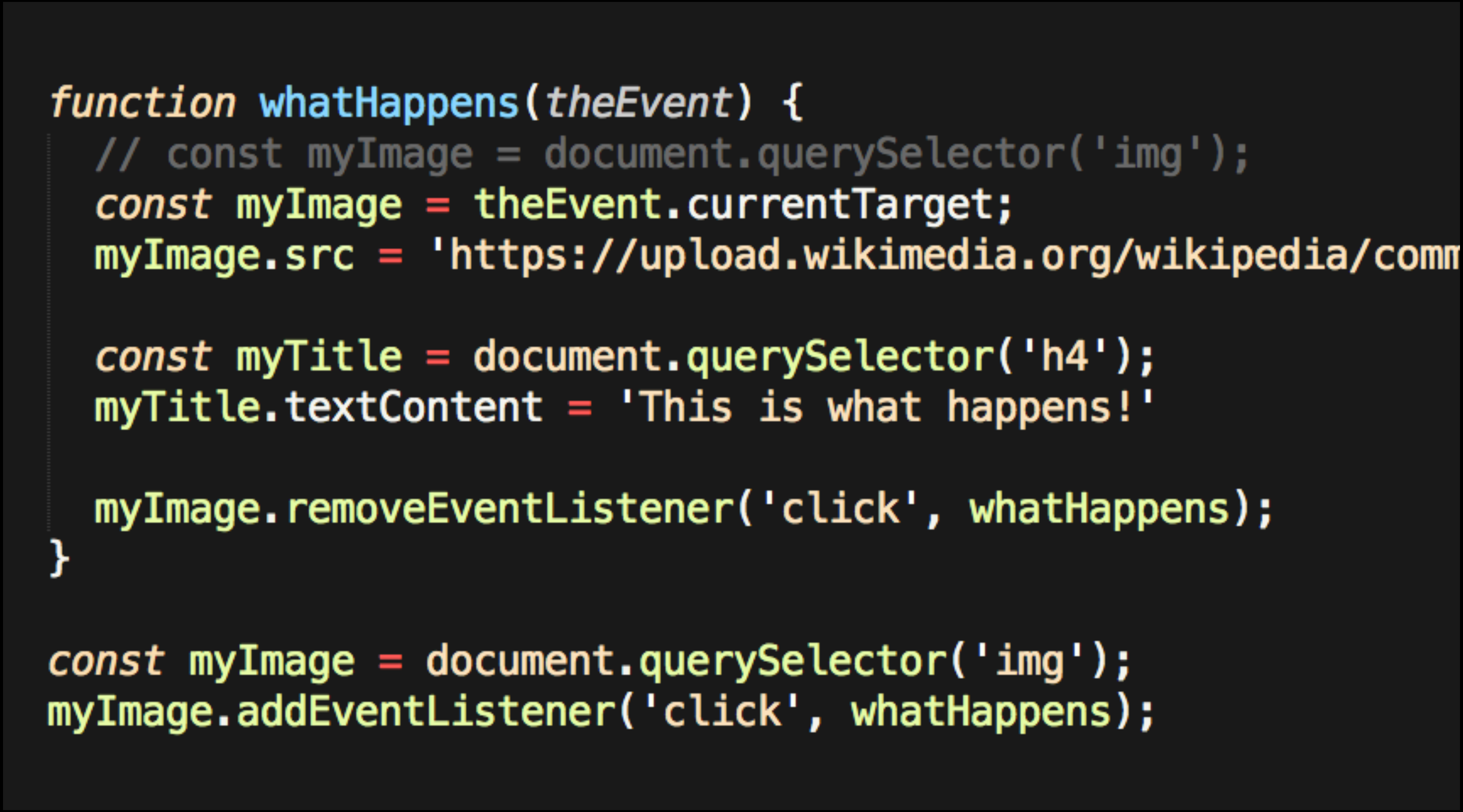


What Happens When We Click...

It would be neat if we could change the text after we click.

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



```
function whatHappens(theEvent) {
  // const myImage = document.querySelector('img');
  const myImage = theEvent.currentTarget;
  myImage.src = 'https://upload.wikimedia.org/wikipedia/comm

  const myTitle = document.querySelector('h4');
  myTitle.textContent = 'This is what happens!'

  myImage.removeEventListener('click', whatHappens);
}

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

We can select the h4 element then set its
textContent to change what is displayed in the h4

Another approach: Dynamically changing the elements

Add elements via DOM

We can create elements dynamically and add them to the web page via createElement and appendChild:

```
// document is when the page is the parent node
document.createElement( HTML tag string ) ;
// when adding a child to a parent element
HTMLelement.appendChild( another HTML element );
```

Technically you can also add elements to the webpage via innerHTML, but it poses a security risk.

```
// Try not to use innerHTML like this:
element.innerHTML = '<h1>Hooray!</h1>';
```

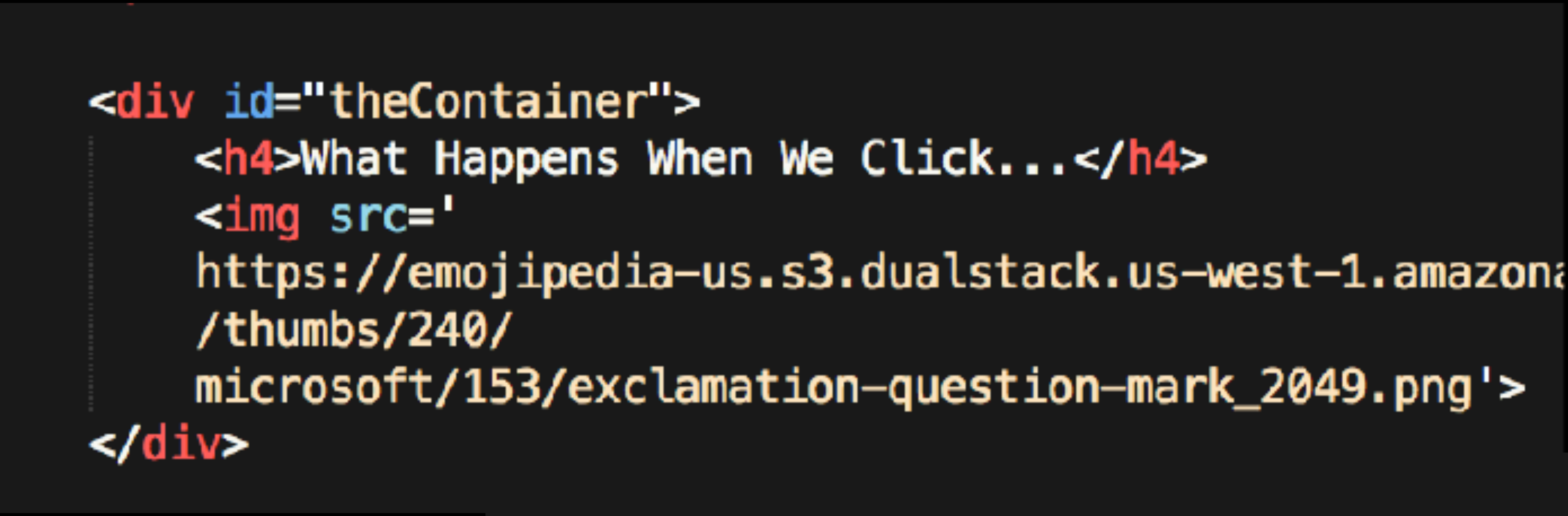
Remove elements via DOM

We can also call remove elements from the DOM by calling the remove( ) method on the DOM object:

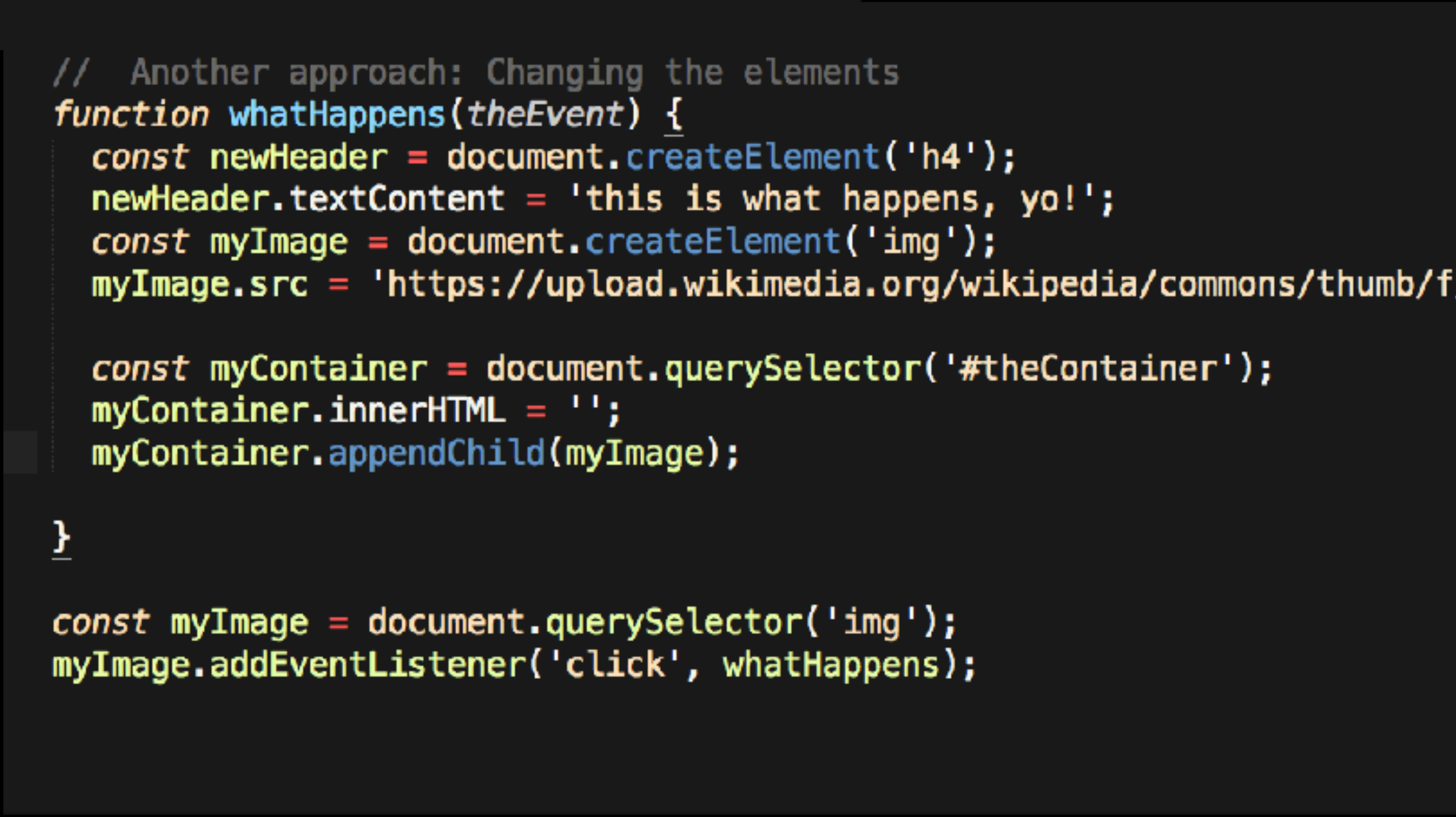
```
element.remove( );
```

And actually setting the innerHTML of an element to an empty string is a fine way of removing all children from a parent node:

```
// This is fine and poses no security risk.
    element.innerHTML = '';
```

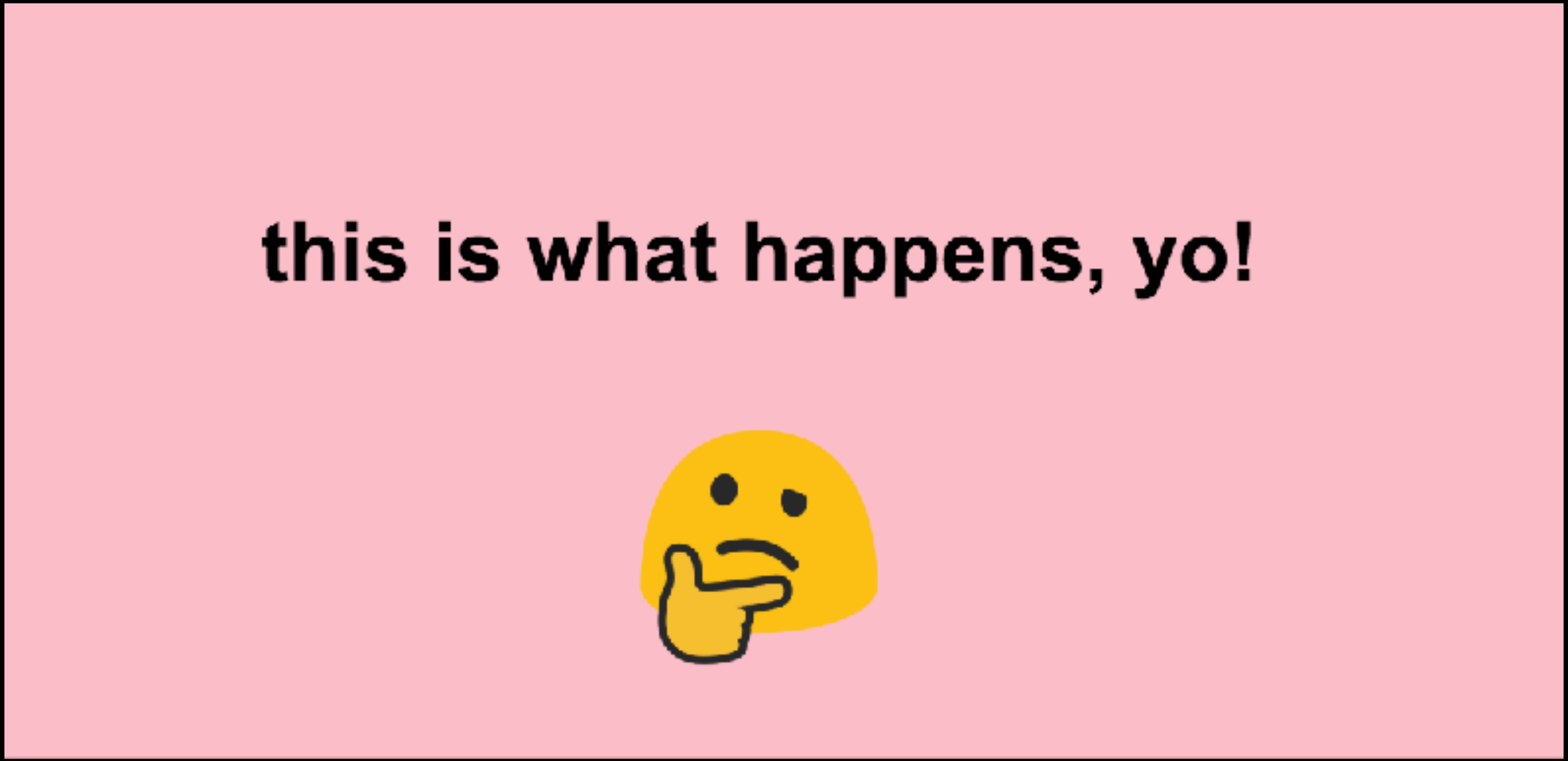


```
<div id="theContainer">
    <h4>What Happens When We Click...</h4>
    <img src='
    https://emojipedia-us.s3.dualstack.us-west-1.amazona
    /thumbs/240/
    microsoft/153/exclamation-question-mark_2049.png'>
</div>
```



```
//  Another approach: Changing the elements
function whatHappens(theEvent) {
  const newHeader = document.createElement('h4');
  newHeader.textContent = 'this is what happens, yo!';
  const myImage = document.createElement('img');
  myImage.src = 'https://upload.wikimedia.org/wikipedia/commons/thumb/f

  const myContainer = document.querySelector('#theContainer');
  myContainer.innerHTML = '';
  myContainer.appendChild(myImage);

}

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

Add a container to hold + append dynamically created html elements


this is what happens, yo!

Hmm, the effect is slightly wonky though: The text changes faster than the image loads.
Q: How do we fix this issue?

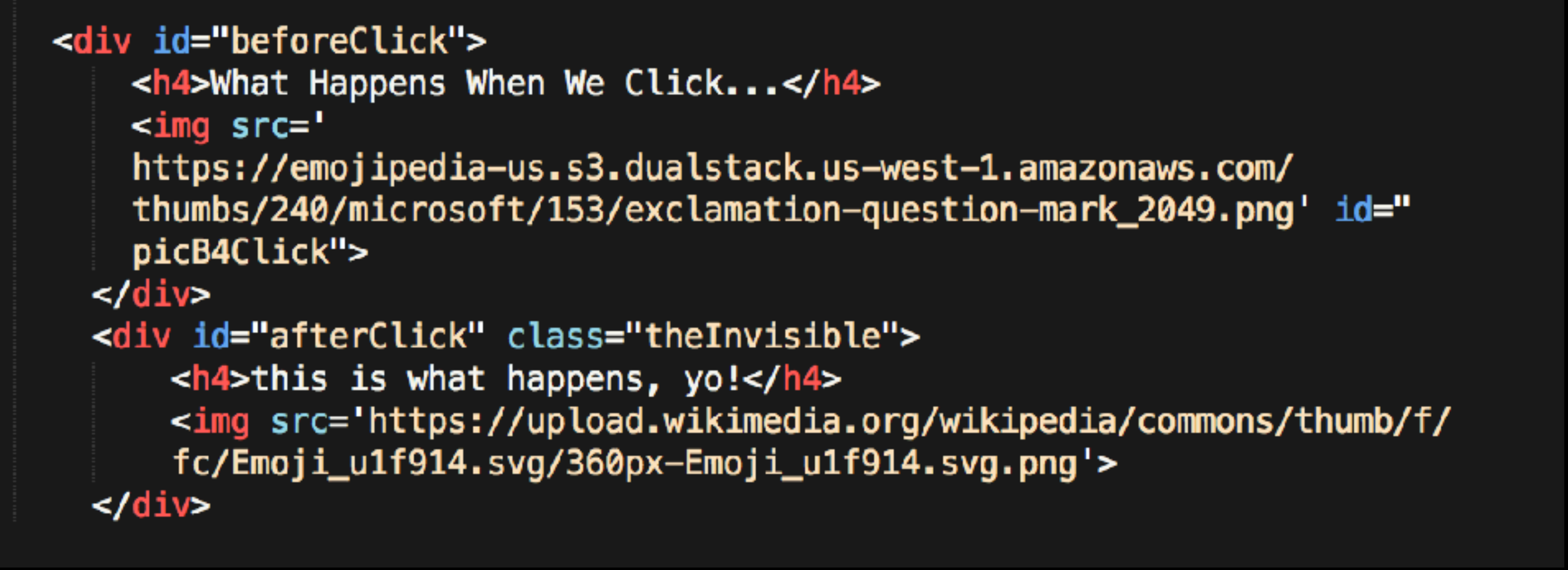
display: none;

There is yet another super helpful value for display:

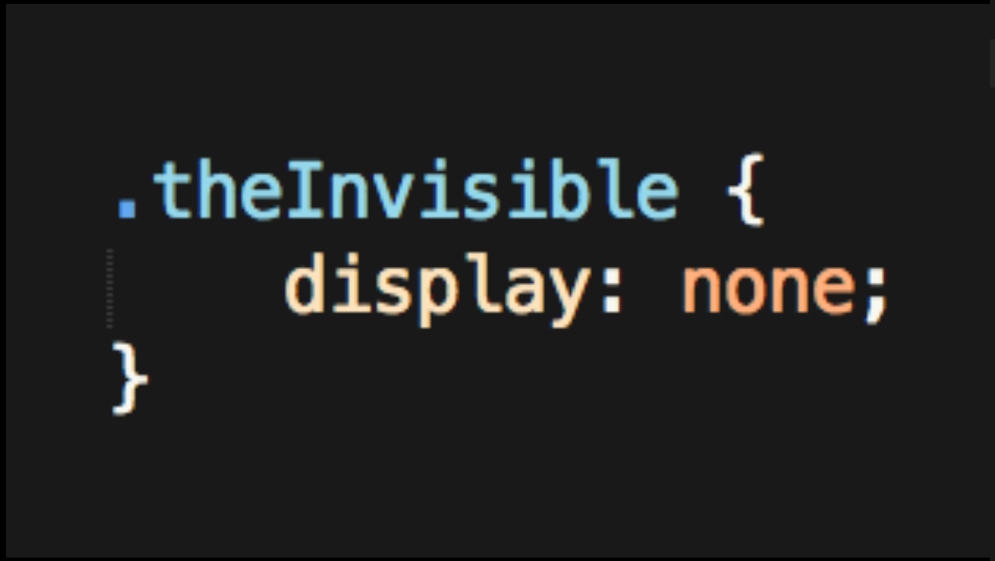
display: block;
display: inline;
display: inline-block;
display: flex;
display: none;

display: none; turns off rendering for the element and all its children. It's treated as if the element were not in the document at all..

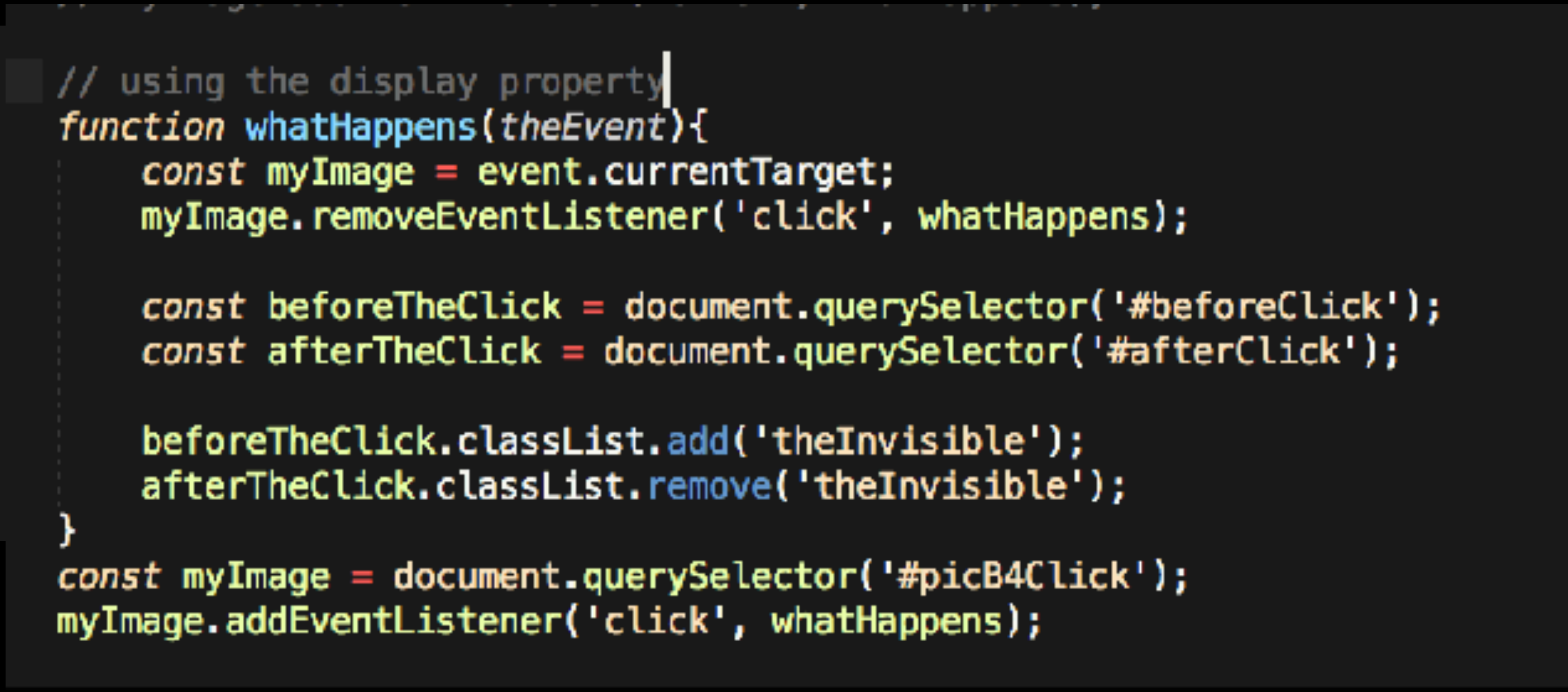
...but the content (such as the images) is still loaded.



```
<div id="beforeClick">
    <h4>What Happens When We Click...</h4>
    <img src='
    https://emojipedia-us.s3.dualstack.us-west-1.amazonaws.com/
    thumbs/240/microsoft/153/exclamation-question-mark_2049.png' id="
    picB4Click">
  </div>
  <div id="afterClick" class="theInvisible">
      <h4>this is what happens, yo!</h4>
      <img src='https://upload.wikimedia.org/wikipedia/commons/thumb/f/
      fc/Emoji_u1f914.svg/360px-Emoji_u1f914.svg.png'>
  </div>
```



```
.theInvisible {
    display: none;
}
```



```
// using the display property
function whatHappens(theEvent){
    const myImage = event.currentTarget;
    myImage.removeEventListener('click', whatHappens);

    const beforeTheClick = document.querySelector('#beforeClick');
    const afterTheClick = document.querySelector('#afterClick');

    beforeTheClick.classList.add('theInvisible');
    afterTheClick.classList.remove('theInvisible');
}
const myImage = document.querySelector('#picB4Click');
myImage.addEventListener('click', whatHappens);
```

Several Strategies for updating HTML elements in JS:

1. Change content of existing HTML elements in page:
    - Good for simple text updates

2. Add elements via createElement and appendChild - Needed if you're adding a variable number of elements

3. Put all "views" in the HTML but set inactive ones to hidden, then update display state as necessary.
    - Good when you know ahead of time what element(s) you want to display
    - Can be used in conjunction with (1) and/or (2)