COMP 431/531: Web Development

Lecture 3: JavaScript

Mack Joyner (mjoyner@rice.edu)

https://www.clear.rice.edu/comp431

JavaScript

- Single-threaded client-side scripting language with C-like syntax
- No requirement on organization
  - functions, objects, and modules can all be in the same file
- Semi-colons are optional
  - Interpreters perform automatic semi-colon insertion (watch out for run-on statements).
  - I recommend using semi-colons.

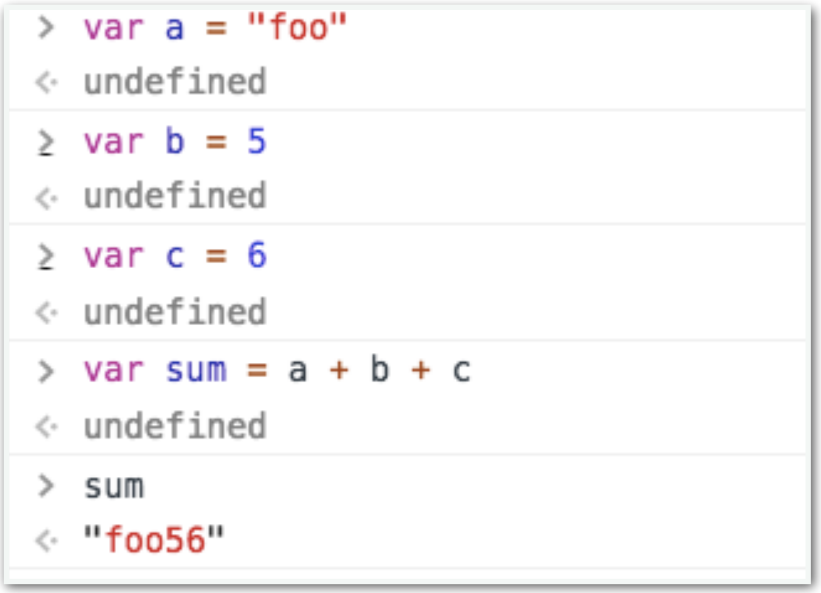
JavaScript

- Dynamically Typed
- Prototyped-based
- Functional
- Engine evaluated script

JavaScript

- **Dynamically Typed**
- Prototyped-based
- Functional
- Engine evaluated script

Types are associated with values not variables



```
> var a = "foo"
<· undefined
> var b = 5
<· undefined
> var c = 6
<· undefined
> var sum = a + b + c
<· undefined
> sum
<· "foo56"
```

JavaScript

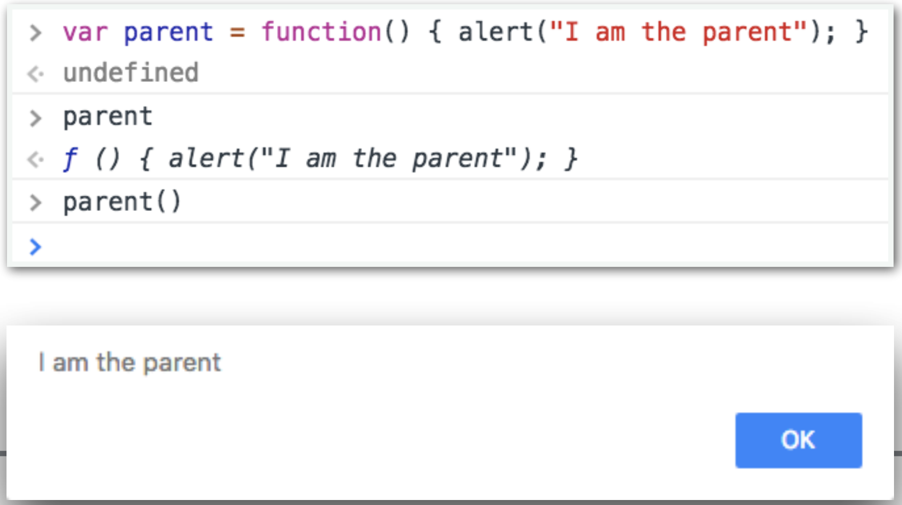
- Dynamically Typed
- **Prototyped-based**
- Functional
- Engine evaluated script

- Object-oriented
- Inheritance performed via prototype object cloning
- Runtime prototype reassignment (Dynamic)

JavaScript

- Dynamically Typed
- Prototyped-based
- **Functional**
- Engine evaluated script

Functions are treated as ‘first class’ objects



```
> var parent = function() { alert("I am the parent"); }
<· undefined
> parent
<· f () { alert("I am the parent"); }
> parent()
>
```

I am the parent

OK

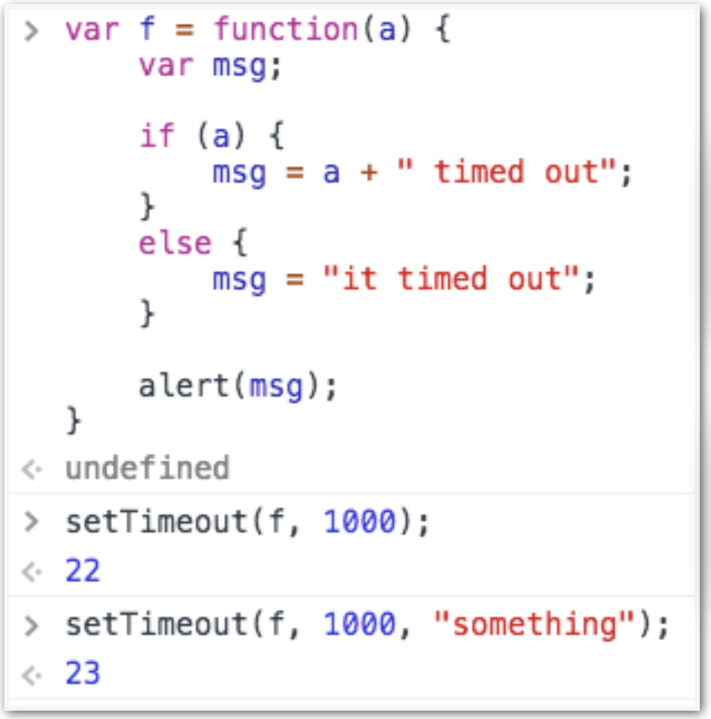
JavaScript

- Dynamically Typed
- Prototyped-based
- Functional
- **Engine evaluated script**

- JavaScript is interpreted at runtime by a JS engine
- Google's V8 (Chrome)
- Spidermonkey (Firefox)
- Apples's JavaScriptCore (Safari)
- …

Timeout

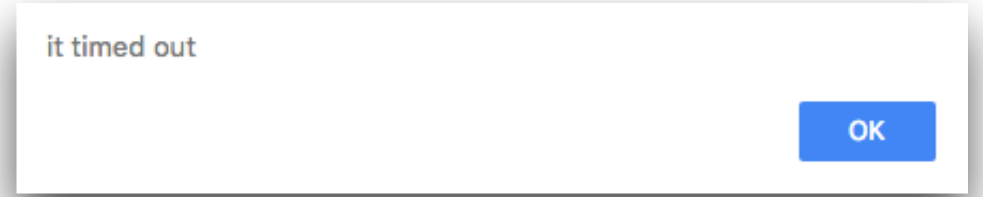
Used when we want something to occur after a certain amount of time



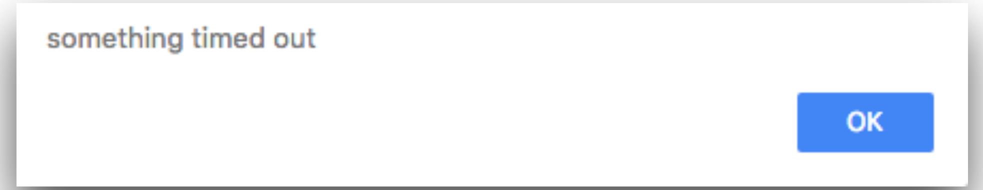
```
> var f = function(a) {
      var msg;

      if (a) {
          msg = a + " timed out";
      }
      else {
          msg = "it timed out";
      }

      alert(msg);
  }
<· undefined
> setTimeout(f, 1000);
<· 22
> setTimeout(f, 1000, "something");
<· 23
```

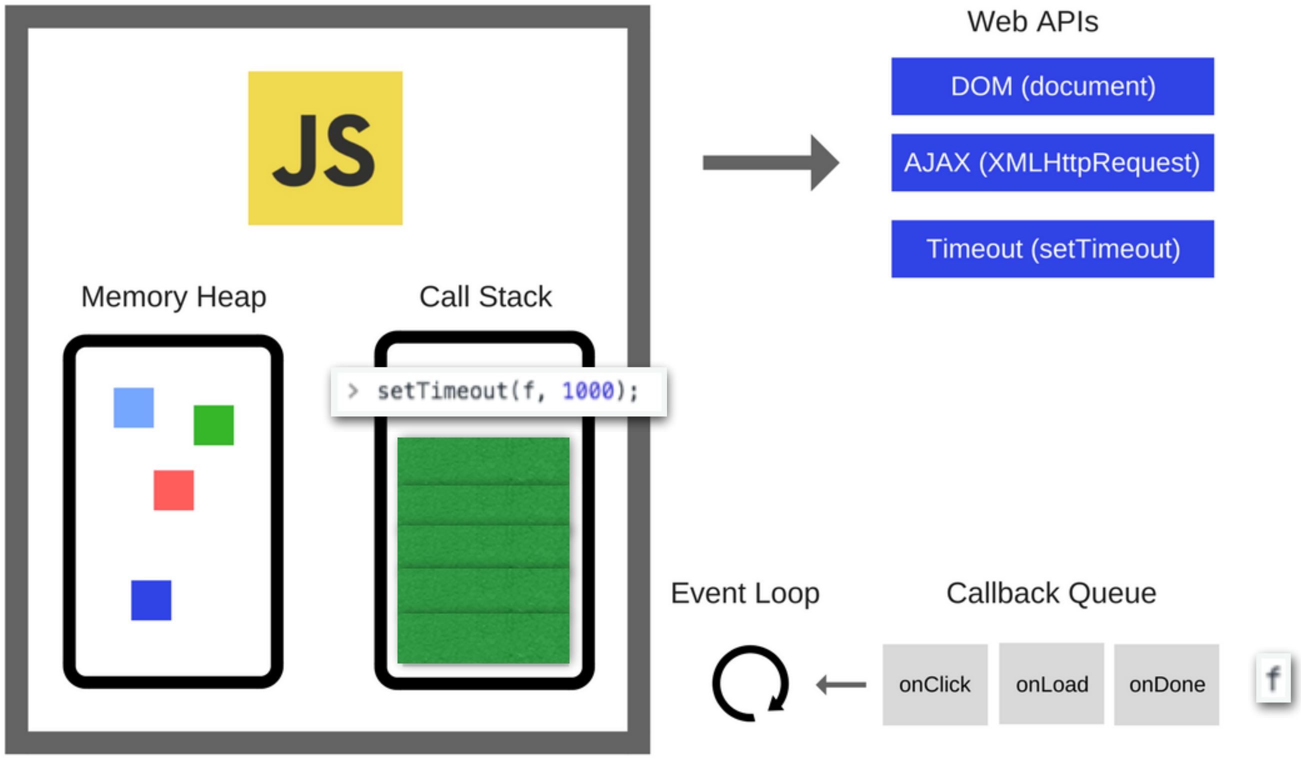


it timed out



something timed out

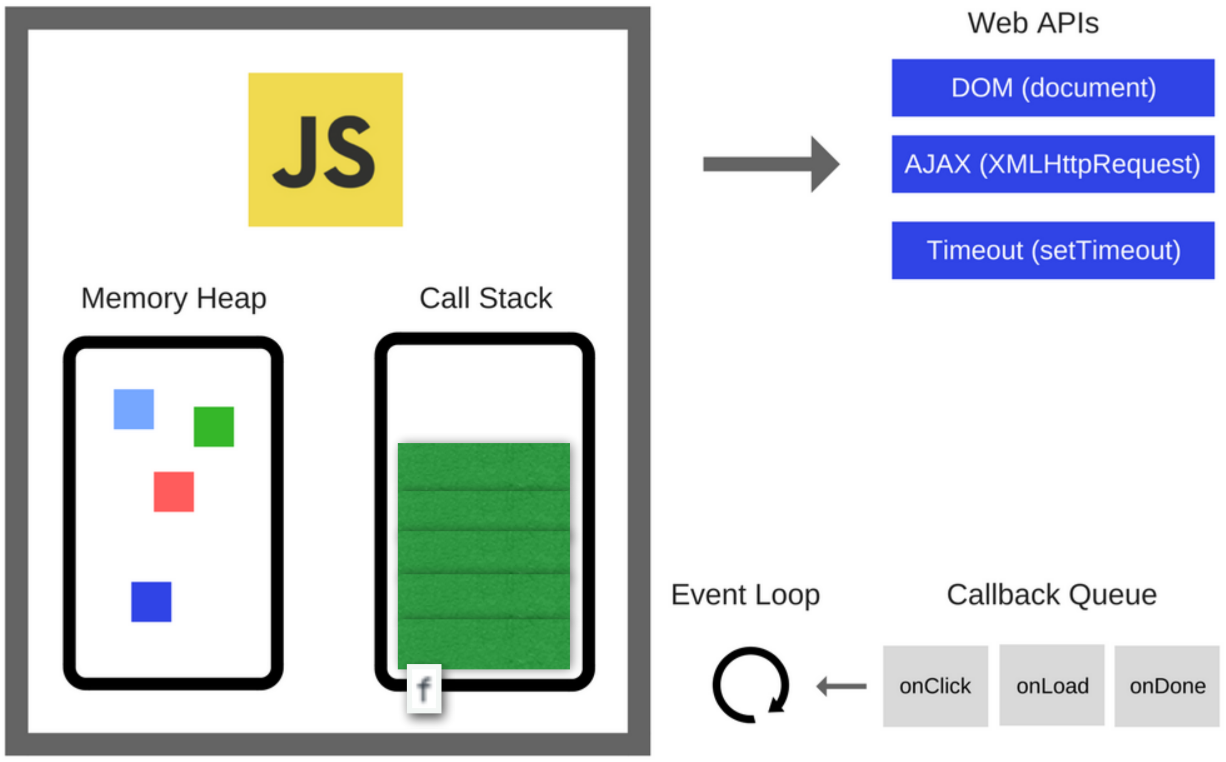
How does JavaScript work?



```
> setTimeout(f, 1000);
```

Source: https://blog.sessionstack.com/how-does-javascript-actually-work-part-1-b0bacc073cf

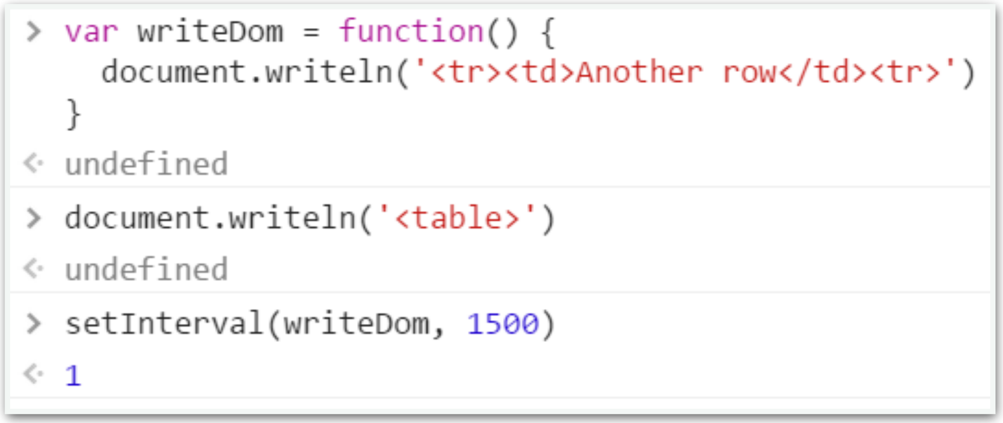
How does JavaScript work?



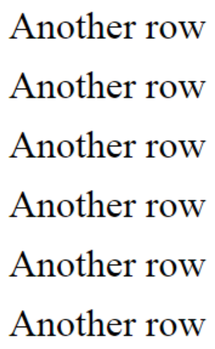
Source: https://blog.sessionstack.com/how-does-javascript-actually-work-part-1-b0bacc073cf

Interval

- Create periodic executions (exact time not guaranteed)
- Most common mistake: passing function call to *setInterval*
- Supports functions with arguments
  - setInterval(func, time, arg1, arg2)
  - setInterval(function(arg1, arg2) {…}, time)



```
> var writeDom = function() {
    document.writeln('<tr><td>Another row</td><tr>')
  }
<· undefined
> document.writeln('<table>')
<· undefined
> setInterval(writeDom, 1500)
<· 1
```



Another row

Another row

Another row

Another row

Another row

Another row

Where does the JavaScript go?

- <head>
  - The body doesn’t exist yet so don’t look for it
  - Execution is before the loading of the body
- <body onload=“go()”>
  - When the body finishes loading, the onload function is executed
  - This is obtrusive JavaScript
- <script> after body
  - No guarantee that the body is loaded

JavaScript Data Types

- boolean (true, false)
- null
- undefined
- number
- String
- Object (includes Array, Function)

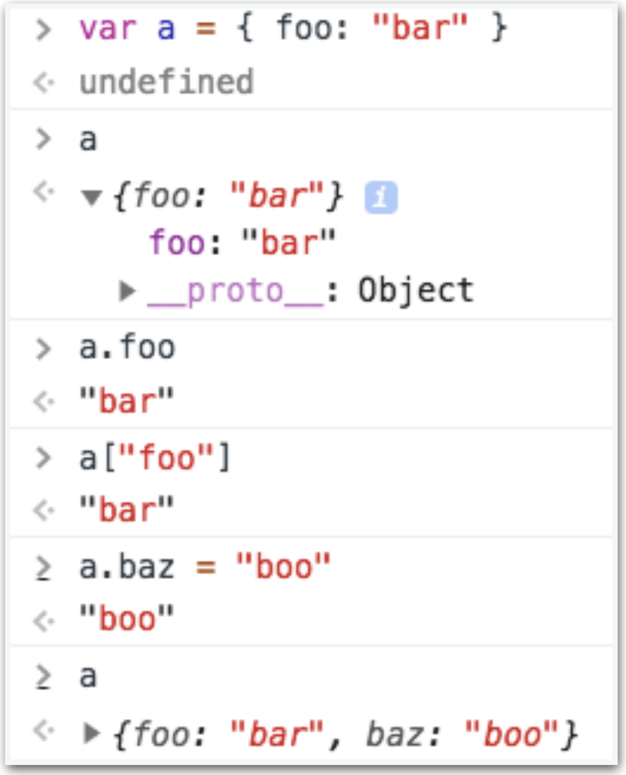
null vs undefined

- *undefined* - the value given to anything that has not been defined, e.g., declared but not initialized variable

- *null* - a special value that indicates a variable has the null value

JavaScript Objects

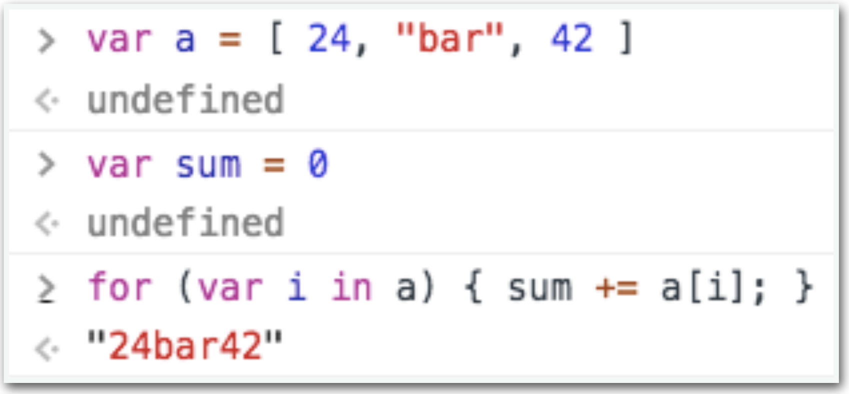
- No need to create new Object()
- Outside of primitives, everything is an object



```
> var a = { foo: "bar" }
<· undefined
> a
<· ▼{foo: "bar"} ℹ
      foo: "bar"
    ▶ __proto__: Object
> a.foo
<· "bar"
> a["foo"]
<· "bar"
> a.baz = "boo"
<· "boo"
> a
<· ▶{foo: "bar", baz: "boo"}
```

JavaScript Arrays

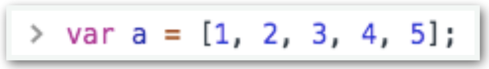
- No need to create new Array()
- Array traversal
  - for-in provides index values
  - forEach provides the values themselves
- Add element to an array
  - add to end: *push*
  - add to front: *unshift*
- Remove element from an array
  - remove from end: *pop*
  - Remove to front: *shift*



```
> var a = [ 24, "bar", 42 ]
<· undefined
> var sum = 0
<· undefined
> for (var i in a) { sum += a[i]; }
<· "24bar42"
```

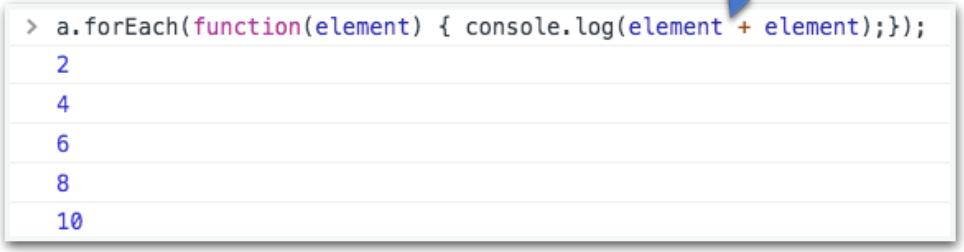
Array forEach

array.forEach(function(element) { .... }

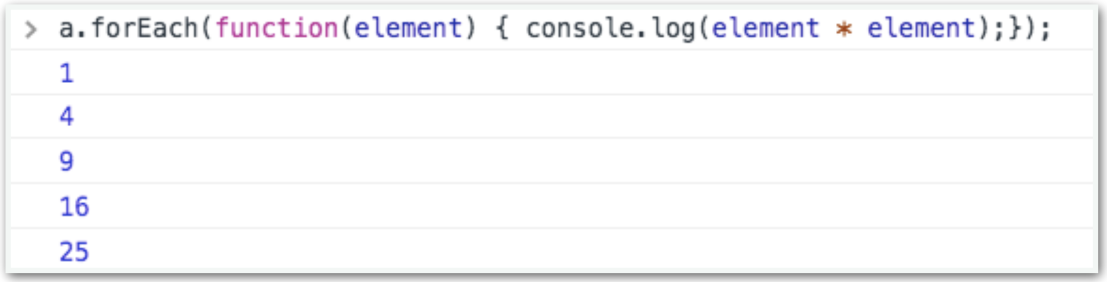


```
> var a = [1, 2, 3, 4, 5];
```

element will represent each value in array



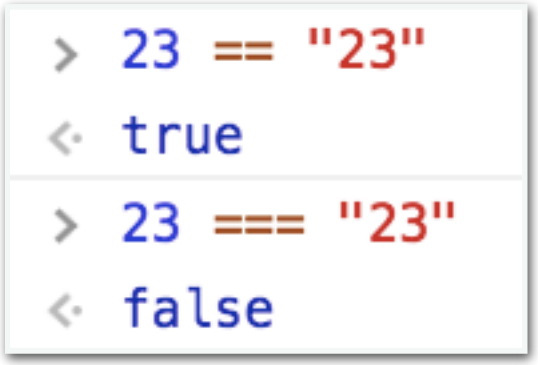
```
> a.forEach(function(element) { console.log(element + element);});
  2
  4
  6
  8
  10
```



```
> a.forEach(function(element) { console.log(element * element);});
  1
  4
  9
  16
  25
```

=, ==, ===

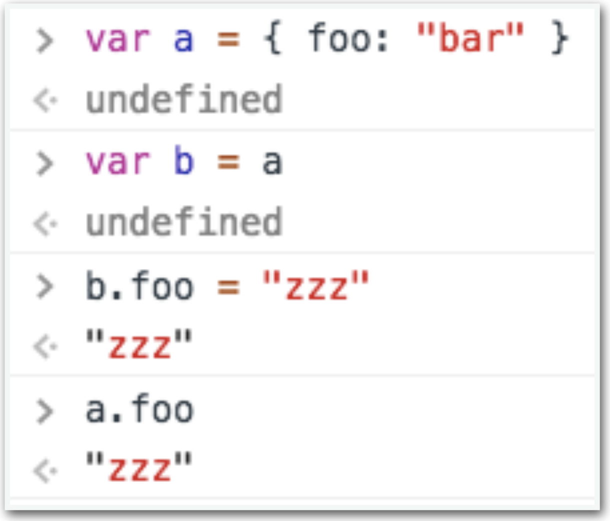
- =
  - Assignment operator
- ==, !=
  - Equality operator with type coercion
- ===, !==
  - Strict equality with no type coercion



```
> 23 == "23"
<· true
> 23 === "23"
<· false
```

References

- Primitives are accessed by value
- Objects are accessed by reference

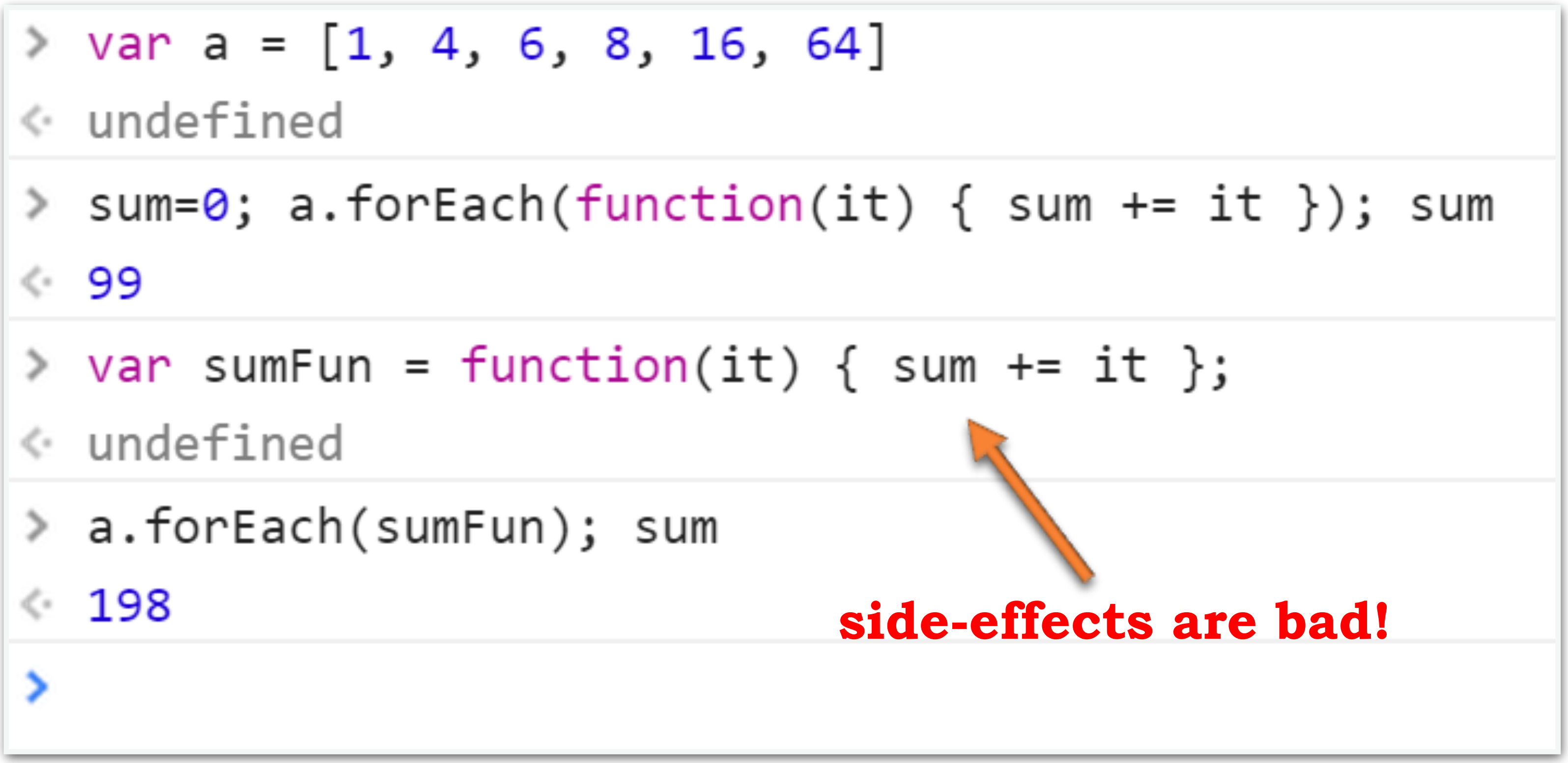


```
> var a = { foo: "bar" }
<· undefined
> var b = a
<· undefined
> b.foo = "zzz"
<· "zzz"
> a.foo
<· "zzz"
```

Control Structures

- if (condition) {…} else if (condition){…} else {…}
- var a = (condition) ? tValue: fValue;
- for (initializer; conditional; update) {…}
- while (conditional) {…}
- do {…} while (conditional)
- switch (value) { case <constant>:…; break;…default: …}
  - compares with ===
- try {…} catch (error) {…} finally {…}

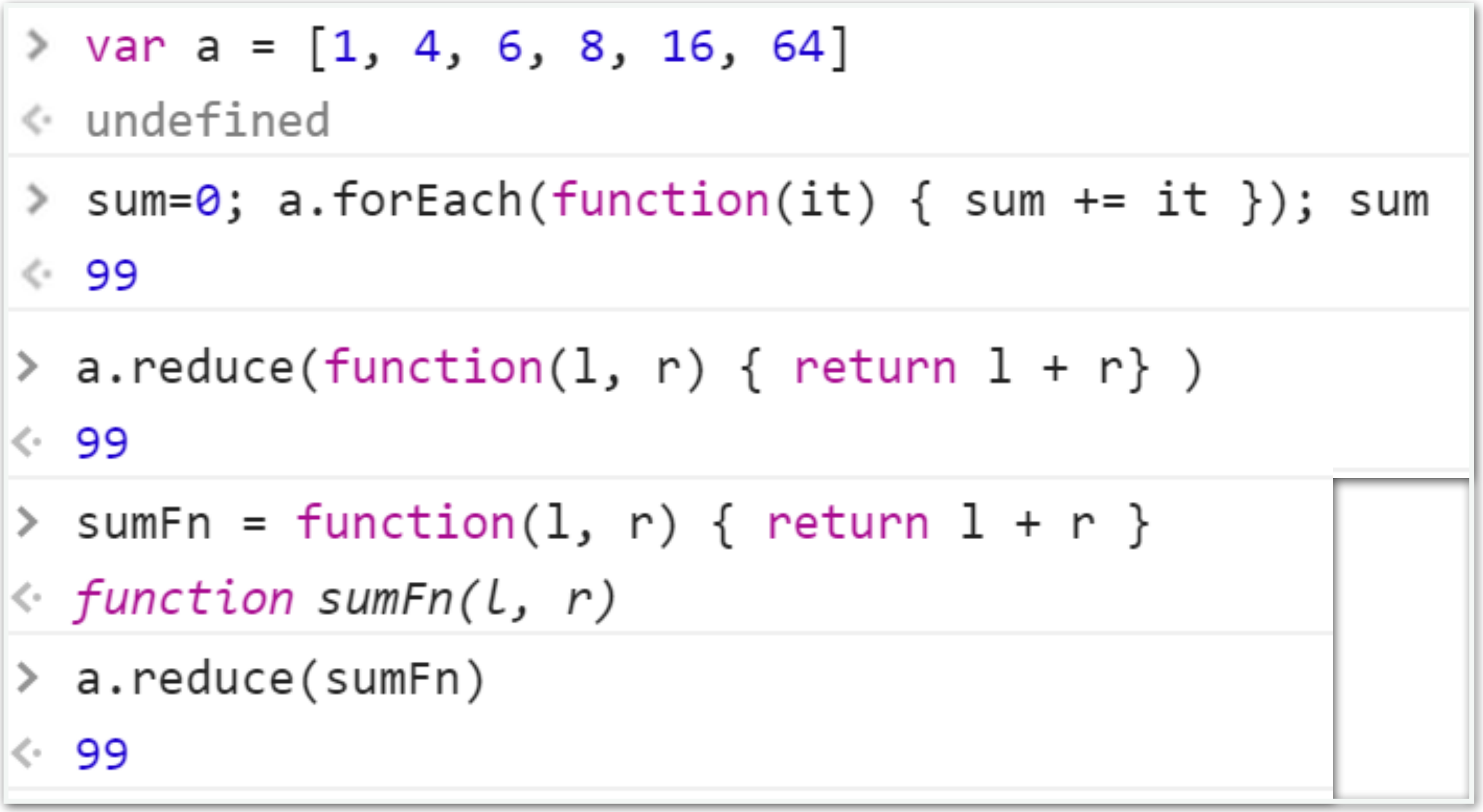
Array Functions: forEach



```
> var a = [1, 4, 6, 8, 16, 64]
<· undefined
> sum=0; a.forEach(function(it) { sum += it }); sum
<· 99
> var sumFun = function(it) { sum += it };
<· undefined
> a.forEach(sumFun); sum
<· 198
>
```

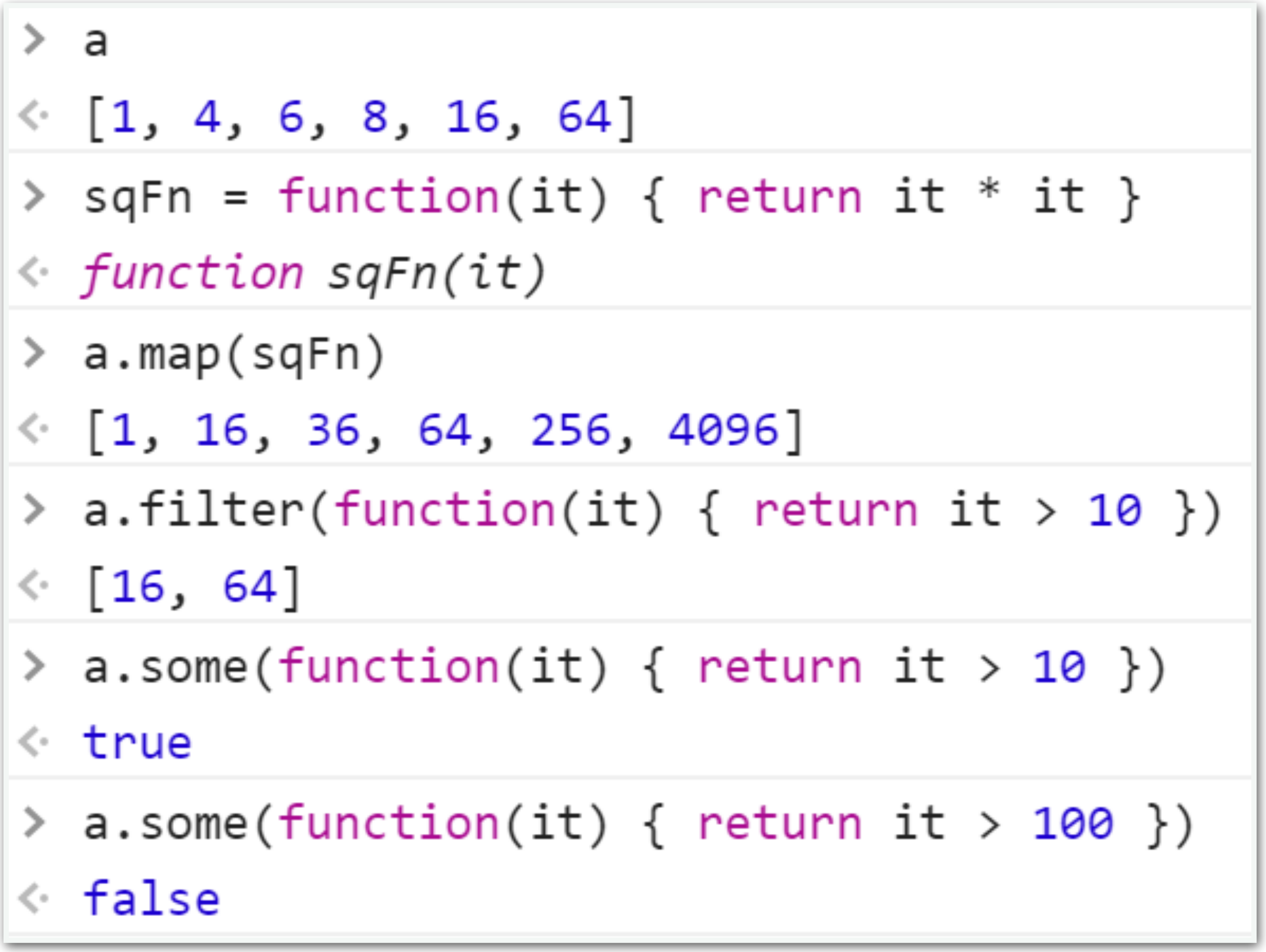
side-effects are bad!

Array Functions: reduce



```
> var a = [1, 4, 6, 8, 16, 64]
<· undefined
> sum=0; a.forEach(function(it) { sum += it }); sum
<· 99
> a.reduce(function(l, r) { return l + r} )
<· 99
> sumFn = function(l, r) { return l + r }
<· function sumFn(l, r)
> a.reduce(sumFn)
<· 99
```

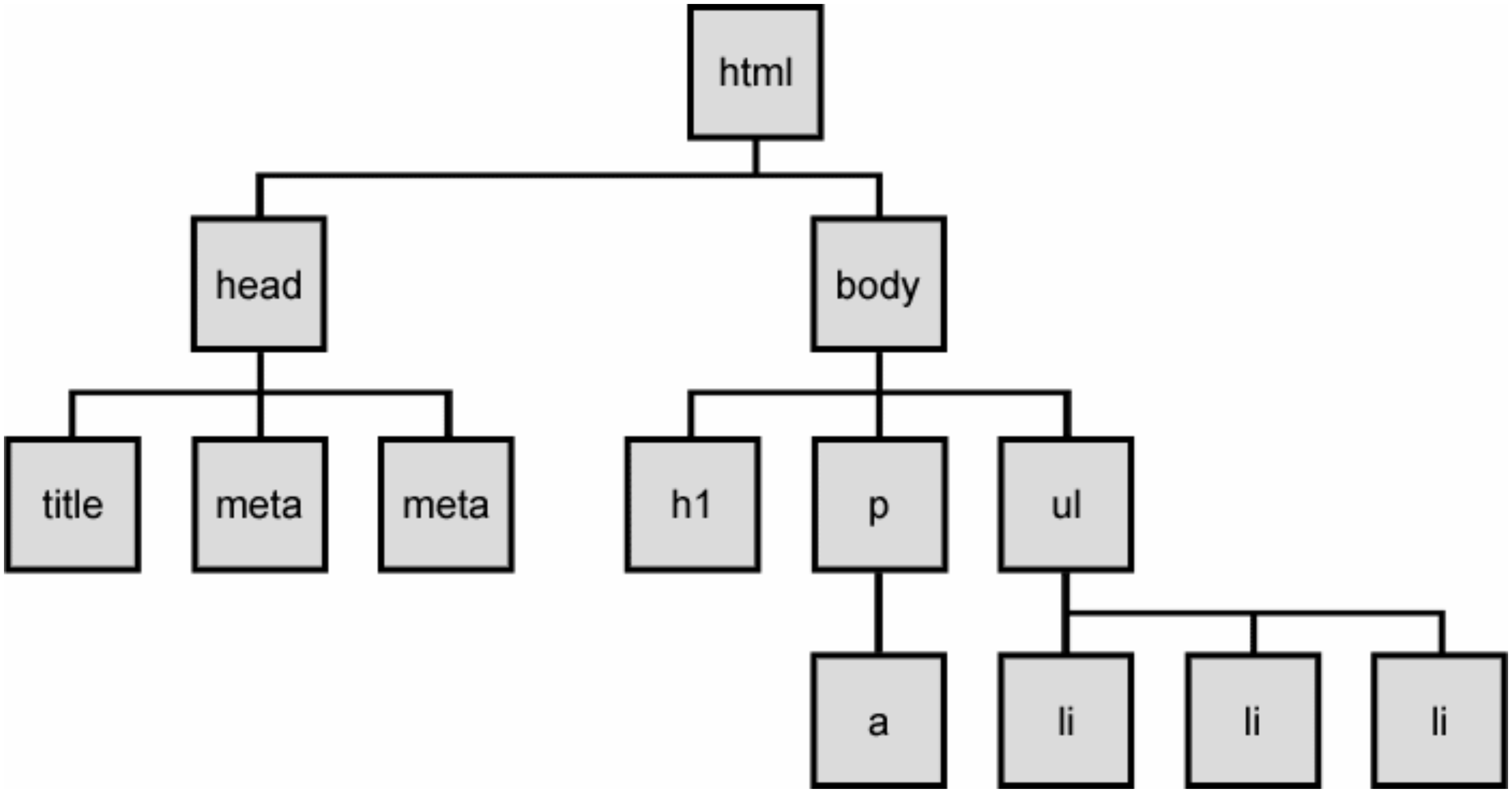
Array Functions: map, filter, some



```
> a
<· [1, 4, 6, 8, 16, 64]
> sqFn = function(it) { return it * it }
<· function sqFn(it)
> a.map(sqFn)
<· [1, 16, 36, 64, 256, 4096]
> a.filter(function(it) { return it > 10 })
<· [16, 64]
> a.some(function(it) { return it > 10 })
<· true
> a.some(function(it) { return it > 100 })
<· false
```

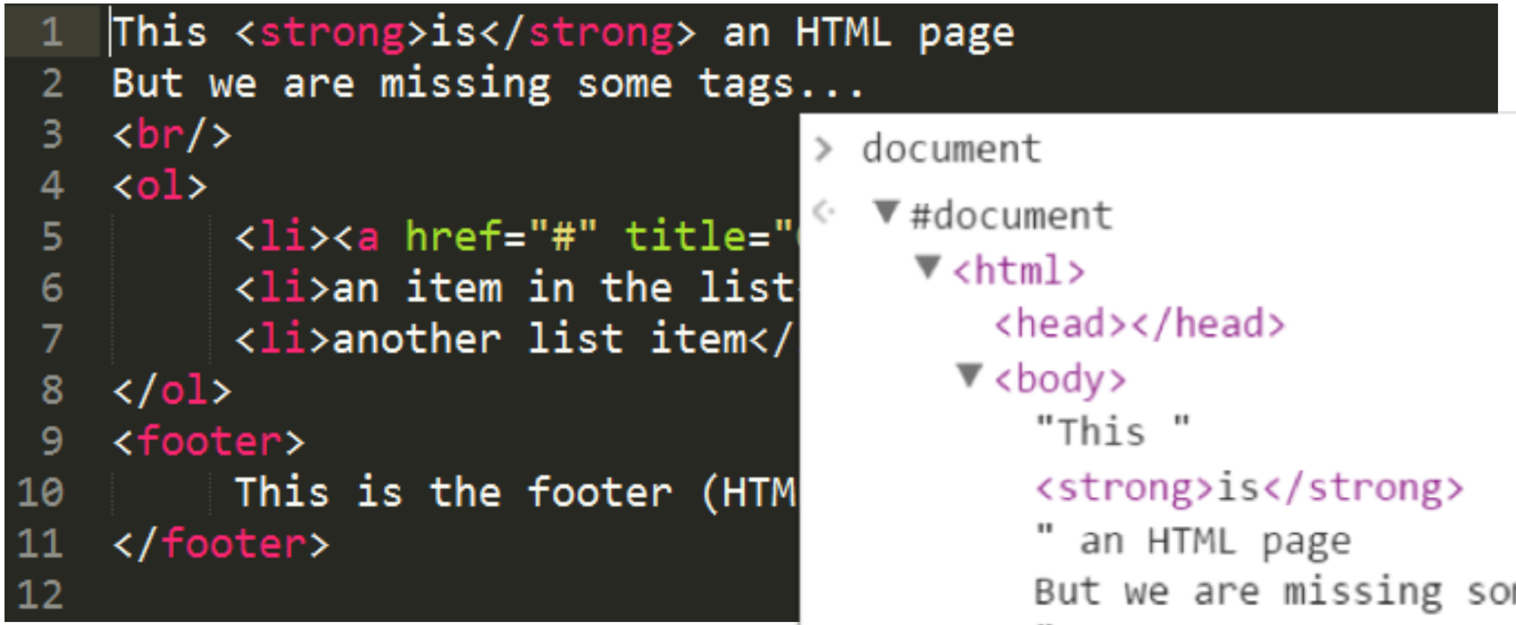
Document Object Model

- *document* provides a reference to the root of the tree



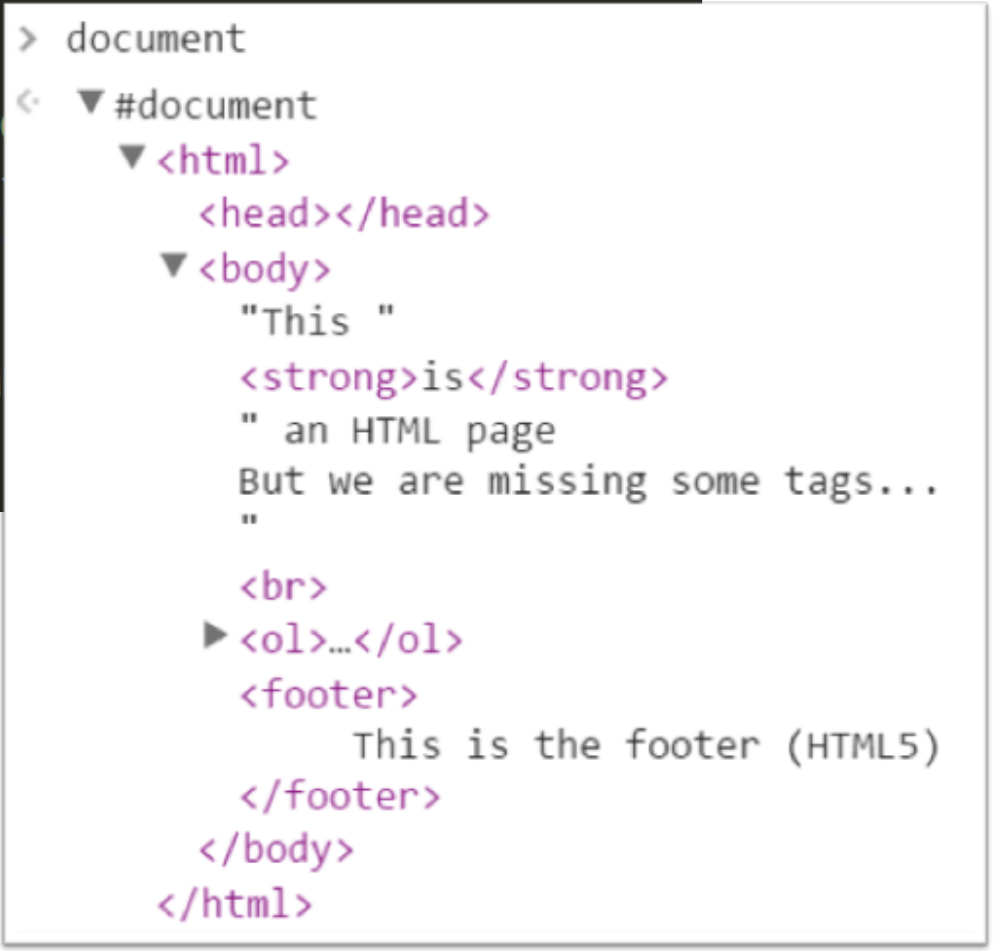
https://developer.mozilla.org/en-US/docs/Web/API/document

The DOM



```
This <strong>is</strong> an HTML page
But we are missing some tags...
<br/>
<ol>
	<li><a href="#" title="
	<li>an item in the list
	<li>another list item</
</ol>
<footer>
	This is the footer (HTM
</footer>

```

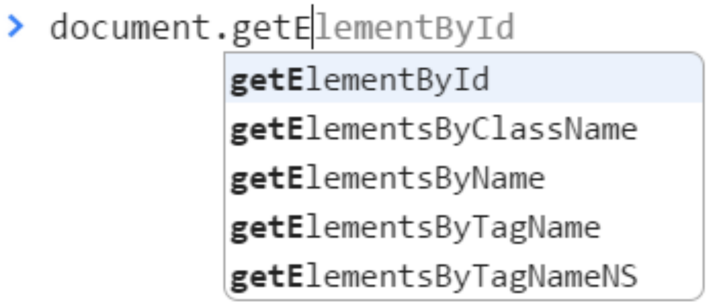


```
> document
▼ #document
  ▼ <html>
      <head></head>
    ▼ <body>
        "This "
        <strong>is</strong>
        " an HTML page
        But we are missing some tags...
        "
        <br>
      ▶ <ol>…</ol>
        <footer>
              This is the footer (HTML5)
        </footer>
      </body>
  </html>
```

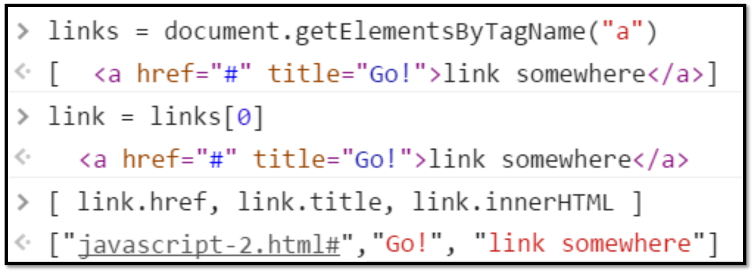
DOM Access

- Document *getElementById* returns HTMLElement
  - Assess value of InputElement: *value*
  - Access value of non-InputElement: *innerHTML*

- May need to cast result of *getElementById*



```
> document.getElementById
    getElementById
    getElementsByClassName
    getElementsByName
    getElementsByTagName
    getElementsByTagNameNS
```



```
> links = document.getElementsByTagName("a")
<· [  <a href="#" title="Go!">link somewhere</a>]
> link = links[0]
<·   <a href="#" title="Go!">link somewhere</a>
> [ link.href, link.title, link.innerHTML ]
<· ["javascript-2.html#","Go!", "link somewhere"]
```