HTML: Hypertext Markup Language
Markup language uses tags to define content
Example Paragraph
Brackets a modern text editor designed for markup and code
WWW standards
Standards are published byW3C World Wide Web Connection

Saving work

H drive on computer follows no matter what computer you are on

9/6/18

HTML5 tags
<article> = self-contained independent content. As the article should be independent from the web page
Comments forum posts
<aside> =defines some content article aside from the related content it is placed in
(Side information like in a history textbook)

<audio> = allows audio/sounds to be added one of 3 different file MP3 WAV OGG
<bdi> = bio directional isolation isolates part that could formatted differently from other text outside it.
<canvas> = allows to you to create a graphic image

Also allows using the script to have the picture be drawn.
height and width
<datalist> = creates a list
Opening tag, an id for values contained within the list is provided
In between the closing and opening
<data> = translate content of an element into a common language for computer and human
<detail> = specifies additional details that the user can view or hide in demand
Can be used to create an interactive widget that the user can open and close
<header> = form of interactive tool or visual for website users
There can be multiple headers in a webpage.
<main> =very similar to a title like a <article> <footer> <header>
<mark> = defined a marked highlighted text like a marker
<meter> = defines a scale movement within a known range (a gauge) (progress bar)
<Nav> = used to navigation links linking to external websites or navigation on your website
Mainly used for advertisement for social media
<output>= able to create the solution to a math equation requires 2 <input> tags to know what #s to plug into the equation
<picture>= gives web developers more flexibility in specifying image resources
<progress> = progress bar on a task
<ruby> = annotations, adding words/ notes on top of text
The words will come out small but still readable mostly used for indicating the pronunciation of a character
<rt> = exploration/pronunciation for characters used in the ruby tag
Uses the <rp> and <ruby> tage to work
<rp> = makes parenthesis around the ruby text so it can help show on a website that doesn't support ruby
<section> = defines the sections of a website like chapters to an article
Used in team with the <H1> tags and <p> tags
<source> = allows you to add a source to a audio or video or image
<summary> = puts all details to the summary in a drop down menu
<template> = works when a content is hidden from the client
Visible when there is a button commend
<time> = defines a date or time of something

9-17

CSS cascading style sheets
Last definition or specification in CSS takes over the importance

- Defined on website 3 different ways
- <link> outside style sheet
- Internal style sheet (only affects page that it is on) goes in the head of the website
- Personal HTML style codes

Selector (not official){
    Property1:value;
    Property2 :value;
} rule
*/ (comment)
Bad habit- inline style

- Class attribute
    - Target tag/class/IDs(#id name)(id=attribute)
    - Style in several places then target tag
    - IDs only used once in HTML
    - More specific in to style
    - Over 200 properties that you can change in CSS
    - 55 selectors
    - Best practice is to write CSS files in alphabetical order

1.
2. : checked
    a. Checkboxes
    b. Radio Buttons
    c. The option of selected elements
3. : enabled
    a. Makes it where a text box will be able to be typed in
    b. Should be only used in HTML elements that are supported by the disabled attribute <button>
4. Read-only
    a. Makes a text box a read-only text.
    b. No changing the words
5. Read-write
    a. The opposite of read-only
    b. You are allowed to type in this box
6. attribute=value
    a. Used to select an element that has an attribute value
    b. It will still work if it has something else with it on the same
7. ; first-line
    a. Used as a selector to add style to your first line of text

    b. Used for color, background, height
    c. Used only on back lettered text
8. Not-selector
    a. Selects every element that is specified element or selector
    b. Every element that is not an <P> element
9. : First-of-type
    a. The first of type is a selector that matches every first child to its parent
    b. Same use or nth-of-type
10. nth-child(n)
    a. The nth-child is a selector that matches every element that is the nth child it doesn't
11. Lang-selector
    a. Used to select elements with a lang attribute with the specified value
    b. After specifying the value of the language you can also add the color to the background
12. attribute^=value
    a. Used to match every element whose attribute value begins with a specified value
13. attribute|=value
    a. Used to select elements with the specified attribute with the specified value
    b. Value has to be a whole word, either alone or followed by a hyphen
14. attribute$=value
    a. Used to specify classes and adding variability in what is affected in HTML
15. attribute*=value
    a. Shortens the name of the class\
    b. Deals with the first two letters of the word
16. Active
    a. Hold the click on a link will highlight the link
17. Focus
    a. When clicking on a clickable box or folder it will highlight that box or folder
    b. Doesn't have t be an input
18. : valid
    a. Targets all values in an element indicating as "valid" in the element attribute
    b. Opposite to the invalid selector
19. : visited
    a. Targets all visited links
    b. Default style for a visited link is a purple text color
20. Attribute
    a. In CSS by doing this will allow you to target that has a target selector make anything you want it to do.

21. Attribute=value
    a. By doing this in CSS, it will allow you to make a specific target and do anything with it
22. Root Selector
    a. Selects the element topmost in the document may also target the body element
23. Selection selector
    a. This may target whatever the user selects with tap slides or drags
24. Disabled
    a. a. Matches every disabled element
25. Empty
    a. It will customize any element that has no children
26. Element+element
    a. Used for an element that is placed right after another tag
27. Element1~element2
    a. A selector that selects a; occurrences of element 2 presented by element
28. :: before
    a. Adds content before a selected element
    b. You have to use this content properly yo specifically input what you want after the selected tag
29. :: after
    a. Used to insert content after a selected element
    b. Use the content properly to specify what content you want to insert
30. :in-range
    a. Only works for elements with range limitations, such input elements
31. :only-child
    a. Matches every element that is the only child of the parent
    b. It specifies background color for every <p> element
32. Target
    a. Used to style the active target element
33. Selection
    a. Takes effect only on the parts that are been selected
34. Required
    a. Allows you to be able to select form elements
35. Root
    a. Selects the element topmost in the document may also target the body element
36. :: placeholder
    a. Gives you a box where you type and change the color
    b. Select form, an element which can hold a placeholder text
37. :nth-last-type()

    a. Colors of the background form countdown to up
    b. Allows the background to be colored for every <p> element form the second element from the parents to the last child.
38. :nth-last-child
    a. Selects every last child of the selected tag, counting up from the last child
39. :only-of-type
    a. Selects every chosen tag that is the only element of its parent
40. :last-child
    a. Applies to a specific property to the last child of each element
    b. It changes the last line of the element
41. :last-of-type
    a. Applies an attribute to the everlast child form a specific type.
    b. It is essentially the last of the child tag
42. Hover
    a. When your pointer mover over text or image
43. Link
    a. A link to a website or image or article

9/19/18

.box are blocks by default
Inline: only the size of the text in the box
Block elements can have inline and padding applied to them

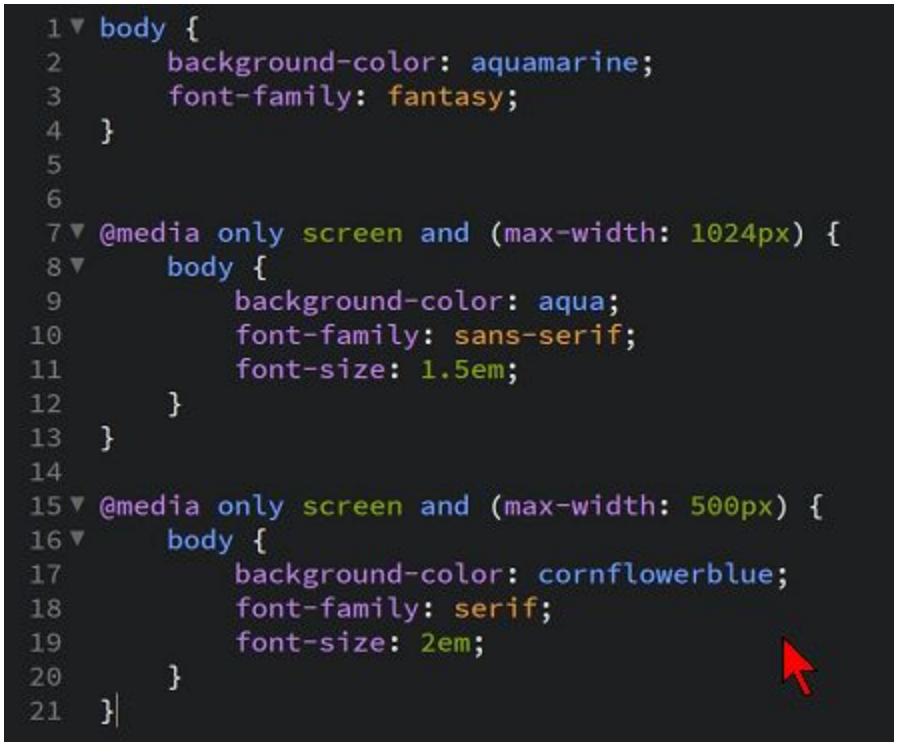
Inline-block cannot
Box models are most websites are made of
Objects are basically the box
Padding is the stuff around the box
Border goes around the border
Outside the border is the margin

CSS

Floats

9/25/18

Responsive design alters the image from the
Breaking point when a website can't adjust to as certain size
<meta http-equiv= “X-UA-Compatible” content=”ie=edge”>
<meta name=:viewport” content=”width=device-width, initial-scale=1.0”>
Without this tag , meta viewport how to tell a website to adjust for a mobile device
@media



```css
body {
    background-color: aquamarine;
    font-family: fantasy;
}


@media only screen and (max-width: 1024px) {
    body {
        background-color: aqua;
        font-family: sans-serif;
        font-size: 1.5em;
    }
}

@media only screen and (max-width: 500px) {
    body {
        background-color: cornflowerblue;
        font-family: serif;
        font-size: 2em;
    }
}
```

Two max widths (1024px 500px)
Last one should be smallest one to make sure it overrides everything.
The only instance where {} will nest to work in CSS without breaking the CSS page
Media needs to be written at the end after the base has been written

10/12 reference forms.html

How we collect data from website users
Numerous tags and attributes that allow us to adapt
To start a form

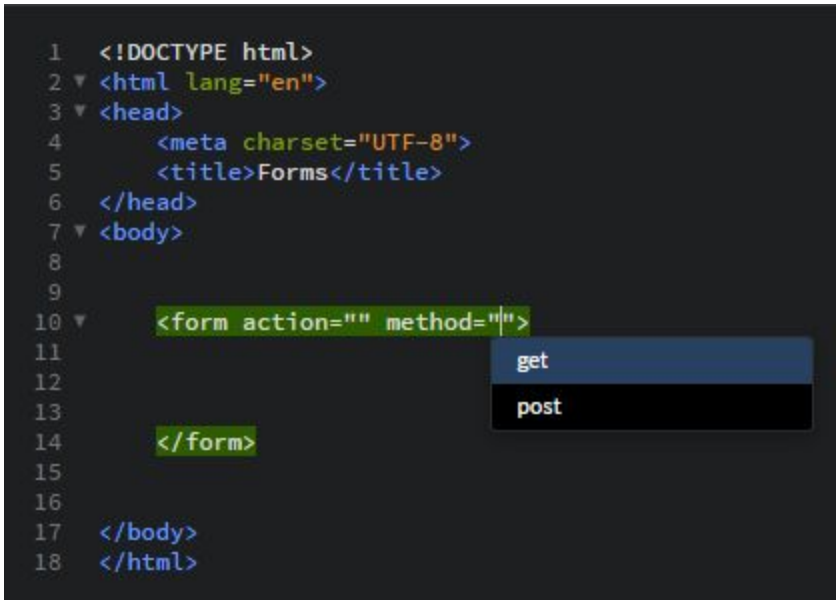
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Forms</title>
</head>
<body>

    <form action="">


    </form>


</body>
</html>
```

Action attribute is where you set a place for where data is collected into.
Forms require action and method



Get is unstable and sends out all info into the address bar
The post is more secure
If you aren't getting any important info then get would be good
Do not use get

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Forms</title>
</head>
<body>


    <form action="" method="post">
        <label for="fn">First Name:</label>
        <input type="text" name="fn">

    </form>


</body>
</html>
```

For and name **must** match

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Forms</title>
</head>
<body>


    <form action="" method="post">
        <label for="fn">First Name:</label>
        <input type="text" name="fn">

          <label for="ln">Last Name:</label>
        <input type="text" name="ln">

        <lable for="sex">Gender:</lable>
        <input type="radio" name="sex"
        value="M">Male<br>

            <lable for="sex">Gender:</lable>
        <input type="radio" name="sex"
        value="F">Female<br>

            <lable for="sex">Gender:</lable>
        <input type="radio" name="sex"
        value="O">Other<br>

    </form>


</body>
</html>
```

```html
<label for="Gender2">Gender:</label>
<select name="Gender2" id="Gender2">
    <option value="M">Male</option>
    <option value="F">Female</option>
    <option value="O">Other</option>
    <option value="A">Apache Helicopter</option>
    <option value="P">Potato</option>
</select>
```

Another way

```html
<lable for="sex">Gender:</lable>
<input type="radio" name="sex"
value="M" checked>Male<br>
```

pre select

```html
<option value="A" selcted>Apache Helicopter</option>
```

```
  <label for="Gender2">Gender:</label>
  <select name="Gender2" id="Gender2"
      size="3" multiple>
  </select>

  <label for="email">Email Address</label>
  <input type="email" name="email" placeholder="address@domain.com">
form>
```

```
<label for="un">Username:</label>
<input type="text" name="un">

<label for="pw">Password:</label>
<input type="password" name="pw">

<label for="bio">Bio:</label>
<textarea name="bio" id="bio" cols="30" rows="10"></textarea>
<!-------Thirty words long 10 rows high----->

<label for="food">Fast Food:</label>
<input type="checkbox" name="food"
value="McD">McDonalds<br>

 <input type="checkbox" name="food"
value="TB">Taco Bell<br>

  <input type="checkbox" name="food"
value="CJ">Carl's Junior<br>

  <input type="checkbox" name="food"
value="INN">In n Out<br>

  <input type="checkbox" name="food"
value="W">Wendy's<br>

  <input type="checkbox" name="food"
value="DT">Del Taco<br>

  <input type="checkbox" name="food"
value="BK">Burger King<br>

<label for="color">Fav. Color</label>
<input type="color" name="color">

<label for="BM">Birth Month:</label>
<input type="month" name="BM">

<input type="date">

<input type="datetime-local"><br>
```

```
<label for="qty">Quanity</label>
<input type="number" name="qty" min="1" max="20" step="4">
```

```
<input type="range">

<label for="browsers">Favorite Browser:</label>
<input list="browsers">
<datalist id="browsers"></datalist>
<option value="Edge">
 <option value="Internet Explorer">
  <option value="Chrome">
   <option value="FireFox">
    <option value="Canary">
```

10/30

- HTML
    - Content structure
    - No specific details
        - In car speak, this is the chassis
- CSS
    - Style layout
        - Body and paint
- Javascript
    - Functionally interactive animation
        - Motor

Many names it goes by

- Mocha
- LiveScript
- JScript
- ECMAScript

The world's most misunderstood programming language

- Often derided as being a toy
- Deceptively simple to start
- Wickedly powerful once mastered
- “Script” does not mean a lesser language

Created in ‘95 by Eich at Netscape

First released in early ‘96 w/ Netscape Navigator 2 (web browser)

First called LiveScript but renamed due to the popularity of Java

- Marketing move has caused confusion ever since

Javascript and Java(another language)

- Have nothing, zilch, nada, zip, zero in common

A few months later, Microsoft released equivalent JScript language with Internet Explorer 3

This led to submit JavaScript to EMCA International, a European Standards Organization(like W3C). Thus the first standardized version of JavaScript was born.

The code should be self-documenting

ECMAScript

- Official name
    - JS

JS definition

JavaScript is a dynamic untyped and interpreted a programming language

It is prototyped based and supports both objects

```
> alert("Hello, World!");
< undefined
```

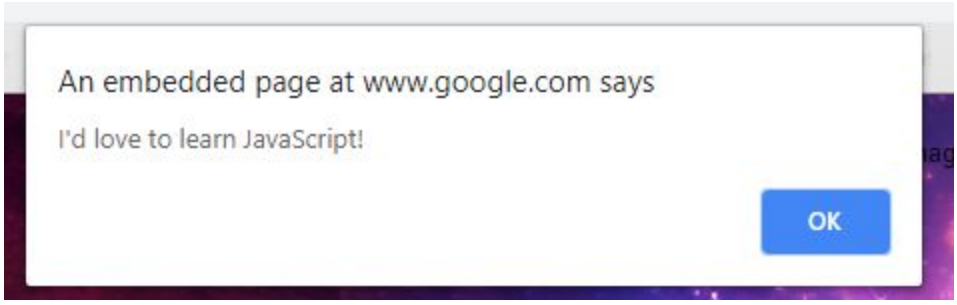
All JS commands end in semicolons

```
> console.log("Javascript is also known as ECMAScript");
  Javascript is also known as ECMAScript
```

console.log()
Displays the requested info in the console

A large part of programming is experimentation -- we come up with a hypothesis and test it
JavaScript doesn't know the difference between a single quote and apostrophe

```
> alert("i'd love to learn JavaScript!");
< undefined
> alert('I/'d love to learn JavaScript!');
✖ Uncaught SyntaxError: missing ) after argument list
> alert('I\'d love to learn JavaScript!');
>
```



11-5

Objectives

Define data type

What is a data type?
At the machine level, all data on a computer is bits-- 1s and 0s. Humans, it turns out, prefer not to work to the metal, so we have data types for describing different bits of information. Data types classify

Data types only go into certain holes.

It turns out that we are adding things of two different types. 2 is a number “2” is a string

Why with types?
As humans we see 23445, we provide content we can see it as a sequence of integers in our heads or two thousand three hundred forty-five we understand that 2345+2 would give us 2347. Javascript can't do that. All it knows is that it is a number. Anything in quotations marks as a string. We can provide the context for JAvaScript according to the rules that JavaScript follows-- one of those rules happen to describe what happens when we add a number and a string.
Type of
Other types

- Boolean - true or false
- Undefined- no value
- Null - nothing

```
> typeof 1
"number"
> typeof 10
"number"
> typeof 1.123
"number"
> typeof "Albert"
"string"
> typeof "123"
"string"
> typeof "what's my type?"
"string"
> typeof true
"boolean"
> typeof false
"boolean"
> typeof undefined
"undefined"
> typeof null
"object"
```

```
> typeof 1.123.45
Uncaught SyntaxError: Unexpected number
> typeof'i'm not going to work'
Uncaught SyntaxError: Unexpected identifier
```

JavaScript uses numbers
Java treats all numbers as if they have decimal points (even when they don't).

```
> 4
< 4
> 4.0000000000000001
< 4
```

Strings are a collection of characters.
We have to escape quotation marks when they are inside of a string
'I'm happy to be here'
When we wrap a string in double quotation marks, we don't need to escape single quotation marks.

```
> 'I\'m practiing strings!'
< "I'm practiing strings!"
```

Quote in strings

```
> ' I said, "strings are pretty nifty."'
< " I said, "strings are pretty nifty.""
```

Add strings together

```
> 'Hello, ' + 'World'
< "Hello, World"
> 'High ' + 5 + '!!!'
< "High 5!!!"
```

Called concatenation
We can also insert a string into other things called interpolation
Template literals backticks ‘ and shares the ~ key

```
`High ${3 + 2}!` // 'High 5!'
```

String interpolation whole string is wrapped in backticks and the part that we interpolate is wrapped in ${} it signals to the JavaScript interpreter that it should evaluate (that is, run the code) whatever is inside of it.

11-7

Objectives

- Explain what a function is in JavaScript
- Write a function from scratch
- Explain

What if?

Using the console is a great way

There *is* a better way!

We can use a function

Functions are ways of gaining JavaScript instructions that it can run over and over again. When we run a function's instructions, we say that we have called the function. In JavaScript functions are written with the function keyword.

Function doNothing() {}

When we declare a function, we start with the function keyword, followed by a name for the function (doNothing), followed by a pair of parentheses. Then we have a pair of curly braces.

The function above, as its name implies doesn't do anything

Nothing happened

```
> function sayHelloToJared() {
      console.log("Hello, Jared!")}
< undefined
> sayHelloToJared
< ƒ sayHelloToJared() {
      console.log("Hello, Jared!")}
> satHelloToJared
✖ ▶Uncaught ReferenceError: satHelloToJar
      at <anonymous>:1:1
> sayHelloToJared()
  Hello, Jared!
```

```
> function sayHelloToJodi() {
      console.log("Hello, Jodi!")}
< undefined
> sayHelloToJodi()
  Hello, Jodi!
< undefined
```

```
> sayHelloToJared(); sayHelloToJodi();
  Hello, Jared!
  Hello, Jodi!
< undefined
```

We can pass arguments to a function between its parentheses

```
function doSomething(thing) {
  console.log(thing)
}
doSomething('anything')
```

```
> function doSomething(thing) {
      console.log(thing)}
<- undefined
> doSomething('anything')
  anything
<- undefined
```

```
> function sayHelloTo(firstname) {
      console.log(`Hello, ${firstname}!`)}
<- undefined
> sayHelloTo('R2D2')
  Hello, R2D2!
<- undefined
```

```
> sayHelloTo('Ben')
  Hello, Ben!
<- undefined
> sayHelloTo('Rey')
  Hello, Rey!
<- undefined
> sayHelloTo('Antilles')
  Hello, Antilles!
<- undefined
```

Parameters are placeholders that we put between the parentheses when declaring a function. When we invoke that function, we can pass arguments to it that get stored as local function-level variables that are available anywhere in the function.
Note that we can only access arguments within the function.

```
> function say(greet, firstName) {
      console.log(`${greet}, ${firstName}!`)}
<- undefined
> say("Goodbye", "Han")
  Goodbye, Han!
<- undefined
```

```
> say("Skywalker", "We meet again")
  Skywalker, We meet again!
<- undefined
```

JavaScript cannot tell between the two when we flip it.

World's most okay function
These functions are pretty much okay, but our functions are only logging the information in their local scope.

```
> function add(x, y){
      return x + y}
<- undefined
> add(i, 2)
✖ ▶Uncaught ReferenceEr
      at <anonymous>:1:
> add(1, 2)
<- 3
```

adds to arguments together. Function finally is defined
When we return inside a function, we are giving that value back to the world outside the function.

```
> var z = add(18, 3)
<- undefined
> add(z, 21)
<- 42
```

Note the use if a template literal, which we learned about previously, in order to make the string easier to read.

```
> function say(greet, firstName) {
      return`${greet}, ${firstName}!`}
<- undefined
> say("Hello", "Senator"
✖ Uncaught SyntaxError: missing ) after argume
> say("Hello", "Senator")
<- "Hello, Senator!"
```

BY wrapping the return in quotes the console is letting us know we are returning a string instead of just logging the values.
If we don't

```
> function say(greet, firstName) {
      return`${greet}, ${firstName}!`
      console.log('3,720 to 1')}
<- undefined
> say("Never", "tell me the odds")
<- "Never, tell me the odds!"
```

What happened to our log?

This is because returns end the execution inside the function, meaning that if we return, nothing will happen after that.

```
> function say(greet, firstName) {  console.log('3,720 to 1')
      return`${greet}, ${firstName}!`}

<- undefined
> say("Never", "tell me the odds")
  3,720 to 1
<- "Never, tell me the odds!"
```

11-13

Objectives

- Define comparisons
- Write if statements
- Write if-else if -else statements
- Use the ternary operator
- Write switch statements

Sometimes we only want to run code under certain conditions.
When you're driving a car you can only go through a light if the light is green. Else if the light is yellow you prepare to a slow down: and if the light is red, you stop.

IN programming, when we check for a statement in this way, we check to see whether the statement is true or false. JavaScript, being the friendly language that it is, use true and false directly to mean exactly what they say.

```
var light = prompt("Is the light Green, Yellow, or Red?")

if (light == Green) {
  go()
} else if (light == Yellow) {
  slow()
} else {
  stop()
}
```

When we get down to it, every if statement like the above is saying, "If the thing in the parentheses is true, then do what's between the curly braces.

How do things become true or false?
JavaScript lets us compare things. Most of these comparisons come straight from math: Is something less than something?

We can't use = because it is reserved for setting the value of a variable
We use === (3 equal signs) to make an exact comparison.
This tells JavaScript to match the value and data type.

You might see === used for comparisons, this tries to coerce values and may not always compare what you expect.

It's best practice to use ===.

We can combine comparisons using && (pronounced "and") and || ("or').

With &&, both statements (to the left and right of &&) must be true in order for the entire expression (that is, the entire phrase) to be true.

With ||, only one of the statements needs to be true for the expression to be true.

JavaScript reads these comparisons from left to right, returns the last statement it saw, and only evaluates as many statements as necessary.

```
5 === 4 && 0
```

Will return false because it stops evaluating the && expression on its first false encounter

200<100||alphabet'

Will return alphabet, because it needs to evaluate the right-hand side of ||

JavaScript lets us control what blocks of code to execute using if statements, if-else statements.if -else if-else statements, ternary operators, and switch statements.

If statements look like this:

```
if (something) {
  // do something
}
```

If something is truthy (so the boolean true or anything other than the empty string('), 0, false, null, or undefined), the code in between the curly braces runs; if not (false), the code between the curly braces is skipped.

You will often see an if statement used in combination with an else clause. An else clause will only be executed if the previous if statement is false.

```
if (conditionToTest) {
   // executed if `conditionToTest` is truthy
} else {
   // executed if `conditionToTest` is false
}
```

If statements can also be combined with an else if clause. This is like an else statement, but with its own condition. It will only run if its condition is true, and the previous statement's condition was false.

```
if (conditionToTest1) {
    // if condition is false code is not executed, otherwise code runs and stops here
} else if (conditionToTest2) {
  // execute this code if `conditionToTest1` statement is false AND `conditionToTest2` is truthy
}
```

You can optionally add a final else statement after all of your else if statements. You can probably guess what will happen: if all of the other statements are false, this final else block

```
if (conditionToTest1) {
  // if condition is false code is not executed
} else if (conditionToTest2) {
  // execute this code if `conditionToTest1` statement is false AND `conditionToTest2` is truthy
} else {
  // execute this code if none of the other conditions are met
}
```

Remember, if you place a return statement before the end of the function, anything after return won't get executed. We can use this to make code terser:

```
function canGo(lightColor) {
 if (lightColor === 'green') {
   return true
 }
 return false
}
```

You can think of it as a shortcut for the if-else statement.
This operator tests a condition; if the condition is truthy, it evaluates the left-hand side of the colon; otherwise, it evaluates the right-hand side of the colon.

Syntax:

```
conditionToTest ? valueToBeReturnedIfTrue : valueToBeReturnedIfFalse
```

Switch statements

Switch statements are like a big if/else if/else chain. The switch expression is evaluated once and the value of the expression is compared with the values of each case. If there is a match, the associated block if the code is executed.

```
switch (expression) {
  case n:
    // code to be executed if case n is true
    break; // optional, stop processing switch statement once code is executed
  case m:
    // code to be executed if case m is true
    break; // optional, break chain of commands once code is executed
  default:  // all other cases
    // code to be executed if case n and case m false
}
```

```
var mood = "hungry"
switch(mood) {
  case "happy":
    console.log("Dance to Pharrell's 'Happy'");
    break;
  case "sad":
    console.log("You should eat a pint of ice cream");
    break;
  case "anxious":
    console.log("Take some deep breaths");
    break;
  case "hungry":
    console.log("You should eat a big chocolate cake");
    break;
  default:
    console.log("That's not a mood we support");
}
```

As with any function, the return will halt execution at any point.

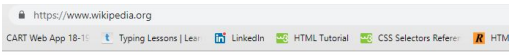
```
function feelings(mood) {
  switch(mood) {
    case "happy":
      return "Dance to Pharrell's 'Happy'"
    Default:
      return "I don't recognize that mood."
  } console.log("Let us know how you're feeling tomorrow!")
}
```

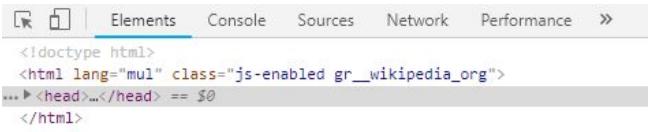
The console.log() statement at the bottom of the function will never run. This is a major difference between return and break: return exits the function and returns a value; break exits a block and does not (generally speaking) have a value associated with it.

11-26

Document object model

Gives us access to javascript in real time





What are you trying to say?

That by making changes to the Document Object Model, we can change the way our webpage displays. And we can do this even if our HTML is unchanged.

What happened?

The changes that the developer console caused, and the changes the developer console currently displays are changes in the Document Object Model, but not in our HTML. Our webpage now looks blank, reflecting the missing body in our DOM, even though our HTML still has content in the body tags.

The Document Object Model is a representation of the current view of the browser, and can be manipulated without reloading a page.

HTML is the starting point of the page's content. But as we just saw by deleting the body of the page, what is displayed can change. When we change it, we change the Document Object Model, and that changes the appearance in the browser. The HTML, however, once loaded on a webpage, does not change.

Is it still JS?

Yes with JS we can:

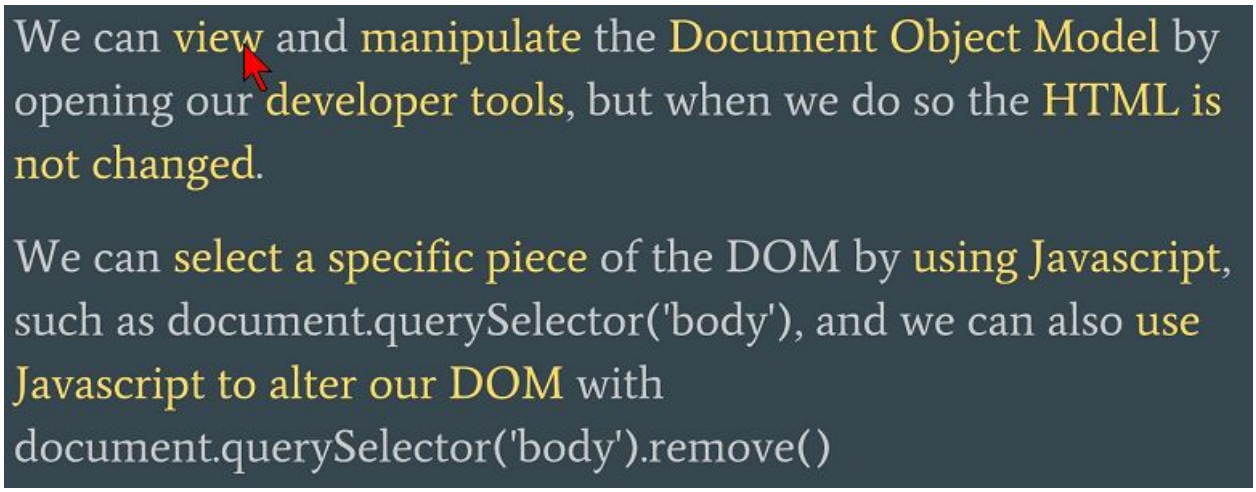
1. View a current representation of our document object Model
2. Select specific portions of the DOM and manipulate them which changes what shows up in the browser window

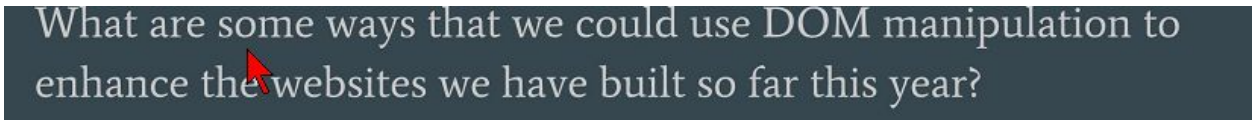
```
> document
<- ▼#document
      <!doctype html>
      <html lang="mul" class="js-enabled gr__wikipedia_org">
      ▶<head>…</head>
      ▶<body id="www-wikipedia-org" class=" jsl10n-visible" data-gr-c-s-
      loaded="true">…</body>
      </html>
```

```
> document.querySelector('body')
<- ▶<body id="www-wikipedia-org" class=" jsl10n-visible" data-gr-c-s-loaded=
   "true">…</body>
```

```
> document.querySelector('body').remove()
<- undefined
```

As you now know, the HTML never changes after it is first rendered. Instead, we accessed the Document Object Model, altered the model and that altered the appearance of our web page.



1. 

We can use to replace an image or possible change a piece of code from anywhere for that moment.

2. 

   Right now it would just be used to fix small things that we can't in HTML or CSS.

11-28

JS DOM part 2

All of this is done using JavaScript to traverse and manipulate the DOM.
Everything google related is JS heavy.
The DOM provides a structural representation if the document in tree form, enabling you to modify its content and visual presentation.
The highest level of the DOM tree is the window object
Think of the window as the browser window.
The window contains the entire DOM document. All components of your HTML file will be accessible from

The window object has a large number of properties that return information about the object.
Window.document;
//returns the entire HTML document

window.innerWidth;
//returns the inner width of the browser window

window.innerHeight;
//returns the inner height of the browser window

The document object represents any web page loaded in the browser.
The document contains all the nodes that represent the HTML elements of the page.
We use the document object to traverse through the HTML and manipulate elements.
There are many document properties that allow us to obtain information about the document programmatically
Document.all;
//returns all the nodes inside the document object

document .contentType;
//returns the type of content contained. Most web pages should “text/html”
Below the document are all the nodes representing the HTML or XML elements on the page.
We can alter the DOM through different methods:

1. Add/remove HTML elements in the page
    a. You can add elements with the function like appendChild
    b. You can remove elements with the similarly named removeChild
    c. Both of these functions can be called on any noise in the DOM tree

2. Add/remove/change HTML attributes
    a. If you have a DOM node called element, element. attributes give
3. Add/remove/change CSS styles
    a. You can modify any DOM node's style property.
4. Listen for any key presses or mouse events on Elements
    a. You can add a listener directly using addEventListener
    b. Elements emit an extensive array of events

Selecting elements

- document.getElementsByTagName("p")
    - 👀 //returns all p tags on a page
    - 👀 //alternatively
        - 💡 document.querySelectorAll('p')
            - //the results of these two function are the same in this example but as we'll see later, getElementsByTagName and querySelectorAll have different uses
- `document.getElementById('id_here')`

  This method provides the quickest access to a node, but it requires that we know something very specific about it — its id. Since IDs must be unique, this method only returns one element. (If you have two elements with the same ID, this method returns the first one — keep your IDs unique!)
- `document.getElementsByClassName('class_name')`

  This method, as its name implies, finds elements by their className. Unlike id, className does not need to be unique; as such, this method returns an HTMLCollection (basically a list of DOM nodes — *note that it is not an array*, even though it has a length property) of all the elements with the given class. You can iterate over an HTMLCollection with a simple for loop.
- The HTML property class is referred to as className in JS.
- It's... unfortunate.

What is Flexbox

Floats are evil
Problem was compatibility
Prior to 2016 websites used floats
How do I Use flexbox?
.flex-container {
Display: flex;}
/*simple right*/
See brackets for the rest of the notes
  flex-direction: row;
    display: flex;
    height: 100vh;
      flex-wrap: wrap;
    align-items: center;
    align-content: space-around;
Flex-container usage

Content order:

```
.box1 {
  width: 100%;
     flex-grow: 1;
}

.box2 {
order: -1;
     flex-grow: 3;
}
.box3 {
order: -3;
}
.box4 {
order: 3;
}
.box5 {
flex-grow: 5;
}
```

Flex-shrink never gets smaller than it’s content.

flexbox playground address

JS 2/6:

A series of instructions = a script
Like a recipe/handbook/manual
Recipe: instruction, that when followed in sequence allow cooks to create a dish they never made before

. Recipes:
Some are simple and only deal with the individual scenarios
Others are more complex, like making a 3-course meal
Similar to programming, there is a lot of terminologies to learn.

Handbooks:
A hotel handbook may include steps for different scenarios
Checking in guests
When a room needs to be tidied
Fire alarm procedure… etc.

In any of the above scenarios, such as guest checking in, employees need to follow the steps for that particular event. They don't do everything in the manual for check-in.

Manuals:
Scenario: mechanic testing break on a car:

1. Follow the steps in the manual to test the brake system.
    a. If they pass the test, move on to the next check.
    b. Else, follow instructions to repair system.
2. If repairs are made the mechanic can go back to the beginning of the test procedure to verify the repair was successful.

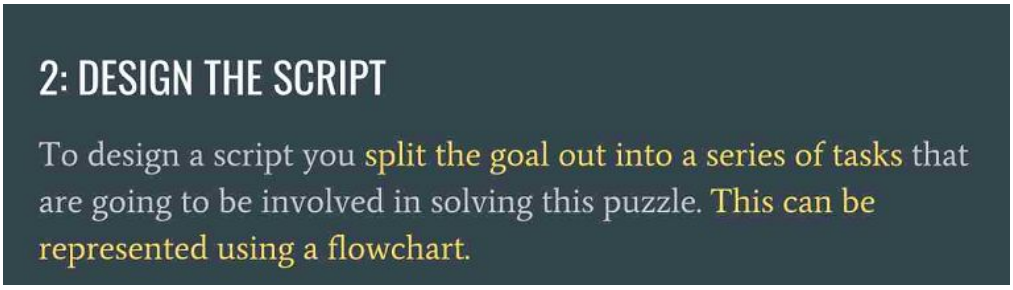
Scripts
Are made of instructions a computer can follow step by step
A browser may use different parts of a script depending on how a user interacts with it.
Scripts can run different sections of code in response to the situation around them.

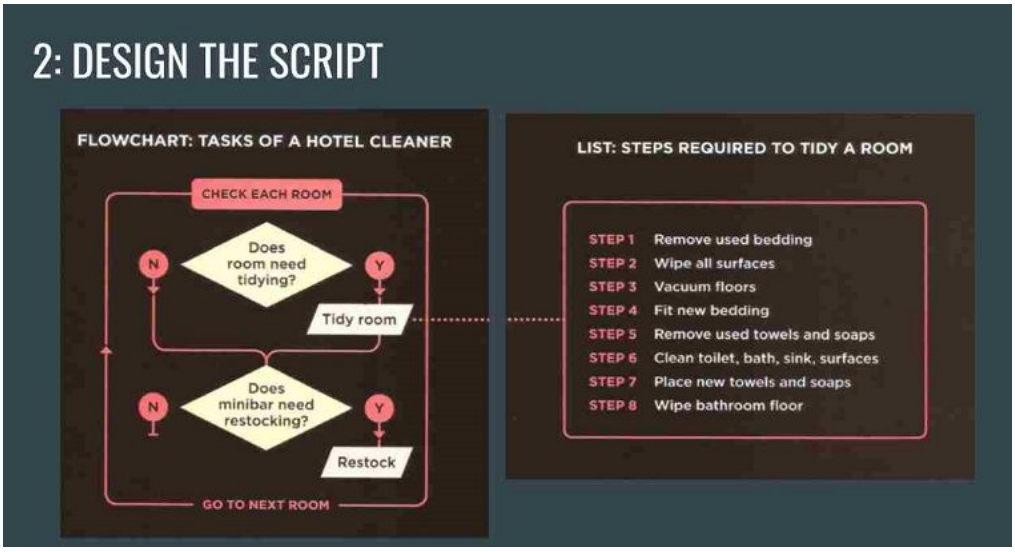
To write a script, you first need to first state your goal and then list the tasks that need to be completed in order to achieve it.
Start with the big picture of what you want to achieve and break that down into smaller steps.

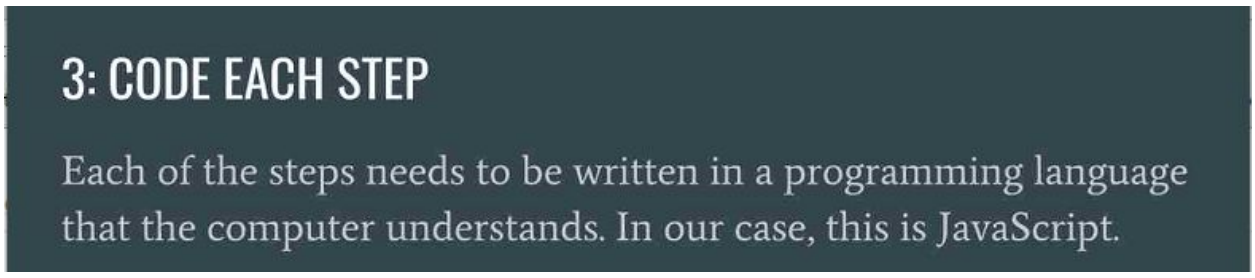
1. Define the goal

a. first, you need to define the task you want to achieve. You can think of this as a puzzle for the

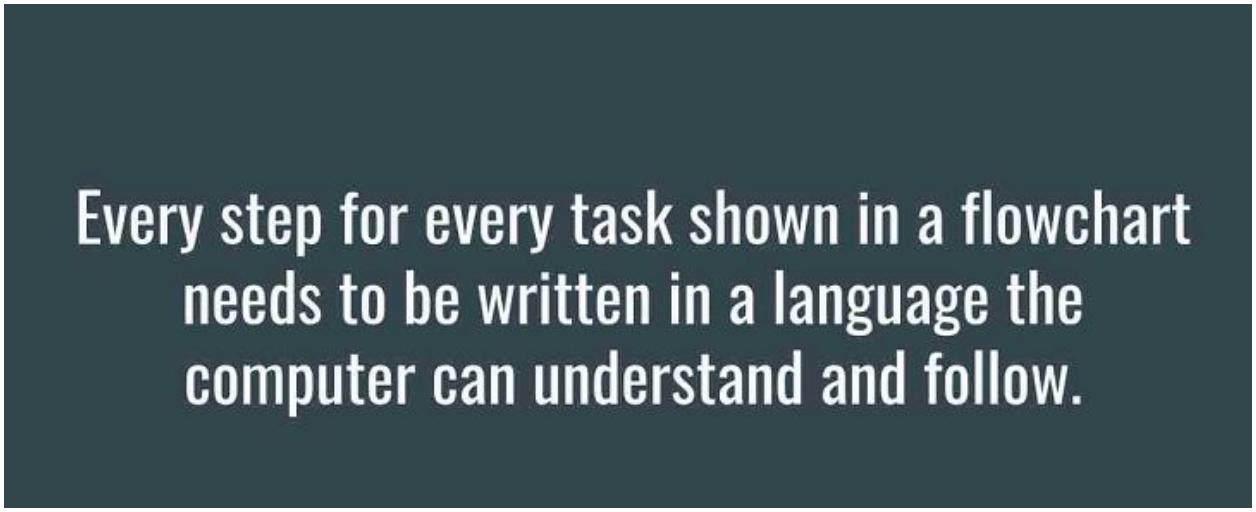
2. Design the script

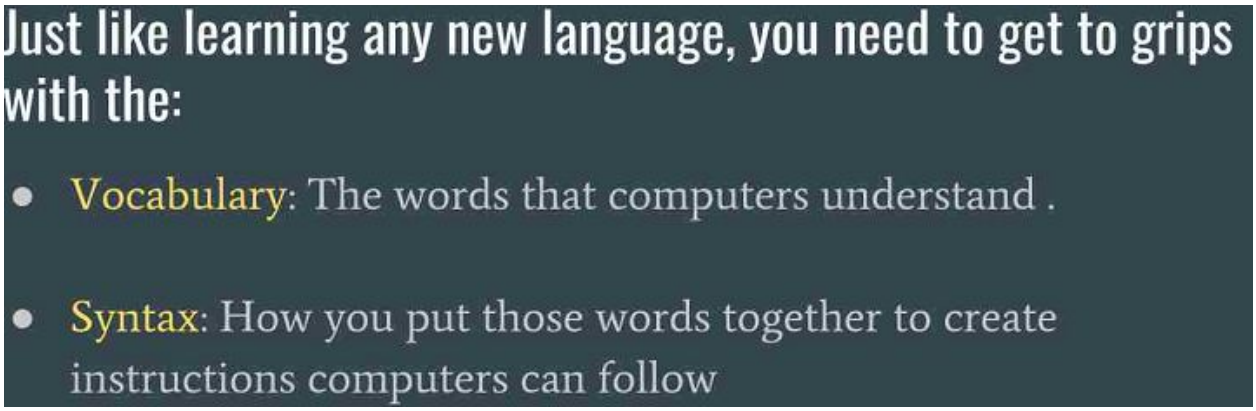
a. 

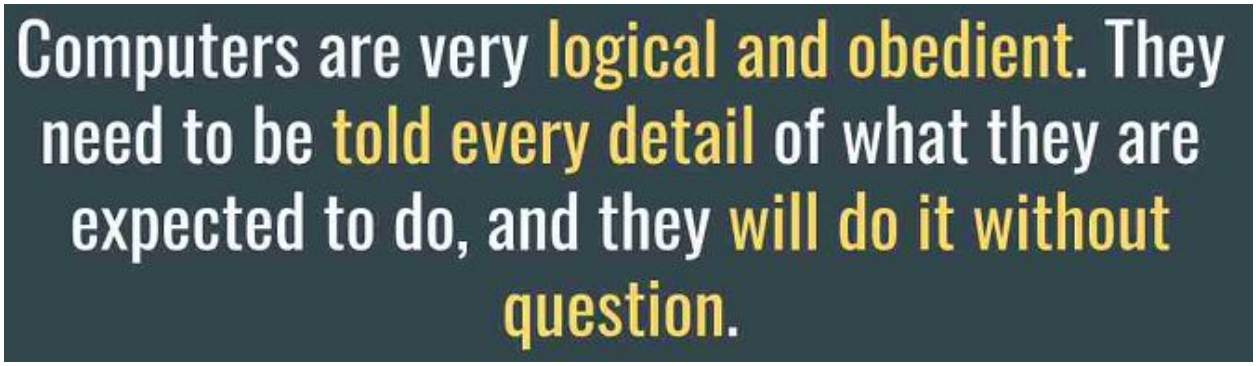
b. 

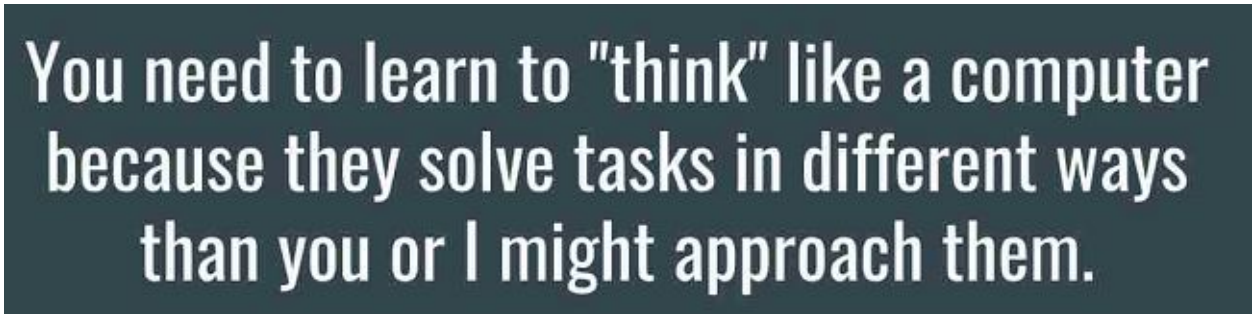
3. 

4. 

5. 
Every step for every task shown in a flowchart needs to be written in a language the computer can understand and follow.

6. 
Just like learning any new language, you need to get to grips with the:
Vocabulary: The words that computers understand .
Syntax: How you put those words together to create instructions computers can follow

7. 
Computers are very logical and obedient. They need to be told every detail of what they are expected to do, and they will do it without question.

8. 
You need to learn to "think" like a computer because they solve tasks in different ways than you or I might approach them.

9. When you look at the picture on the right how do you tell which person is the tallest?

10.
1. Find the height of the first person
2. Assume he or she is the "tallest person"
3. Look at the height of the remaining people one-by-one and compare their height to the "tallest person" you have found so far
4. At each step, if you find someone whose height is greater than the current "tallest person", he or she becomes the new "tallest person"
5. Once you have checked all the people, tell me which one is the tallest

2/25

Git removes the need to copy files to and from the class share and your “H” drive as you collaborate on a project.
Git is similar to using your camera to take a snapshot of your files at a specific point in time that you can magically go back to if terrible things happen
Checkpoint for your files
Got exist so you can modify/change/break/improve your code, secure in the knowledge that you can not ruin your work too badly because you created save points along the way

Git is a collaboration tool that allows different people to work on all the parts of a project at the same time.
(more on this tomorrow)

Git is a tool that protects yourself and others from yourself and others.

The Local Workflow

Getting Started

We need to tell Git to start tracking our files:

1. Open File Explorer.
2. Create a folder named “practice” in your “H” drive.
3. Type “cmd” in the address bar.
4. A Command Prompt should open to H:\practice >
5. Tell git to watch this folder by initializing it:

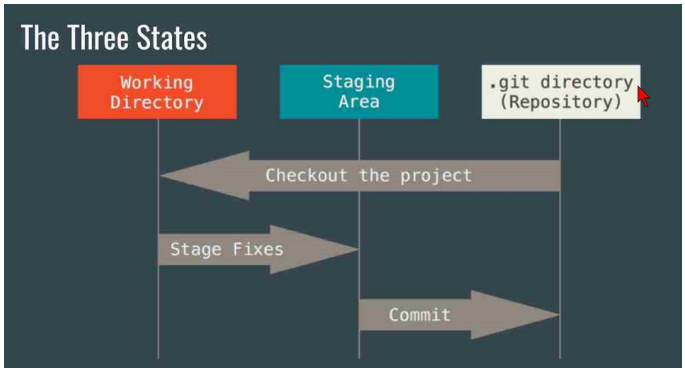
```
git init
```

Git init creates a repository in the folder you ran the command on. Often shortened to repo, this is a hidden location where file checkpoints will be stored.
(do not delete this folder!)

Three main “states”

Now that git is tracking your working directory, files in it will exist in 3 states:

- Modified - files that are new or have changes not yet saved by Git
- Staged - the current version of a file, tagged to be included in the next commit.
- Committed - files that are safely stored by Git



Using Git

Open Brackets and save an index.html file in your practice folder.

Add the basic HTML structure, but nothing else and save the file.

```
No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        git-practice.html

nothing added to commit but untracked files present (use "git add" to track)

H:\practice>
```

Using Git

Switch to your command prompt window and check the status of your files with:

git status

Let's hit that checkpoint.

In order to save our progress, we need to tell Git to stage our file.

We do this by adding the file.

git add index.html

How do we store the box?

We commit the box to storage and note what it contains.

git commit -m "description goes here"

Until we commit our work, there is no checkpoint to save us from errors...

Git commit physically moves the box of copies into long-term storage, like my grandma's attic... so make sure you describe what's in the box just in case you need it...

It does not move or remove files in your working directory.

Git commit physically moves the box of copies into long-term storage, like my grandma's attic... so make sure you describe what's in the box just in case you need it...

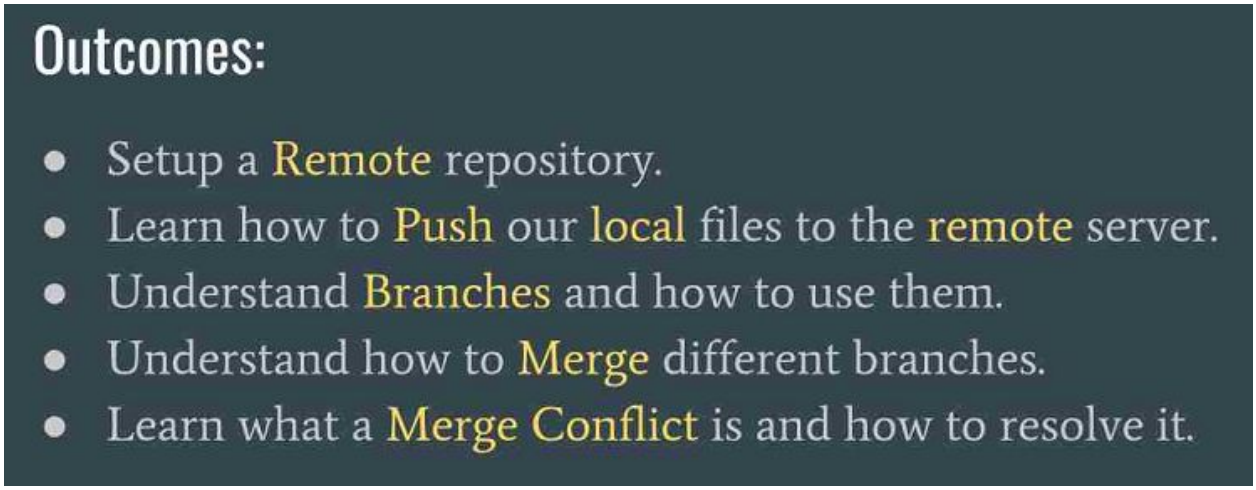
It does not move or remove files in your working directory.

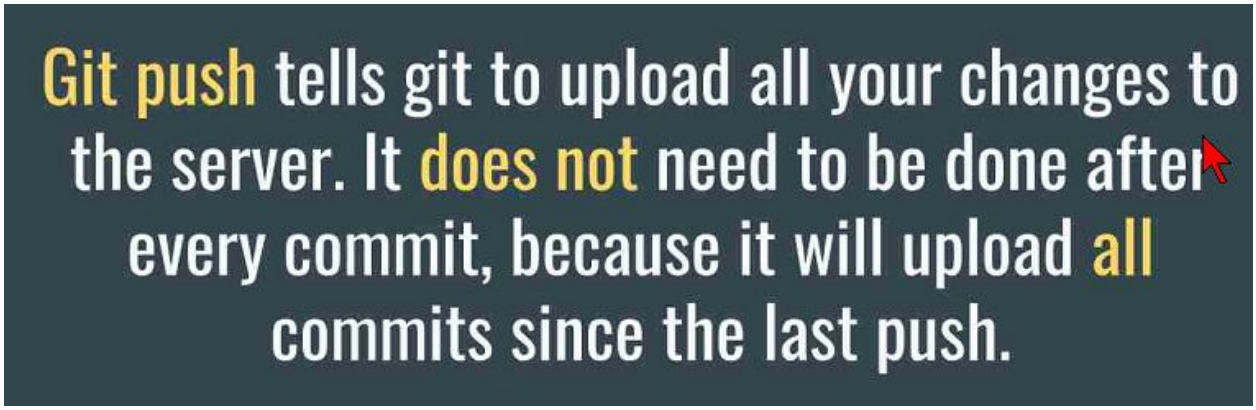
git log

Returns a list of all the commits (boxes) you have saved.
The gibberish is the unique name of the box.
What else do you see?

2/26

Git





Push at the end of the class period or when partner needs code from you

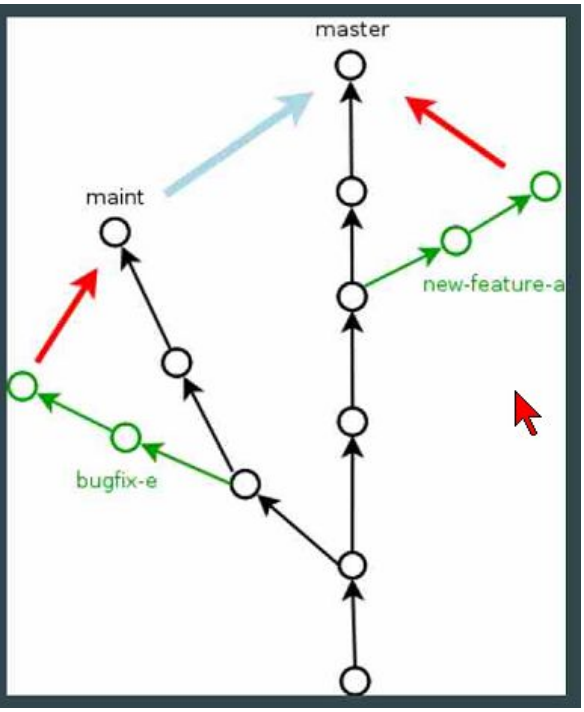
Branches are exactly what they sound like, smaller bits extending from a tree trunk.

They represent different versions of our code.

Branches allow us to work on code fixes and features without breaking what we already have (presumably) working.

Fixes and new features should always start on a branch.

The master branch is the "trunk" of your code tree and should only contain clean code ready for deployment (use on the web).



Git branch <name> tells git to maintain a new copy of our code with the given name.

Git branch on its own will list the branches available and display an asterisk next to the one we are currently working on.

Git checkout <branch> tells git to switch our working folder to the branch name specified.

Working with branches:

We're going to add media queries to our flexbox page:

1. Create a new branch so we don't break our working code:
   a. git branch mobile

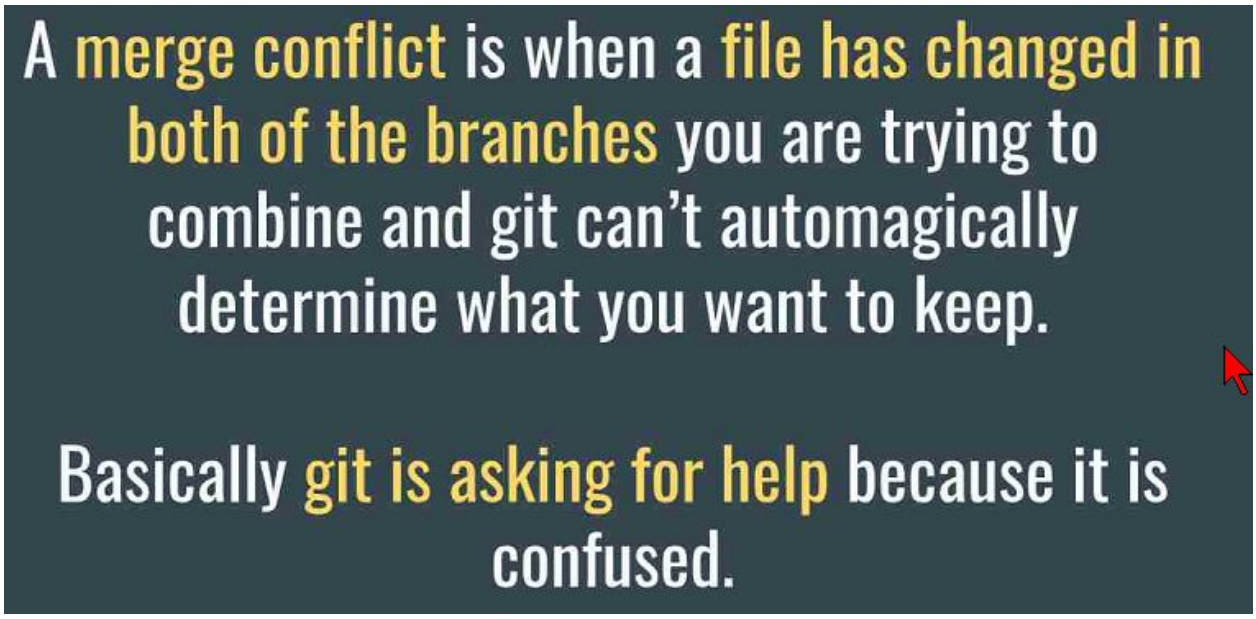
2. IMPORTANT! We have to switch to our new branch:
   a. git checkout mobile

Working with branches:

1. Open Brackets and add a MQ line to flex.css:
   a. @media screen and (max-width:480px) {}
   b. Save the file.

We use the merge command to combine branches.

Git merge <branch> combines the file changes in branch we name into our current working branch.



I think by having these set up we are able to work together better. I know that Andrew and I had had some problems in the showcase project due to code not appearing for one person til the person restarted brackets. I think that this would be easier to use than ever. I think I would give this a 3 for understanding.