Making Websites for Beginners

Clarissa Littler

What we'll cover

- The basic technology that goes into a webpage
- Simple examples of how to use HTML and CSS and maybe a little JavaScript
- Resources to continue your learning

What we'll cover

- The basic technology that goes into a webpage
- Simple examples of how to use HTML and CSS and maybe a little JavaScript
- Resources to continue your learning

What we'll cover

- The basic technology that goes into a webpage
- Simple examples of how to use HTML and CSS and maybe a little JavaScript
- Resources to continue your learning

What we'll cover

- The basic technology that goes into a webpage
- Simple examples of how to use HTML and CSS and maybe a little JavaScript
- Resources to continue your learning

What we won't cover

- How to build the back-end of a site
- How to program in JavaScript in general
  - Though there are free supplements for that
- A majority of CSS and HTML

What we won't cover

- How to build the back-end of a site
- How to program in JavaScript in general
    - Though there are free supplements for that
- A majority of CSS and HTML

What we won't cover

- How to build the back-end of a site
- How to program in JavaScript in general
    - Though there are free supplements for that
- A majority of CSS and HTML

What we won't cover

- How to build the back-end of a site
- How to program in JavaScript in general
  - Though there are free supplements for that
- A majority of CSS and HTML

Client and server

Two pieces that talk to each other to make a site

Server

- Sends data to the browser
- Saves information for long term use
- Receives requests from the client

Client

- Receives data from the server
- Renders server data into a usable page
- Handles the user interface

Client and server

Two pieces that talk to each other to make a site

Server

- Sends data to the browser
- Saves information for long term use
- Receives requests from the client

Client

- Receives data from the server
- Renders server data into a usable page
- Handles the user interface

Client and server

Two pieces that talk to each other to make a site

Server

- Sends data to the browser
- Saves information for long term use
- Receives requests from the client

Client

- Receives data from the server
- Renders server data into a usable page
- Handles the user interface

Client and server

Two pieces that talk to each other to make a site

Server

- Sends data to the browser
- Saves information for long term use
- Receives requests from the client

Client

- Receives data from the server
- Renders server data into a usable page
- Handles the user interface

Client and server

Two pieces that talk to each other to make a site

Server

- Sends data to the browser
- Saves information for long term use
- Receives requests from the client

Client

- Receives data from the server
- Renders server data into a usable page
- Handles the user interface

Client and server

Two pieces that talk to each other to make a site

Server

- Sends data to the browser
- Saves information for long term use
- Receives requests from the client

Client

- Receives data from the server
- Renders server data into a usable page
- Handles the user interface

Client and server

Two pieces that talk to each other to make a site

Server

- Sends data to the browser
- Saves information for long term use
- Receives requests from the client

Client

- Receives data from the server
- Renders server data into a usable page
- Handles the user interface

How do you share a site?

- You can load a site locally in your browser
- To share a site you need a server to host
- Free hosting option: `neocities.org`

How do you share a site?

- You can load a site locally in your browser
- To share a site you need a server to host
- Free hosting option: `neocities.org`

How do you share a site?

- You can load a site locally in your browser
- To share a site you need a server to host
- Free hosting option: `neocities.org`

The three pieces of a web page

- HTML
- CSS
- JavaScript

The three pieces of a web page

- HTML
- CSS
- JavaScript

The three pieces of a web page

- HTML
- CSS
- JavaScript

HTML

What does HTML do?

HTML describes the content of the page, but not how it looks

HTML

What does HTML do?

HTML describes the content of the page, but not how it looks

CSS

What does CSS do?

CSS describes how a page looks, but not its content

CSS

What does CSS do?

CSS describes how a page looks, but not its content

JavaScript

What does JavaScript do?

The dynamics and the user interface of the page

What is HTML?

HyperText Markup Language

- HyperText
- Markup

What is HTML?

HyperText Markup Language

- HyperText
- Markup

Tags and Elements

```
<body>
 <h1>This is a heading</h1>
 <p>
     This is a paragraph of text,
     where some of the text is <b>bold</b>, and
     after this paragraph, there will be a numbered list
 </p>

 <ol>
   <li>lists are made of "list items"</li>
   <li>like these</li>
 </ol>
</body>
```

Tags and Elements

```
<body>
 <h1>This is a heading</h1>
 <p>
     This is a paragraph of text,
     where some of the text is <b>bold</b>, and
     after this paragraph, there will be a numbered list
 </p>

 <ol>
   <li>lists are made of "list items"</li>
   <li>like these</li>
 </ol>
</body>
```

Tags and Elements

```
<body>
 <h1>This is a heading</h1>
 <p>
     This is a paragraph of text,
     where some of the text is <b>bold</b>, and
     after this paragraph, there will be a numbered list
 </p>


 <ol>
   <li>lists are made of "list items"</li>
   <li>like these</li>
 </ol>
</body>
```

Tags and Elements

```
<body>
 <h1>This is a heading</h1>
 <p>
     This is a paragraph of text,
     where some of the text is <b>bold</b>, and
     after this paragraph, there will be a numbered list
 </p>

 <ol>
   <li>lists are made of "list items"</li>
   <li>like these</li>
 </ol>
</body>
```

Tags and Elements

```
<body>
 <h1>This is a heading</h1>
 <p>
     This is a paragraph of text,
     where some of the text is <b>bold</b>, and
     after this paragraph, there will be a numbered list
 </p>

 <ol>
   <li>lists are made of "list items"</li>
   <li>like these</li>
 </ol>
</body>
```

Tags and Elements

```
<body>
 <h1>This is a heading</h1>
 <p>
     This is a paragraph of text,
     where some of the text is <b>bold</b>, and
     after this paragraph, there will be a numbered list
 </p>

 <ol>
   <li>lists are made of "list items"</li>
   <li>like these</li>
 </ol>
</body>
```

Tags and Elements

```
<body>
 <h1>This is a heading</h1>
 <p>
     This is a paragraph of text,
     where some of the text is <b>bold</b>, and
     after this paragraph, there will be a numbered list
 </p>

 <ol>
   <li>lists are made of "list items"</li>
   <li>like these</li>
 </ol>
</body>
```

Tags and Elements

```
<body>
 <h1>This is a heading</h1>
 <p>
     This is a paragraph of text,
     where some of the text is <b>bold</b>, and
     after this paragraph, there will be a numbered list
 </p>

 <ol>
   <li>lists are made of "list items"</li>
   <li>like these</li>
 </ol>
</body>
```

Whence closing tags

```
<body>
  <ol>
    <li>This is a list
    <li>but
    <li>there's ambiguity here

  <ol>
   <li> where does this part go?
   <li> is it a sublist or a second list?
```

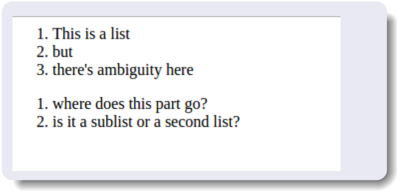
Whence closing tags

```
<body>
  <ol>
    <li>This is a list</li>
    <li>but</li>
    <li>there's ambiguity here</li>
  </ol>
  <ol>
   <li> where does this part go?</li>
   <li> is it a sublist or a second list?</li>
  </ol>
```

Whence closing tags

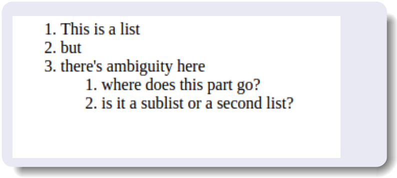
```
<body>
  <ol>
    <li>This is a list</li>
    <li>but</li>
    <li>there's ambiguity here

  <ol>
   <li> where does this part go?</li>
   <li> is it a sublist or a second list?</li>
  </ol>
  </li>
  </ol>
```

Whence closing tags



1. This is a list
2. but
3. there's ambiguity here

1. where does this part go?
2. is it a sublist or a second list?



1. This is a list
2. but
3. there's ambiguity here
    1. where does this part go?
    2. is it a sublist or a second list?

The basic template

```
<!doctype html>
<html>
  <head>
    ...
  </head>
  <body>
    ...
  </body>
</html>
```

The basic template

```
<!doctype html>
<html>
  <head>
    ...
  </head>
  <body>
    ...
  </body>
</html>
```

The basic template

```
<!doctype html>
<html>
  <head>
    ...
  </head>
  <body>
    ...
  </body>
</html>
```

The basic template

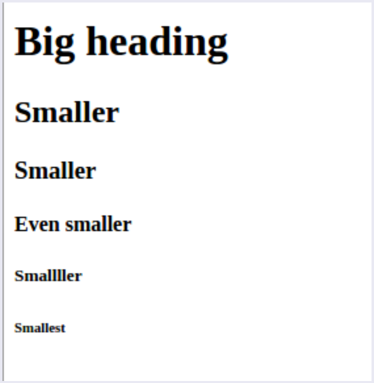
```
<!doctype html>
<html>
  <head>
    ...
  </head>
  <body>
    ...
  </body>
</html>
```

The basic template

```
<!doctype html>
<html>
  <head>
    ...
  </head>
  <body>
    ...
  </body>
</html>
```

Headings

```
<!doctype html>
<html>
  <body>
    <h1>Big heading</h1>
    <h2>Smaller</h2>
    <h3>Smaller</h3>
    <h4>Even smaller</h4>
    <h5>Smallller</h5>
    <h6>Smallest</h6>
  </body>
</html>
```



Big heading

Smaller

Smaller

Even smaller

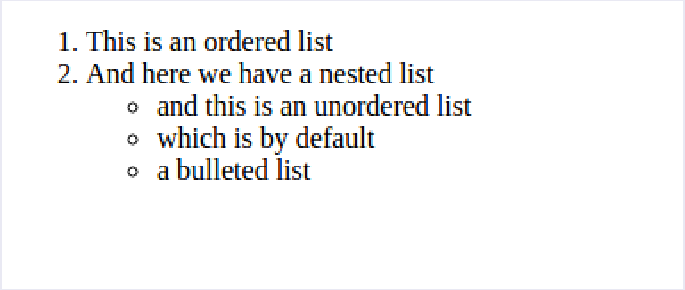
Smallller

Smallest

Lists

```
<!doctype html>
<html>
  <body>
    <ol>
      <li>This is an ordered list</li>
      <li>And here we have a nested list
        <ul>
          <li>and this is an unordered list</li>
          <li>which is by default</li>
          <li>a bulleted list</li>
        </ul>
      </li>
    </ol>
  </body>
</html>
```

Lists



1. This is an ordered list
2. And here we have a nested list
    - and this is an unordered list
    - which is by default
    - a bulleted list

Exercise 1

Let's try making a simple web page ourselves!

- Open notepad++
- Type along the instructions
- Save the file in the F drive (end the file in .html)
- Right click and open in the browser

Exercise 1

Let's try making a simple web page ourselves!

- Open notepad++
- Type along the instructions
- Save the file in the F drive (end the file in .html)
- Right click and open in the browser

```
<!doctype html>
```

Exercise 1

Let's try making a simple web page ourselves!

- Open notepad++
- Type along the instructions
- Save the file in the F drive (end the file in .html)
- Right click and open in the browser

```
<!doctype html>
<html>
```

Exercise 1

Let's try making a simple web page ourselves!

- Open notepad++
- Type along the instructions
- Save the file in the F drive (end the file in .html)
- Right click and open in the browser

```
<!doctype html>
<html>
 <body>
```

Exercise 1

Let's try making a simple web page ourselves!

- Open notepad++
- Type along the instructions
- Save the file in the F drive (end the file in .html)
- Right click and open in the browser

```
<!doctype html>
<html>
 <body>
  <h1>This is our heading</h1>
```

Exercise 1

Let's try making a simple web page ourselves!

- Open notepad++
- Type along the instructions
- Save the file in the F drive (end the file in .html)
- Right click and open in the browser

```
<!doctype html>
<html>
 <body>
  <h1>This is our heading</h1>
  <p>Here is our text.</p>
```

Exercise 1

Let's try making a simple web page ourselves!

- Open notepad++
- Type along the instructions
- Save the file in the F drive (end the file in .html)
- Right click and open in the browser

```
<!doctype html>
<html>
 <body>
  <h1>This is our heading</h1>
  <p>Here is our text.</p>
  <p>Here's more <b>text</b></p>
```

Exercise 1

Let's try making a simple web page ourselves!

- Open notepad++
- Type along the instructions
- Save the file in the F drive (end the file in .html)
- Right click and open in the browser

```
<!doctype html>
<html>
 <body>
  <h1>This is our heading</h1>
  <p>Here is our text.</p>
  <p>Here's more <b>text</b></p>
 </body>
```

Exercise 1

Let's try making a simple web page ourselves!

- Open notepad++
- Type along the instructions
- Save the file in the F drive (end the file in .html)
- Right click and open in the browser

```
<!doctype html>
<html>
 <body>
  <h1>This is our heading</h1>
  <p>Here is our text.</p>
  <p>Here's more <b>text</b></p>
 </body>
</html>
```

Exercise 2

Try making your own simple page using

- `<p>`
- `<h1>`
- `<ol>`
- `<ul>`
- `<li>`

tags, following the process of the last example

Anchors and Attributes

```
<a href="https://multcolib.org">This is a link</a>
```

Exercise 3

Create your own page that uses at least two links and test them to ensure they work

Cascading Style Sheets

What is CSS?

Cascading style sheets control the appearance of elements

CSS Entries

```
selector {
    property: value;
    property: value;
    property: value;
}
```

CSS Entries

```
selector {
    property: value;
    property: value;
    property: value;
}
```

CSS Entries

```
selector {
    property: value;
    property: value;
    property: value;
}
```

Adding CSS to a page

Style tags

```
<!doctype html>
<html>
  <head>
    <style>
      ...
    </style>
  </head>
  <body>
    ...
  </body>
</html>
```

Adding CSS to a page

Style tags

```
<!doctype html>
<html>
  <head>
    <style>
      ...
    </style>
  </head>
  <body>
    ...
  </body>
</html>
```

Selecting elements by ID

```
<!doctype html>

<html>
  <head>
    <style>
      #para {
         color: blue;
      }
    </style>
  </head>
  <body>
    <p id="para">This is the text within our paragraph.</p>
  </body>
</html>
```

Selecting elements by ID

```
<!doctype html>

<html>
  <head>
    <style>
      #para {
         color: blue;
      }
    </style>
  </head>
  <body>
    <p id="para">This is the text within our paragraph.</p>
  </body>
</html>
```

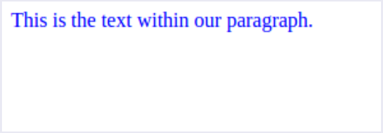
Selecting elements by ID

```
<!doctype html>

<html>
  <head>
    <style>
      #para {
         color: blue;
      }
    </style>
  </head>
  <body>
    <p id="para">This is the text within our paragraph.</p>
  </body>
</html>
```

Selecting elements by ID

```
<!doctype html>

<html>
  <head>
    <style>
      #para {
         color: blue;
      }
    </style>
  </head>
  <body>
    <p id="para">This is the text within our paragraph.</p>
  </body>
</html>
```

Selecting elements by ID


This is the text within our paragraph.

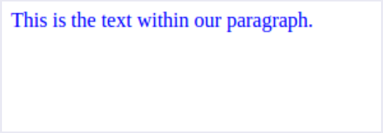
Exercise 4

Let's use CSS

- Open a new file in the text editor
- Copy the template on this slide
- Fill in the style element within the <head> tags
- Turn the middle heading green

```
<!doctype html>
<html>
  <head>
    <style>
      fill this in
    </style>
  </head>
  <body>
    <h1 id="heading1">First</h1>
    <h2 id="heading2">Second</h2>
    <h3 id="heading3">Third</h3>
  </body>
</html>
```

Selecting elements by ID


This is the text within our paragraph.

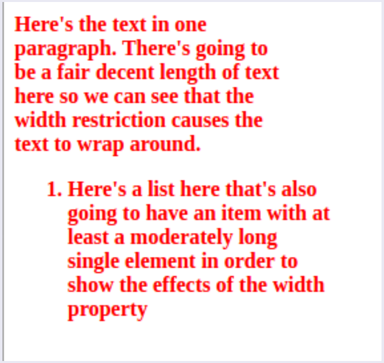
Selecting elements by class

```
.ourClass {
    color: red;
    width: 200px;
    font-weight: bold;
}
```

Selecting elements by class

```
<p class="ourClass">Here's the
text in one paragraph.
There's going to be a fair
decent length of text here so we
can see that the width
restriction causes the text to wrap around.</p>

<ol class="ourClass">
  <li>Here's a list here that's
  also going to have an item
  with at least a moderately long
  single element
  in order to show the
  effects of the width property</li>
</ol>
```

Selecting elements by class



Here's the text in one paragraph. There's going to be a fair decent length of text here so we can see that the width restriction causes the text to wrap around.

1. Here's a list here that's also going to have an item with at least a moderately long single element in order to show the effects of the width property

Exercise 5

Open a new file, follow the template on this slide, then add in CSS declarations to make both paragraphs have `width: 200px` and the first paragraph have a color of `blue`

```
<!doctype html>
<html>
  <head>
  </head>
  <body>
    <p class="theClass" id="firstPara">
    This is a paragraph that has some text in it
    and, y’know, stuff and things</p>
    <p class="theClass" id="sndPara">
      This is the second paragraph by gum</p>
  </body>
</html>
```

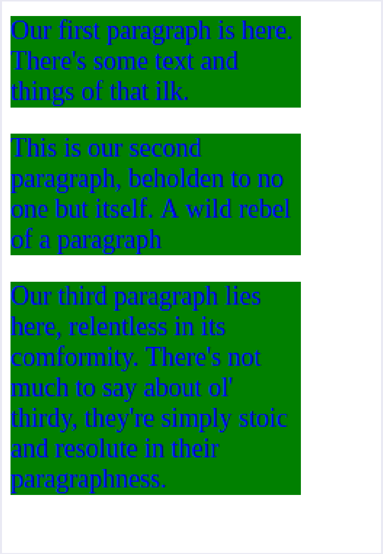
Selecting elements by type

```
p {
    font-size: large;
    background-color: green;
    color: blue;
    width: 200px;
}
```

Selecting elements by type

```
<p>Our first paragraph is here.
  There's some text and things of that ilk.</p>
<p>This is our second paragraph,
  beholden to no one but itself.
  A wild rebel of a paragraph</p>
<p>Our third paragraph lies here,
  relentless in its comformity.
  There's not much to say about ol' thirdy,
  they're simply stoic and
  resolute in their paragraphness.</p>
```

Selecting elements by type


Our first paragraph is here. There's some text and things of that ilk.

This is our second paragraph, beholden to no one but itself. A wild rebel of a paragraph

Our third paragraph lies here, relentless in its comformity. There's not much to say about ol' thirdy, they're simply stoic and resolute in their paragraphness.

Specificity

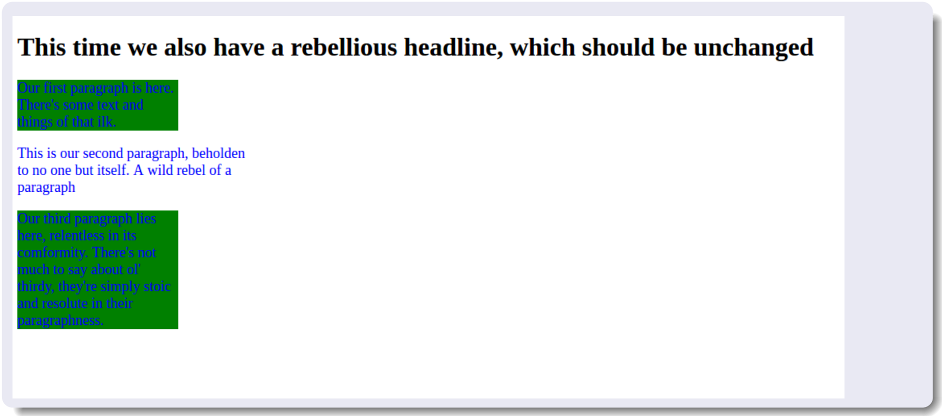
combining type and class

```
p {
    font-size: large;
    background-color: green;
    color: blue;
    width: 200px;
}
p.rebel {
    width: 300px;
    background-color: white;
}
```

Specificity

```
<h1 class="rebel">This time we also have a rebellious heading
which should be unchanged</h1>

<p>Our first paragraph is here.
  There’s some text and things of that ilk.</p>
<p class="rebel">This is our second paragraph,
  beholden to no one but itself.
  A wild rebel of a paragraph</p>
<p>Our third paragraph lies here,
  relentless in its comformity.
  There’s not much to say about ol’ thirdy,
  they’re simply stoic and resolute
  in their paragraphness.</p>
</div>
```

Specificity



This time we also have a rebellious headline, which should be unchanged

Our first paragraph is here. There's some text and things of that ilk.

This is our second paragraph, beholden to no one but itself. A wild rebel of a paragraph

Our third paragraph lies here, relentless in its comformity. There's not much to say about ol' thirdy, they're simply stoic and resolute in their paragraphness.

Div and span

- Div and span are used to group related elements together
- *But they don't have an appearance themselves*

Specificity

choosing children of an element

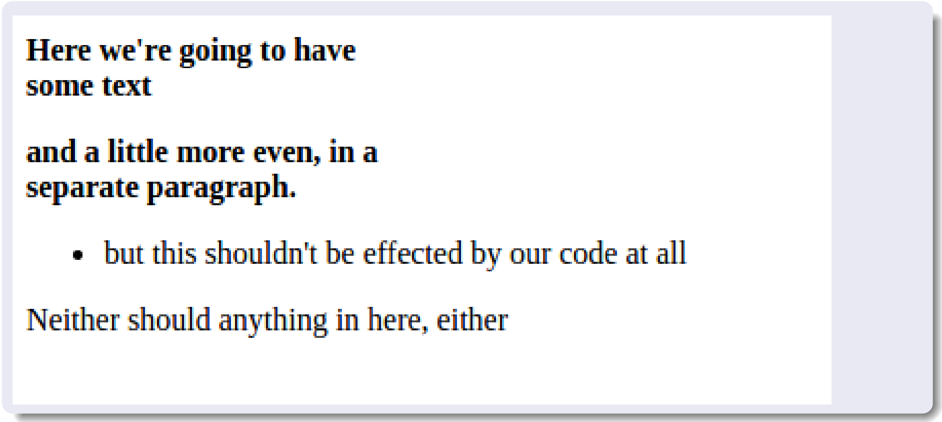
```
#divvy p{
  width: 200px;
  font-weight: bold;
}
```

Specificity

choosing children of an element

```
<div id="divvy">
  <p> Here we're going to have some text </p>
  <p> and a little more even, in a separate paragraph. </p>

  <ul>
    <li>but this shouldn't be effected by our code at all</li
  </ul>
</div>
<p>Neither should anything in here, either</p>
```

Specificity


Here we're going to have some text

and a little more even, in a separate paragraph.

- but this shouldn't be effected by our code at all

Neither should anything in here, either

Exercise 6

Using the following skeleton, found in exer6.html, add CSS declarations so that the first paragraph has *blue* text, the second paragraph has *red* text, and the third paragraph has *green* text.

```
<body>
  <p>our first paragraph</p>
  <div>
    <p>our second paragraph</p>
    <div>
      <p>our third paragraph </p>
    </div>
</body>
```

What is the Document Object Model?

The Document Object Model (DOM) gives you the ability to write code that changes the web page dynamically

What is the Document Object Model?

The Document Object Model (DOM) gives you the ability to write code that changes the web page dynamically

What is the Document Object Model?

The Document Object Model (DOM) gives you the ability to write code that changes the web page dynamically

What is the Document Object Model?

The Document Object Model (DOM) gives you the ability to write code that changes the web page dynamically

What does the DOM do?

- Change CSS classes
- Create and remove HTML elements
- Respond to user interface events

What does the DOM do?

- Change CSS classes
- Create and remove HTML elements
- Respond to user interface events

What does the DOM do?

- Change CSS classes
- Create and remove HTML elements
- Respond to user interface events

What does the DOM do?

- Change CSS classes
- Create and remove HTML elements
- Respond to user interface events

What is JavaScript?

JavaScript is a programming language that runs in the browser

What is JavaScript?

JavaScript is a programming language that runs in the browser

What are programming languages?

A programming language is. . .

- a formal language with rules and grammar
- that has meaning as computation
- and can be used to talk to a computer

What are programming languages?

A programming language is. . .

- a formal language with rules and grammar
- that has meaning as computation
- and can be used to talk to a computer

What are programming languages?

A programming language is. . .

- a formal language with rules and grammar
- that has meaning as computation
- and can be used to talk to a computer

Script tag

```
<!doctype html>

<html>
  <head>
    <script>
      ...
    </script>
  </head>
  <body>
    ...
  </body>
</html>
```

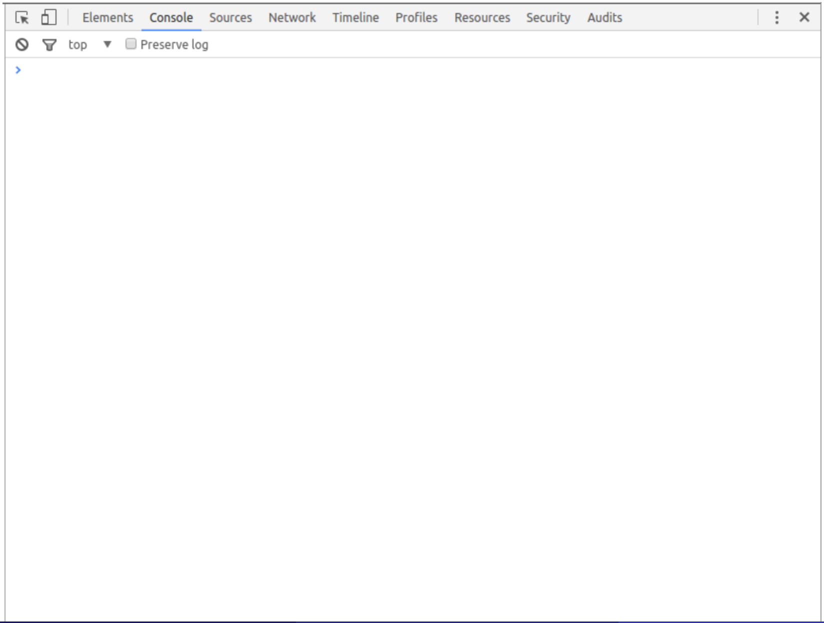
Script tag

```
<!doctype html>

<html>
  <head>
    <script>
      ...
    </script>
  </head>
  <body>
    ...
  </body>
</html>
```

Script tag

```
<!doctype html>

<html>
  <head>
    <script>
      ...
    </script>
  </head>
  <body>
    ...
  </body>
</html>
```

JavaScript console



Basic programming constructs

Data

- Numbers
- Text
- Lists
- Dictionaries

Actions

- Arithmetic
- Creating and using storage
- Performing actions multiple times
- Making choices about what to do
- Naming routine tasks to easily perform them again

Basic programming constructs

Data

- Numbers
- Text
- Lists
- Dictionaries

Actions

- Arithmetic
- Creating and using storage
- Performing actions multiple times
- Making choices about what to do
- Naming routine tasks to easily perform them again

Basic programming constructs

Data

- Numbers
- Text
- Lists
- Dictionaries

Actions

- Arithmetic
- Creating and using storage
- Performing actions multiple times
- Making choices about what to do
- Naming routine tasks to easily perform them again

Basic programming constructs

Data

- Numbers
- Text
- Lists
- Dictionaries

Actions

- Arithmetic
- Creating and using storage
- Performing actions multiple times
- Making choices about what to do
- Naming routine tasks to easily perform them again

Basic programming constructs

Data

- Numbers
- Text
- Lists
- Dictionaries

Actions

- Arithmetic
- Creating and using storage
- Performing actions multiple times
- Making choices about what to do
- Naming routine tasks to easily perform them again

Basic programming constructs

Data

- Numbers
- Text
- Lists
- Dictionaries

Actions

- Arithmetic
- Creating and using storage
- Performing actions multiple times
- Making choices about what to do
- Naming routine tasks to easily perform them again

Basic programming constructs

Data

- Numbers
- Text
- Lists
- Dictionaries

Actions

- Arithmetic
- Creating and using storage
- Performing actions multiple times
- Making choices about what to do
- Naming routine tasks to easily perform them again

Basic programming constructs

Data

- Numbers
- Text
- Lists
- Dictionaries

Actions

- Arithmetic
- Creating and using storage
- Performing actions multiple times
- Making choices about what to do
- Naming routine tasks to easily perform them again

Basic programming constructs

Data

- Numbers
- Text
- Lists
- Dictionaries

Actions

- Arithmetic
- Creating and using storage
- Performing actions multiple times
- Making choices about what to do
- Naming routine tasks to easily perform them again

Basic programming constructs

Data

- Numbers
- Text
- Lists
- Dictionaries

Actions

- Arithmetic
- Creating and using storage
- Performing actions multiple times
- Making choices about what to do
- Naming routine tasks to easily perform them again

Basic programming constructs

Data

- Numbers
- Text
- Lists
- Dictionaries

Actions

- Arithmetic
- Creating and using storage
- Performing actions multiple times
- Making choices about what to do
- Naming routine tasks to easily perform them again

Basic programming constructs

Data

- Numbers
- Text
- Lists
- Dictionaries

Actions

- Arithmetic
- Creating and using storage
- Performing actions multiple times
- Making choices about what to do
- Naming routine tasks to easily perform them again

Strings

Strings are text-as-data, useful for:

- error messages
- writing output

```
"this is a string"
'this is also a string'
"even this 'is a string'"
```

Strings

Strings are text-as-data, useful for:

- error messages
- writing output

```
"this is a string"
'this is also a string'
"even this 'is a string'"
```

Strings

Strings are text-as-data, useful for:

- error messages
- writing output

```
"this is a string"
'this is also a string'
"even this 'is a string'"
```

Strings

Strings are text-as-data, useful for:

- error messages
- writing output

```
"this is a string"
'this is also a string'
"even this 'is a string'"
```

Strings

Strings are text-as-data, useful for:

- error messages
- writing output

```
"this is a string"
'this is also a string'
"even this 'is a string'"
```

Strings

Strings are text-as-data, useful for:

- error messages
- writing output

```
"this is a string"
'this is also a string'
"even this 'is a string'"
```

Strings exercise

Type the following into the console:

- `"hi there everybody"`
- `"it’s such a ’nice’ day"`
- `"I’m in this class" + " and I’m typing"`

Variables and Storage

Variables...

- are names given to data
- are storage containers
- can change in value

Type along

```
var thisVariable="a string"
thisVariable
thisVariable = 10
thisVariable
```

Variables and Storage

Variables...

- are names given to data
- are storage containers
- can change in value

Type along

```
var thisVariable="a string"
thisVariable
thisVariable = 10
thisVariable
```

Variables and Storage

Variables...

- are names given to data
- are storage containers
- can change in value

Type along

```
var thisVariable="a string"
thisVariable
thisVariable = 10
thisVariable
```

Variables and Storage

Variables...

- are names given to data
- are storage containers
- can change in value

Type along

```
var thisVariable="a string"
thisVariable
thisVariable = 10
thisVariable
```

Variables and Storage

Variables...

- are names given to data
- are storage containers
- can change in value

Type along

```
var thisVariable="a string"
thisVariable
thisVariable = 10
thisVariable
```

Variables and Storage

Variables...

- are names given to data
- are storage containers
- can change in value

Type along

```
var thisVariable="a string"
thisVariable
thisVariable = 10
thisVariable
```

Variables and Storage

Variables...

- are names given to data
- are storage containers
- can change in value

Type along

```
var thisVariable="a string"
thisVariable
thisVariable = 10
thisVariable
```

Variables and Storage

Variables...

- are names given to data
- are storage containers
- can change in value

Type along

```
var thisVariable="a string"
thisVariable
thisVariable = 10
thisVariable
```

Lists and Arrays

In JavaScript, the data type for lists are called arrays

Arrays

- have a beginning and end
- are in order
- can be accessed by index

Type along

```
var myArr = [1,2,3]
myArr
myArr[0]
myArr[1]
myArr[2]
myArr[0] = 20
myArr[0]
```

Lists and Arrays

In JavaScript, the data type for lists are called arrays

Arrays

- have a beginning and end
- are in order
- can be accessed by index

Type along

```
var myArr = [1,2,3]
myArr
myArr[0]
myArr[1]
myArr[2]
myArr[0] = 20
myArr[0]
```

Lists and Arrays

In JavaScript, the data type for lists are called arrays

Arrays

- have a beginning and end
- are in order
- can be accessed by index

Type along

```
var myArr = [1,2,3]
myArr
myArr[0]
myArr[1]
myArr[2]
myArr[0] = 20
myArr[0]
```

Lists and Arrays

In JavaScript, the data type for lists are called arrays

Arrays

- have a beginning and end
- are in order
- can be accessed by index

Type along

```
var myArr = [1,2,3]
myArr
myArr[0]
myArr[1]
myArr[2]
myArr[0] = 20
myArr[0]
```

Lists and Arrays

In JavaScript, the data type for lists are called arrays

Arrays

- have a beginning and end
- are in order
- can be accessed by index

Type along

```
var myArr = [1,2,3]
myArr
myArr[0]
myArr[1]
myArr[2]
myArr[0] = 20
myArr[0]
```

Lists and Arrays

In JavaScript, the data type for lists are called arrays

Arrays

- have a beginning and end
- are in order
- can be accessed by index

Type along

```
var myArr = [1,2,3]
myArr
myArr[0]
myArr[1]
myArr[2]
myArr[0] = 20
myArr[0]
```

Lists and Arrays

In JavaScript, the data type for lists are called arrays

Arrays

- have a beginning and end
- are in order
- can be accessed by index

Type along

```
var myArr = [1,2,3]
myArr
myArr[0]
myArr[1]
myArr[2]
myArr[0] = 20
myArr[0]
```

Lists and Arrays

In JavaScript, the data type for lists are called arrays

Arrays

- have a beginning and end
- are in order
- can be accessed by index

Type along

```
var myArr = [1,2,3]
myArr
myArr[0]
myArr[1]
myArr[2]
myArr[0] = 20
myArr[0]
```

Lists and Arrays

In JavaScript, the data type for lists are called arrays

Arrays

- have a beginning and end
- are in order
- can be accessed by index

Type along

```
var myArr = [1,2,3]
myArr
myArr[0]
myArr[1]
myArr[2]
myArr[0] = 20
myArr[0]
```

Lists and Arrays

In JavaScript, the data type for lists are called arrays

Arrays

- have a beginning and end
- are in order
- can be accessed by index

Type along

```
var myArr = [1,2,3]
myArr
myArr[0]
myArr[1]
myArr[2]
myArr[0] = 20
myArr[0]
```

Lists and Arrays

In JavaScript, the data type for lists are called arrays

Arrays

- have a beginning and end
- are in order
- can be accessed by index

Type along

```
var myArr = [1,2,3]
myArr
myArr[0]
myArr[1]
myArr[2]
myArr[0] = 20
myArr[0]
```

Objects and Dictionaries

Objects...

- are like dictionaries
- associate names and data
- are used to collect information

Type along

```
var obj = {names : "chicken",
           petSpecies : "dog",
           age : 10}
obj.names
obj.petSpecies
obj.age = 10
obj.age
```

Objects and Dictionaries

Objects...

- are like dictionaries
- associate names and data
- are used to collect information

Type along

```
var obj = {names : "chicken",
           petSpecies : "dog",
           age : 10}
obj.names
obj.petSpecies
obj.age = 10
obj.age
```

Objects and Dictionaries

Objects...

- are like dictionaries
- associate names and data
- are used to collect information

Type along

```
var obj = {names : "chicken",
           petSpecies : "dog",
           age : 10}
obj.names
obj.petSpecies
obj.age = 10
obj.age
```

Objects and Dictionaries

Objects...

- are like dictionaries
- associate names and data
- are used to collect information

Type along

```
var obj = {names : "chicken",
           petSpecies : "dog",
           age : 10}
obj.names
obj.petSpecies
obj.age = 10
obj.age
```

Objects and Dictionaries

Objects...

- are like dictionaries
- associate names and data
- are used to collect information

Type along

```
var obj = {names : "chicken",
           petSpecies : "dog",
           age : 10}
obj.names
obj.petSpecies
obj.age = 10
obj.age
```

Objects and Dictionaries

Objects...

- are like dictionaries
- associate names and data
- are used to collect information

Type along

```
var obj = {names : "chicken",
           petSpecies : "dog",
           age : 10}
obj.names
obj.petSpecies
obj.age = 10
obj.age
```

Objects and Dictionaries

Objects...

- are like dictionaries
- associate names and data
- are used to collect information

Type along

```
var obj = {names : "chicken",
           petSpecies : "dog",
           age : 10}
obj.names
obj.petSpecies
obj.age = 10
obj.age
```

Objects and Dictionaries

Objects...

- are like dictionaries
- associate names and data
- are used to collect information

Type along

```
var obj = {names : "chicken",
           petSpecies : "dog",
           age : 10}
obj.names
obj.petSpecies
obj.age = 10
obj.age
```

Objects and Dictionaries

Objects...

- are like dictionaries
- associate names and data
- are used to collect information

Type along

```
var obj = {names : "chicken",
           petSpecies : "dog",
           age : 10}
obj.names
obj.petSpecies
obj.age = 10
obj.age
```

Functions

Functions...

- are code that can be used again and again
- are data that can be assigned to variables
- use the keyword `return` to give back a value
- are made up of arguments and a body

```
function () {
//this function doesn't have a name
   var x = 10;
   return x + x;
}

function thisFun (x,y) {
//but this function does
   return (x + y);
}
```

Functions

Functions...

- are code that can be used again and again
- are data that can be assigned to variables
- use the keyword `return` to give back a value
- are made up of arguments and a body

```
function () {
//this function doesn't have a name
   var x = 10;
   return x + x;
}

function thisFun (x,y) {
//but this function does
   return (x + y);
}
```

Functions

Functions...

- are code that can be used again and again
- are data that can be assigned to variables
- use the keyword `return` to give back a value
- are made up of arguments and a body

```
function () {
//this function doesn't have a name
   var x = 10;
   return x + x;
}

function thisFun (x,y) {
//but this function does
   return (x + y);
}
```

Functions

Functions...

- are code that can be used again and again
- are data that can be assigned to variables
- use the keyword `return` to give back a value
- are made up of arguments and a body

```
function () {
//this function doesn't have a name
   var x = 10;
   return x + x;
}

function thisFun (x,y) {
//but this function does
   return (x + y);
}
```

Functions

Functions...

- are code that can be used again and again
- are data that can be assigned to variables
- use the keyword `return` to give back a value
- are made up of arguments and a body

```
function () {
//this function doesn't have a name
   var x = 10;
   return x + x;
}

function thisFun (x,y) {
//but this function does
   return (x + y);
}
```

Functions

Functions...

- are code that can be used again and again
- are data that can be assigned to variables
- use the keyword return to give back a value
- are made up of arguments and a body

```
function () {
//this function doesn't have a name
   var x = 10;
   return x + x;
}

function thisFun (x,y) {
//but this function does
   return (x + y);
}
```

Functions

Functions...

- are code that can be used again and again
- are data that can be assigned to variables
- use the keyword `return` to give back a value
- are made up of arguments and a body

```
function () {
//this function doesn't have a name
   var x = 10;
   return x + x;
}

function thisFun (x,y) {
//but this function does
   return (x + y);
}
```

Functions

Functions...

- are code that can be used again and again
- are data that can be assigned to variables
- use the keyword `return` to give back a value
- are made up of arguments and a body

```
function () {
//this function doesn't have a name
    var x = 10;
    return x + x;
}

function thisFun (x,y) {
//but this function does
    return (x + y);
}
```

Functions

Functions...

- are code that can be used again and again
- are data that can be assigned to variables
- use the keyword `return` to give back a value
- are made up of arguments and a body

```
function () {
//this function doesn't have a name
    var x = 10;
    return x + x;
}

function thisFun (x,y) {
//but this function does
    return (x + y);
}
```

Functions

Functions...

- are code that can be used again and again
- are data that can be assigned to variables
- use the keyword `return` to give back a value
- are made up of arguments and a body

```
function () {
//this function doesn't have a name
   var x = 10;
   return x + x;
}

function thisFun (x,y) {
//but this function does
   return (x + y);
}
```

Functions

Functions...

- are code that can be used again and again
- are data that can be assigned to variables
- use the keyword `return` to give back a value
- are made up of arguments and a body

```
function () {
//this function doesn't have a name
   var x = 10;
   return x + x;
}

function thisFun (x,y) {
//but this function does
   return (x + y);
}
```

Functions

Functions...

- are code that can be used again and again
- are data that can be assigned to variables
- use the keyword `return` to give back a value
- are made up of arguments and a body

```
function () {
//this function doesn't have a name
   var x = 10;
   return x + x;
}

function thisFun (x,y) {
//but this function does
   return (x + y);
}
```

How we'll proceed

From here on, we'll be presenting examples of JavaScript interacting with the DOM and practice more JavaScript from there

How we'll proceed

From here on, we'll be presenting examples of JavaScript interacting with the DOM and practice more JavaScript from there

What is the Document Object Model?

The DOM

The document object model (DOM) is the representation of the web page *as JavaScript objects*

Putting the document in DOM

The document object

`document` is the object that holds most of the important methods for controlling web pages

Our first example

In our first example we'll

- create an HTML element in JavaScript
- create text to put inside the element
- insert the HTML element in the web page

Our first example

In our first example we'll

- create an HTML element in JavaScript
- create text to put inside the element
- insert the HTML element in the web page

Our first example

In our first example we'll

- create an HTML element in JavaScript
- create text to put inside the element
- insert the HTML element in the web page

Our first example

In our first example we'll

- create an HTML element in JavaScript
- create text to put inside the element
- insert the HTML element in the web page

When to load code

```
window.onload = function () {
    ...
};
```

Creating elements

Relevant functions

- `document.createElement`
- `document.createTextNode`
- `element.appendChild`
- `document.body`

Creating elements

```
<!doctype html>

<html>
  <head>
    <script>
      window.onload = function () {
        var elm = document.createElement("p");
        var text = document.createTextNode("this is text");
        elm.appendChild(text);
        document.body.appendChild(elm);
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

Creating elements

```
<!doctype html>

<html>
  <head>
    <script>
      window.onload = function () {
        var elm = document.createElement("p");
        var text = document.createTextNode("this is text");
        elm.appendChild(text);
        document.body.appendChild(elm);
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

Creating elements

```
<!doctype html>

<html>
  <head>
    <script>
      window.onload = function () {
        var elm = document.createElement("p");
        var text = document.createTextNode("this is text");
        elm.appendChild(text);
        document.body.appendChild(elm);
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

Creating elements

```
<!doctype html>


<html>
  <head>
    <script>
      window.onload = function () {
        var elm = document.createElement("p");
        var text = document.createTextNode("this is text");
        elm.appendChild(text);
        document.body.appendChild(elm);
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

Creating elements

```
<!doctype html>

<html>
  <head>
    <script>
      window.onload = function () {
        var elm = document.createElement("p");
        var text = document.createTextNode("this is text");
        elm.appendChild(text);
        document.body.appendChild(elm);
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

Creating elements

```
<!doctype html>

<html>
  <head>
    <script>
      window.onload = function () {
        var elm = document.createElement("p");
        var text = document.createTextNode("this is text");
        elm.appendChild(text);
        document.body.appendChild(elm);
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

Creating elements

```
<!doctype html>

<html>
  <head>
    <script>
      window.onload = function () {
        var elm = document.createElement("p");
        var text = document.createTextNode("this is text");
        elm.appendChild(text);
        document.body.appendChild(elm);
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

Your turn

- Create a new html file
- Leave the body empty
- Create two elements and put them in the body

```
<!doctype html>
<html>
  <head>
    <script>
      window.onload = function () {
        ...
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

Your turn

- Create a new html file
- Leave the body empty
- Create two elements and put them in the body

```
<!doctype html>
<html>
  <head>
    <script>
      window.onload = function () {
        ...
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

Getting existing elements

- `document.getElementById`
- `document.getElementsByTagName`
- `element.firstChild`
- `node.nodeValue`

Getting existing elements

- `document.getElementById`
- `document.getElementsByTagName`
- `element.firstChild`
- `node.nodeValue`

Getting existing elements

- `document.getElementById`
- `document.getElementsByTagName`
- `element.firstChild`
- `node.nodeValue`

Getting existing elements

- `document.getElementById`
- `document.getElementsByTagName`
- `element.firstChild`
- `node.nodeValue`

getElementById

```
<body>
  <ol id="list1">
    <li>This is a list</li>
  </ol>
  <ol id="list2">
    <li>This is our second list</li>
  </ol>
</body>
```

getElementById

```
window.onload = function () {
    var newItem =
      document.createElement("li");
    var newText =
        document
        .createTextNode("item in the second list");
    newItem.appendChild(newText);
    var secondList = document.getElementById("list2");
    secondList.appendChild(newItem);
};
```

Your turn

- Create a new html file
- Follow the template to the right
- Add an element to the list

```
<!doctype html>
<html>
  <head>
    <script>
      window.onload = function () {
        ...
      }
    </script>
  </head>
  <body>
    <ol id="list">
    </ol>
  </body>
</html>
```

Changing CSS properties

Important properties and methods

- `elm.style`
- `elm.classList`
- `elm.classList.add`
- `elm.classList.remove`

Changing CSS properties

Important properties and methods

- `elm.style`
- `elm.classList`
- `elm.classList.add`
- `elm.classList.remove`

Changing CSS properties

Important properties and methods

- `elm.style`
- `elm.classList`
- `elm.classList.add`
- `elm.classList.remove`

Changing CSS properties

Important properties and methods

- `elm.style`
- `elm.classList`
- `elm.classList.add`
- `elm.classList.remove`

Changing CSS properties

Important properties and methods

- `elm.style`
- `elm.classList`
- `elm.classList.add`
- `elm.classList.remove`

Changing CSS properties

```
<!doctype html>
<html>
  <head>
    <script>
      window.onload = function () {
        var h = document.getElementById("heading");
        h.style.color = "red";
      }
    </script>
  </head>
  <body>
    <h1 id="heading">This is a heading!</h1>
  </body>
</html>
```

Changing the CSS class

```
<head>
  <style>
    .reddish {
      color: red;
    }
  </style>
  <script>
    window.onload = function () {
       var h = document.getElementById("heading");
       h.classList.add("reddish");
    };
  </script>
</head>
```

Events

What are events?

Events connect user interfaces to code

- Mouse clicks
- Keys pressed
- Moving your cursor
- Focusing on an element

Events

What are events?

Events connect user interfaces to code

- Mouse clicks
- Keys pressed
- Moving your cursor
- Focusing on an element

Events

What are events?

Events connect user interfaces to code

- Mouse clicks
- Keys pressed
- Moving your cursor
- Focusing on an element

Events

What are events?

Events connect user interfaces to code

- Mouse clicks
- Keys pressed
- Moving your cursor
- Focusing on an element

Events

What are events?

Events connect user interfaces to code

- Mouse clicks
- Keys pressed
- Moving your cursor
- Focusing on an element

Events

Listening for events

- `elem.addEventListener`
- `elem.removeEventListener`

Events

Listening for events

- `elem.addEventListener`
- `elem.removeEventListener`

Events

Listening for events

- `elem.addEventListener`
- `elem.removeEventListener`

Listening to events

```
<head>
  <script>
    window.onload = function () {
       var h = document.getElementById("heading");
       h.addEventListener("mouseover", function () {
          this.style.color = "red";
       });
       h.addEventListener("mouseleave", function () {
          this.style.color = "black";
       });
    };
  </script>
</head>
<body>
  <h1 id="heading">This is our heading!</h1>
</body>
```

Collapsing list

```
<body>
  <div id="content">
    <h3>Our list is below here</h3>
    <ol id="list">
      <li>First item</li>
      <li>Second item</li>
      <li>Third item</li>
      <li>Fourth item</li>
    </ol>
  </div>
</body>
```

Collapsing list

```
window.onload = function () {
    var list = document.getElementById("list");
    var div = document.getElementById("content");
    div.addEventListener("mouseover", function () {
        list.style.display = "block";
    });
    div.addEventListener("mouseleave", function () {
        list.style.display = "none";
    });
};
```

To-do list

```
<body>
  <h1>Welcome to your to-do list</h1>
  <ol id="list">
  </ol>
  <input id="input" type="text"></input>
  <button id="add">Add element</button>
</body>
```

To-do list

```
var inputElement = document.getElementById("input");
var todoList = document.getElementById("list");
var addButton = document.getElementById("add");

addButton.addEventListener("click", function () {
  var itemText = document.createTextNode(inputElement.value);
  var newItem = document.createElement("li");
  newItem.appendChild(itemText);
  todoList.appendChild(newItem);
  inputElement.value = "";
});
```

To-do list

```
inputElement.addEventListener("focus", function () {
   inputElement.style.fontWeight = "bold";
});

inputElement.addEventListener("blur", function () {
   inputElement.style.fontWeight = "normal";
});
```

What we've learned

- What a webpage is
    - HTML
    - CSS
    - JavaScript

What we've learned

- What a webpage is
    - HTML
    - CSS
    - JavaScript

What we've learned

- What a webpage is
  - HTML
  - CSS
  - JavaScript

What we've learned

- What a webpage is
    - HTML
    - CSS
    - JavaScript

What we've learned

- HTML
    - Elements
    - Tags
    - Semantic markup
    - Content, not appearance

What we've learned

- HTML
  - Elements
  - Tags
  - Semantic markup
  - Content, not appearance

What we've learned

- HTML
  - Elements
  - Tags
  - Semantic markup
  - Content, not appearance

What we've learned

- HTML
    - Elements
    - Tags
    - Semantic markup
    - Content, not appearance

What we've learned

- HTML
  - Elements
  - Tags
  - Semantic markup
  - Content, not appearance

What we've learned

- CSS
  - Style, not substance
  - Selectors
  - Classes

What we've learned

- CSS
  - Style, not substance
  - Selectors
  - Classes

What we've learned

- CSS
  - Style, not substance
  - Selectors
  - Classes

What we've learned

- CSS
    - Style, not substance
    - Selectors
    - Classes

What we've learned

- JavaScript
  - A general purpose programming language
  - Can be run by every browser
  - Connects to HTML via Document Object Model

What we've learned

- JavaScript
  - A general purpose programming language
  - Can be run by every browser
  - Connects to HTML via Document Object Model

What we've learned

- JavaScript
    - A general purpose programming language
    - Can be run by every browser
    - Connects to HTML via Document Object Model

What we've learned

- JavaScript
  - A general purpose programming language
  - Can be run by every browser
  - Connects to HTML via Document Object Model

What to learn next

- More HTML tags
- So much more CSS
- Frameworks for styling
    - Bootstrap is a very popular one
- JavaScript programming

What to learn next

- More HTML tags
- So much more CSS
- Frameworks for styling
  - Bootstrap is a very popular one
- JavaScript programming

What to learn next

- More HTML tags
- So much more CSS
- Frameworks for styling
  - Bootstrap is a very popular one
- JavaScript programming

What to learn next

- More HTML tags
- So much more CSS
- Frameworks for styling
    - Bootstrap is a very popular one
- JavaScript programming

What to learn next

- More HTML tags
- So much more CSS
- Frameworks for styling
  - Bootstrap is a very popular one
- JavaScript programming

Thanks for attending!

Thanks for being in this class