

<PICTURE>

Prof. Andrew Sheehan
Boston University/MET
Computer Science Dept.

WORKS EVERYWHERE

Picture element - LS

A responsive images method to control which image resource a user agent presents to a user, based on resolution, media query and/or support for a particular image format

Current aligned	Usage relative	Date relative	Filtered	All	
Chrome	Edge *	Safari	Firefox	Opera	IE
4-36			2-33	10-23	
1 37	12	3.1-9	3 34-37	2 24	
38-103	13-103	9.1-15.5	38-102	25-88	6-10
104	104	15.6	103	89	11
105-107		16.0-TP	104-105	90	
					16.0
					Samsung Internet
					3.2-9.2
					9.3-15.4
					4-17.0
					104
					15.5
					18.0

The HTML `<picture>` element contains zero or more `<source>` elements and one `` element to provide versions of an image for different display/device scenarios. The browser will consider each child `<source>` element and choose the best match among them; if no matches are found, the URL of the `` element's `src` attribute is selected. The selected image is then presented in the space occupied by the `` element.

MORE ABOUT |

HOW DOES IT WORK?

The browser will decide based on the device characteristics



EXAMPLE



```
<picture>
  <source srcset="fruits_250x75.jpeg" type="image/jpeg" media="(max-width: 300px)">
  <source srcset="fruits_500x150.jpeg" type="image/jpeg" media="(max-width: 600px)">
    
</picture>
```

VARIABLES, HOISTING, CLOSURES & SCOPE

Prof. Andrew Sheehan

Boston University/MET
Computer Science Dept.



var



WHAT IS
A VARIABLE?

Purpose:

Stores a
value in a
computer
program



A DYNAMIC
LANGUAGE

NOT STRONGLY
TYPED

DATA TYPES

- Six **Data Types** that are primitives, checked by `typeof` operator:

- `undefined` : `typeof instance === "undefined"`
- `Boolean` : `typeof instance === "boolean"`
- `Number` : `typeof instance === "number"`
- `String` : `typeof instance === "string"`
- `BigInt` : `typeof instance === "bigint"`
- `Symbol` : `typeof instance === "symbol"`

- **Structural Types**:

- `Object` : `typeof instance === "object"`. Special non-data but **Structural type** for any constructed object instance also used as data structures: new `Object`, new `Array`, new `Map`, new `Set`, new `WeakMap`, new `WeakSet`, new `Date` and almost everything made with new keyword;
- `Function` : a non-data structure, though it also answers for `typeof` operator: `typeof instance === "function"`. This is merely a special shorthand for Functions, though every Function constructor is derived from Object constructor.

- **Structural Root Primitive**:

- `null` : `typeof instance === "object"`. Special primitive type having additional usage for its value: if object is not inherited, then `null` is shown;

”Welcome to the course”
‘Welcome to the course’
‘Welcome to the course’

CONST VAR & LET

`const`  Assigned?
Can't change

Block scoped.
Use it. `let` 

 `var`
larger scope!



Difference Between `var`, `let`, and `const`

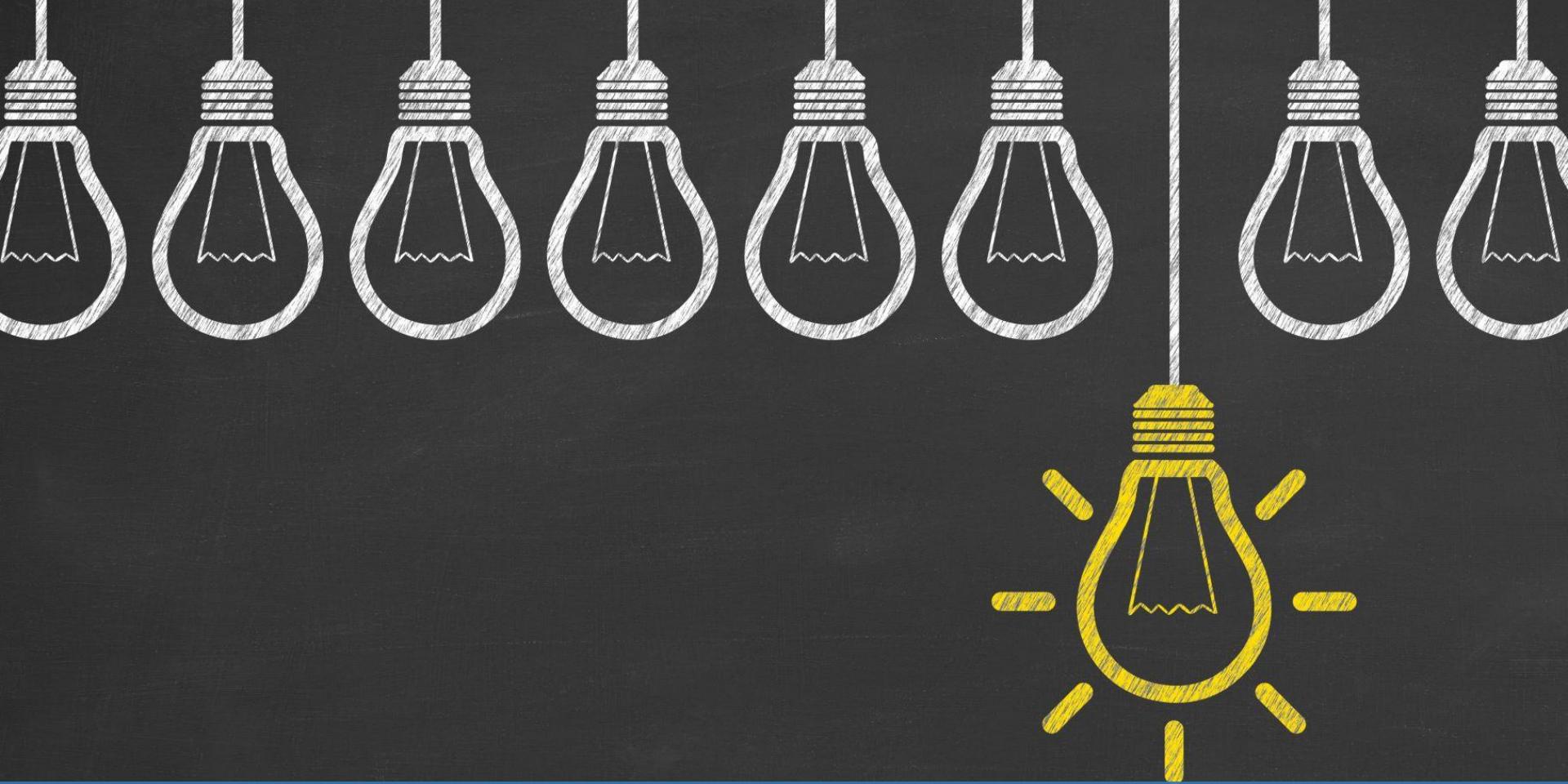
JavaScript has three different keywords to declare a variable, which adds an extra layer of intricacy to the language. The differences between the three are based on scope, hoisting, and reassignment.

Keyword	Scope	Hoisting	Can Be Reassigned	Can Be Redeclared
<code>var</code>	Function scope	Yes	Yes	Yes
<code>let</code>	Block scope	No	Yes	No
<code>const</code>	Block scope	No	No	No

You may be wondering which of the three you should use in your own programs. A commonly accepted practice is to use `const` as much as possible, and `let` in the case of loops and reassignment. Generally, `var` can be avoided outside of working on legacy code.

TYPES OF SCOPE

1. Inside a block or function (local)
2. Outside a block (global)



GLOBAL SCOPE

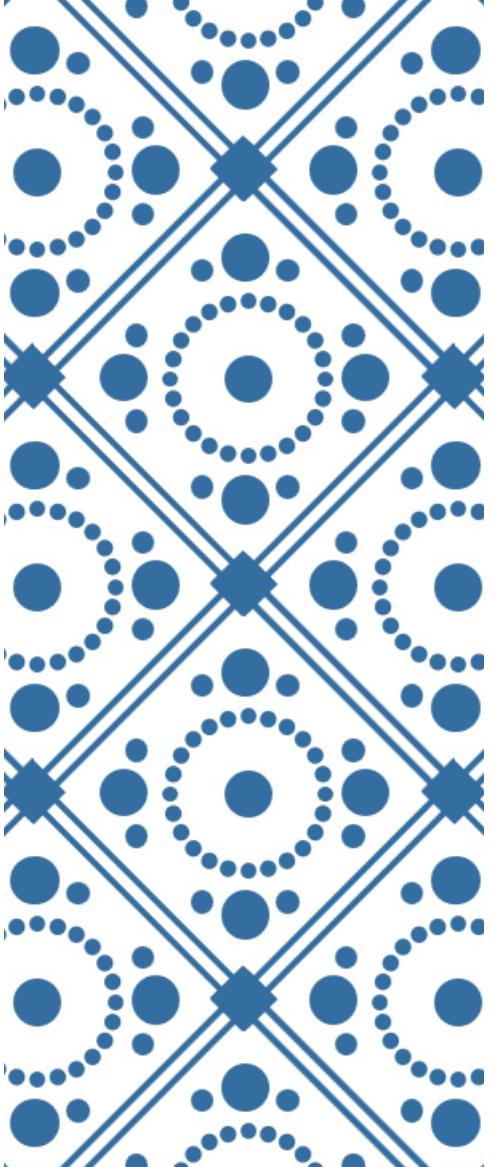
Outside any
class, function
or object

GLOBAL SCOPE

```
function check (A = 0) {
```

```
    return A > MAX;
```

```
}
```



When you use a variable before
you assign something to it, the
default value is **undefined**

**undefined IS A
PRIMITIVE TYPE**



NULL

A place
where
unicorns
live *and*
null, *as*
well

```
// Initialize x in the global scope
var x = 100;

function hoist() {
  // A condition that should not affect the outcome of the code
  if (false) {
    var x = 200;
  }
  console.log(x);
}

hoist();
```

Output

undefined

HOISTING



Hosting means the object declaration is moved up to the top of its scope.

HOISTING

All undeclared
variables are
global variables

//Notice lack of var, let or const
message = 'happy';

HOISTING EXAMPLE

```
function hoist() {  
    a = 20;  
    var b = 100;  
}  
  
hoist();  
  
console.log(a);  
/*  
Accessible as a global variable outside hoist() function  
Output: 20  
*/  
  
console.log(b);  
/*  
Since it was declared, it is confined to the hoist() function scope.  
We can't print it out outside the confines of the hoist() function.  
Output: ReferenceError: b is not defined  
*/
```

CLOSURE



A **closure** is the combination of a function bundled together (enclosed) with references to its surrounding state (the **lexical environment**). In other words, a closure gives you access to an outer function's scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.

```
function init() {  
  var name = 'Mozilla'; // name is a local variable created by init  
  function displayName() { // displayName() is the inner function, a closure  
    alert(name); // use variable declared in the parent function  
  }  
  displayName();  
}  
init();
```

EXAMPLE |

EXAMPLE

```
var counter = (function() {
  var privateCounter = 0;
  function changeBy(val) {
    privateCounter += val;
  }

  return {
    increment: function() {
      changeBy(1);
    },
    decrement: function() {
      changeBy(-1);
    },
    value: function() {
      return privateCounter;
    }
  };
})();

console.log(counter.value()); // 0.

counter.increment();
counter.increment();
console.log(counter.value()); // 2.

counter.decrement();
console.log(counter.value()); // 1.
```

FUNCTIONS

Prof. Andrew Sheehan

Boston University/MET
Computer Science Dept.

A chalkboard with several mathematical equations and a graph. The graph shows a curve labeled $y = g(x)$ and two straight lines: a tangent line at point x labeled "Tangent Line" and a secant line from x to $x+h$ labeled "Secant Lines".

Derivative definitions:

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$
$$f'(x) = \lim_{h \rightarrow 0} \frac{(x+h)^2 - x^2}{h}$$
$$= \lim_{h \rightarrow 0} \frac{x^2 + 2xh + h^2 - x^2}{h}$$
$$= \lim_{h \rightarrow 0} \frac{2xh + h^2}{h}$$
$$= \lim_{h \rightarrow 0} h(2x + h)$$

FUNCTION DECLARATIONS

```
function equationOfLine(m, x, b) {  
    return (m * x) + b;
```

FUNCTIONS ARE OBJECTS

Name of the object
(*The function name*)

Parameters
(*Optional*)

```
function validateScreenName(screamName) {  
    /* This is the function body */  
}
```

RETURNING A VALUE

```
function validateScreenName(screenName) {  
}  
} 
```

Nothing returned?
undefined

Parentheses:
always used

NO SEMICOLON

```
function validateName(name) {
```

```
}
```

 Semicolon at the end of a function
declaration is not required.

PARAMETERS

```
const now = new Date();
healthCheck(message, now);
```

```
function healthCheck(message) {
  let timestamp = arguments[1] || new Date();
}
```

PASSING FUNCTIONS AS PARAMETERS

```
function health(message) {  
  message = message || 'no message found';  
}
```

```
function check(time) {  
  let timestamp = time || new Date();  
}
```

```
function ping(health, check) {  
  health("Pinging the server");  
  check();  
}
```

SHADOWING

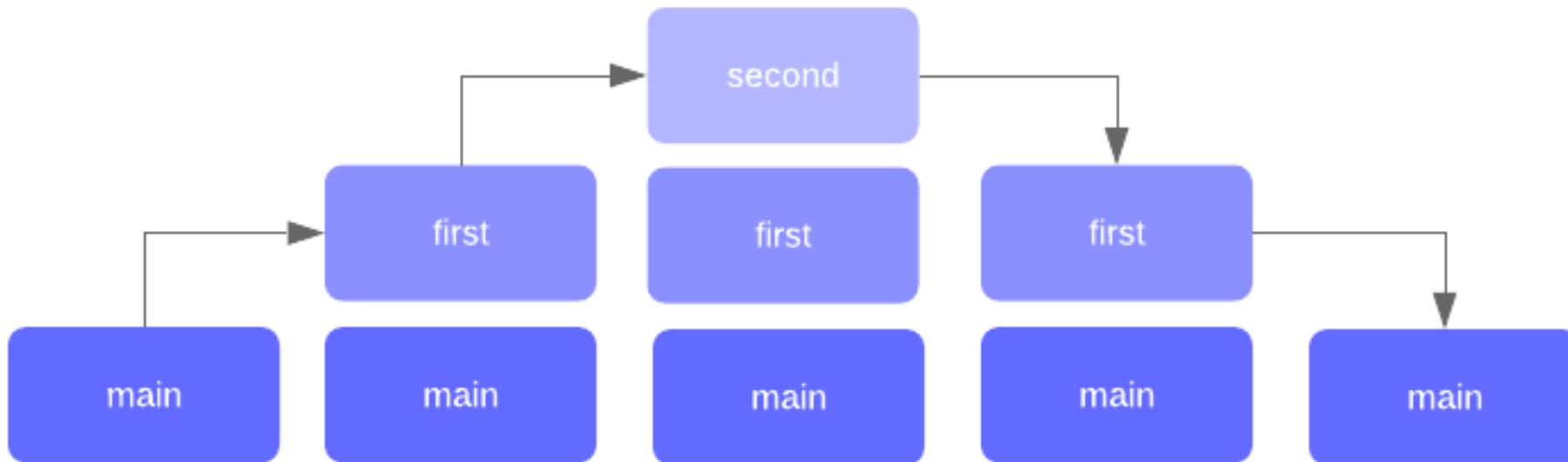
```
flag = 'On' // Global

function isLucky() {
    var flag = 'Off' // shadow value!

    return flag === 'On';
}
```

FUNCTIONS THE CALL STACK

Most recently invoked will be on the top of the call stack.



CORE JQUERY

Prof. Andrew Sheehan

Boston University/MET
Computer Science
Department

AGENDA

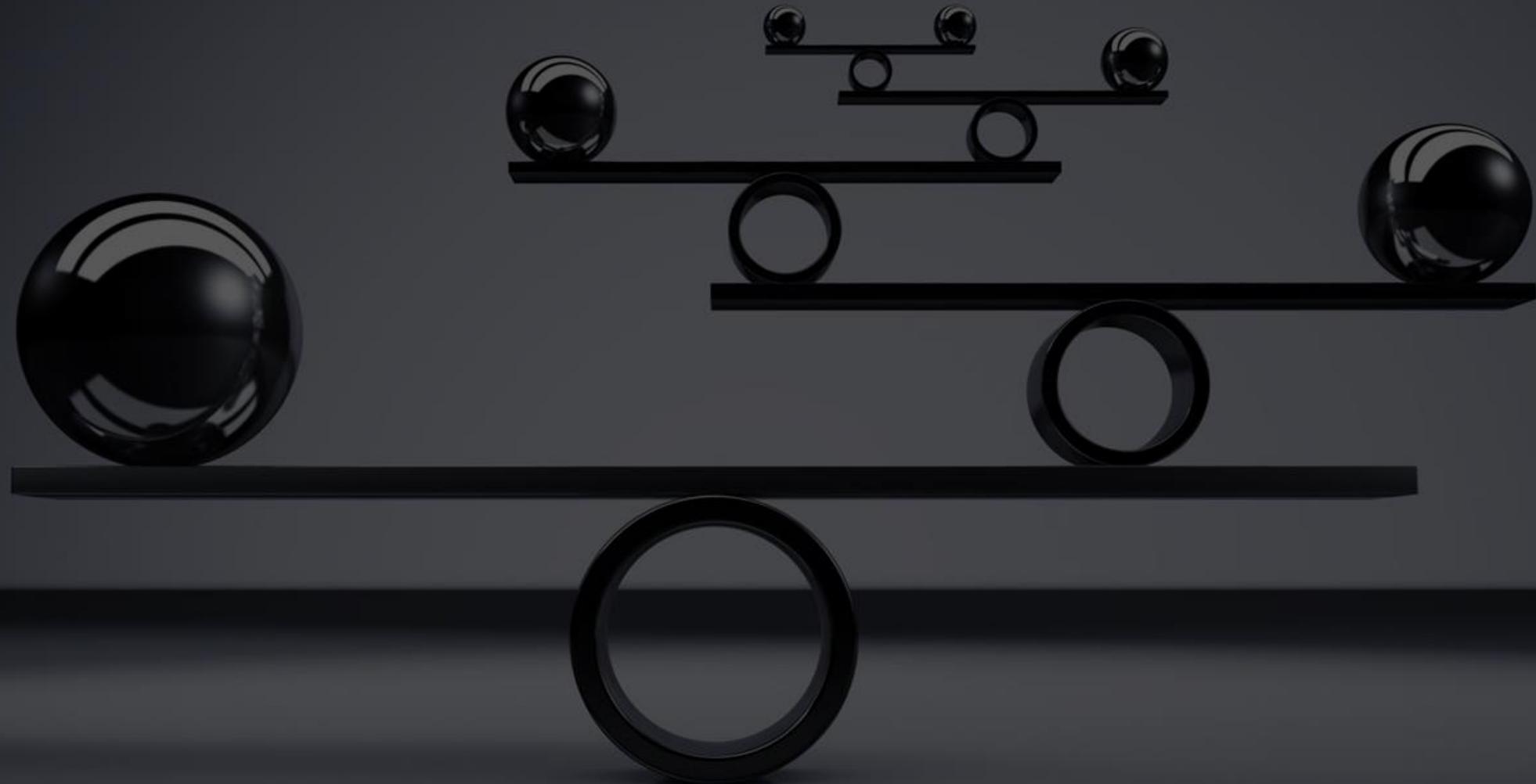
A high-level overview
of jQuery core



WHY USE JQUERY

Make front-end
development easier.





EASY DOM
MANIPULATION

TL;DR

JQUERY: LOADING

```
<script>  
<link>
```



Current Active Support

Desktop

- Chrome: (Current - 1) and Current
- Edge: (Current - 1) and Current
- Firefox: (Current - 1) and Current, ESR
- Internet Explorer: 9+
- Safari: (Current - 1) and Current
- Opera: Current

Support



DOM READY

Use the
DOMContentLoaded
event

CORE FEATURES

Event binding, DOM selection and UI controls



example

```
getElementById("target")
    .style.display = "block";
$('#target').show();

getElementById("target")
    .innerHTML = "<b>hi</b>";
$('#target').html("<b>h</b>");
```

jQuery object

(CSS Selector)

the target

```
// Sets up click behavior on all button elements with the alert class  
// that exist in the DOM when the instruction was executed  
$( "button.alert" ).on( "click", function() {  
    console.log( "A button with the alert class was clicked!" );  
});
```

Event
(What you are
interested in)

(Event Handler)
the callback

CSS MANIPULATION

`$(‘CSS Selector goes
here’).[DO SOMETHING]`

JQUERY AND CSS

```
$(‘selector’).css(‘target’, ‘value');
```

```
const hVal = $(‘selector’).css(‘height’);
```

EVENTS

Most/All events are available for use with jQuery

tutorialsteacher.com/jquery/jquery-event
use the `on()` API

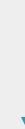
The use of the older methods – like ‘bind()’ is not recommended. (It is an ancient version of jQuery’s API methods.)

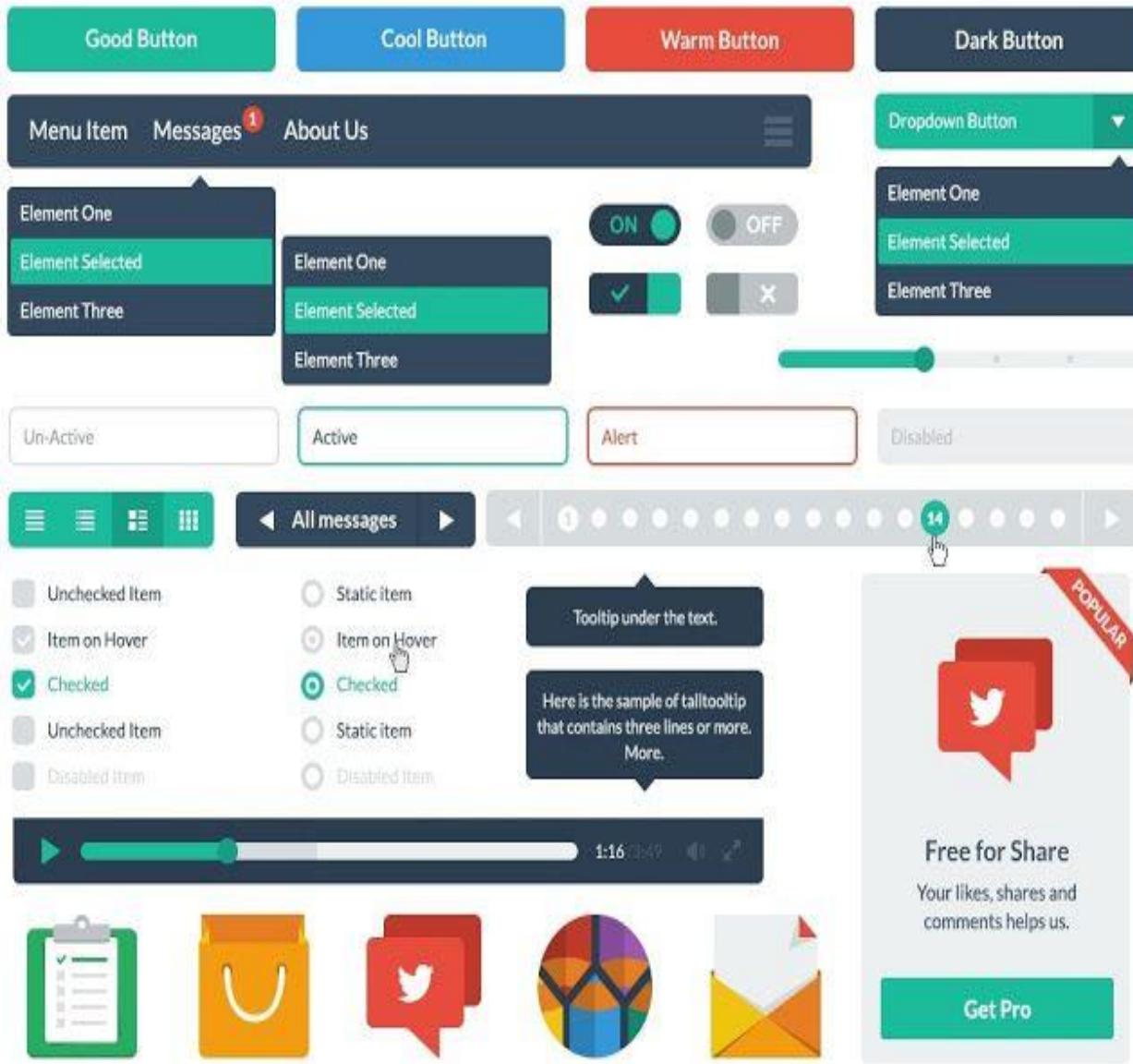
EXAMPLES USING ON()

```
1 | $( "p" ).on( "click", function() {  
2 |   alert( $( this ).text() );  
3 | });
```



```
1 | function notify() {  
2 |   alert( "clicked" );  
3 | }  
4 | $( "button" ).on( "click", notify );
```





UI controls

Just a sampling...



```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>css demo</title>
6   <style>
7     p {
8       color: blue;
9       font-weight: bold;
10      cursor: pointer;
11    }
12  </style>
13  <script src="https://code.jquery.com/jquery-3.5.0.js"></script>
14 </head>
15 <body>
16
17 <p>
18 Once upon a time there was a man
19 who lived in a pizza parlor. This
20 man just loved pizza and ate it all
21 the time. He went on to be the
22 happiest man in the world. The end.
23 </p>
24
25 <script>
26 var words = $( "p" ).first().text().split( /\s+/ );
27 var text = words.join( "</span> <span>" );
28 $( "p" ).first().html( "<span>" + text + "</span>" );
29 $( "span" ).on( "click", function() {
30   $( this ).css( "background-color", "yellow" );
31 });
32 </script>
33
34 </body>
35 </html>
```

CSS EXAMPLES

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>css demo</title>
6   <style>
7     p {
8       color: green;
9     }
10  </style>
11  <script src="https://code.jquery.com/jquery-3.5.0.js"></script>
12 </head>
13 <body>
14
15 <p>Move the mouse over a paragraph.</p>
16 <p>Like this one or the one above.</p>
17
18 <script>
19 $( "p" )
20   .on( "mouseenter", function() {
21     $( this ).css({
22       "background-color": "yellow",
23       "font-weight": "bolder"
24     });
25   })
26   .on( "mouseleave", function() {
27     var styles = {
28       backgroundColor : "#ddd",
29       fontWeight: ""
30     };
31     $( this ).css( styles );
32   });
33 </script>
34
35 </body>
36 </html>
```



```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <title>css demo</title>
6     <style>
7       div {
8         width: 20px;
9         height: 15px;
10        background-color: #f33;
11      }
12    </style>
13    <script src="https://code.jquery.com/jquery-3.5.0.js"></script>
14  </head>
15  <body>
16
17  <div>click</div>
18  <div>click</div>
19
20 <script>
21 $( "div" ).on( "click", function() {
22   $( this ).css({
23     width: function( index, value ) {
24       return parseFloat( value ) * 1.2;
25     },
26     height: function( index, value ) {
27       return parseFloat( value ) * 1.2;
28     }
29   });
30 });
31 </script>
32
33 </body>
34 </html>
```

CDN'S

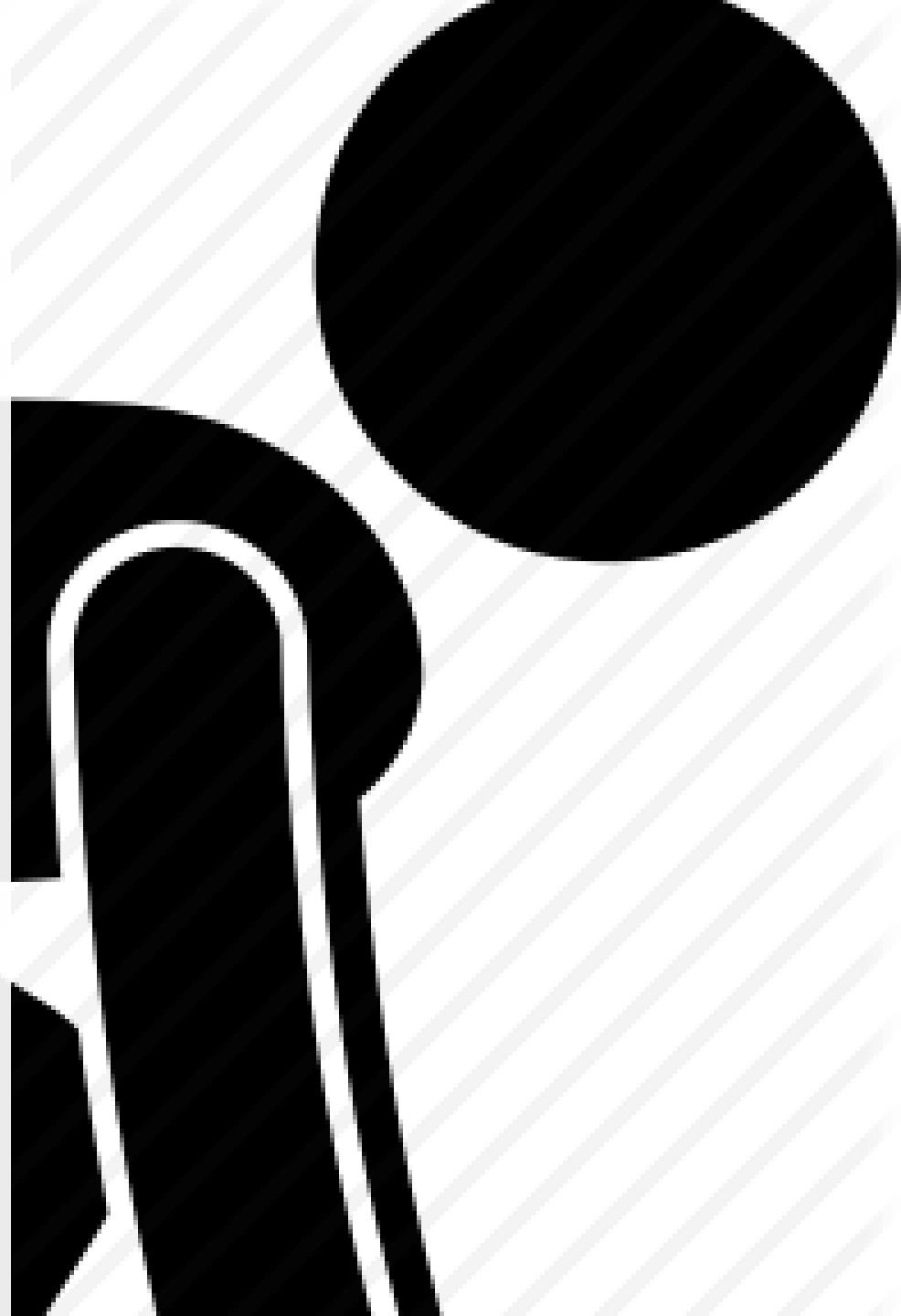
cdnjs.com/libraries

code.jquery.com

developers.google.com/speed/libraries#libraries

READY() ?

When your DOM is
ready to be
manipulated.



INTRODUCTION TO CSS

Prof. Andrew Sheehan

Boston University/MET
Computer Science Dept.



Bert Bos
Co-creator of CSS



Håkon Wium Lie
Co-creator of CSS



A STANDARD

Cascading Style Sheets (CSS) is a language used for describing the look and formatting of a document written in HTML

BENEFITS

We get faster downloads after 1st time with external css files.

Separation of style and structure and greater control of the presentation tier.

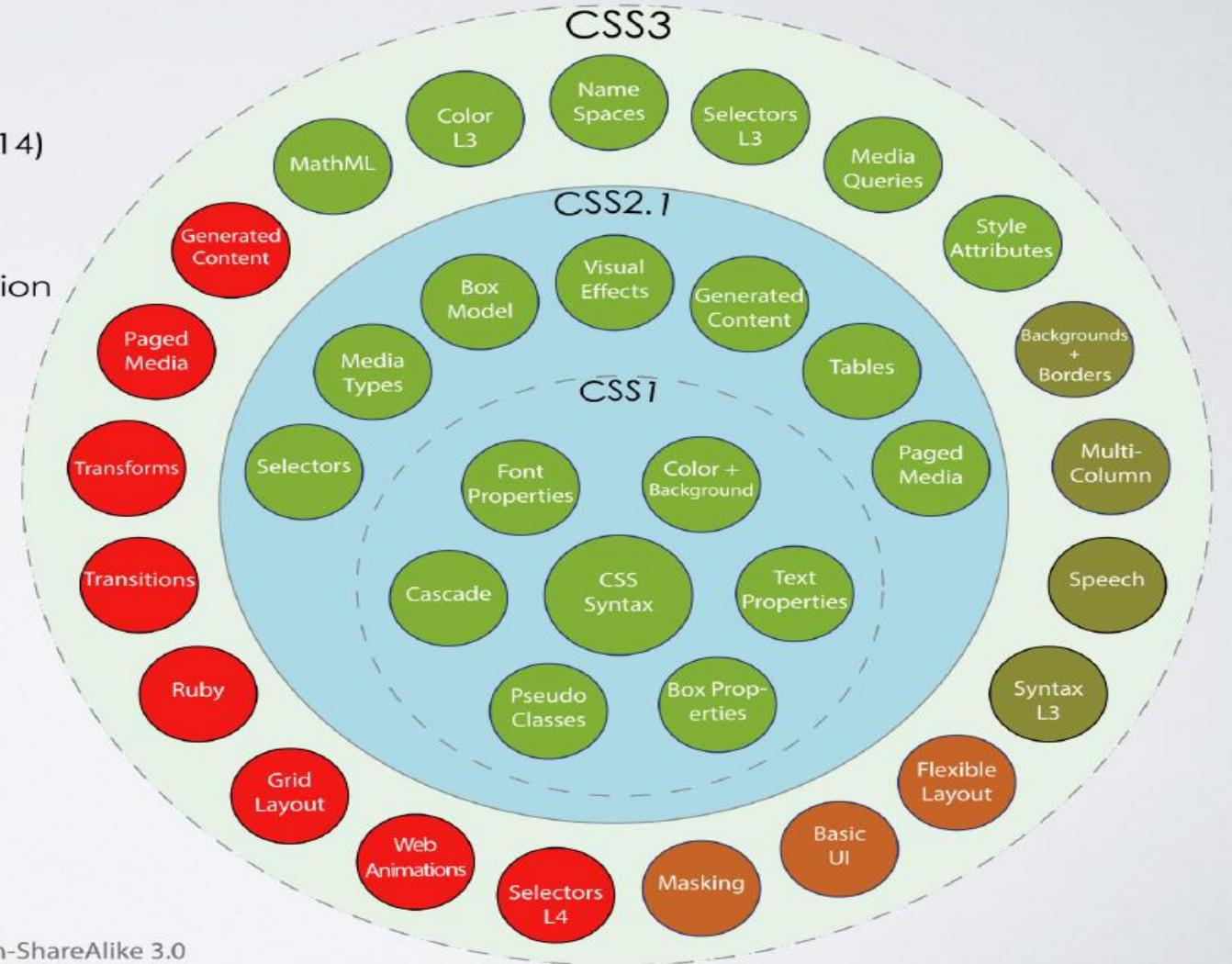
Simpler page means your site will be indexed/categorized better by Bing, Google and other search engines

Each level of CSS builds on the previous, refining definitions and adding more features.

CSS3

Taxonomy & Status (October 2014)

- W3C Recommendation
- Candidate Recommendation
- Last Call
- Working Draft
- Obsolete or inactive





NOT FULLY SUPPORTED YET

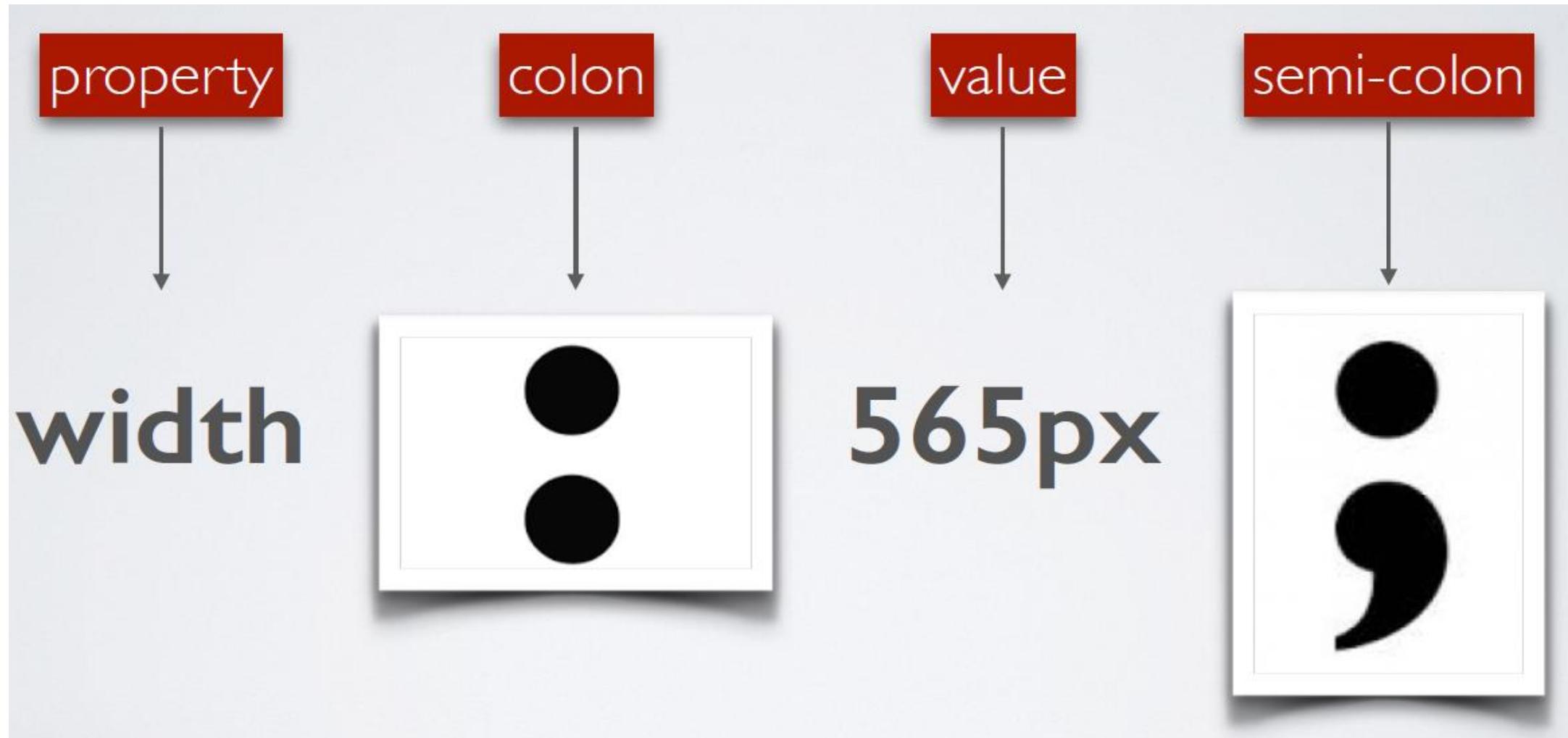


TEST. TEST. TEST.

Your clients should not
find your problems.

Test before you
release.

PROPERTY DECLARATION/SYNTAX



Selector

h1

Declaration

{ color:blue; font-size:12px; }

Property

Value

Declaration

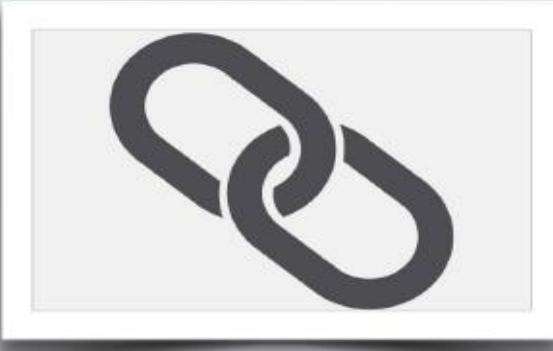
↑

Property

↑

Value

SYNTAX |



Linked

<link href="..../styles/style.css" type="text/css" rel="stylesheet"/>



Embedded

<style>....</style>



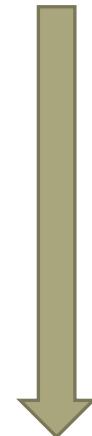
Inline

<div style="float: right; width: 545px; height: 455px">...</div>

CHANGING

Declaring a rule at the beginning of your CSS can be updated or changed below it's declaration

```
hr {  
    color: blue;  
}
```

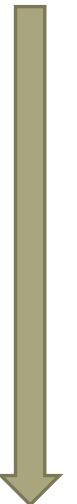


```
hr {  
    color: red;  
}
```



AUGMENTING

Declaring a rule can be added to later down in your CSS

```
hr {  
    color: blue;  
}  
  
  
  
hr {  
    color: blue;  
    width: 80%;  
    letter-spacing: 2px;  
}
```



USER AND AGENT SENSORS



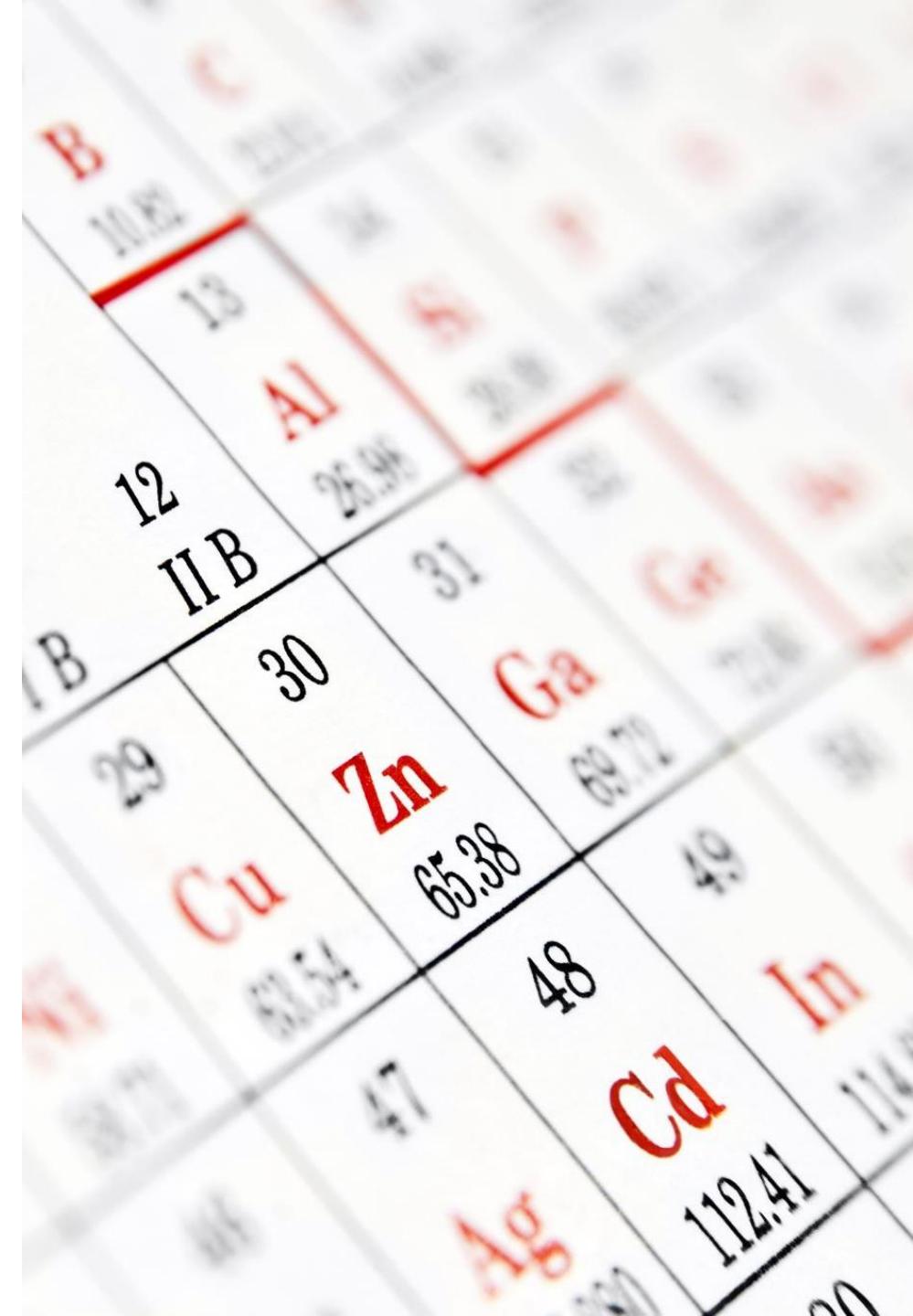
```
    if operation == "MIRROR_X":  
        mirror_mod.use_x = True  
        mirror_mod.use_y = False  
        mirror_mod.use_z = False  
    elif operation == "MIRROR_Y":  
        mirror_mod.use_x = False  
        mirror_mod.use_y = True  
        mirror_mod.use_z = False  
    elif operation == "MIRROR_Z":  
        mirror_mod.use_x = False  
        mirror_mod.use_y = False  
        mirror_mod.use_z = True  
  
    #selection at the end - add  
    #_ob.select= 1  
    #mirror_ob.select=1  
    context.scene.objects.active = bpy.data.objects["Selected" + str(modifier_index)]  
    mirror_ob.select = 0  
    bpy.context.selected_objects.append(mirror_ob)  
    data.objects[one.name].select = 1  
  
    print("please select exactly one object")  
  
-- OPERATOR CLASSES --
```

Browsers have defaults. They have to.
Every property has a value of some type.

THE RUBRIC USED IN CSS DETERMINATIONS

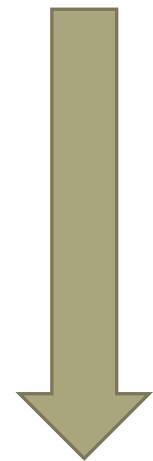
The CSS specification describes a set of rules to determine which style rules apply if more than one rule matches against a particular element.

Priorities or weights are calculated and assigned. Results are reliable and predictable.



ORDER OF IMPORTANCE

highest



lowest

- Author inline styles
- Author embedded styles
- Author external style sheet
- User style sheet
- Default browser style sheet

ORDER MATTERS

```
1. p {color: black;}  
2. ul {border: 1px solid pink;}  
3. p.intro {color: brown;}  
4. p {color: red;}
```

In the code above, we have created rules for paragraphs to be three different colors. Clearly, these rules conflict with one another.

Rule #3 is the most specific because it specifies all paragraphs that also have the class attribute value of `intro`.

Rules #1 and #4 conflict. They are from the same style sheet and they have the same level of specificity. If all else is equal in the cascade, we sort by order specified.

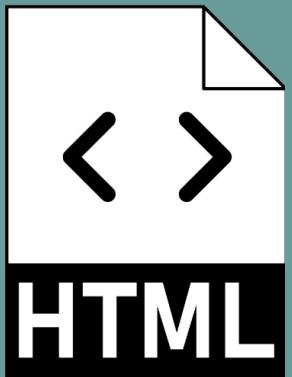
Rule #4 is declared last in the CSS document and therefore, overrides the previously declared Rule #1.

THE CASCADE

Linked CSS rules will be trumped by embedded and finally inline CSS trumps them all



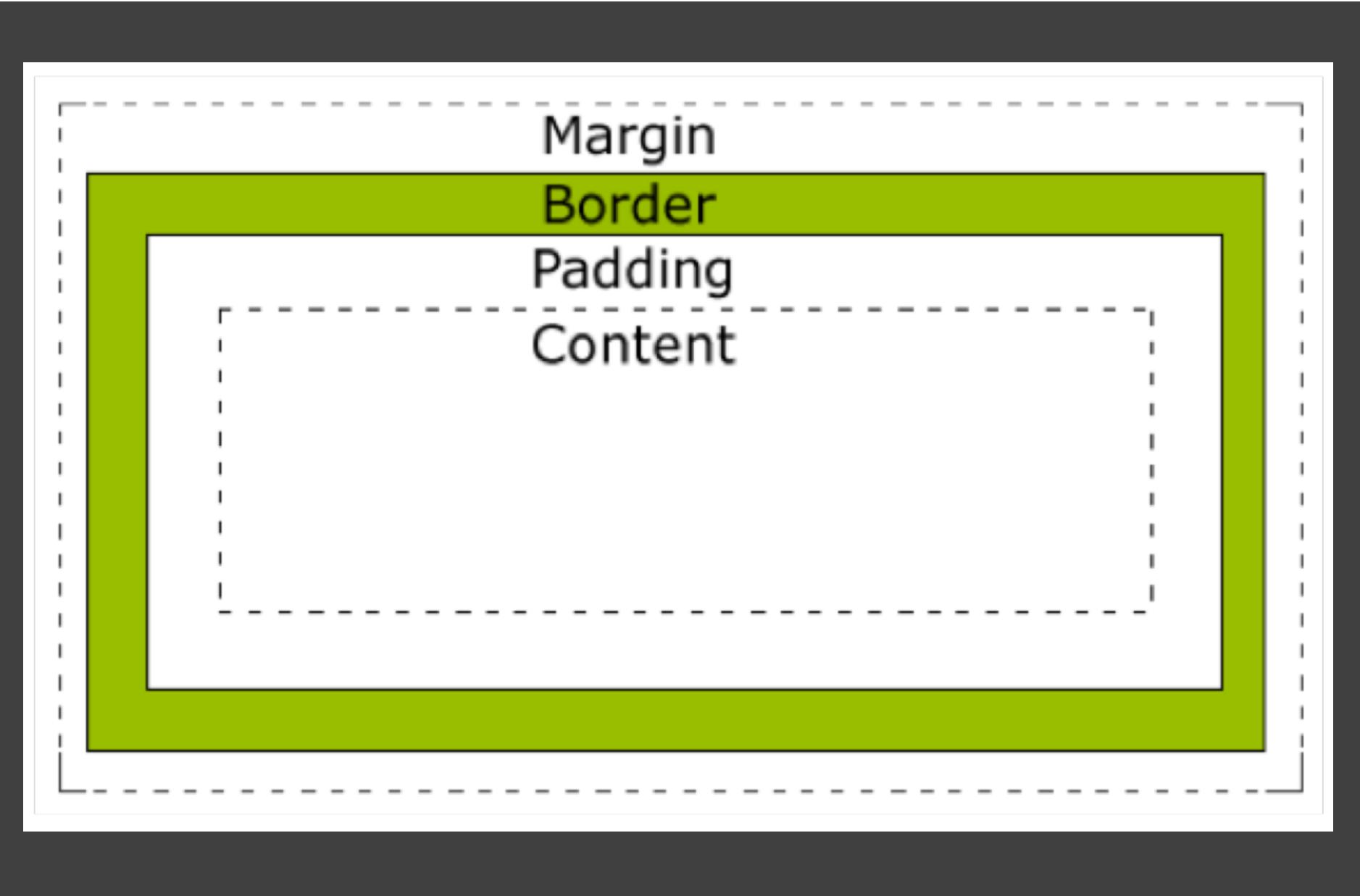
<link>



<style>

<p style=""></p>

THE BOX MODEL



IMPORTANT

```
/* The important rule can be used to  
override your styling rules. */  
  
p > span.odd {  
    font-weight: bold !important;  
}
```

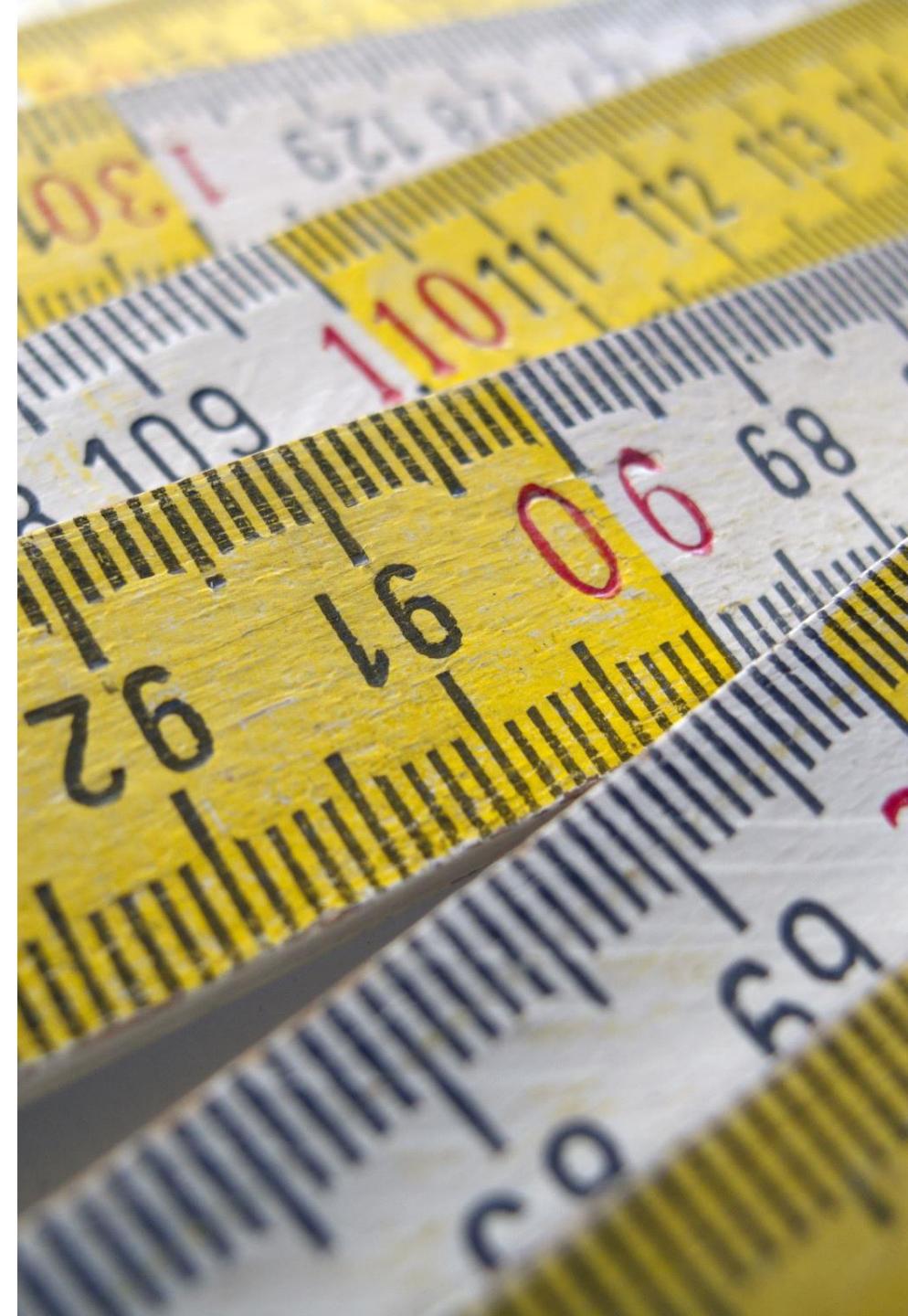
MEASUREMENTS

developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Values_and_units



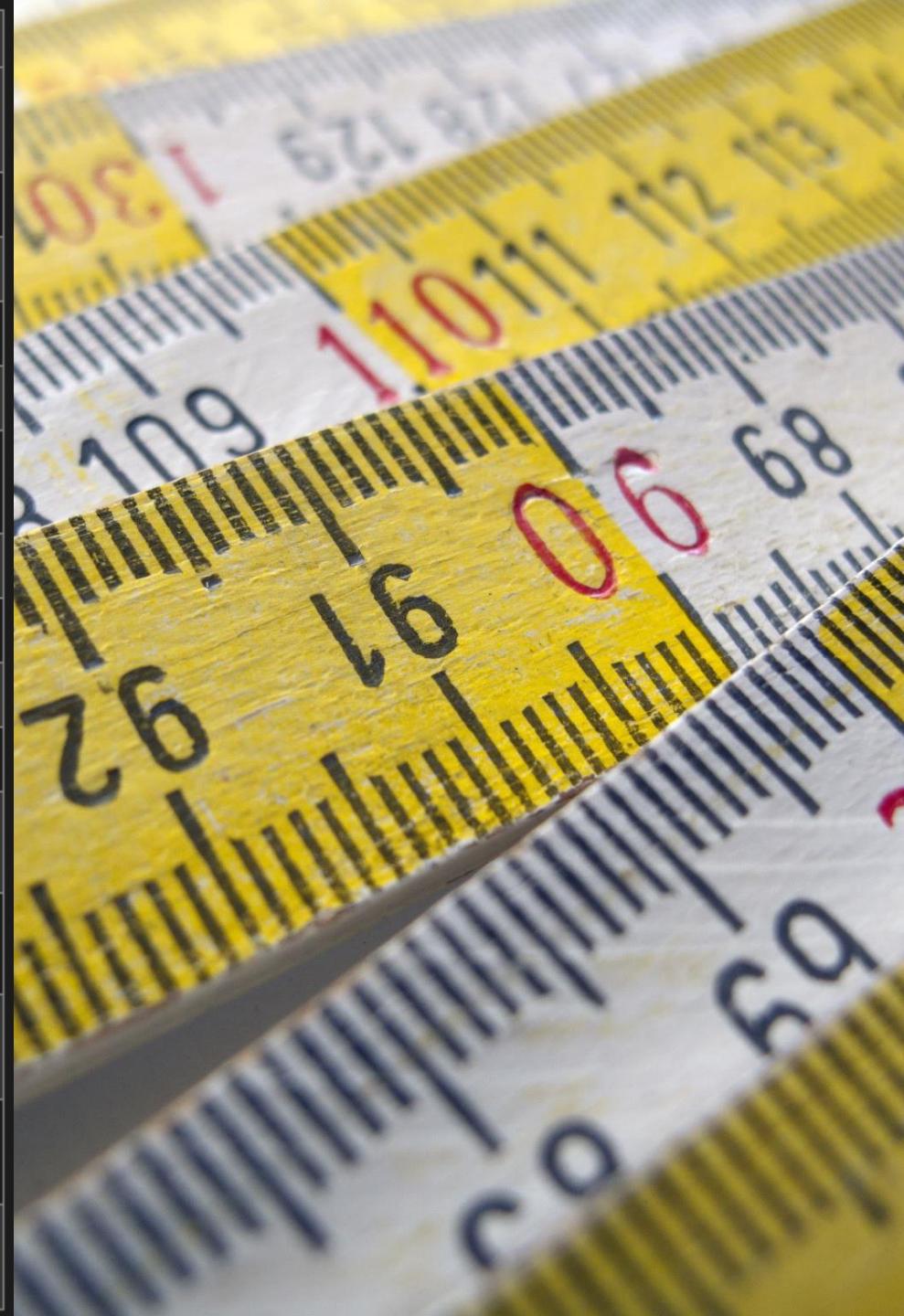
ABSOLUTE MEASUREMENTS

| Unit | Name | Equivalent to |
|------|---------------------|--------------------------|
| cm | Centimeters | 1cm = 37.8px = 25.2/64in |
| mm | Millimeters | 1mm = 1/10th of 1cm |
| Q | Quarter-millimeters | 1Q = 1/40th of 1cm |
| in | Inches | 1in = 2.54cm = 96px |
| pc | Picas | 1pc = 1/6th of 1in |
| pt | Points | 1pt = 1/72nd of 1in |
| px | Pixels | 1px = 1/96th of 1in |



RELATIVE MEASUREMENTS

| Unit | Relative to |
|---------------------------|---|
| <code>em</code> | Font size of the parent, in the case of typographical properties like <code>font-size</code> , and font size of the element itself, in the case of other properties like <code>width</code> . |
| <code>ex</code> | x-height of the element's font. |
| <code>ch</code> | The advance measure (width) of the glyph "0" of the element's font. |
| <code>rem</code> | Font size of the root element. |
| <code>lh</code> | Line height of the element. |
| <code>rlh</code> | Line height of the root element. When used on the <code>font-size</code> or <code>line-height</code> properties of the root element, it refers to the properties' initial value. |
| <code>vw</code> | 1% of the viewport's width. |
| <code>vh</code> | 1% of the viewport's height. |
| <code>vmin</code> | 1% of the viewport's smaller dimension. |
| <code>vmax</code> | 1% of the viewport's larger dimension. |
| <code>vb</code> | 1% of the size of the initial containing block in the direction of the root element's <code>block axis</code> . |
| <code>vi</code> | 1% of the size of the initial containing block in the direction of the root element's <code>inline axis</code> . |
| <code>svw,
svh</code> | 1% of the <code>small viewport</code> 's width and height, respectively. |
| <code>lvw,
lvh</code> | 1% of the <code>large viewport</code> 's width and height, respectively. |
| <code>dvw,
dhv</code> | 1% of the <code>dynamic viewport</code> 's width and height, respectively. |



VALIDATOR . W3 . ORG

The screenshot shows the W3C Markup Validation Service interface. At the top, there's a blue header bar with the W3C logo and the text "Markup Validation Service" and "Check the markup (HTML, XHTML, ...) of Web documents". Below the header, there are three tabs: "Validate by URI" (selected), "Validate by File Upload", and "Validate by Direct Input". Under the "Validate by URI" tab, there's a section titled "Validate by URI" with the sub-instruction "Validate a document online:". It includes a text input field labeled "Address:" and a "More Options" link. At the bottom of this section is a "Check" button.

W3C®

Markup Validation Service

Check the markup (HTML, XHTML, ...) of Web documents

Validate by URI Validate by File Upload Validate by Direct Input

Validate by URI

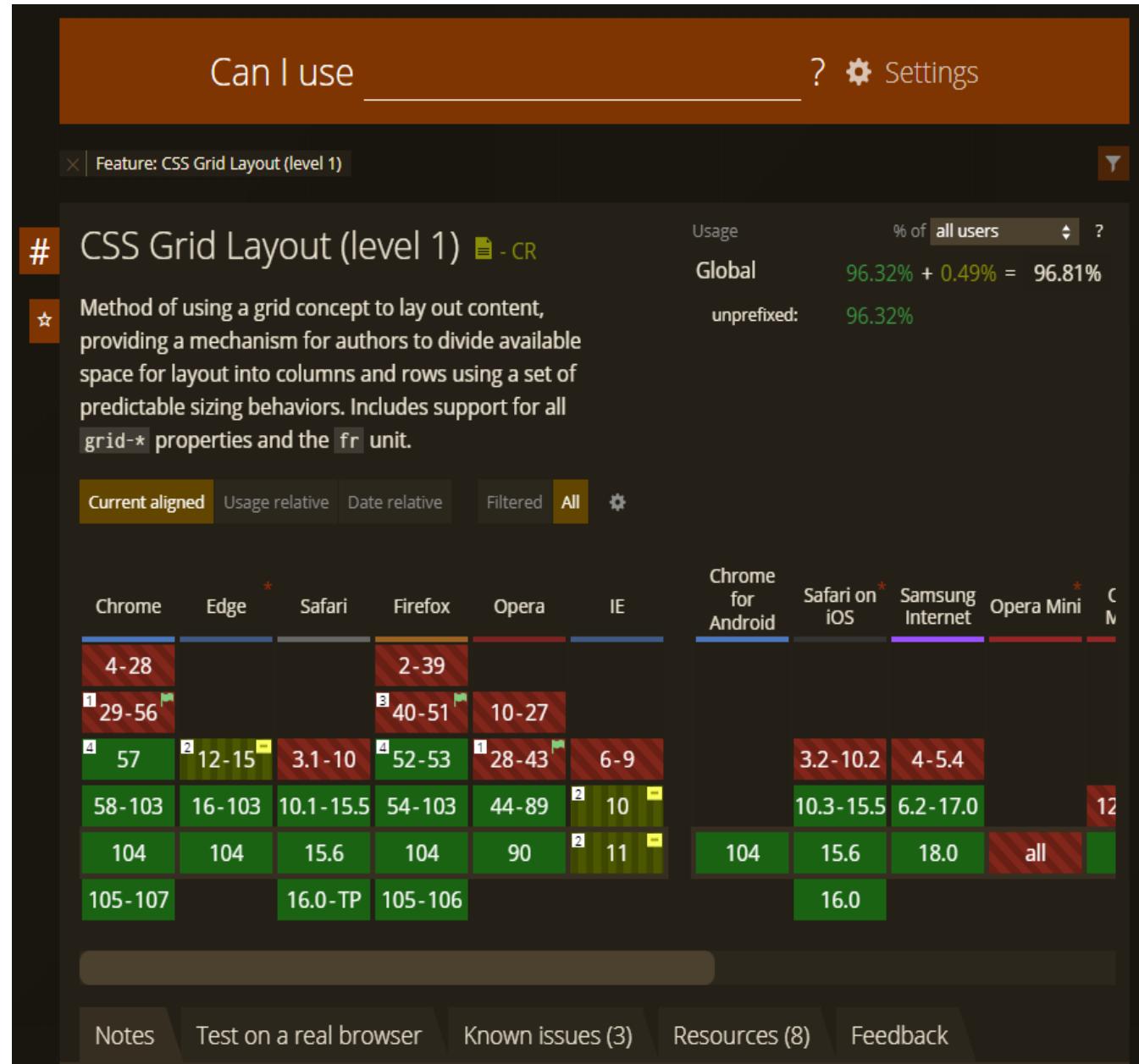
Validate a document online:

Address:

▶ More Options

Check

CANIUSE.COM



UNSUPPORTED CSS?

Your CSS will be ignored if the CSS engine cannot interpret it.



CSS Values and Units

Prof. Andrew Sheehan
Boston University/MET
Computer Science Dept.

2 TYPES OF LENGTH UNITS

- a. Relative
- b. Absolute

COMPUTED VALUES

The **computed** value is the value that is transferred from parent to child during inheritance

The **computed** value of a length will be resolved to an absolute unit

It will be represented by a canonical unit of measurement – the pixel

MEASUREMENTS AND NUMBERS

Numbers are signed (-10)
or unsigned (244) and also
can be decimal (.1)

PERCENTAGES

Percentages are based on the parent width

The parent could be a nested `<div>` or the overall viewport of the device

ABSOLUTE UNITS OF MEASUREMENT

| Unit | Name | Equivalent to |
|-----------|---------------------|---|
| <u>cm</u> | Centimeters | $1\text{cm} = 96\text{px}/2.54$ |
| <u>mm</u> | Millimeters | $1\text{mm} = 1/10\text{th} \text{ of } 1\text{cm}$ |
| <u>Q</u> | Quarter-millimeters | $1\text{Q} = 1/40\text{th} \text{ of } 1\text{cm}$ |
| <u>in</u> | Inches | $1\text{in} = 2.54\text{cm} = 96\text{px}$ |
| <u>pc</u> | Picas | $1\text{pc} = 1/6\text{th} \text{ of } 1\text{in}$ |
| <u>pt</u> | Points | $1\text{pt} = 1/72\text{th} \text{ of } 1\text{in}$ |
| <u>px</u> | Pixels | $1\text{px} = 1/96\text{th} \text{ of } 1\text{in}$ |

RELATIVE UNITS

| unit | relative to: |
|------------|--|
| <u>em</u> | Relative to the font size, i.e. 1.5em will be 50% larger than the base computed font size of its parent. (Historically, the height of the capital letter "M"). |
| <u>ex</u> | Heuristic to determine whether to use the x-height, a letter "x", or `5em` in the current computed font size of the element. |
| <u>cap</u> | Height of the capital letters in the current computed font size of the element. |
| <u>ch</u> | Average <u>character advance</u> of a narrow glyph in the element's font (represented by the "0" glyph). |
| <u>ic</u> | Average <u>character advance</u> of a full width glyph in the element's font, as represented by the "𠮾" (CJK water ideograph, U+6C34) glyph. |
| <u>rem</u> | Font size of the root element (default is 16px). |
| <u>lh</u> | Line height of the element. |
| <u>rlh</u> | Line height of the root element. |

VIEWPORT UNITS

| unit | relative to |
|-------------|---|
| <u>vw</u> | 1% of viewport's width. People use this unit to do cool font tricks, like resizing a header font based on the width of the page so as the user resizes, the font will also resize. |
| <u>vh</u> | 1% of viewport's height. You can use this to arrange items in a UI, if you have a footer toolbar for example. |
| <u>vi</u> | 1% of viewport's size in the root element's <u>inline axis</u> . Axis refers to writing modes. In horizontal writing modes like English, the inline axis is horizontal. In vertical writing modes like some Japanese typefaces, the inline axis runs top to bottom. |
| <u>vb</u> | 1% of viewport's size in the root element's <u>block axis</u> . For the block axis, this would be the directionality of the language. LTR languages like English would have a vertical block axis, since English language readers parse the page from top to bottom. A vertical writing mode has a horizontal block axis. |
| <u>vmin</u> | 1% of the viewport's smaller dimension. |
| <u>vmax</u> | 1% of the viewport's larger dimension. |

RESPONSIVE

Using relative units over absolute measurements/unit allows your content to be more responsive



CSS PIXELS != DEVICE PIXELS

Since the 1980s, the PC market has determined a CSS inch to be equivalent to 96 pixels.

This calculation of pixels was directly tied to the DPI/PPI standard of Microsoft Windows for monitors at the time



THE PIXEL

When we size in mobile, we measure according to CSS pixels, not according to device pixels

- > CSS pixels are logical pixels
- > Device pixels are (real) physical pixels

iPhone 1
(2007)



Device Resolution
320x480

320px screen width
320px CSS width

1x density

iPhone 4
(2010)

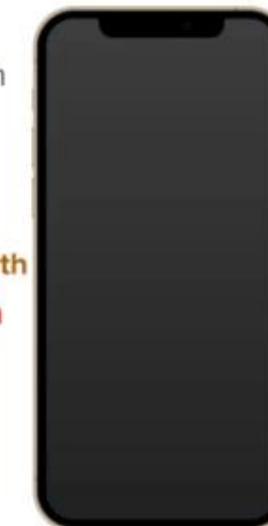


Device Resolution
640x960

640px screen width
320px CSS width

2x density

iPhone 12 Pro
(2020)



Device Resolution
1170x2532

1170px screen width
390px CSS width

3x density

DIMENSIONS

REM

REM stands for ‘root em’

It represents the font size of
the root element: <HTML>

Most browsers use the default: 16px

Units of measurement with the ch designation are not
'character' units.

The zero "0" character - that's the measure of one ch

THE CH UNIT |

EXAMPLE

C

Take a look between
Courier (fixed-width
characters) and
Georgia (variable-width)

Courier

Look, 20 characters.
abcdefghijklmnopqrstuvwxyz
12345678901234567890
iiiiiiiiiiiiiiiiii
mmmmmmmmmmmmmmmmmm

Helvetica

Look, 20 characters.
abcdefghijklmnopqrstuvwxyz
12345678901234567890
iiiiiiiiiiiiiiii
mmmmmmmmmmmmmmmm

Georgia

Look, 20 characters.
abcdefghijklmnopqrstuvwxyz
12345678901234567890
iiiiiiiiiiiiiiii
mmmmmmmmmmmmmmmm

RELATIVE.. EM UNITS

em units for font-size will
be relative to the font-size of
its parent

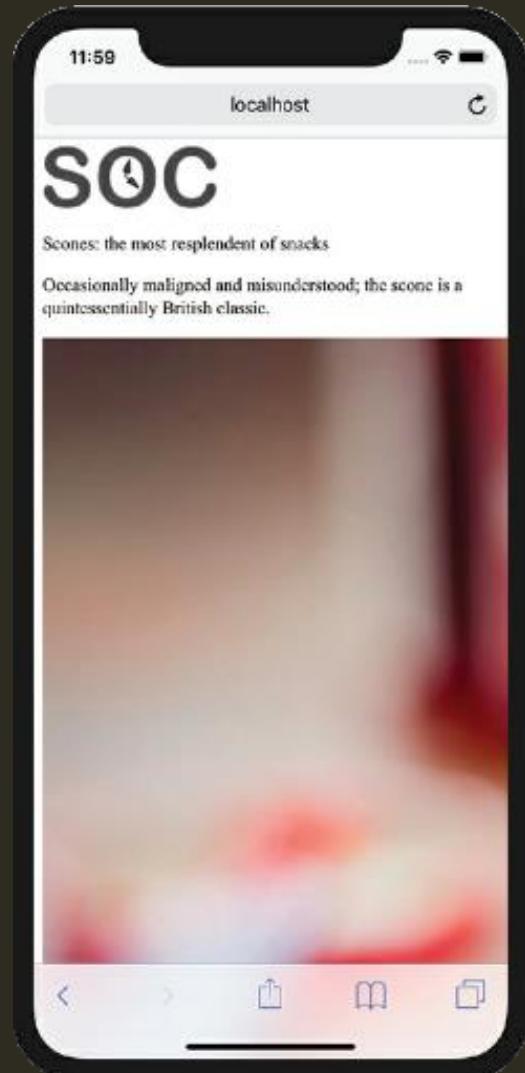
em units on other properties than
font-size will be relative to the
font-size of the current element.
(em can be used anywhere)

EM UNIT PARENT OR ELEMENT

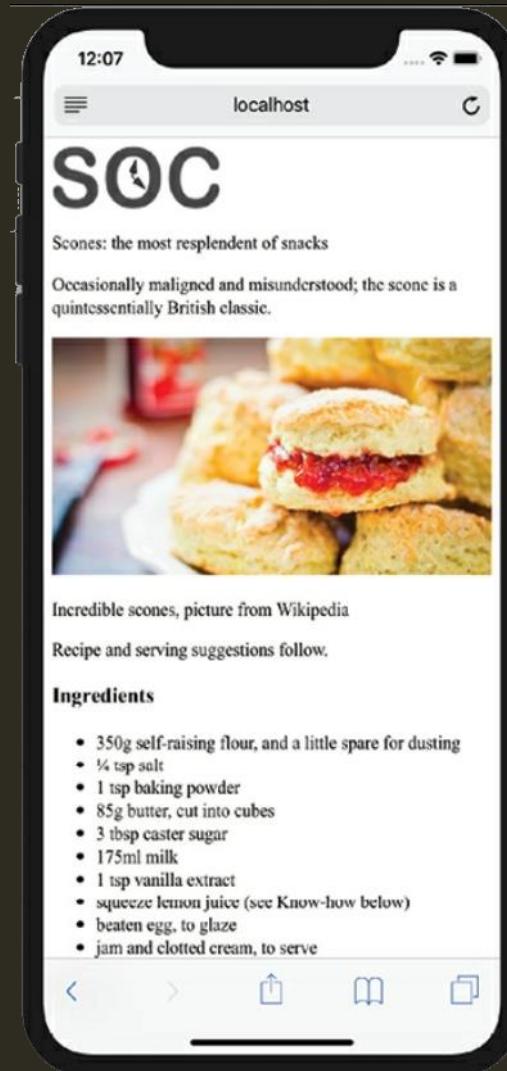
Remember: when em units are used on font-size, the size is relative to the font-size of the parent. When used on other properties, it's relative to the font-size of the element itself.

SIZING IMAGES

Most times, images are not sized where you want them to be.



```
img#soc {  
    max-width: 100%;  
}
```



SIZING IMAGES

max-width stipulates an image should grow to a maximum of 100% of their containing size.

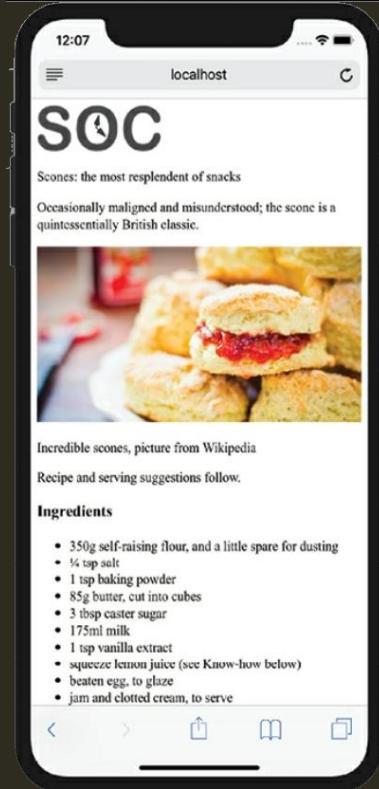
width

Will allow the image be grow larger than its original size. Typically, you do not want that to happen



Remember, test in portrait and landscape mode **on your test phones**.

SIZING IMAGES



Media Queries

media queries adjust the layout depending upon the screen width

Pseudo-elements and classes

Prof. Andrew Sheehan
Boston University/MET
Computer Science Dept.

A CSS pseudo-class is a keyword added to a selector that specifies when your target element is in a special state

It is used for selecting elements

They are not case sensitive. Lowercase use is best practice

```
div.targeted:hover {  
    text-decoration: underline;  
}
```

EXPLAINED |

SYNTAX

Pseudo-class selectors have a colon preceding them

```
button.active:hover {  
    border-radius: 50%;  
}
```

| Selector Style | Name | Does |
|----------------|----------------|---|
| :: | pseudo element | selects/creates some actual content |
| : | pseudo class | selects elements in certain conditions |

LINK RELATED

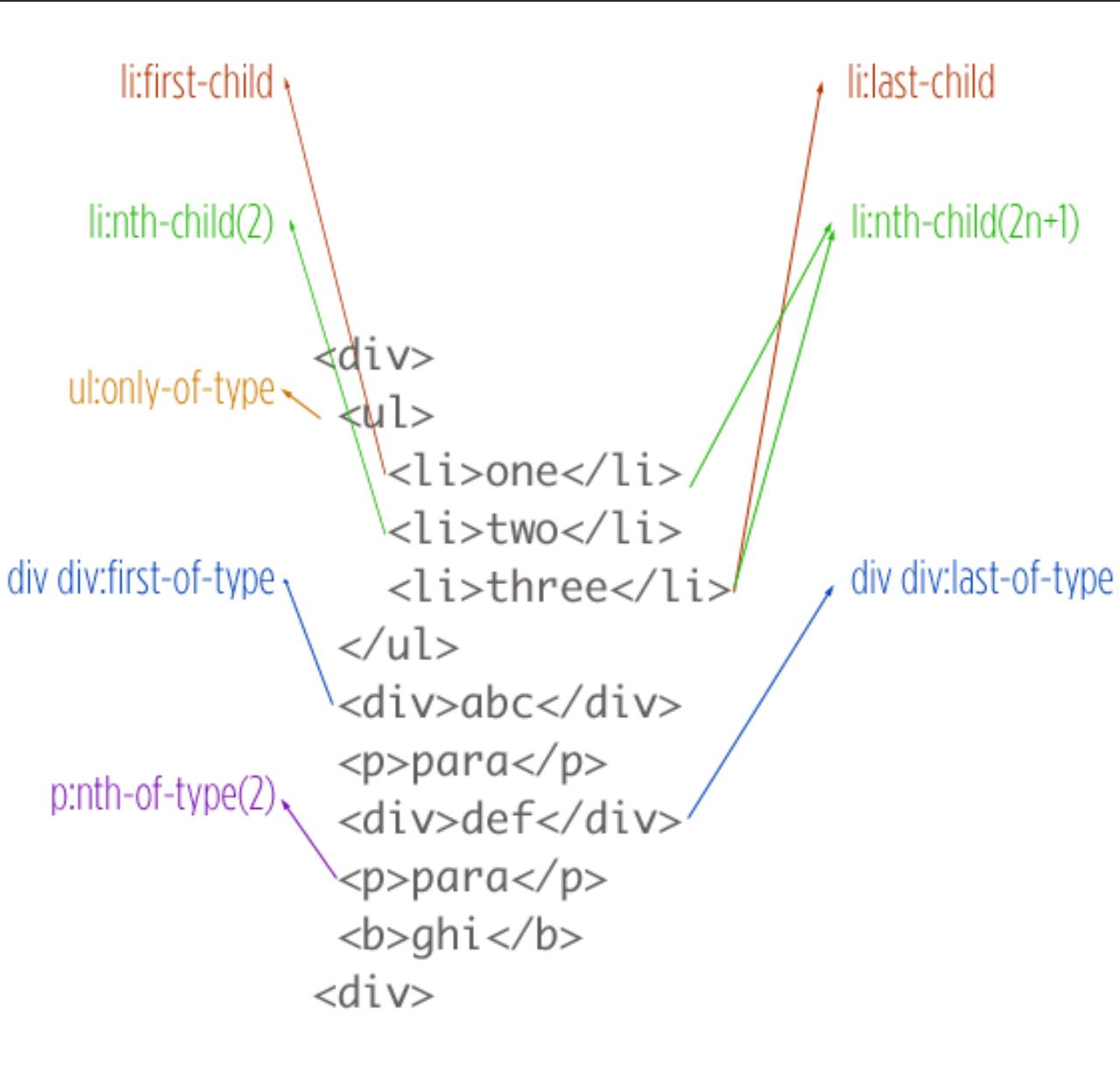
PSEUDO-CLASSES

| Types | Meaning / Use |
|----------|--------------------------------------|
| :link | <a> with href defined |
| :visited | <a> already clicked or being clicked |
| :hover | target has mouse over it |
| :active | when about to be clicked ~ clicking |

PSEUDO-CLASSES & INPUT & LINK

| Types | Meaning / Use |
|----------------|--|
| :focus | Target with current focus |
| :target | Hash in URL matching ID attribute |
| :enabled | Targets that are ready to use (not disabled) |
| :disabled | Opposite of enabled |
| :checked | Checkboxes that are checked |
| :indeterminate | Radios with nothing chosen |
| :required | Inputs with required attribute defined |
| :read-only | Based on a combination of readonly and disabled attributes |

PSEUDO-CLASSES BASED



RELATIONAL

PSEUDO-CLASS

:not()

Removes elements that do not match the selector

```
input:not([disabled]) {  
    font-weight: bolder;  
}
```

:empty()

Has absolutely no children (text or DOM)

TEXT RELATED PSEUDO-ELEMENT

::first-letter || ::first-line

Selects the first letter or line in the text

```
/* The first line of every <p> element. */
p::first-line {
  color: blue;
  text-transform: uppercase;
}
```

It was a dark and stormy night.

IT WAS A DARK AND STORMY NIGHT.

CONTENT-BASED PSEUDO-ELEMENT

::before / ::after

Add inline content before (after) a certain element. It is often used to add cosmetic content **with the content property**

```
<q>Some quotes</q>, he said, <q>are better than none.</q>
```

```
q::before {  
    content: ““;  
}  
q::after {  
    content: ““;  
}
```

PSEUDO-ELEMENTS

A pseudo-element is like adding an extra element without having to add more HTML.

```
article::first-letter {  
    color: rgb(22,122,3);  
    font-size: calc(2em * 2);  
}  
  
/* The calc() is a CSS function  
 * that lets you perform  
 * calculations on several property  
 * values */
```

PSEUDO-ELEMENT

::marker

The ::marker pseudo-element lets you style a <ul,ol> or the arrow of a <summary> element.

PSEUDO-ELEMENT ::MARKER

```
ul ::marker {  
    font-size: 1.5em;  
}  
  
ol ::marker {  
    font-size: 1.1em;  
}  
  
summary::marker {  
    content: '\002B' ' '/* Plus symbol with space */;  
}  
  
details[open] summary::marker {  
    content: '\2212' ' '/* Minus symbol with space */;  
}
```

PSEUDO-ELEMENT

The ::selection allows you to style selected text on your page

```
::selection {  
    background: green;  
    color: white;  
}
```

THERE ARE MORE...

We are not going to cover all of the pseudo-elements and pseudo-classes. Please check the web for what you need.

Look through them to find what you need.



Understanding display, visibility and float

Prof. Andrew Sheehan

Boston University/MET
Computer Science Dept.

PAGE FLOW

Content in
your page
start at the
top-left,
goes right,
flows to the
bottom

The float property places an element to the **left or right** of its container.

Always create a container for your floating elements

EXPLAINED |

POSITIONING

ABSOLUTE

Absolutely positioned content will be removed from the normal flow of a page

```
44 | .info {  
45 |   position: absolute;  
46 |   top: 10px;  
47 |   right: 20px  
48 | }
```

ABSOLUTE

Absolutely positioned elements will not affect the position of other elements

Other elements will not affect them - whether they touch each other or not

FLOAT

EXAMPLE

```
float: none;
```

```
float: left;
```

```
float: right;
```

```
float: inline-start;
```

```
float: inline-end;
```

Float
me

As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

```
float: none;
```

```
float: left;
```

```
float: right;
```

```
float: inline-start;
```

```
float: inline-end;
```

Float
me

As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

```
float: none;
```

```
float: left;
```

```
float: right;
```

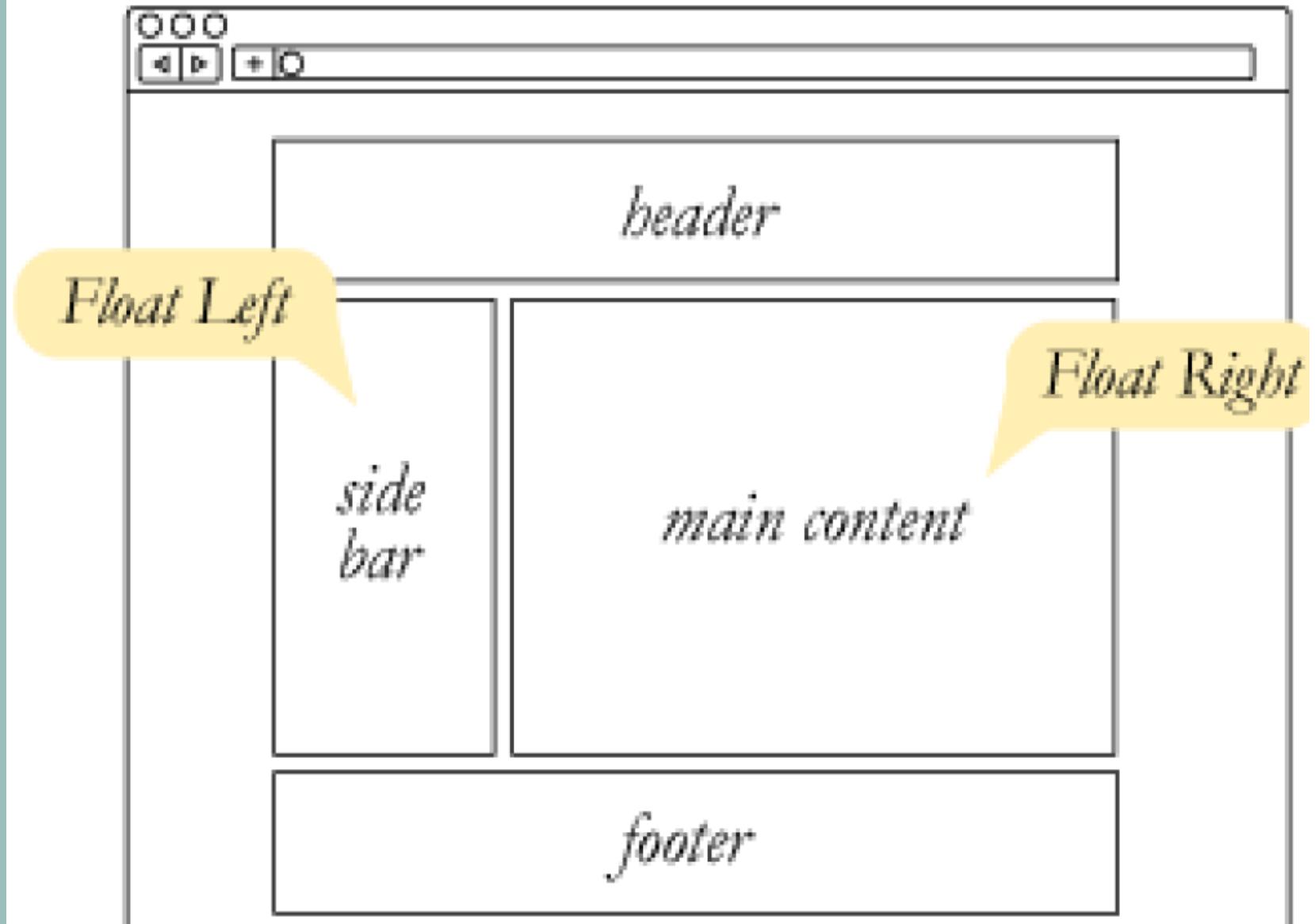
```
float: inline-start;
```

```
float: inline-end;
```

Float
me

As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

IDEA/ EXAMPLE



FLOAT VALUES

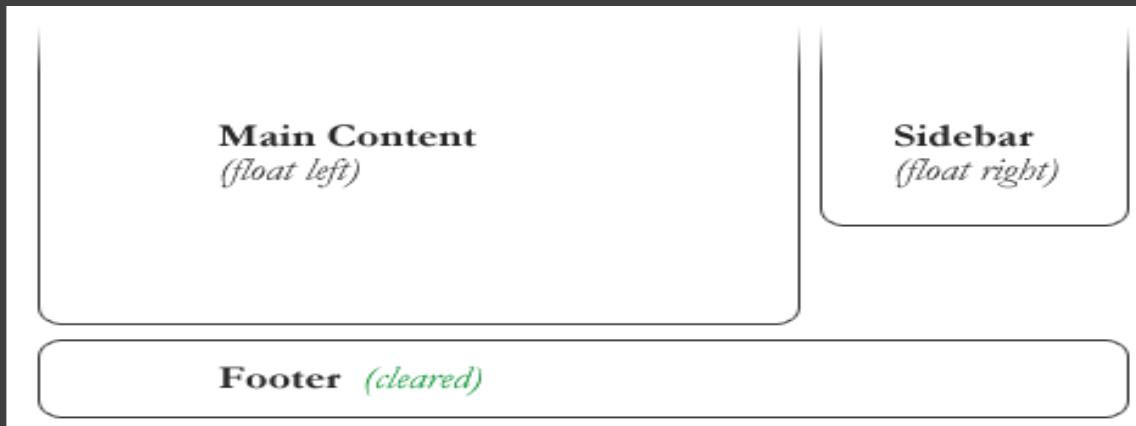
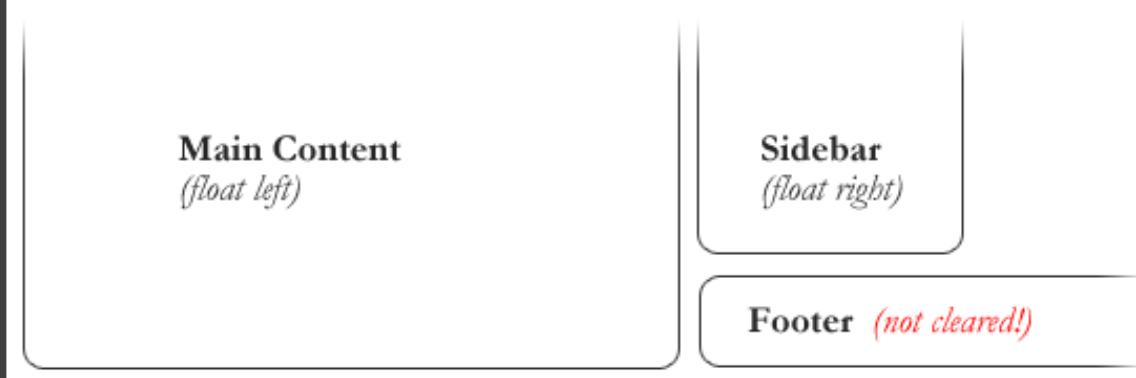
| Specified value | Computed value |
|--------------------|----------------|
| inline | block |
| inline-block | block |
| inline-table | table |
| table-row | block |
| table-row-group | block |
| table-column | block |
| table-column-group | block |
| table-cell | block |
| table-caption | block |
| table-header-group | block |
| table-footer-group | block |
| inline-flex | flex |
| inline-grid | grid |
| other | unchanged |

FLOAT IMPLICATIONS

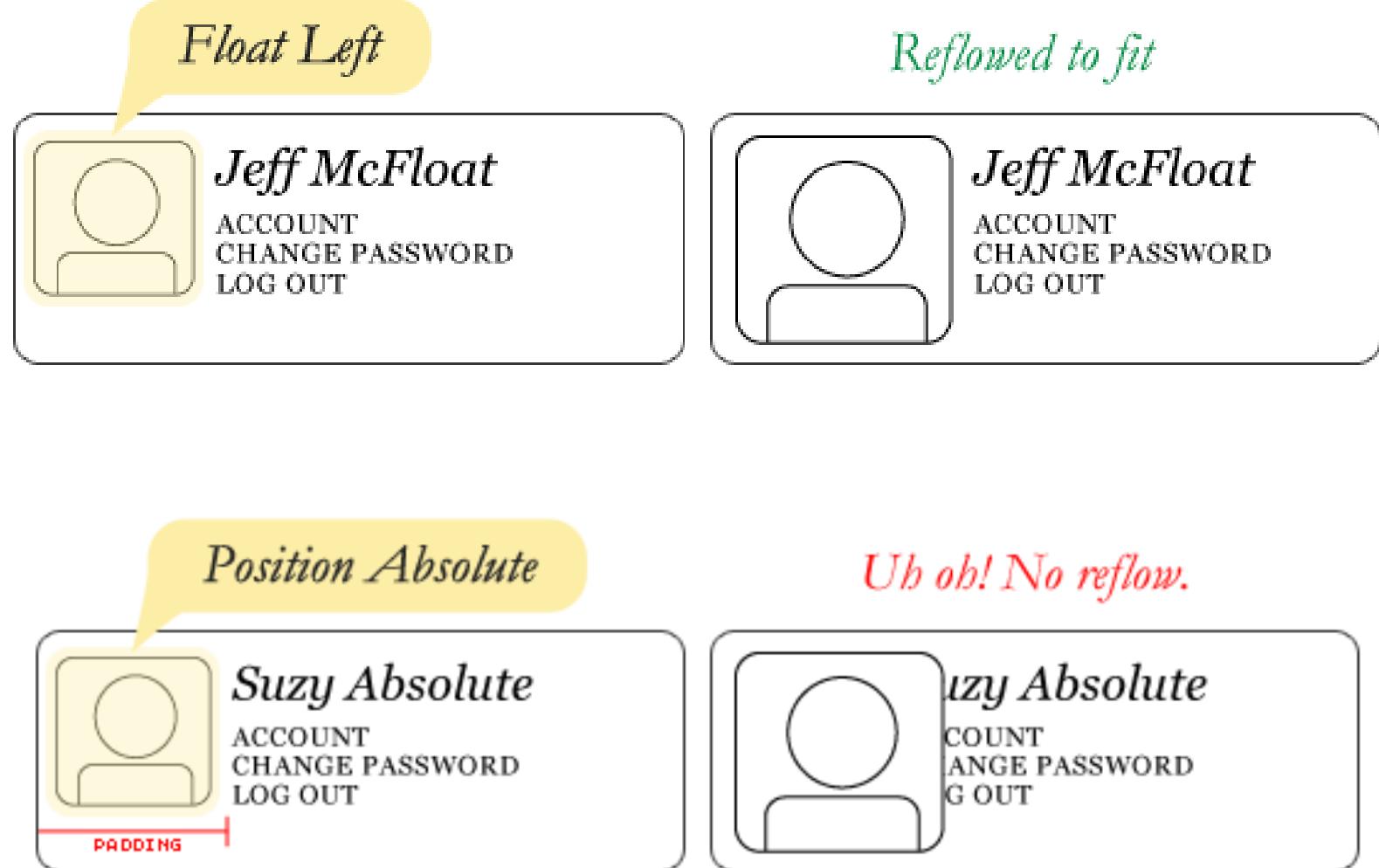
Absolutely positioned content assumes you are using block-level containment.

CLEARING THE FLOAT

`clear: [both | right | left]`



ABSOLUTE FAIL



INLINE-START

float: inline-start

The element must float on the start side
(left) of its containing block

float: inline-end

The element must float on the end side
(right) of its containing block

Show or hide your element

The DOM remains - as is

Nothing moves around

VISIBILITY

POSSIBLE VALUES

```
visibility: visible;  
visibility: hidden;  
visibility: collapse;  
visibility: inherit;  
visibility: initial;  
visibility: revert;  
visibility: revert-layer;  
visibility: unset;
```

Assistive reading
software/devices **will not work**
with content that is marked
with visibility: hidden

Bootstrap

Prof. Andrew Sheehan

Boston University/MET
Computer Science Dept.

HTML5 & VIEWPORT

The HTML5 doctype
should be used.

Use the viewport

SUPPORTED BROWSERS

Bootstrap supports the latest, stable releases of all major browsers and platforms

STANDARD BASELINE TEMPLATE

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link href="bootstrap.min.css" rel="stylesheet">
  </head>
  <body>
    <h1>Hello, world!</h1>
    <script src="bootstrap.bundle.min.js" ></script>
  </body>
</html>
```

While the Bootstrap CSS can be used with any framework, **the Bootstrap JavaScript is not fully compatible with JavaScript frameworks like React, Vue, and Angular** which assume full knowledge of the DOM. Both Bootstrap and the framework may attempt to mutate the same DOM element, resulting in bugs like dropdowns that are stuck in the "open" position.

A better alternative for those using this type of frameworks is to use a framework-specific package **instead of** the Bootstrap JavaScript. Here are some of the most popular options:

- React: [React Bootstrap](#)
- Vue: [BootstrapVue](#) (currently only supports Vue 2 and Bootstrap 4)
- Angular: [ng-bootstrap](#)

CONTAINERS

Containers are the most basic element in bootstrap

You must use them



OVERVIEW OF CONTAINERS

	Extra small <code><576px</code>	Small <code>≥576px</code>	Medium <code>≥768px</code>	Large <code>≥992px</code>	X-Large <code>≥1200px</code>	XX-Large <code>≥1400px</code>
<code>.container</code>	100%	540px	720px	960px	1140px	1320px
<code>.container-sm</code>	100%	540px	720px	960px	1140px	1320px
<code>.container-md</code>	100%	100%	720px	960px	1140px	1320px
<code>.container-lg</code>	100%	100%	100%	960px	1140px	1320px
<code>.container-xl</code>	100%	100%	100%	100%	1140px	1320px
<code>.container-xxl</code>	100%	100%	100%	100%	100%	1320px
<code>.container-fluid</code>	100%	100%	100%	100%	100%	100%

THE

CONTAINER

At every breakpoint,
bootstrap will adjust
the width of your
container element

```
<div id='container'>  
    <!-- your elements/content -->  
</div>
```

FLUID CONTAINER

The container (and your content)
will span the entire viewport

100% width

```
<div id='container-fluid'>  
    <!-- your elements/content -->  
</div>
```

Bootstrap's grid system uses a series of containers, rows, and columns to layout and align content

Built with Flexbox

GRIDS |

3 COLUMN LAYOUT

```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6     <title>Bootstrap Grid</title>
7     <link href="bootstrap.min.css">
8     <style>
9       .col {
10         background-color: blue;
11         color: white;
12         border: 1px dotted white;
13         height: 12vh;;
14       }
15     </style>
16   </head>
17   <body>
18     <div class="container text-center">
19       <div class="row">
20         <div class="col">
21           I like
22         </div>
23         <div class="col">
24           Met Cs601
25         </div>
26         <div class="col">
27           A LOT
28         </div>
29       </div>
30     </div>
31     <script src="bootstrap.bundle.min.js"></script>
32   </body>
33 </html>
```

I like	Met cs601	A LOT
--------	-----------	-------

T
H
E
T
H

There are 12 columns per row

You do not need to use all 12 columns.
Each column uses percentages

COLUMN EXAMPLE

```
<div class="container">
  <div class="row">
    <div class="col">
      1 of 2
    </div>
    <div class="col">
      2 of 2
    </div>
  </div>
  <div class="row">
    <div class="col">
      1 of 3
    </div>
    <div class="col">
      2 of 3
    </div>
    <div class="col">
      3 of 3
    </div>
  </div>
</div>
```

The diagram illustrates the layout defined by the code. It shows a container divided into two horizontal rows. The first row contains two columns, each labeled 'col' below it. The second row contains three columns, each labeled 'col' below it. The total width of the columns in each row adds up to 12 units, which is the standard width for a single row in a 12-column grid system.

ALIGNMENT IN ROWS

One of three columns One of three columns One of three columns

One of three columns One of three columns One of three columns

One of three columns One of three columns One of three columns

ALIGNMENT IN ROWS (CODE)

```
<div class="container">
  <div class="row align-items-start">
    <div class="col">
      One of three columns
    </div>
    <div class="col">
      One of three columns
    </div>
    <div class="col">
      One of three columns
    </div>
  </div>
  <div class="row align-items-center">
    <div class="col">
      One of three columns
    </div>
    <div class="col">
      One of three columns
    </div>
    <div class="col">
      One of three columns
    </div>
  </div>
  <div class="row align-items-end">
    <div class="col">
      One of three columns
    </div>
    <div class="col">
      One of three columns
    </div>
    <div class="col">
      One of three columns
    </div>
  </div>
</div>
```

PRACTICAL EXAMPLE

```
<body>
  <div class="clearfix">
    <img class="col-md-6 float-md-end mb-3 ms-md-3">
    <p>A paragraph of placeholder text. We're using it here to show the use of the clearfix class. We're adding quite a few meaningless phrases here to demonstrate how the columns interact here with the floated image.</p>
    <p>As you can see the paragraphs gracefully wrap around the floated image. Now imagine how this would look with some actual content in here, rather than just this boring placeholder text that goes on and on, but actually conveys no tangible information at. It simply takes up space and should not really be read.</p>

    <p>And yet, here you are, still persevering in reading this placeholder text, hoping for some more insights, or some hidden easter egg of content. A joke, perhaps.</p>
  </div>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/js/bootstrap.bundle.min.js">
</body>
```

A paragraph of placeholder text. We're using it here to show the use of the clearfix class. We're adding quite a few meaningless phrases here to demonstrate how the columns interact here with the floated image.

As you can see the paragraphs gracefully wrap around the floated image. Now imagine how this would look with some actual content in here, rather than just this boring placeholder text that goes on and on, but actually conveys no tangible information at. It simply takes up space and should not really be read.

And yet, here you are, still persevering in reading this placeholder text, hoping for some more insights, or some hidden easter egg of content. A joke, perhaps. Unfortunately, there's none of that here.



Responsive floated image

RESPONSIVE IMG

```
<img src='bu-logo.png' class='img-fluid' />
```

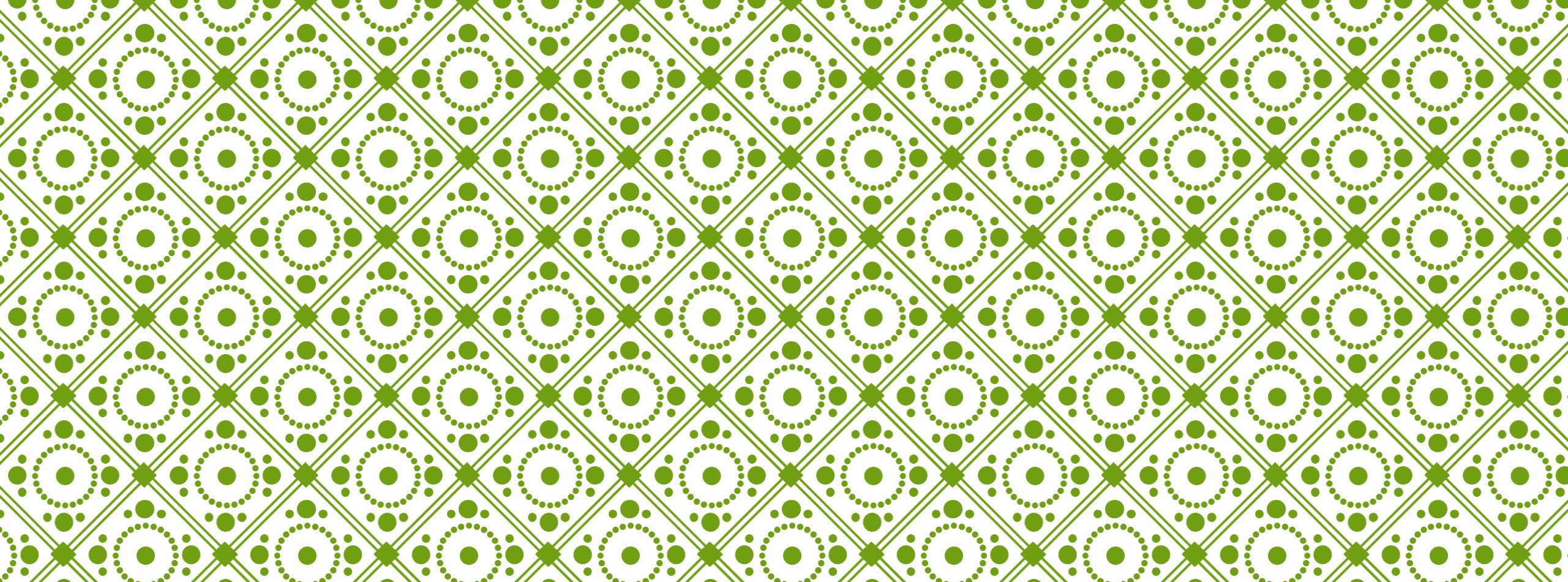


UTILITY CLASSES

Utilities

- API
- Background
- Borders
- Colors
- Display
- Flex
- Float
- Interactions
- Overflow
- Position
- Shadows
- Sizing
- Spacing
- Text
- Vertical align
- Visibility

Online.



SCOPE, HOISTING & CLOSURES EXPLAINED

Prof. Andrew Sheehan

Boston University/MET
Computer Science Dept.

WHAT IS A CLOSURE

A closure is the combination of a function bundled together (enclosed) with references to its surrounding state

A closure gives you access to an outer function's scope from its inner function(s)

Closures are created every time a function is created

EXAMPLE

```
function init() {  
    var name = 'Mozilla';  
  
    function displayName() // inner function (closure)  
    {  
        console.log(name);  
    }  
    displayName();  
}  
  
init();
```

LEXICAL SCOPE

Lexical scoping uses the location where a variable is declared

How a parser resolves variable names when functions are nested.

Scope means where something is accessible from its declaration point

var

Can be either function scope or global.

let, const

Not function scope, but block scope.

SCOPE

CLOSURE CHAIN

- ↓
Local scope (within function body)
- ↓
Enclosing scope (block, function or module)
- ↓
Global scope



PERFORMANCE CONSIDERATIONS

Do not create functions within other functions if closures are not needed.

It will negatively affect script performance both in terms of processing speed and memory consumption.

HOSTING

Refers to the process whereby the interpreter appears to move the declaration of functions, variables or classes to the top of their scope, prior to execution of the code.

All undeclared variables will be placed into global scope

```
function hoist() {  
    a = 20;  
    var b = 100;  
}  
  
hoist();  
  
console.log(a);  
/*  
Accessible as a global variable outside hoist() function  
Output: 20  
*/  
  
console.log(b);  
/*  
Since it was declared, it is confined to the hoist() function scope.  
We can't print it out outside the confines of the hoist() function.  
Output: ReferenceError: b is not defined  
*/
```

UNDECLARED
VARIABLES

FUNCTION HOISTING

Hoisting allows functions to be safely used in code before they are declared

```
sayHi("Andrew");

function sayHi(username = "Guest") {
    console.log(`Hi, ${username}`);
}

// The result of the code above is: "Hi, Andrew"
```



‘STRICT MODE’

By enabling strict mode, we opt into a restricted variant of JavaScript that will not tolerate the usage of variables before they are declared

```
'use strict';

console.log(hoist); // Output: ReferenceError: hoist is not defined
hoist = 'Hoisted';
```

VAR HOTSTING

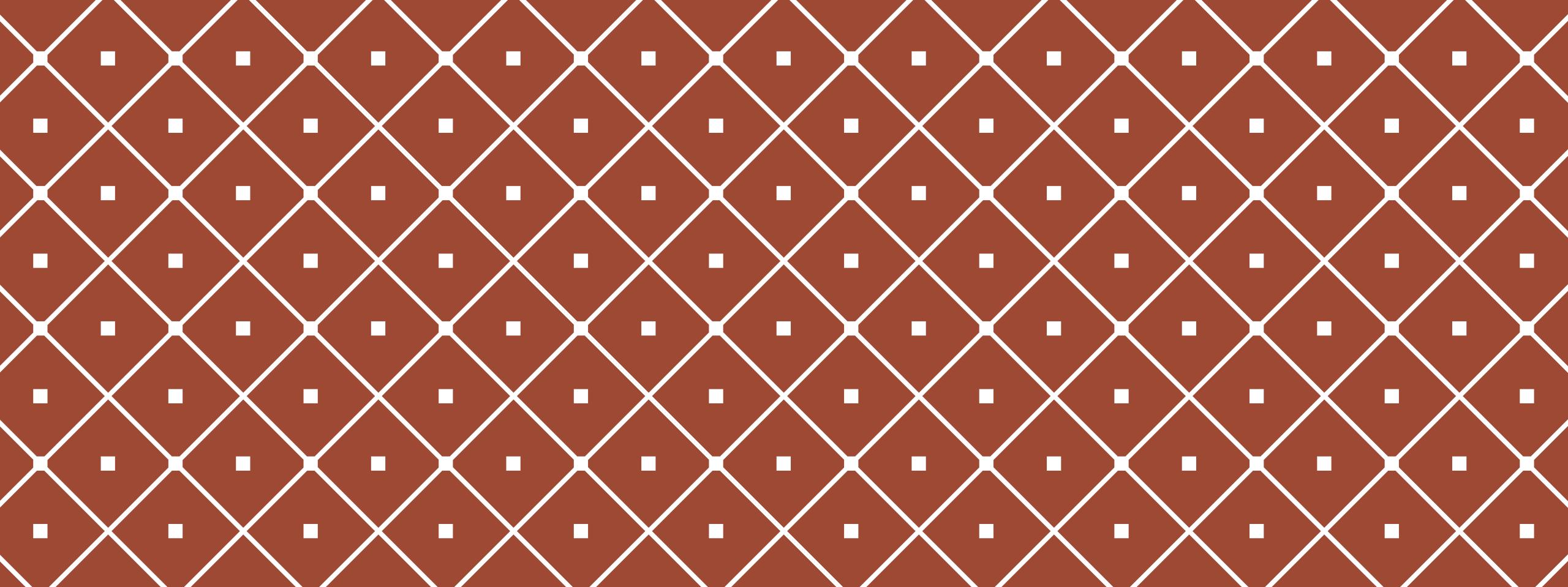
Hoisting allows functions to be safely used in code before they are declared

```
console.log(randomName); // undefined  
  
var randomName; // undefined  
  
randomName = “Frank”;  
  
Console.log(randonName); // Frank;
```

|

ES6 CLASSES AND HOTSWAPPING

Class
expressions
are not
hoisted



CSS MEDIA QUERIES

Andrew Sheehan

Boston University
Metropolitan College

WHAT ARE MEDIA QUERIES



A **media query** is method to change the way your webapp looks on different devices.

MOBILE FIRST > THEN OTHERS



CONTENT CAN ADAPT TO THE DEVICE

-  Based on the device width & height.
-  The orientation of the viewport
-  Resolution (pixel density)

BASIC PRINCIPALS

Media type:
(all, print, screen and speech)

Matching expression:
to trigger the media change

MEDIA QUERIES

DEFAULT

When no
media type
indicated



“all” will be used
(all media types)

MEDIA EXPRESSIONS MUST EVALUATE TO TRUE



- i. The media type of the media query matches the media type of the device.
- ii. All expressions in the media query eval to true.
- iii. If the media query list is empty (i.e. the declaration is the empty string or consists solely of whitespace) it evaluates to true.

MEDIA QUERIES

EXAMPLE

```
@media (max-width: 2082px) {  
    #main_content_area {  
        display: block;  
    }  
}
```

Display that target a block-level element
when the viewport width is equal to or less
than 2082 pixels

MEDIA TYPES: PRINT

When a user wants to print, they ~probably~ do not want to waste their color ink on graphics or unnecessary content.

```
@media print {  
    #header, #left-side-ads, #menu {  
        display: none;  
    }  
}
```



```
<link rel="stylesheet" type="text/css"  
      media="screen" href="sans-serif.css">
```

```
<link rel="stylesheet" type="text/css"  
      media="print" href="serif.css">
```

EXAMPLE |
USING THE LINK ELEMENT |

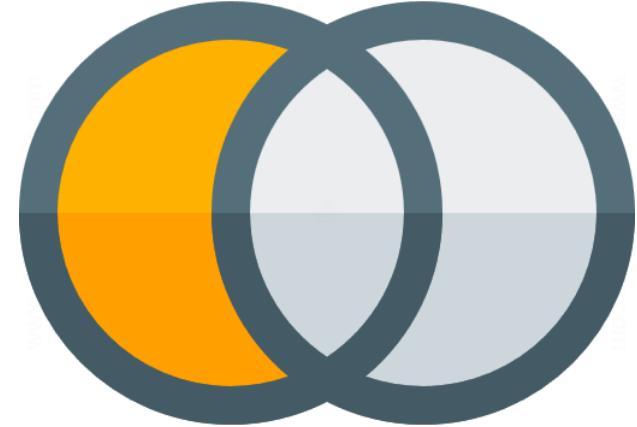
EXAMPLE COLOR SCREENS

```
<link rel="stylesheet" href="styles.css"  
      media="screen and (color)" />
```

This style sheet applies to devices of a certain media type 'screen', with a certain feature (it must be a color screen).

Here the same media query written in an @import-rule

```
@import url(color.css) screen and (color);
```



LOGICAL NOT (NEGATION)

www.iconexperience.com

It will negate (true/false) the entire expression you are testing.

```
<link rel="stylesheet" href="example.css"  
      media="not screen and (color)" />
```

TAKE NOTICE

The not keyword can't be used
to negate an individual feature
**query, only an entire media
query**



STANDARD AND EXPERIMENTAL ELEMENTS

Misspalling your media type or
having an invalid format or logic
will result in **false**

(the media query will be be
applied or used)



COMPLEX EXPRESSIONS

```
@media (min-width: 30em) and (orientation: landscape) { ... }
```

Viewport set to landscape mode with a width of at least 30em's

```
@media (min-height: 680px), screen and (orientation: portrait) { ... }
```

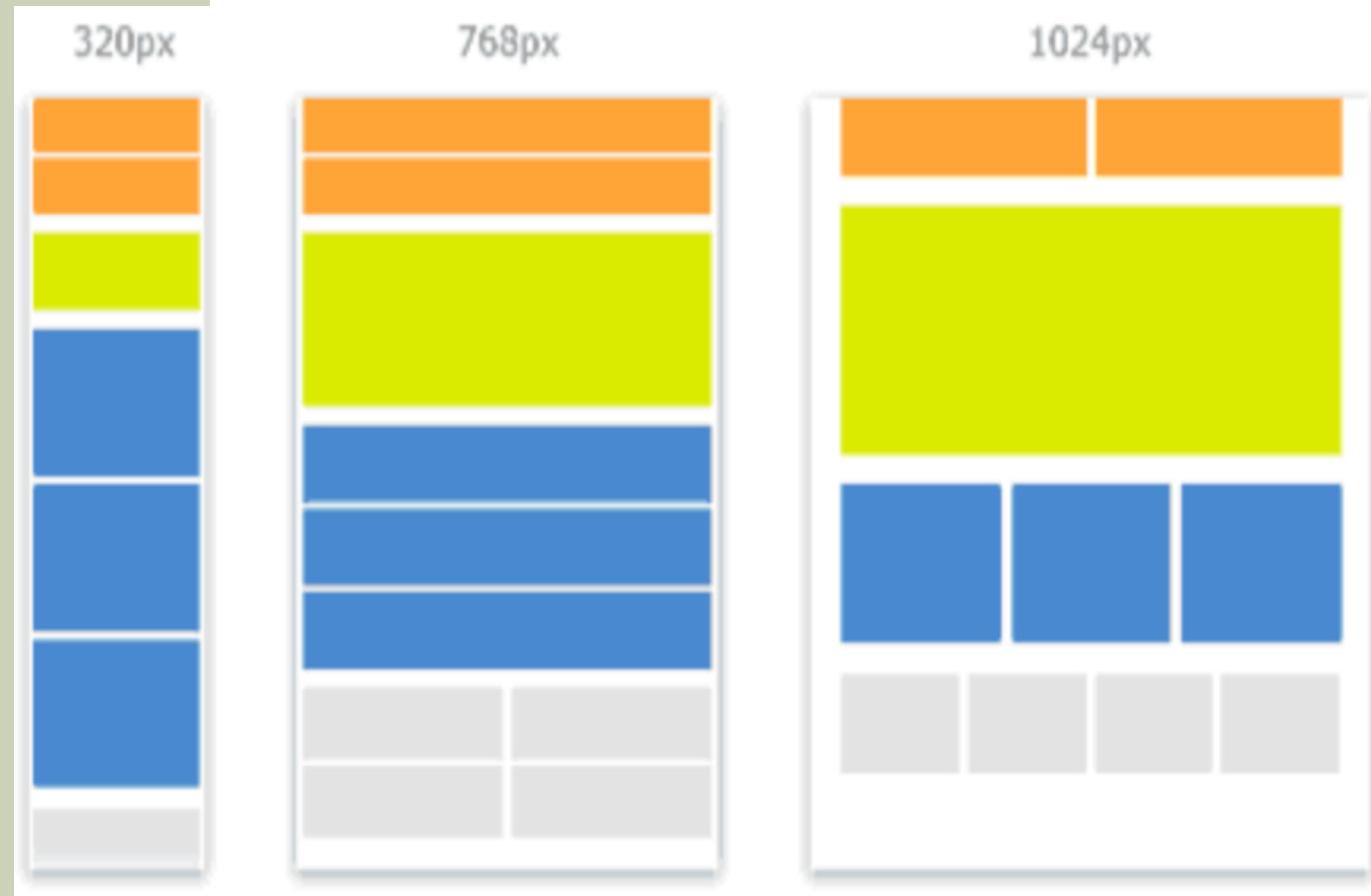
Viewport with at least a height of 680 pixels or using a screen display in portrait mode

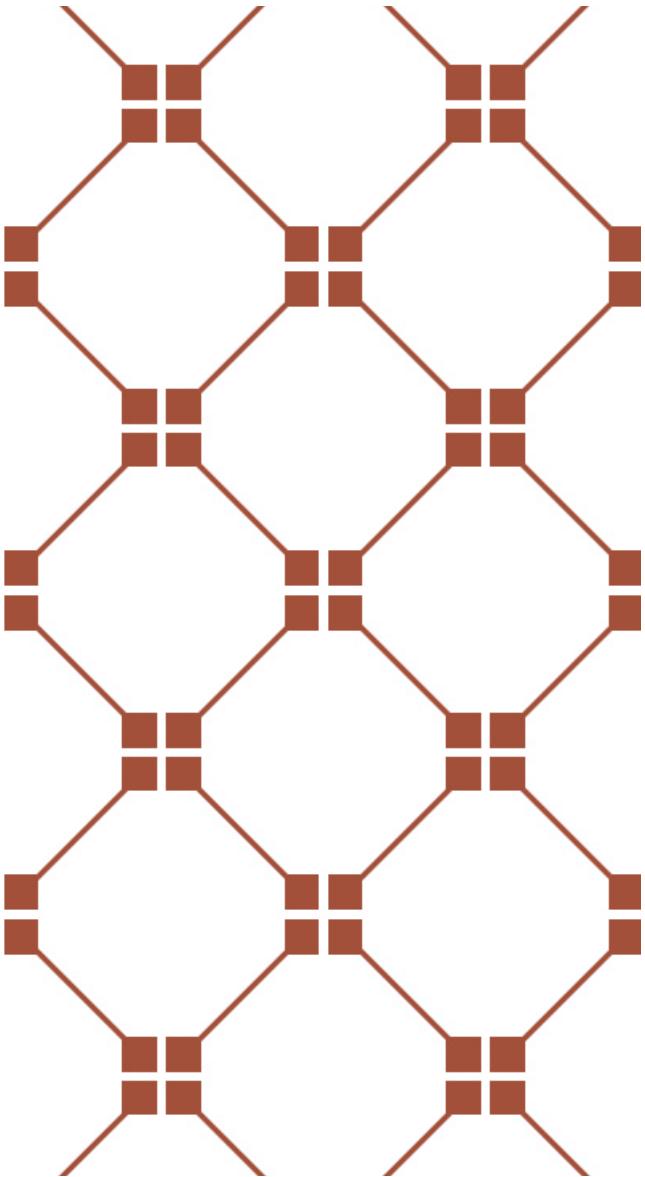
```
@media (30em <= width <= 50em ) { ... }
```

Viewport min-width and max-width between 30em & 50em
(Example using new, Level 4 media query syntax)

GENERAL PRINCIPAL

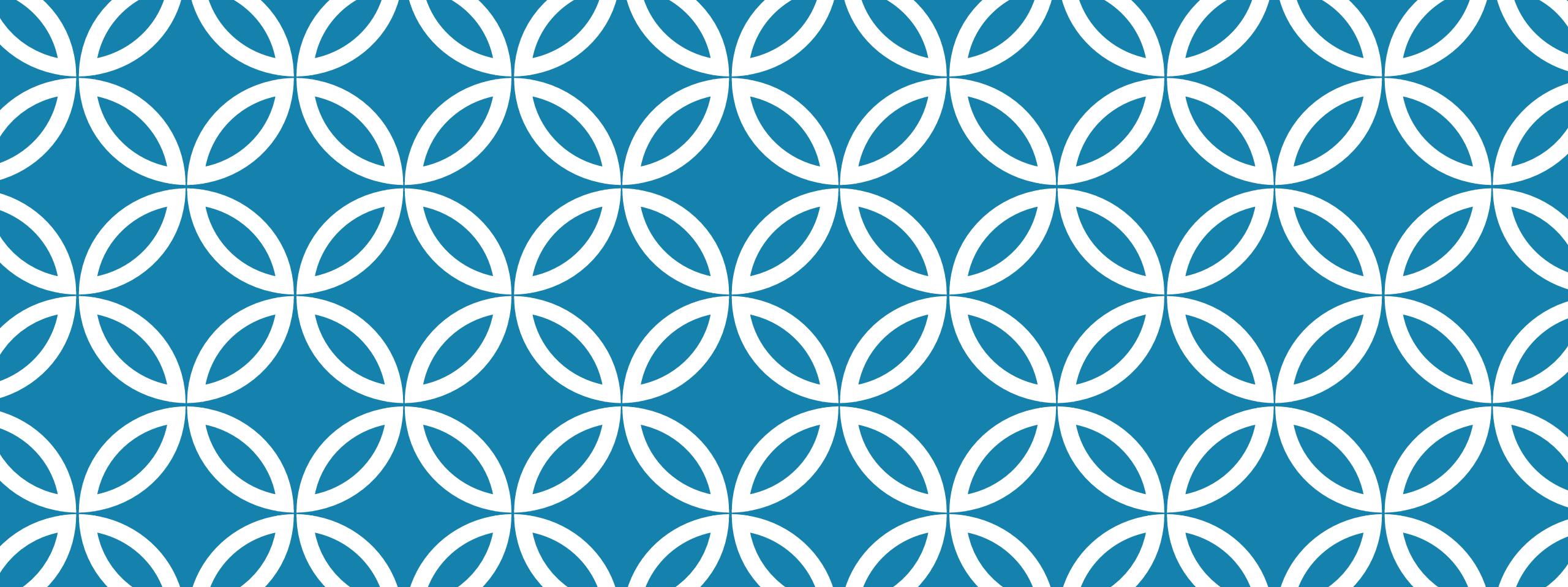
Shift content (or remove) to be the width of your client's device





```
@media screen and (max-width: 520px) {  
    /* CSS rules go inside brackets */  
}  
  
@media screen and (max-width: 980px) {  
    /* CSS rules go inside brackets */  
}  
  
@media screen and (max-width: 1440px) {  
    /* CSS rules go inside brackets */  
}
```

STANDARD USE OR EXAMPLE



ARRAYS

Andrew Sheehan

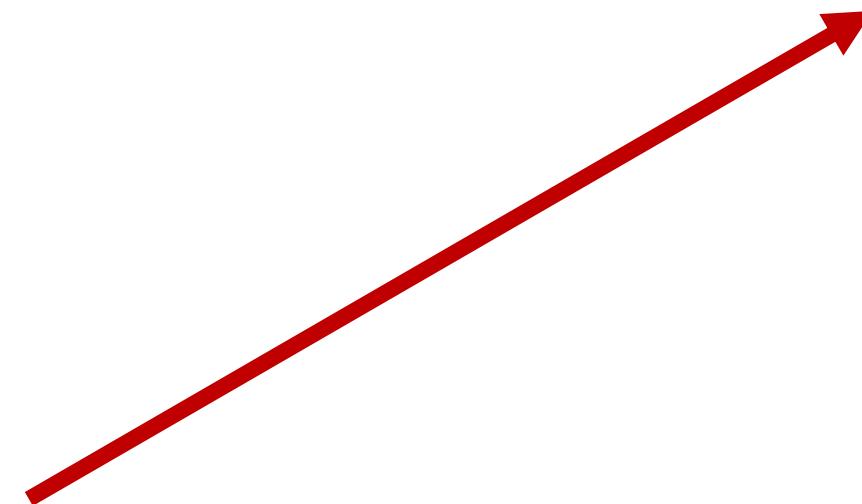
Boston University
Metropolitan College

DATA STRUCTURES

Objects that allow us to group or store ‘things’ (other data types)

SYNTAX

```
const lastNames = [ ];
```



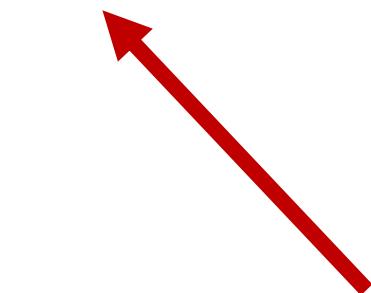
Brackets: indicates an array. This is important to note.

ARRAY EXAMPLE

Variable name



```
const lastNames = ['Sheehan', 'Ali', 'Torreto'];
```



Collection of string values

ITERATION FOREACH

```
const lastNames = [ 'Sheehan', 'Smith' ];  
  
lastNames.forEach( (item, index) => {  
    /* your logic goes in here... */  
});
```

Output: Sheehan, 0
 Smith, 1

FOREACH() ANOTHER WAY USING IT

```
1 | var a = ['a', 'b', 'c'];
2 |
3 | a.forEach(function(element) {
4 |   console.log(element);
5 |});
```

ARRAY.MAP()

Creates a new array, based on what you need to do with the original array

```
1 var numbers = [1, 5, 10, 15];
2 var doubles = numbers.map(function(x) {
3     return x * 2;
4 });
5 // doubles is now [2, 10, 20, 30]
6 // numbers is still [1, 5, 10, 15]
7
8 var numbers = [1, 4, 9];
9 var roots = numbers.map(Math.sqrt);
10 // roots is now [1, 2, 3]
11 // numbers is still [1, 4, 9]
```

MAP

ANOTHER EXAMPLE

```
1 | var numbers = [1, 4, 9];
2 | var roots = numbers.map(Math.sqrt);
3 | // roots is now [1, 2, 3]
4 | // numbers is still [1, 4, 9]
```

MAP USING ARROW FUNCTIONS

```
8      let numbers = [1,2,4,8,16];
9      let doubles = numbers.map((value, index, arr) => {
10         return index === 0 ? -1 : value * 2;
11     });
12     doubles.forEach((value, index) => {
13         if ( index === 0 ) {
14             console.info("zeroth doesn't matter.");
15         } else {
16             console.info("doubled is: " + value);
17         }
18     });

```

REDUCE()

A function that reduces to a single value.

```
1 const array1 = [1, 2, 3, 4];
2 const reducer = (accumulator, currentValue) => accumulator + currentValue
3
4 // 1 + 2 + 3 + 4
5 console.log(array1.reduce(reducer));
6 // expected output: 10
7
8 // 5 + 1 + 2 + 3 + 4
9 console.log(array1.reduce(reducer, 5));
10 // expected output: 15
```

REDUCE EXAMPLE

```
var numbers = [65, 44, 12, 4];

function getSum(total, num) {
    return total + num;
}

function myFunction(item) {
    document.getElementById("demo").innerHTML = numbers.reduce(getSum);
}
```

```
const euros = [29.76, 41.85, 46.5];

const sum = euros.reduce((total, amount) => total + amount);

sum // 118.11
```

REDUCE W/ ARROW FUNCTIONS

```
1 | [0, 1, 2, 3, 4].reduce( (prev, curr) => prev + curr );
```

Your **reducer** function's returned value is assigned to the accumulator, whose value is remembered across each iteration throughout the array and ultimately becomes the final, single resulting value.

ADDING VALUE

```
var fruits = ['Apple', 'Banana'];
var newLength = fruits.push('Orange');
// ["Apple", "Banana", "Orange"]
```

KEYS() IN THE ARRAY

KEYS()

keys() will return an iterator that contains all the keys in the array.

```
1 | var arr = ['a', 'b', 'c'];
2 | console.log(Object.keys(arr)); // console: ['0', '1', '2']
3 |
4 | // array like object
5 | var obj = { 0: 'a', 1: 'b', 2: 'c' };
6 | console.log(Object.keys(obj)); // console: ['0', '1', '2']
7 |
8 | // array like object with random key ordering
9 | var anObj = { 100: 'a', 2: 'b', 7: 'c' };
10 | console.log(Object.keys(anObj)); // ['2', '7', '100']
11 |
12 | // getFoo is property which isn't enumerable
13 | var myObj = Object.create({}, {
14 |   getFoo: {
15 |     value: function () { return this.foo; }
16 |   }
17 | });
18 | myObj.foo = 1;
19 | console.log(Object.keys(myObj)); // console: ['foo']
```

undefined is returned when no match

VALUES() IN THE ARRAY

```
1 const array1 = ['a', 'b', 'c'];
2 const iterator = array1.values();
3
4 for (const value of iterator) {
5   console.log(value); // expected output: "a" "b" "c"
6 }
7
```

```
1 const object1 = {
2   a: 'somestring',
3   b: 42,
4   c: false
5 };
6
7 console.log(Object.values(object1));
8 // expected output: Array ["somestring", 42, false]
9
```

REMOVE VALUE

```
var fruits = ['Apple', 'Banana'];
var last = fruits.pop(); // remove Orange (from the end)
// ["Apple", "Banana"];
```

REMOVE FROM THE FRONT

```
var first = fruits.shift();
```



Returns the removed item to you.

PASSED BY REFERENCE

Modifying an Array in a function
does alter its original declaration

```
var myArray = [ 'zero', 'one', 'two', 'three', 'four', 'five' ];

function passedByReference(arrayObject) {
    arrayObject[1] = 'dog';
}

passedByReference(myArray);

document.writeln(myArray[1]); // output: dog, not one
```

JOINING WITH A TOKEN

```
1 | var a = ['Wind', 'Rain', 'Fire'];
2 | a.join();      // 'Wind,Rain,Fire'
3 | a.join('-'); // 'Wind-Rain-Fire'
```

FIND()

```
1 function isBigEnough(element) {  
2     return element >= 15;  
3 }  
4  
5 [12, 5, 8, 130, 44].find(isBigEnough); // 130
```

Undefined when there is no match.

CONCATENATION

```
var parents = ["John", "Mary"];
var children = ["Susan", "Michael"];

var family = parents.concat(children);
```

USING OF()

OF()

The of() method creates a new array, with a variable number of args, regardless of type.

```
1 | Array.of(7);          // [7]
2 | Array.of(1, 2, 3);   // [1, 2, 3]
3 |
4 | Array(7);           // [ , , , , , , ]
5 | Array(1, 2, 3);    // [1, 2, 3]
```

```
1 | Array.of(1);         // [1]
2 | Array.of(1, 2, 3);   // [1, 2, 3]
3 | Array.of(undefined); // [undefined]
```

INCLUDES SEARCHING

The image shows a screenshot of a code editor with two code blocks. The top code block is enclosed in a light gray box and contains three numbered lines of JavaScript code. The bottom code block is also enclosed in a light gray box and contains five numbered lines of JavaScript code. A tooltip-like box is overlaid on the first line of the bottom block, containing the text "arr.includes(searchElement)" and "arr.includes(searchElement, fromIndex)".

```
1 | var a = [1, 2, 3];
2 | a.includes(2); // true
3 | a.includes(4); // false
```

```
1 | [1, 2, 3].includes(2);      // true
2 | [1, 2, 3].includes(4);      // false
3 | [1, 2, 3].includes(3, 3);  // false
4 | [1, 2, 3].includes(3, -1); // true
5 | [1, 2, NaN].includes(NaN); // true
```

arr.includes(searchElement)
arr.includes(searchElement, fromIndex)

arrayA.sort()

Sorts the array (numerically or alpha)

arrayB.join(byYourDelimiter)

creates string, separated by your delimiter

arrayC.push(object)

places it on the front of array

arrayD.toString()

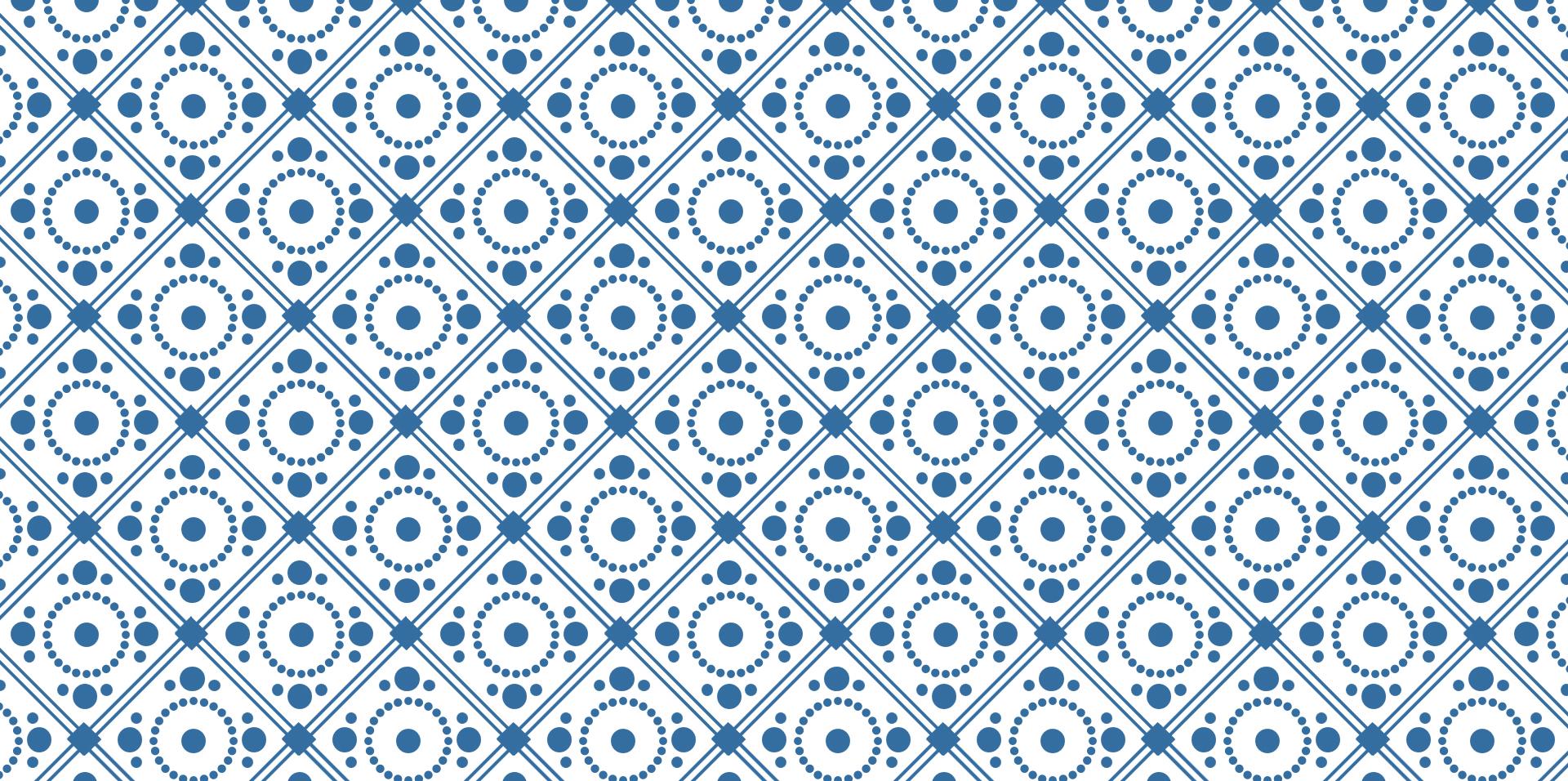
Array elements output as string

arrayE.reverse(object)

returns new array in reversed order.

arrayF.slice(from, [to])

returns new array

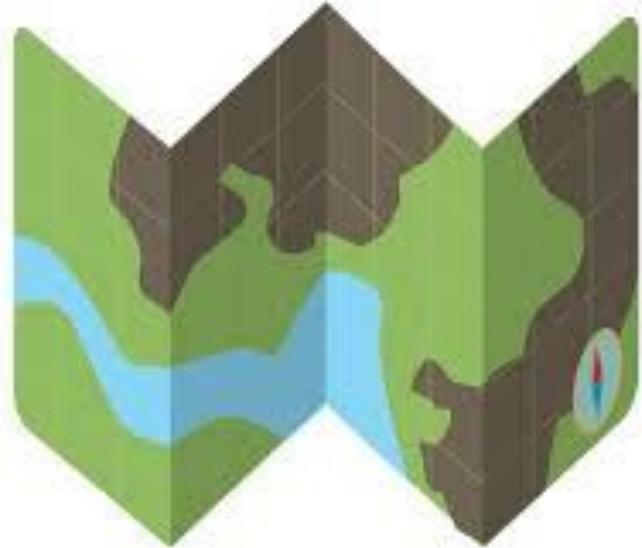


JAVASCRIPT SET

Andrew Sheehan
Boston University
Metropolitan College

WHAT IS A SET?

A Set is a built-in
data type that stores unique
values.

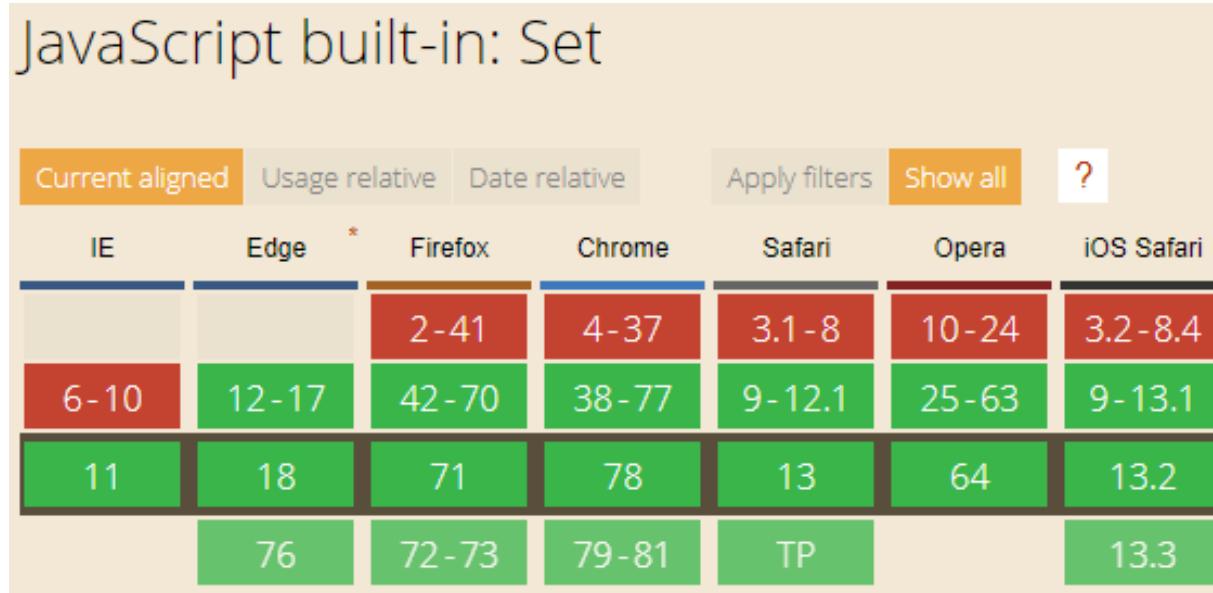


| TAKE NOTE ON THESE TYPES



NaN and **undefined** can also be stored (placed) in your Set.

| CANIUSE? AS OF 2020

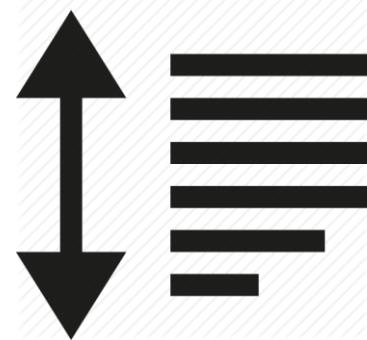


| DUPLICATES? | NOT ALLOWED

Only **unique values** are allowed.



ORDERED INSERTION ORDER



Insertions will be done in order – as you do it.

SETS USE ANYTHING

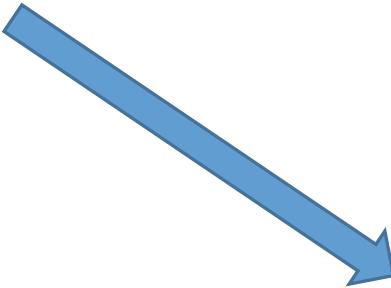
A Set can store any type:

primitives or objects.

```
var a = 10;
```

```
object = {  
    x: 5,  
    y: 6,  
};
```

USING INSTANCES



```
const uniqueNames  
      = new Set();
```

USING EXAMPLE 2

```
const names = ["an", "aj", "an",
"fi", "jk", "smh"];
const uniqueNames
    = new Set(names);
```

USING EXAMPLE (SET OF CHARACTERS)

```
// any dup char's removed  
const name = "Massachusetts";
```

```
const uniqueChars  
= new Set(name);
```

SETS

SET.ADD(VALUE)

```
const numbers = new Set();
```

```
// one expression
```

```
names.add(5);
```

```
// chained expressions
```

```
names.add(67).add(15).add(405);
```

SETS USING ARRAYS

```
1 | let myArray = ['value1', 'value2', 'value3']
2 |
3 | // Use the regular Set constructor to transform an Array into a Set
4 | let mySet = new Set(myArray)
5 |
6 | mySet.has('value1')      // returns true
```

USING DOES THE VALUE EXIST?

```
// key
if ( IdToName.has("U-13-32") ) {
    /* true: Yes, it is within the Set. */
}
```

SETS

REMOVING A VALUE

```
// true: deleted; false did not  
names.delete('Andrew');
```

CLEAR() TO REMOVE EVERYTHING

```
// All key/values are deleted  
IdToName.clear();
```

USING MAPS

RETRIEVING THE SIZE

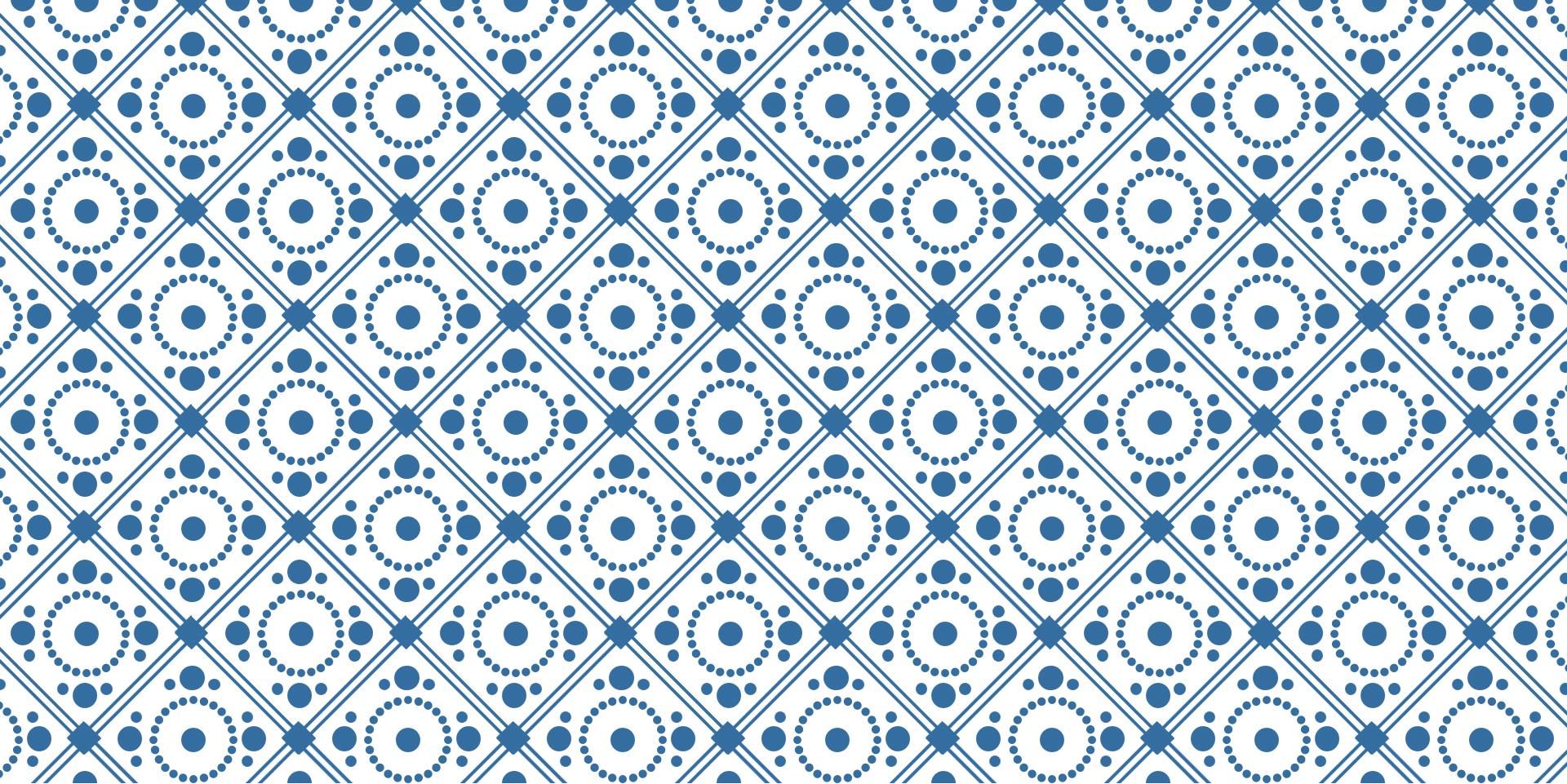
```
If ( IdToName.size > 0) {  
    // logic goes here  
}
```

USING SET ONLY THE VALUES

```
IdToName
    .values()
        .foreach(value => {
            // logic goes here.
        });
    }
```

USING SET EXAMPLE

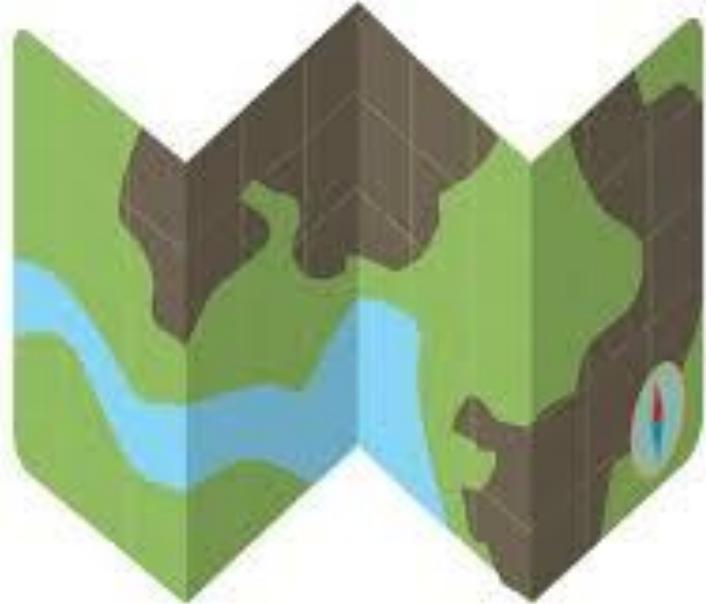
```
1 let mySet = new Set()
2
3 mySet.add(1)          // Set [ 1 ]
4 mySet.add(5)          // Set [ 1, 5 ]
5 mySet.add(5)          // Set [ 1, 5 ]
6 mySet.add('some text') // Set [ 1, 5, 'some text' ]
7 let o = {a: 1, b: 2}
8 mySet.add(o)
9
10 mySet.add({a: 1, b: 2}) // o is referencing a different object, so this is okay
11
12 mySet.has(1)          // true
13 mySet.has(3)          // false, since 3 has not been added to the set
14 mySet.has(5);         // true
15 mySet.has(Math.sqrt(25)) // true
16 mySet.has('Some Text'.toLowerCase()) // true
17 mySet.has(o)          // true
18
19 mySet.size            // 5
20
21 mySet.delete(5)        // removes 5 from the set
22 mySet.has(5)          // false, 5 has been removed
```



JAVASCRIPT MAP

MET CS
Andrew Sheehan

WHAT IS A MAP?



Map is a built-in
ES6 data type that offers a
key, value pairing

CANIUSE?

IE	Edge *	Firefox	Chrome	Safari	Opera	Safari on iOS *	Opera Mini *	Android Browser *
6-10	12-91	2-90	4-91	3.1-14	10-77	3.2-14.4		2.1-4.4.4
11	92	91	92	14.1	78	14.7	all	92
		92-93	93-95	15-TP				

WHAT IS ORDER MATTERS



The keys in Map are ordered.

When iterating, a Map object returns keys in order of insertion

BUILT IN PART OF THE CORE

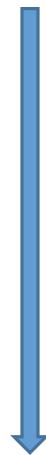
You can use any valid
Javascript data type **as key or**
as value.



DATA TYPE

EXAMPLE OF A MAP

Target a key.



Aug

Get the value.



37.3

KEYS	VALUES
Jan	327.2
Feb	368.2
Mar	197.6
Apr	178.4
May	100.0
Jun	69.9
Jul	32.3
Aug	37.3
Sep	19.0
Oct	37.0
Nov	73.2
Dec	110.9
Annual	1551.0

MAP PURPOSE

Designed as an alternative to using Object Literals for storing key, value pairs.

```
// Example of an Object Literal
const Car = {
    color: Color.UNKNOWN,
    height: 66.2,
    averageWeight: "981 lbs."
}
```

USING MAPS INstantiation

```
// Must use the new keyword
```

```
const nameTolds = new Map();
```



MAP:

ADDING TO MAP

```
const IdToName =  
    new Map();
```

```
IdToName.set("U-13-32", // key  
             "John"); // value
```

MAP: REMOVING FROM MAP

```
IdToName.delete("U-13-32");  
          // key
```

MAP: DOES KEY EXIST?

```
// key  
if ( IdToName.has("U-13-32") ) {  
    // logic goes here...  
}
```

MAP:

TO DELETE ALL KEYS/VALUES

```
// All key/values are deleted  
IdToName.clear();
```

MAP: RETRIEVING THE SIZE

```
If ( IdToName.size > 0) {  
    // logic goes here  
}
```

MAP: INITIALIZE::EXAMPLES

```
const kvArray = [['key1', 'value1'], ['key2', 'value2']]  
  
// Use the regular Map constructor to transform a 2D key-value Array into a map  
const myMap = new Map(kvArray)
```

MAP

ITERATING

```
IdToName.forEach(element => {  
    // logic goes here.  
});
```

USING MAPS

ITERATING – ANOTHER WAY

```
for (const key of m.keys() ) {  
    // logic goes here.  
}
```

USING MAPS

ONLY THE KEYS

```
IdToName.keys().foreach(key  
=> {  
    // logic goes here.  
});
```

USING MAPS ONLY THE VALUES

```
IdToName  
.values()  
.foreach(value => {  
    // logic goes here.  
});
```

WEAKMAP

IT IS DIFFERENT...

Items are never garbage collected in a standard Map.

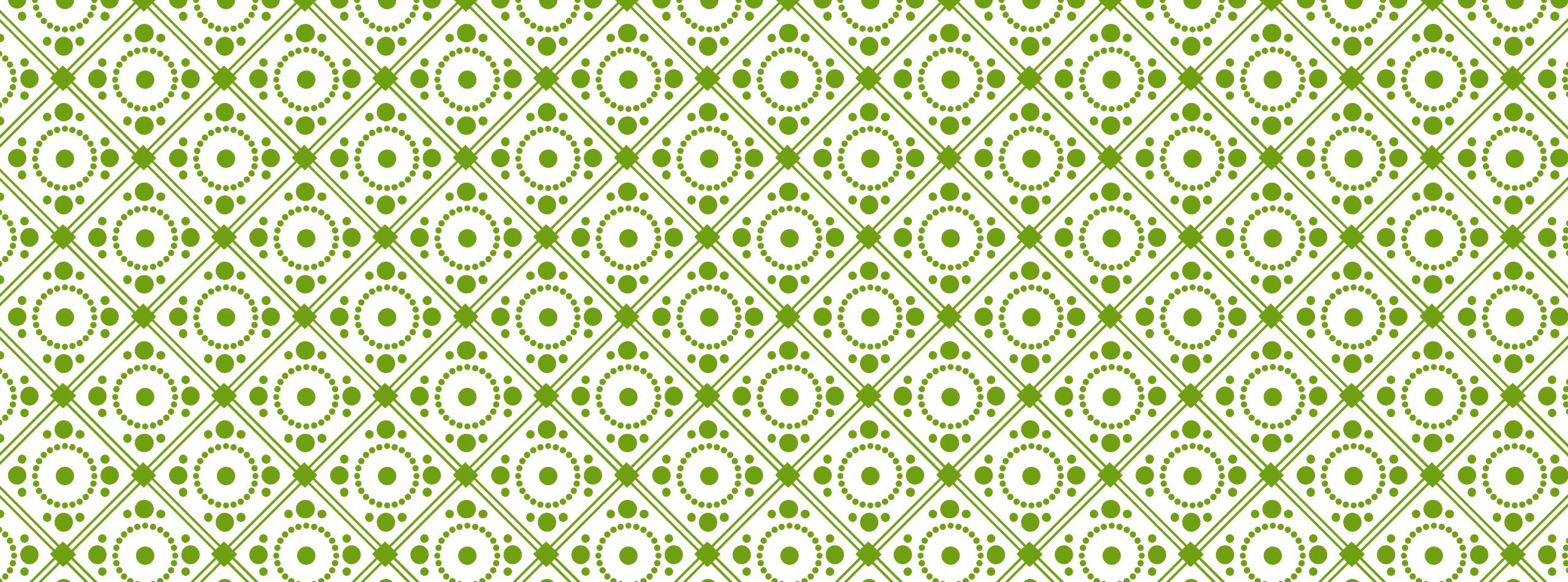
Every key of a WeakMap is an object.
When the reference to this object is lost, the value can be garbage collected

WEAKMAP

IT IS DIFFERENT...

The main differences

- a. Cannot iterate over the keys or values (or key-values) of a WeakMap
- b. Cannot clear all items from a WeakMap
- c. Cannot check the size



PROMISES

ASYNCHRONOUS PROGRAMMING

Prof. Andrew Sheehan

Metropolitan College
Boston University

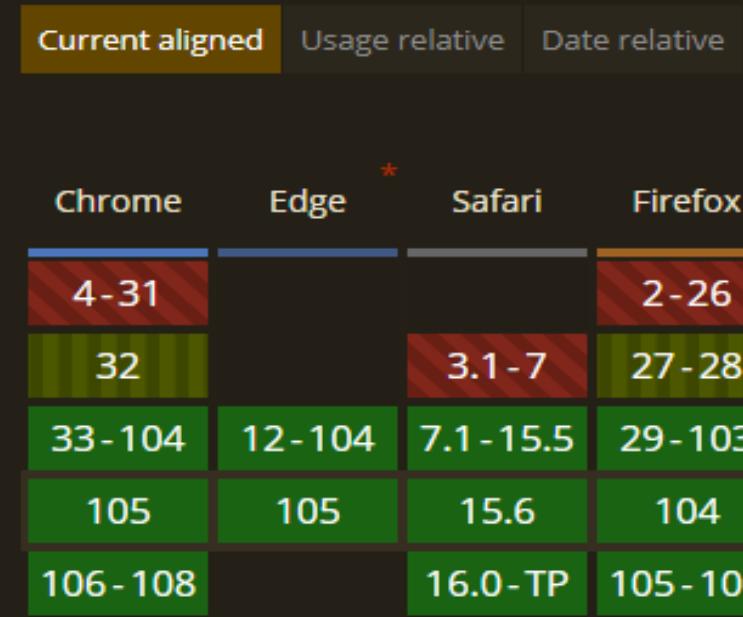
WHAT IS A PROMISE?

A Promise is a proxy for a value not necessarily known when the promise is created.

A promise represents WORK that needs to be done at some point.

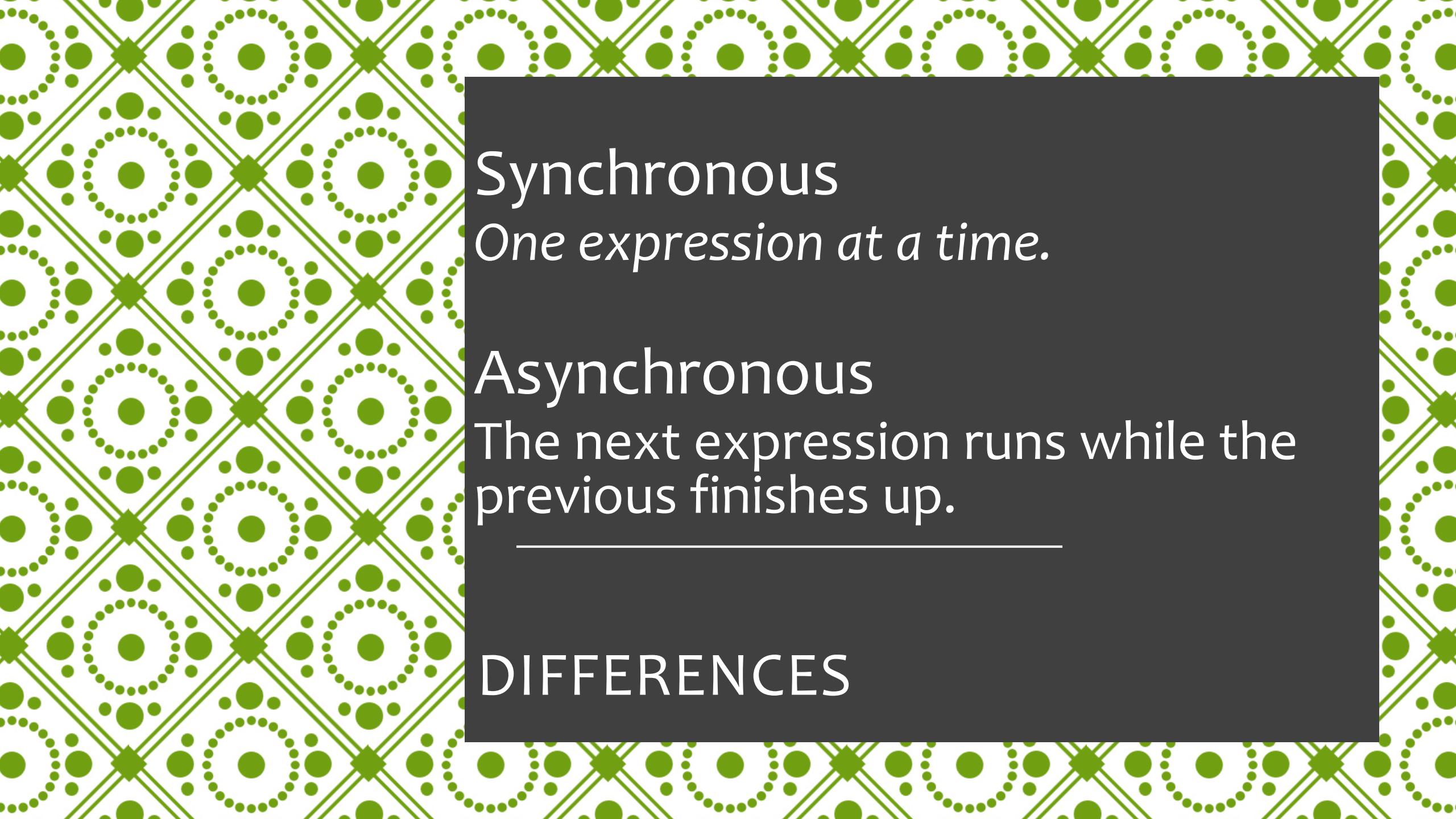
Promises - OTHER

A promise represents the eventual result of an asynchronous operation.



USEABLE
EVERYWHERE

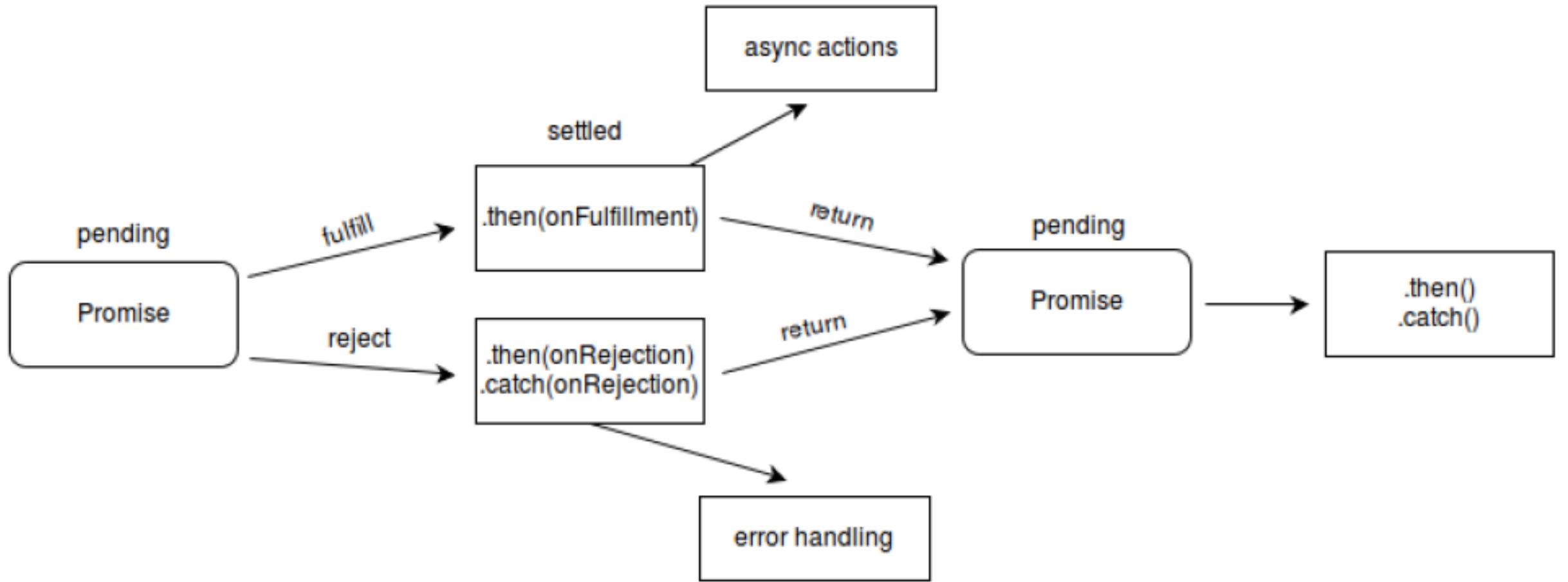
You can use it
everywhere
but not with
Internet
Explorer.



Synchronous
One expression at a time.

Asynchronous
The next expression runs while the previous finishes up.

DIFFERENCES



PROMISE STATES |

STANDARD EXAMPLE

```
var promise = new Promise(function(resolve, reject) {  
    // do a thing, possibly async, then...  
  
    if /* everything turned out fine */ {  
        resolve("Stuff worked!");  
    }  
    else {  
        reject(Error("It broke"));  
    }  
});
```

A `Promise` is in one of these states:

- *pending*: initial state, neither fulfilled nor rejected.
- *fulfilled*: meaning that the operation was completed successfully.
- *rejected*: meaning that the operation failed.

THEN()- CAN BE USED MULTIPLE TIMES

```
> const myPromise = new Promise( (resolve, reject) => {
  resolve('Hello world!');
});

myPromise.then( value => console.log(value));
myPromise.then( value => console.log(value));
myPromise.then( value => console.log(value));

Hello world!
Hello world!
Hello world!

< ▶ Promise {<resolved>: undefined}
```

```
const myPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('foo');
  }, 300);
});

myPromise
  .then(handleResolvedA)
  .then(handleResolvedB)
  .then(handleResolvedC)
  .catch(handleRejectedAny);
```

CHAINING PROMISES

Every use of
.then() will
return a
Promise
that you can
use again.

PROMISE.ALL()

```
<script>
var prom3000 = new Promise(function(resolve, reject) {
    setTimeout(function() { resolve(" 3 seconds out!"); }, 3000);
});
var prom6000 = new Promise(function(resolve, reject) {
    setTimeout(function() { resolve(" 6 seconds out!"); }, 6000);
});
var prom9000 = new Promise(function(resolve, reject) {
    setTimeout(function() { resolve(" 9 seconds out"); }, 9000);
});
$(function() {
    $("button").on("click", function() {
        Promise.all([prom3000,prom6000,prom9000]).then(function(results) {
            console.log("All promises done" + results);
        });
    });
});
</script>
```

PROMISE.ALL()

The `all()` method takes an iterable of promises as input, and returns a single Promise that resolves to an array of the results of the input promises, in order.

Promise.all(iterable)

Wait for all promises to be resolved, or for any to be rejected.

If the returned promise resolves, it is resolved with an aggregating array of the values from the resolved promises, in the same order as defined in the iterable of multiple promises.

If it rejects, it is rejected with the reason from the first promise in the iterable that was rejected.

.ALL() |

RACE() - WAIT UNTIL ANY PROMISE WORKS OR FAILS

Promise.race(iterable)

Wait until any of the promises is fulfilled or rejected.

If the returned promise resolves, it is resolved with the value of the first promise in the iterable that resolved.

If it rejects, it is rejected with the reason from the first promise that was rejected.

CATCHING ERRORS

The `catch()` returns a Promise. Deals w/rejected cases only.

```
1 const promise1 = new Promise(resolve, reject) => {  
2   throw 'Uh-oh!';  
3 });  
4  
5 promise1.catch((error) => {  
6   console.error(error);  
7 });  
8 // expected output: Uh-oh!
```



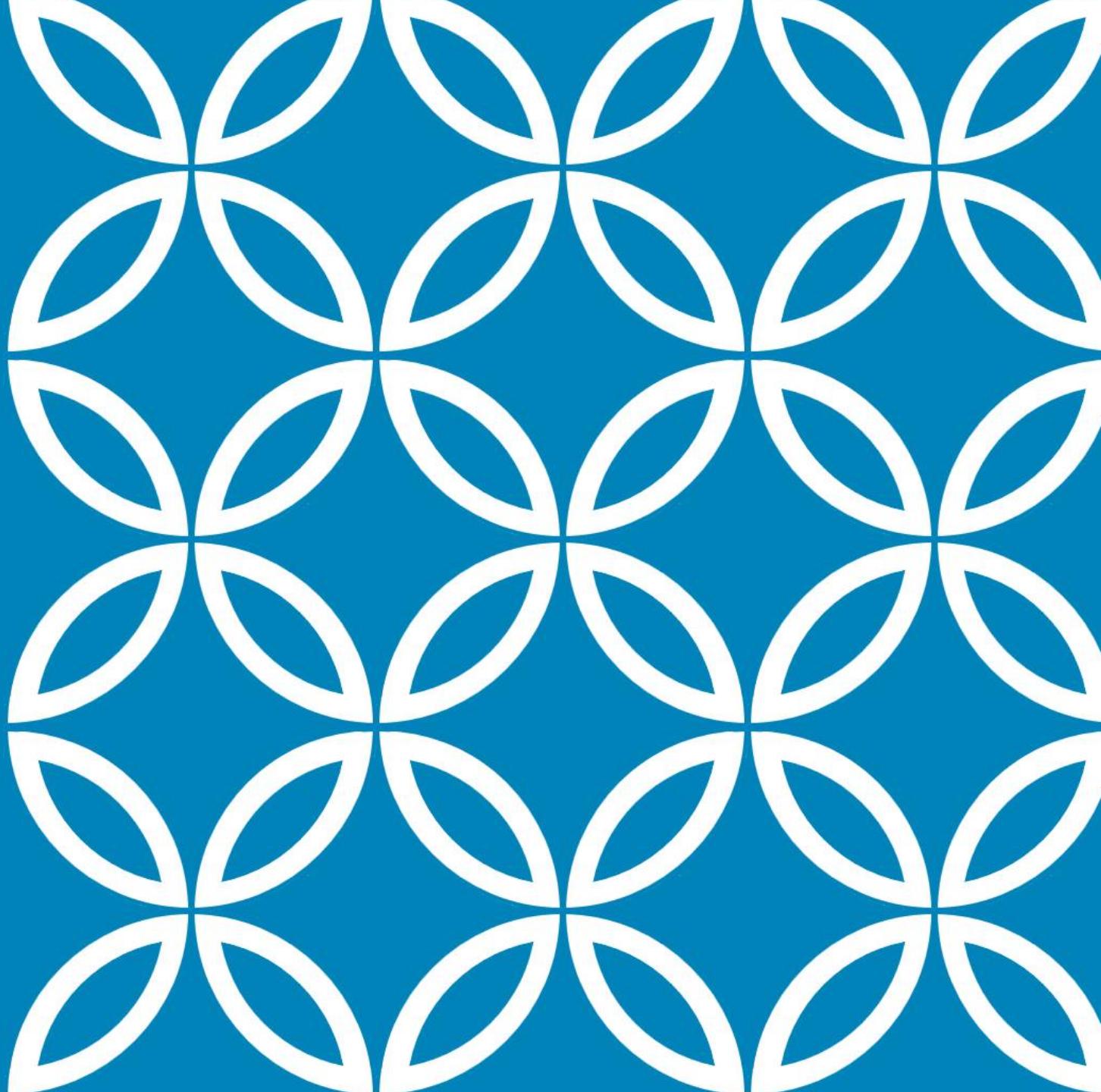
CATCH() IS JUST A FORM OF THEN()

The catch() is just like then() without a callback function for success.

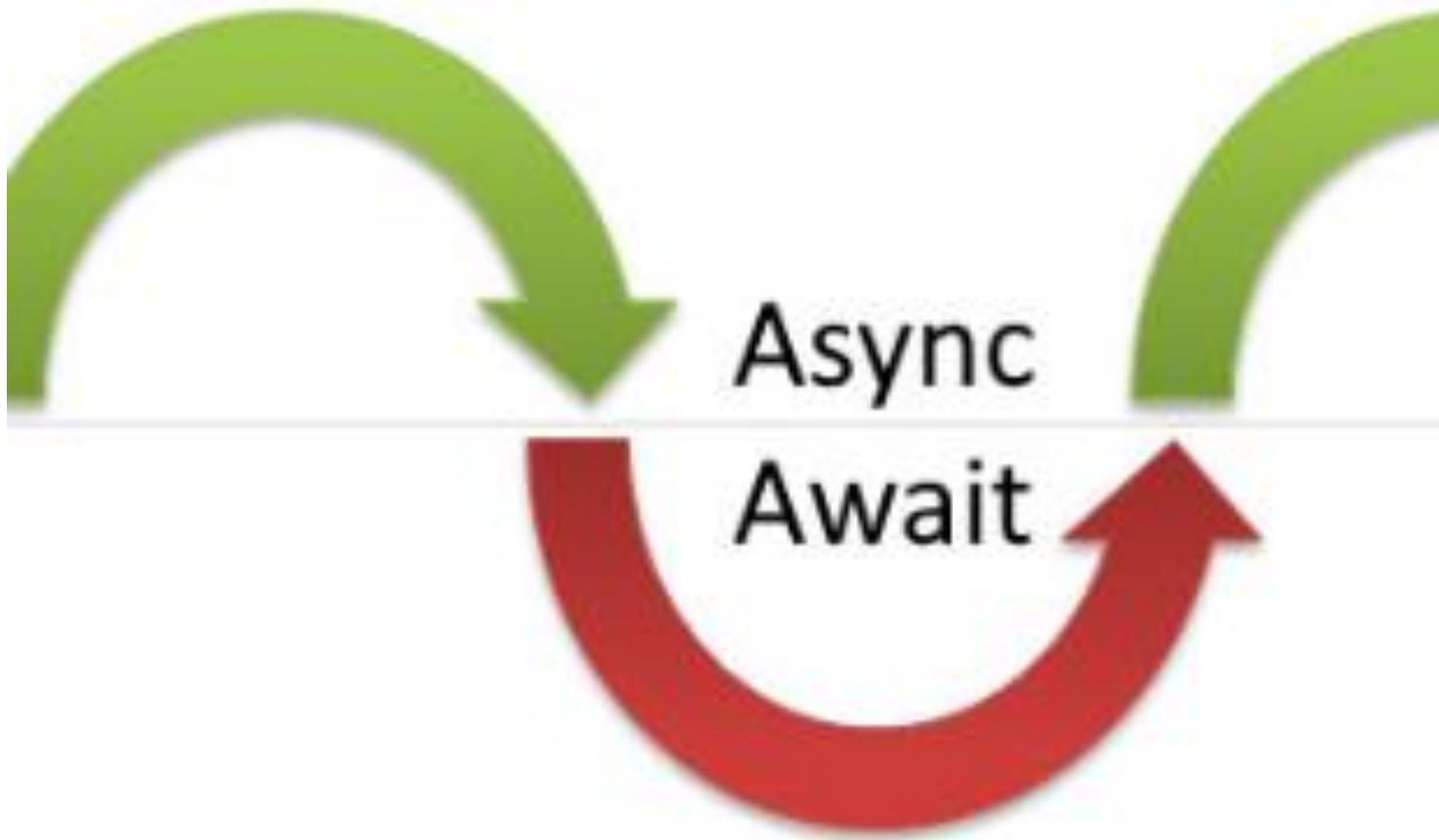
ASYNC & AWAIT

Andrew Sheehan

Boston University
Computer Science Dept.



WHAT ARE THEY?



A different syntax
to code **Promises**.

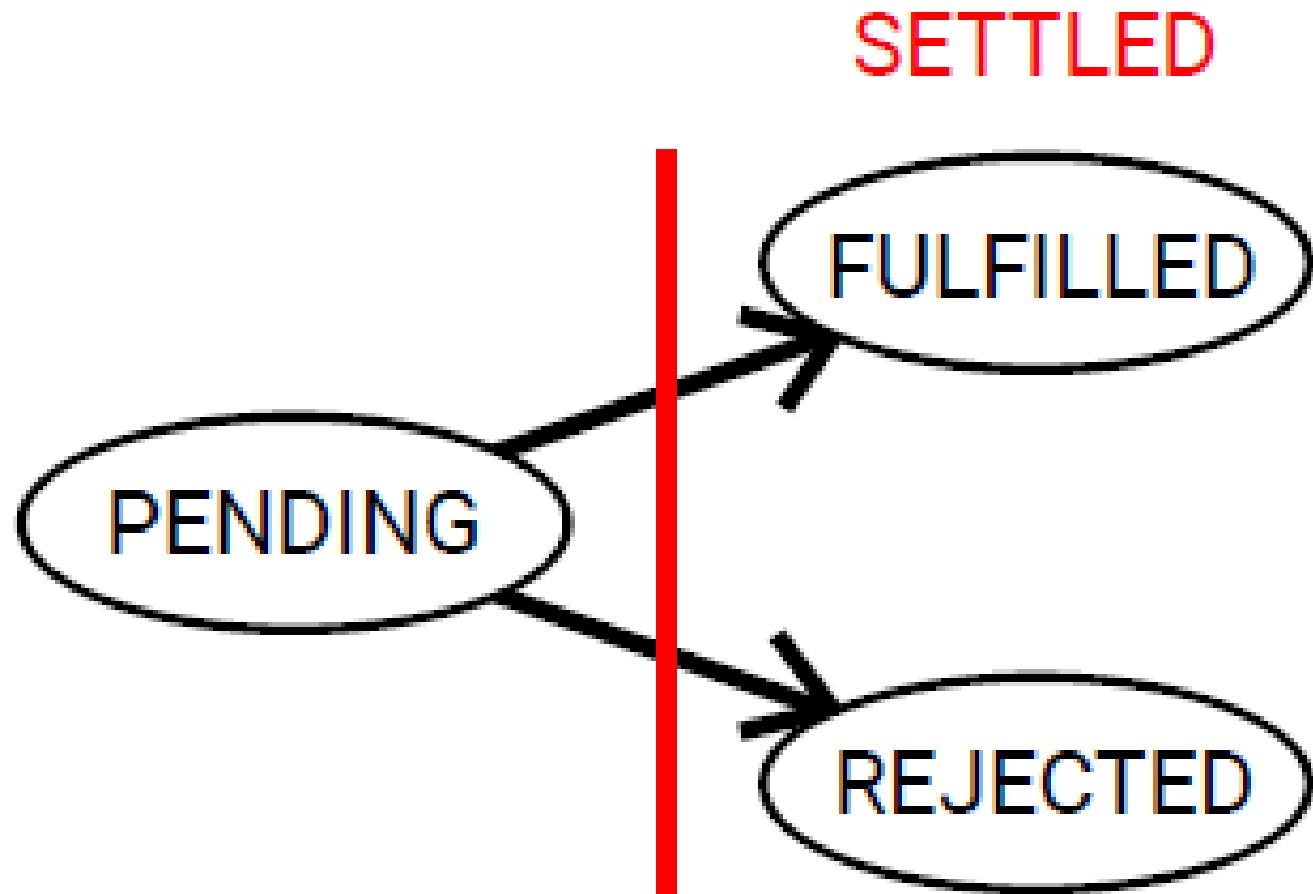
Became part of
the standard in
ECMAScript 2017

THE
CREATOR OF
THE ASYNC
MODULE

Caolan
McMahon



RECAP: PROMISE STATES



- `async` is a keyword for the function declaration
- `await` is used during the promise handling
- `await` must be used within an `async` function, though Chrome now supports "top level" `await`
- `async` functions return a promise, regardless of what the `return` value is within the function
- `async / await` and promises are essentially the same under the hood

TL;DR

EXAMPLE

```
async function save(Something) {  
  try {  
    await Something.save()  
  } catch (ex) {  
    //error handling  
  }  
  console.log('success');  
}
```

EXAMPLE:

AWAITING YOUR FETCH() CALLS.

```
async function fetchContent() {  
    // Instead of using fetch().then, use await  
    let content = await fetch('/');  
    let text = await content.text();  
  
    // Inside the async function text is the request body  
    console.log(text);  
  
    // Resolve this async function with the text  
    return text;  
}  
  
// Use the async function  
var promise = fetchContent().then(...);
```

BEFORE AND AFTER



```
fetch('/users.json')
  .then(response => response.json())
  .then(json => {
    console.log(json);
  })
  .catch(e => { console.log('error!'); })
```

// After: no more callbacks!

```
async function getJson() {
  try {
    let response = await fetch('/users.json');
    let json = await response.json();
    console.log(json);
  }
  catch(e) {
    console.log('Error!', e);
  }
}
```

```
async function test() {  
  // This function will print "Hello, World!" after 1 second.  
  await new Promise(resolve => setTimeout(() => resolve(), 1000));  
  console.log('Hello, World!');  
}  
  
test();
```

EXAMPLE USING A TIMEOUT |



```
async function test() {  
  // Wait 100ms 10 times. This function also prints after 1 sec  
  for (let i = 0; i < 10; ++i) {  
    await new Promise(resolve => setTimeout(resolve, 100));  
  }  
  console.log('Hello, World!');  
}  
  
test();
```

YOU CAN USE AWAIT IN
IF STATEMENTS, FOR
LOOPS, AND TRY/CATCH
BLOCKS |

AWAIT

Await on the
fetch(); await
on the json();

```
async function showAvatar() {  
  
    // read our JSON  
    let response = await fetch('/article/promise-chaining/user.json');  
    let user = await response.json();  
  
    // read github user  
    let githubResponse = await fetch(`https://api.github.com/users/${user.name}`);  
    let githubUser = await githubResponse.json();  
  
    // show the avatar  
    let img = document.createElement('img');  
    img.src = githubUser.avatar_url;  
    img.className = "promise-avatar-example";  
    document.body.append(img);  
  
    // wait 3 seconds  
    await new Promise((resolve, reject) => setTimeout(resolve, 3000));  
  
    img.remove();
```

TYPICAL ERROR SITUATION

There is one major restriction for using await: You can only use await within the body of a **function that's marked `async`**

```
function test() {  
    const p = new Promise(resolve => setTimeout(resolve, 1000));  
    // SyntaxError: Unexpected identifier  
    await p;  
}
```

Anonymous Async Function

```
let main = (async function() {  
  let value = await fetch('/');  
})();
```

Async Function Declaration

```
async function main() {  
  let value = await fetch('/');  
};
```

Async Function Assignment

```
let main = async function() {  
  let value = await fetch('/');  
};
```

DECLARING

Async Function as Argument

```
document.body.addEventListener('click', async function() {  
  let value = await fetch('/');  
});
```

PASSING |
AS ARGUMENTS |

OBJECTS AND METHODS

```
// Object property
let obj = {
  async method() {
    let value = await fetch('/');
  }
};

// Class methods
class MyClass {
  async myMethod() {
    let value = await fetch('/');
  }
}
```

PARALLELISM

```
// Will take 1000ms total!
async function series() {
  await wait(500);
  await wait(500);
  return "done!";
}
```

```
// Would take only 500ms total!
async function parallel() {
  const wait1 = wait(500);
  const wait2 = wait(500);
  await wait1;
  await wait2;
  return "done!";
}
```

- Trigger both wait calls and then use await.
- Allows the async functions to happen concurrently

PROMISE.ALL() EQUIVALENT

```
let [foo, bar] = await Promise.all([getFoo(), getBar()]);
```

ERROR HANDLING

If there is an error, you can use a standard:

try { ... } catch() { ... }

```
async function test() {
  try {
    const p = Promise.reject(new Error('Oops!'));
    // The below 'await' throws
    await p;
  } catch (error) {
    console.log(error.message); // "Oops!"
  }
}
```

ERROR HANDLING

This code:

```
1 async function f() {  
2   await Promise.reject(new Error("Whoops!"));  
3 }
```

...Is the same as this:

```
1 async function f() {  
2   throw new Error("Whoops!");  
3 }
```

We can catch that error using `try...catch`,

```
async function f() {  
  
  try {  
    let response = await fetch('http://no-such-url');  
  } catch(err) {  
    alert(err); // TypeError: failed to fetch  
  }  
}  
  
f();
```



eXtensible

Extend our notion.



Markup

Write using Markups

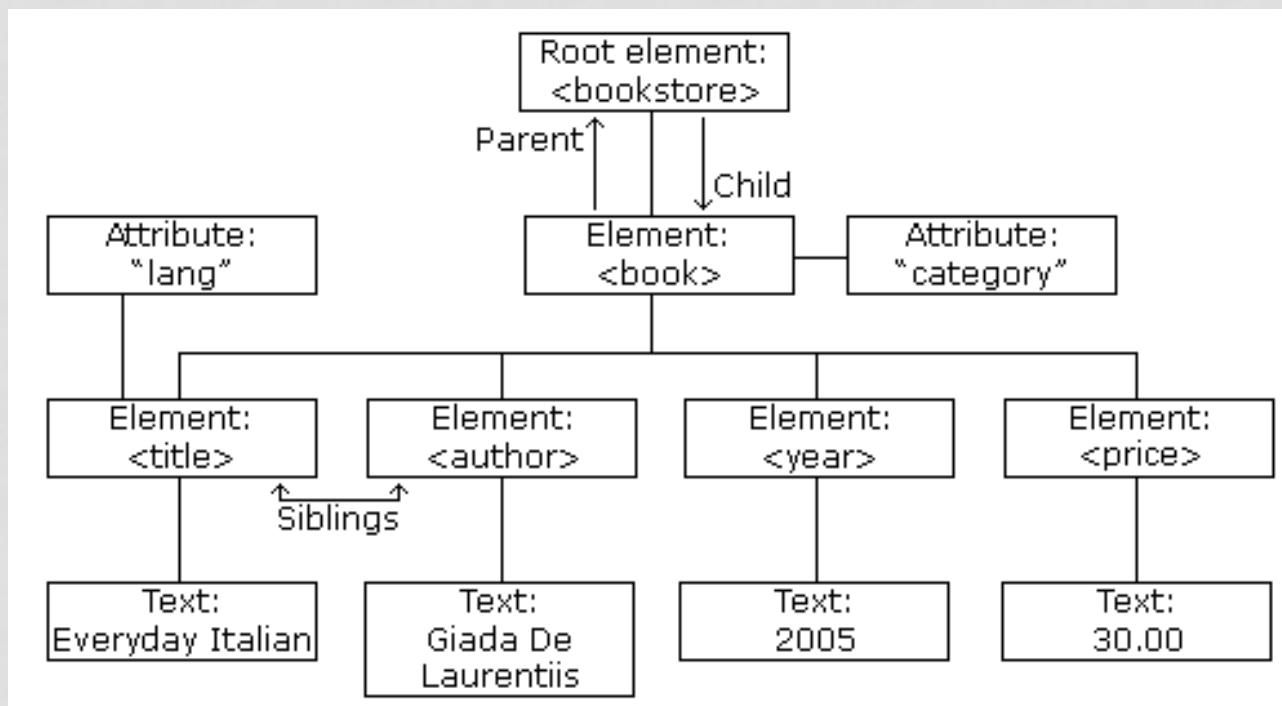


Language

Create your own document.

USING XML WITH JAVASCRIPT

XML IS HIERARCHICAL



THE DOCUMENT OBJECT MODEL

XML elements can be
accessed through the
DOM.

THE XML DOM

XML DOM is:

- A standard object model for XML
- A standard programming interface for XML
- Platform- and language-independent
- A W3C standard

THE XML DOCUMENT (THE ACTUAL DATA)

```
1  <bookstore>
2      <book category="cooking">
3          <title lang="en">Everyday Italian</title>
4          <author>Giada De Laurentiis</author>
5          <year>2005</year>
6          <price>30.00</price> </book>
7      <book category="children">
8          <title lang="en">Harry Potter</title>
9          <author>J. K. Rowling</author>
10         <year>2005</year>
11         <price>29.99</price> </book>
12     <book category="web">
13         <title lang="en">XQuery Kick Start</title>
14         <author>James McGovern</author>
15         <author>Per Bothner</author>
16         <author>Kurt Cagle</author>
17         <author>James Linn</author>
18         <author>Vaidyanathan Nagarajan</author>
19         <year>2003</year>
20         <price>49.99</price> </book>
21     <book category="web" cover="paperback">
22         <title lang="en">Learning XML</title>
23         <author>Erik T. Ray</author>
24         <year>2003</year>
25         <price>39.95</price> </book>
26     </bookstore>
```

USING FETCH

Same folder/location
as the HTML

```
<script>
    fetch("books.xml")
        .then(result => {return result.text();})
        .then(xml => {
            const domParser = new DOMParser();
            const xmlDocument = domParser.parseFromString(xml, "text/xml");
            document.getElementById("demo").innerHTML = xmlDocument.getElementsByTagName("title")[0].childNodes[0].nodeValue;
        });
</script>
```

USING XHR

Object:
XMLDocument

```
<script>
  const xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      const xmlDoc = xml.responseXML;
      const target = document.getElementById("demo");

      target.innerHTML = xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
    }
  };
  xhttp.open("GET", "books.xml", true);
  xhttp.send();|
```

DOM PARSER

The DOMParser parses XML (or HTML) from a string into a DOM Document.

```
1 | var parser = new DOMParser();
2 | var doc = parser.parseFromString(stringContainingXMLSource, "application/xml");
```

DOMPARSER() IS USEFUL.

```
1 var parser = new DOMParser();
2 var doc = parser.parseFromString(stringContainingXMLSource, "application/xml");
3 // returns a Document, but not a SVGDocument nor a HTMLDocument
4
5 parser = new DOMParser();
6 doc = parser.parseFromString(stringContainingSVGSource, "image/svg+xml");
7 // returns a SVGDocument, which also is a Document.
8
9 parser = new DOMParser();
10 doc = parser.parseFromString(stringContainingHTMLSource, "text/html");
11 // returns a HTMLDocument, which also is a Document.
```

LOADING XML - STRINGS

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <title>XML Parsing Example</title>
5      <meta charset="utf-8"/>
6      <meta name="viewport" content="width=device-width, initial-scale=1"/>
7      <link href="https://fonts.googleapis.com/css?family=Mali" rel="stylesheet">
8      <style>
9          * { font-family: 'Mali', cursive; }
10     </style>
11     </head>
12     <body>
13         <p id="demo"></p>
14         <script>
15             const xmlAsString = "<bookstore><book><title>Everyday Ethopian</title><author>Giada De Laurentiis</author><year>2005</year></book></bookstore>";
16             const parser = new DOMParser();
17             const xmlDoc = parser.parseFromString(xmlAsString, "text/xml");
18
19             document.getElementById("demo").innerHTML = xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
20         </script>
21     </body>
22 </html>
```

INTERNET EXPLORER

This approach is old,
but works in IE 5,6,7,8..

```
xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async = false;
xmlDoc.loadXML(txt);
```

ACTIVEXObject IS DEAD

```
var parseXml;
if (typeof window.DOMParser != "undefined") {
    parseXml = function(xmlStr) {
        return ( new window.DOMParser() ).parseFromString(xmlStr, "text/xml");
    };
} else if (typeof window.ActiveXObject != "undefined" &&
           new window.ActiveXObject("Microsoft.XMLDOM")) {
    parseXml = function(xmlStr) {
        var xmlDoc = new window.ActiveXObject("Microsoft.XMLDOM");
        xmlDoc.async = "false";
        xmlDoc.loadXML(xmlStr);
        return xmlDoc;
    };
} else {
    throw new Error("No XML parser found");
}
```

```
var xml = parseXml("<foo>Stuff</foo>");
```

PARSING W/JQUERY

```
var xml = "<music><album>Beethoven</album></music>";  
  
var result = $(xml).find("album").text();
```

XML PARSING

-ANOTHER EXAMPLE-

```
<fruits>
  <fruit name="Apple" colour="Green" />
  <fruit name="Banana" colour="Yellow" />
</fruits>
```

```
function getFruits(xml) {
  var fruits = xml.getElementsByTagName("fruits")[0];
  if (fruits) {
    var fruitsNodes = fruits.childNodes;
    if (fruitsNodes) {
      for (var i = 0; i < fruitsNodes.length; i++) {
        var name = fruitsNodes[i].getAttribute("name");
        var colour = fruitsNodes[i].getAttribute("colour");
        alert("Fruit " + name + " is coloured " + colour);
      }
    }
  }
}
```

JQUERY

jQuery.parseXML()

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>jQuery.parseXML demo</title>
6   <script src="https://code.jquery.com/jquery-1.10.2.js"></script>
7 </head>
8 <body>
9
10 <p id="someElement"></p>
11 <p id="anotherElement"></p>
12
13 <script>
14 var xml = "<rss version='2.0'><channel><title>RSS Title</title></channel></rss>",
15     xmlDoc = $.parseXML( xml ),
16     $xml = $( xmlDoc ),
17     $title = $xml.find( "title" );
18
19 // Append "RSS Title" to #someElement
20 $( "#someElement" ).append( $title.text() );
21
22 // Change the title to "XML Title"
23 $title.text( "XML Title" );
24
25 // Append "XML Title" to #anotherElement
26 $( "#anotherElement" ).append( $title.text() );
27 </script>
28
29 </body>
30 </html>
```

LOADING XML FROM A FILE

```
$.ajax({  
    url: 'xmlfile.xml',  
    dataType: 'xml',  
    success: function(data){  
        // Extract relevant data from XML  
        var xml_node = $('Pages',data);  
        console.log( xml_node.find('Page[Name="test"] > controls > test').text() );  
    },  
    error: function(data){  
        console.log('Error loading XML data');  
    }  
});
```



Same folder as the html resides

XHR OBJECT

The XMLHttpRequest (XHR) object
has a XML property

```
xmlDoc = xmlhttp.responseXML;
```



JSON

Prof. Andrew Sheehan

Boston University/MET
Computer Science Dept.

WHAT IS JSON

Java Script Object Notation

BASICS

EXAMPLE

```
const human = {  
    age: 0,  
    name: 'specify',  
    gender: 'specify'  
};
```

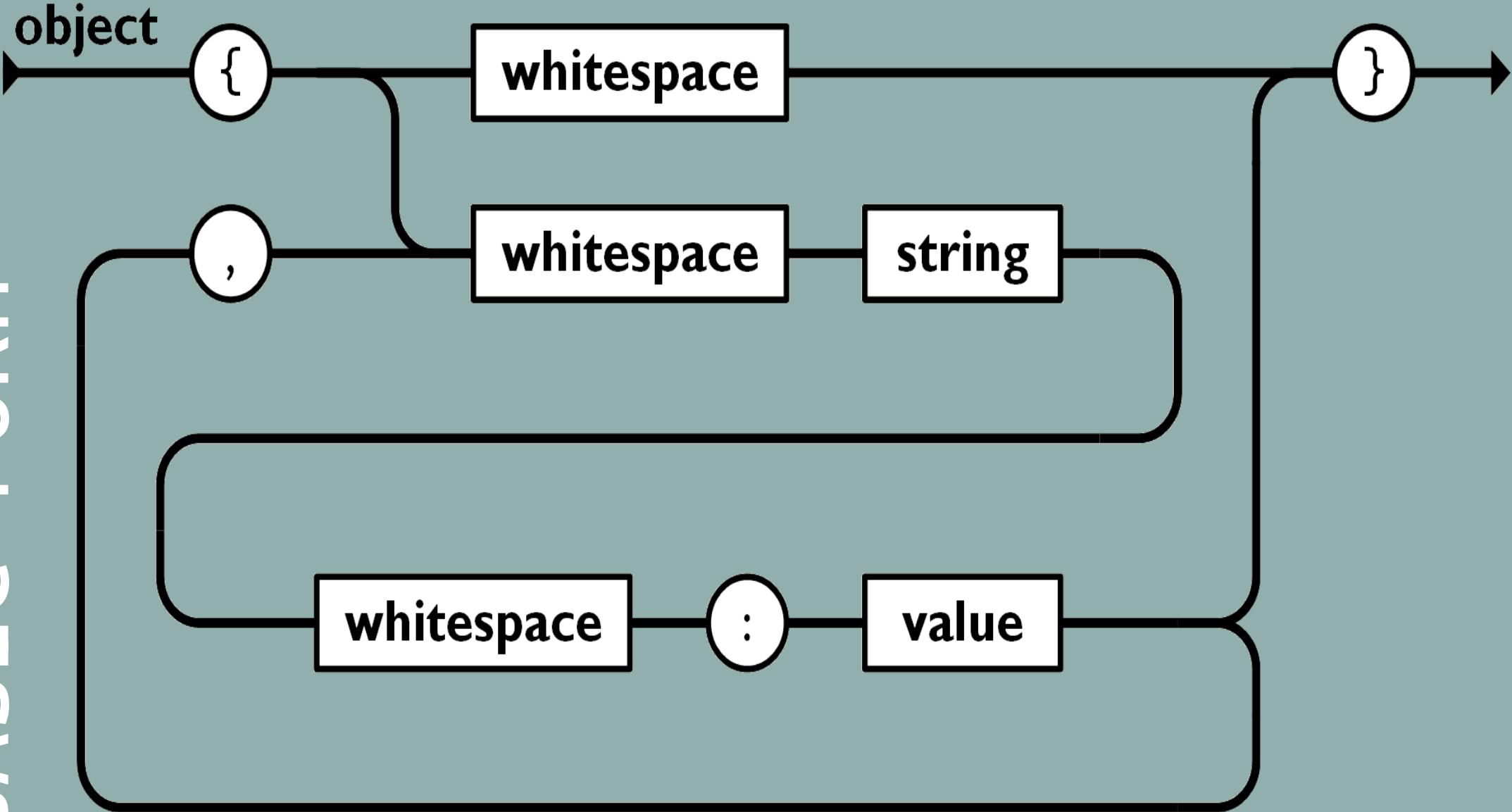
BASICS:

ANOTHER WAY

```
const human = [];  
  
human['age'] = 0;  
human['name'] = 'specify';  
human['gender'] = 'specify';
```

DATA STRUCTURE

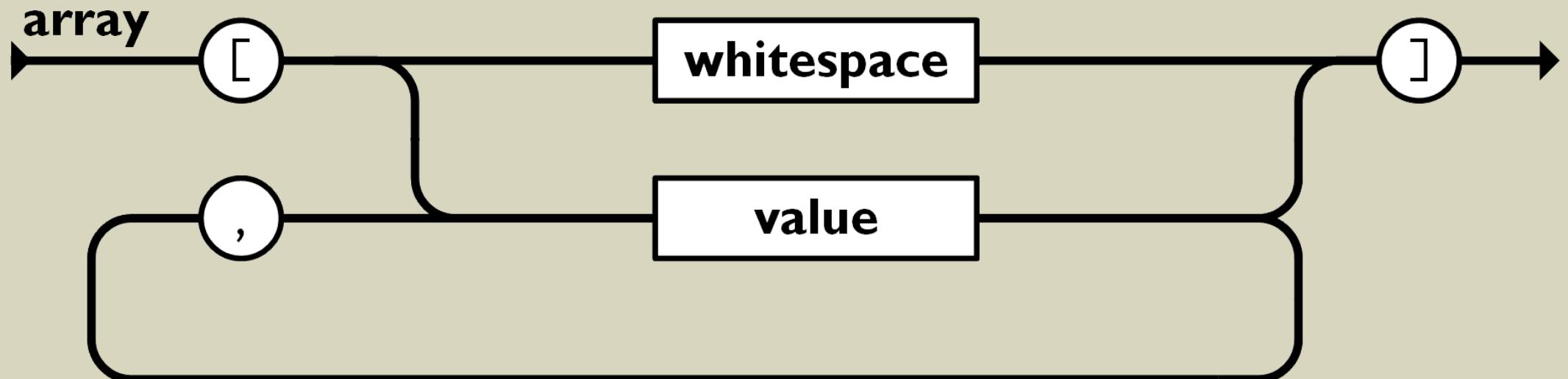
BASIC FORM



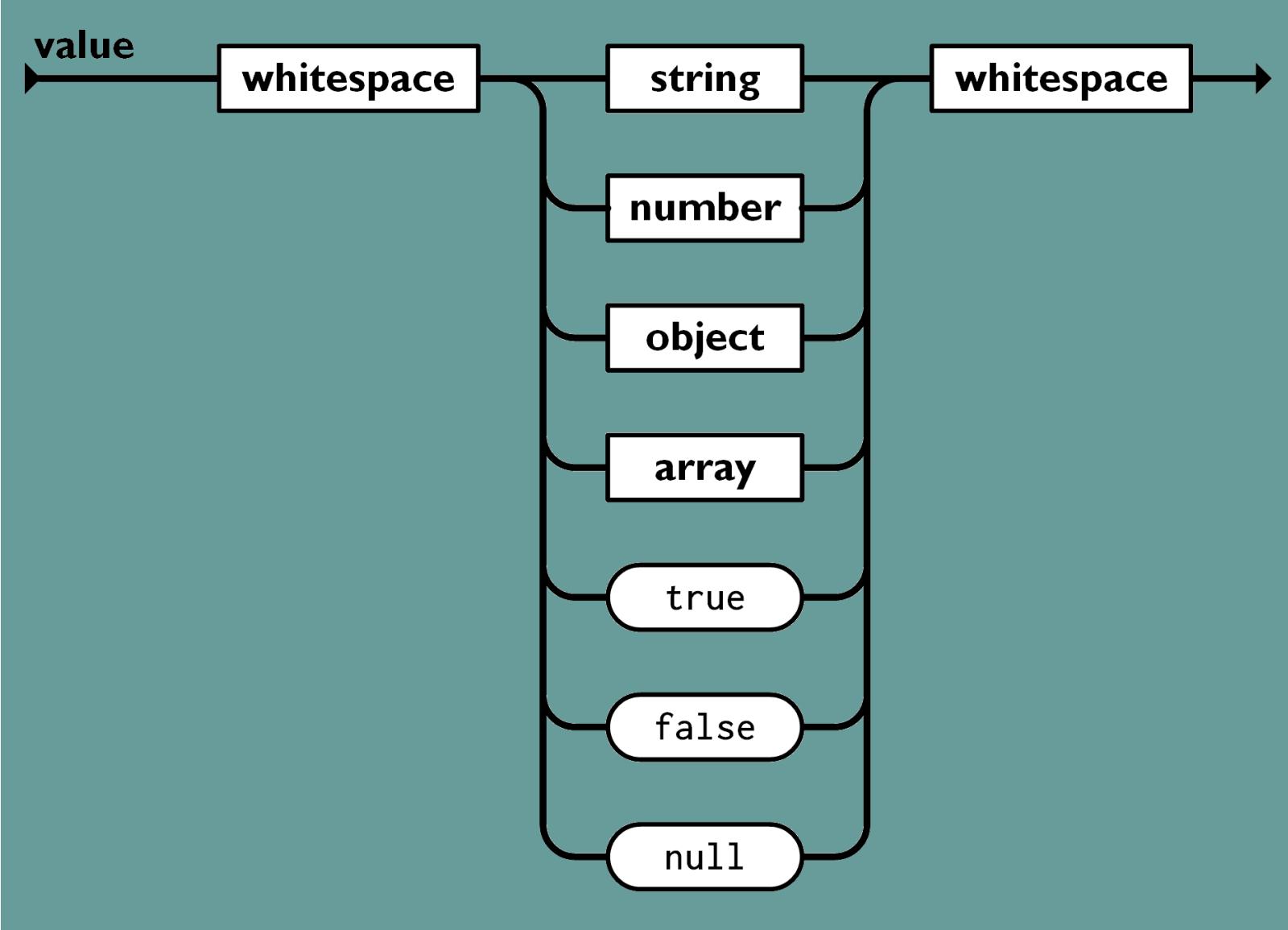
DATA STRUCTURE ARRAY

```
const clients = [  
    { name: 'Fran', age: 1},  
    { name: 'Helen', age: 3},  
    { name: 'Camiel', age: 7},  
    { name: 'Beatrice', age: 11}  
];  
  
const helensAge = clients[1].age;
```

ARRAY FORM



VALUES



ARRAY NOTATION

STANDARD

```
const lotteryNumbers = [  
    23, 11, 7, 45, 32, 21  
]; // this is not json...
```

JSON v. XML

XML: must be (1) loaded,
then (2) parsed. Uses
more memory

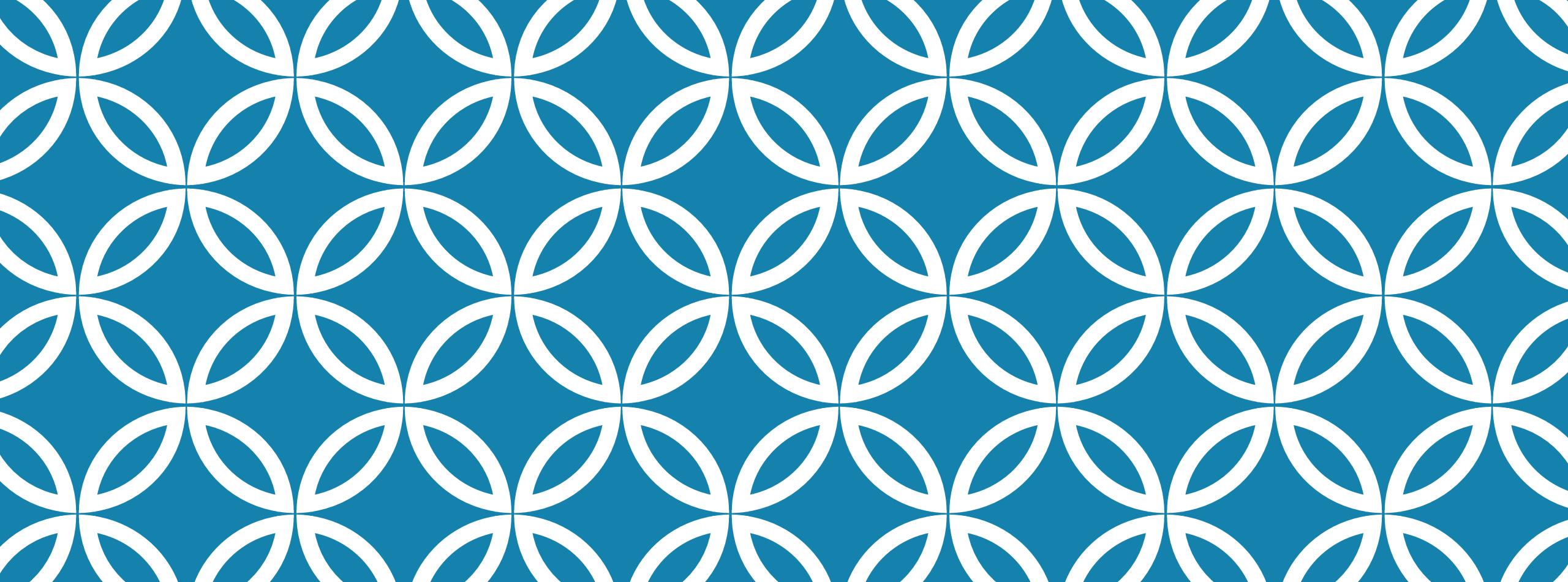
JSON: just use it, as is

Retrieving data from json-formatted data structure is much easier than working with XML

```
const human = {  
  age: 0,  
  name: 'specify',  
  gender: 'specify'  
};
```

```
const age = human.age;  
// object literal destructuring  
let {age} = human;
```

GETTING TO THE DATA |



JAVASCRIPT MODULES

Andrew Sheehan
Boston University
Metropolitan College

ALSO
KNOWN AS

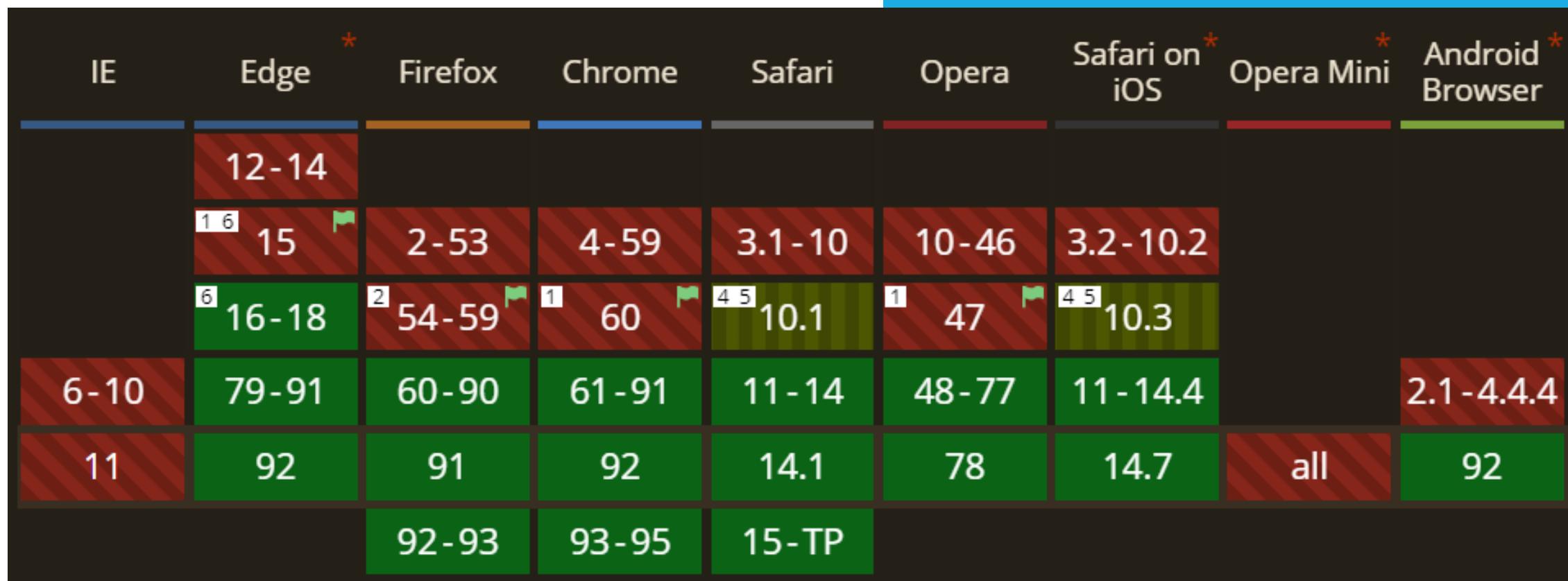


ES Modules

~

ECMAScript Modules

SUPPORT



EXPOSE BY EXPORTING

When something is made available to other modules or pages, it's called an **export**

PARTITION YOUR JS CODE

Provides a means to group your code.

Anything not exported is private.



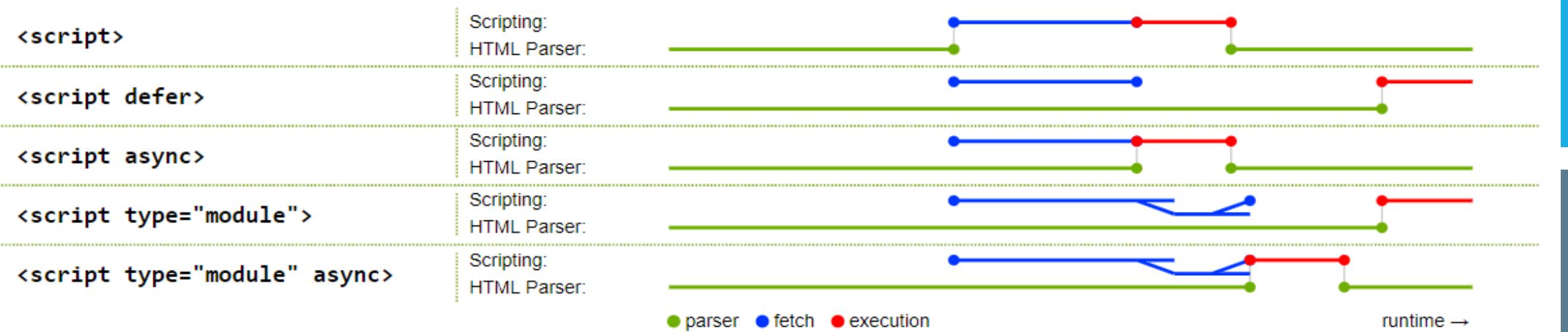
NOT USING MODULES?

When you include multiple .js files into your HTML, you are not provided any scope protection.

Most declarations go into the **global space**

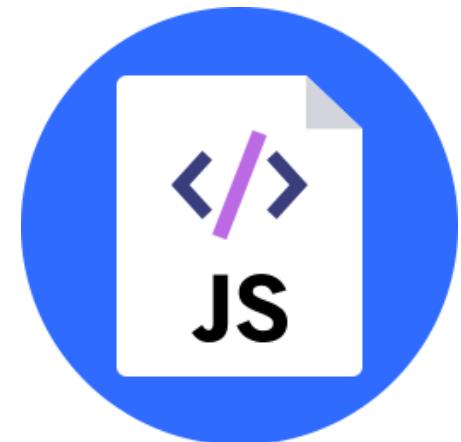
MODULES ‘DEFER’

Classic `<script>`s block the HTML parser by default. You can work around it by adding [the `defer` attribute](#), which ensures that the script download happens in parallel with HTML parsing.



1 MODULE ONE FILE

Declare 1 module per file.
That is the per specification.



DEFAULT EXPORT

NAMED EXPORTS

TYPES OF EXPORTS |

NAMED EXPORTS

```
let company = "TutorialsPoint"

let getCompany = function(){
    return company.toUpperCase()
}

let setCompany = function(newValue){
    company = newValue
}

export {company, getCompany, setCompany}
```

```
import {company as x, getCompany as y} from './company1.js'

console.log(x)
console.log(y())
```

```
//using multiple export keyword  
export function f() {};  
export class g { ... };  
...  
export const PI = 3.14;
```

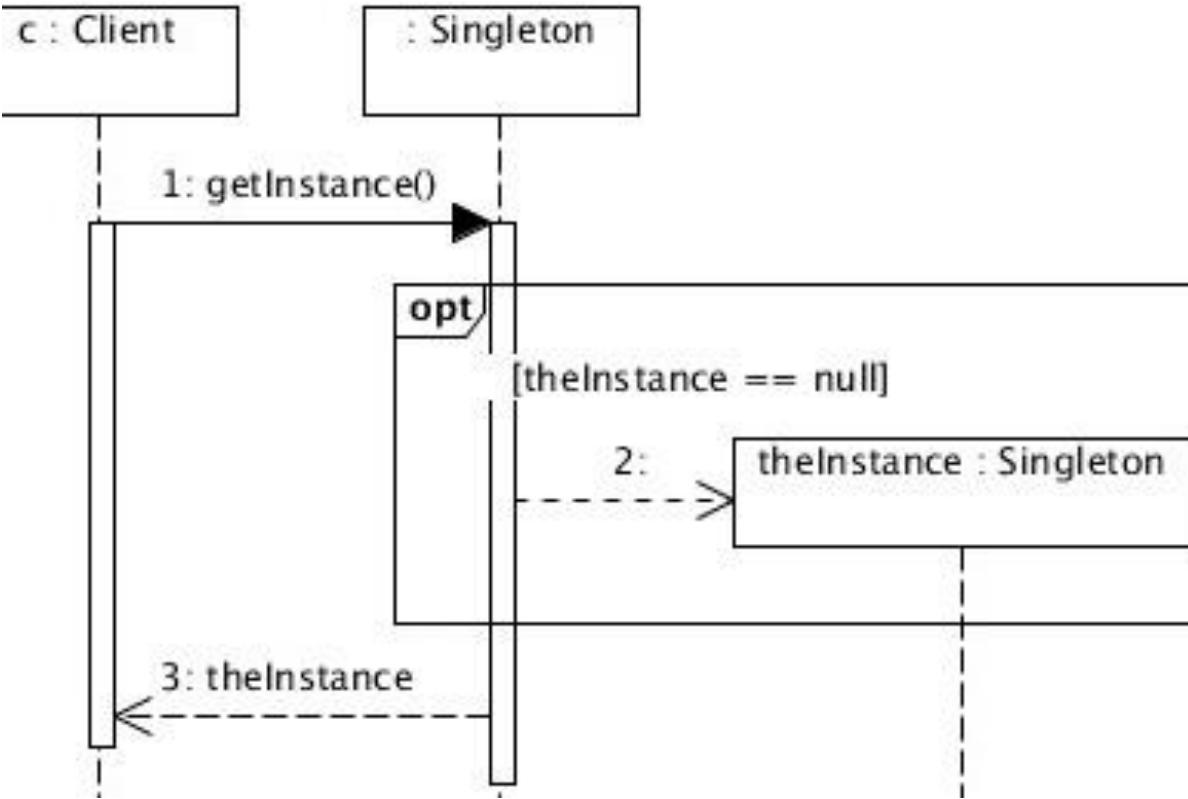
```
//or - using single export keyword  
export {f, g, PI};
```

NAMED EXPORTS |

//there can only be 1 default in your module...

export default function renderCircle(radius, fill, options) {...}

DEFAULT EXPORT |



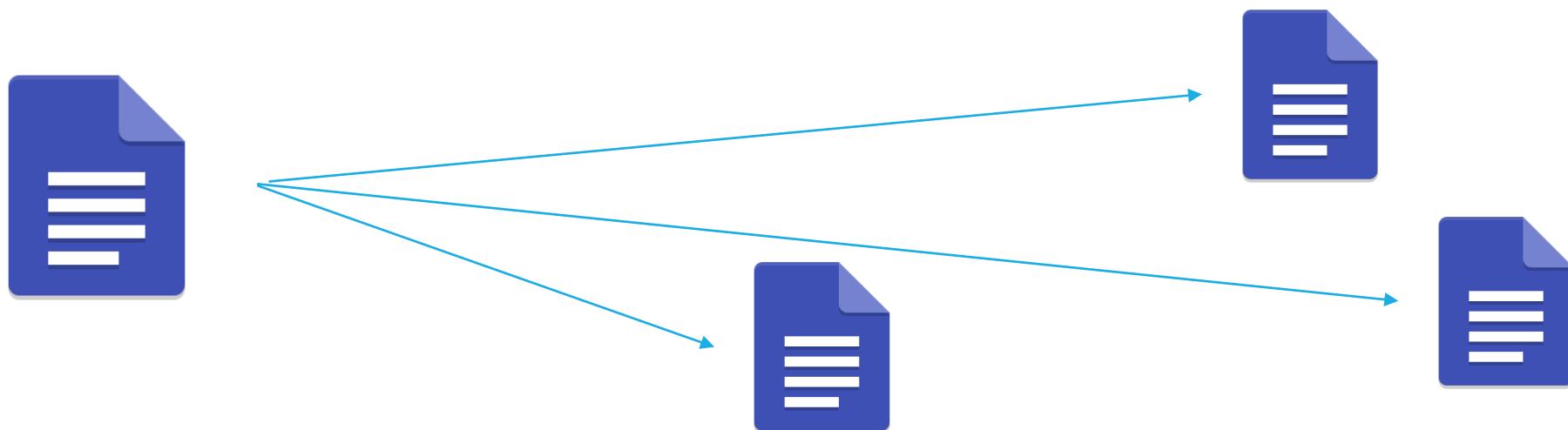
SINGLETON

Once you import a module, it will not be imported again.

*It will be **cached***

MODULES CAN HAVE DIRECT DEPENDENCIES/TRANSITIVE DEPENDENCIES

When a module requires something from another module.



SAME DIRECTORY?

? In the same folder
as the module

Use: `'.'`



COMMONJS USED BY NODE



```
// add.js
function add (a, b) {
  return a + b
}

module.exports = add
```

```
// index.js
const add = require('./add')

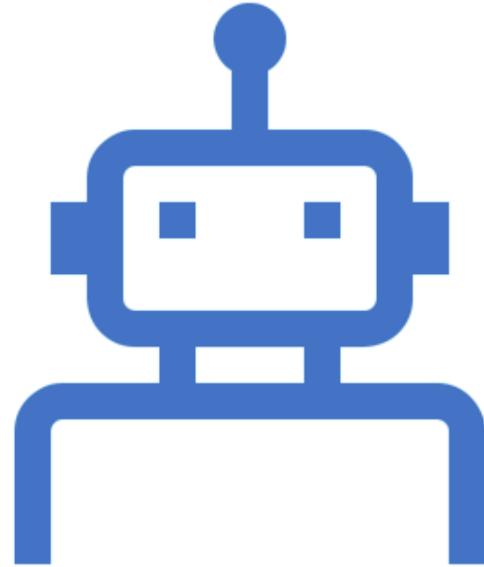
console.log(add(4, 5))
//9
```

| REQUIRE() | PATHNAMES

'..../components'

Up a directory from the module,
then use a folder called 'components'

```
import React from 'react';
```



You will not use require()

ES6 |
MODULES |

STRICTNESS

No need for ‘use strict’ in your modules (it’s the default)



Everything inside an ES6 module is private by default, and runs in strict mode (there's no need for `'use strict'`). Public variables, functions and classes are exposed using `export`.

CONDITIONAL IMPORT

You **cannot...**

```
if (Math.random()) {
    import 'foo'; // SyntaxError
}

// You can't even nest `import` and `export`
// inside a simple block:
{
    import 'foo'; // SyntaxError
}
```

**SMALLER
MODULES =>
PERFORMANCE
GAINS**

Smaller is better with
any download..

Optimize your code

Strict mode – all the
time with modules!

```
import { sum } from './lib.js';

console.log( sum(1,2,3,4) ); // 10
```

ES6 |
SINGLE EXPORT |

DYNAMIC IMPORT()

```
<script type="module">
(async () => {
  const moduleSpecifier = './lib.mjs';
  const {repeat, shout} = await import(moduleSpecifier);
  repeat('hello');
  // → 'hello hello'
  shout('Dynamic import in action');
  // → 'DYNAMIC IMPORT IN ACTION!'
})();
</script>
```

When you do not want to load a module upfront, but on-demand

You can load your modules when you need to use them.

Performance improvement!

.MJS EXTENSION

The Chrome V8 team recommends it's use:

Still, we recommend using the `.mjs` extension for modules, for two reasons:

1. During development, the `.mjs` extension makes it crystal clear to you and anyone else looking at your project that the file is a module as opposed to a classic script. (It's not always possible to tell just by looking at the code.) As mentioned before, modules are treated differently than classic scripts, so the difference is hugely important!
2. It ensures that your file is parsed as a module by runtimes such as [Node.js](#) and [d8](#), and build tools such as [Babel](#). While these environments and tools each have proprietary ways via configuration to interpret files with other extensions as modules, the `.mjs` extension is the cross-compatible way to ensure that files are treated as modules.

```
// Supported:  
import {shout} from './lib.mjs';  
import {shout} from '../lib.mjs';  
import {shout} from '/modules/lib.mjs';  
import {shout} from 'https://simple.example/modules/lib.mjs';
```

```
// Not supported (yet):
```

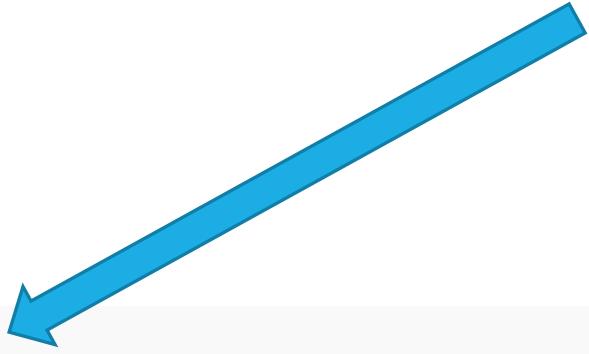
```
import {shout} from 'jquery';  
import {shout} from 'lib.mjs';  
import {shout} from 'modules/lib.mjs';
```

MODULE SPECIFIERS MUST BE| FULL URL'S
OR RELATIVE URL'S WITH | /, ./, OR ../

MODULES

YOUR HTML SCRIPT TAG

```
// html.js
export function tag (tag, text) {
  const el = document.createElement(tag)
  el.textContent = text
  return el
}
```



```
<script type="module">
  import { tag } from './html.js'
  const h1 = tag('h1', 'Hello Modules!')
  document.body.appendChild(h1)
</script>
```

MODULES: START USING THEM TODAY

You do not need to wait.
Works everywhere.
(Nobody cares about IE anymore...)

