Version Control and Git

Commit – a snapshot of a collection of files at a time

Branch – linear sequence of commits

Repository – many branches of a project

Remote – another place a repository is stored

HTML

Hypertext Markup Language

* Block, inline and replace tags

Lists – unordered, ordered and description

Absolute URLs point to a location on the web, including protocol and domain name. relative URLS point somewhere relative to the file.

Accessibility – use alt text for images and titles for links

CSS

Cascading Style Sheets

* Styling HTML elements, colours, fonts etc

Use <link > html tags to connect css to html

Use .classname to apply styles to specific classes.

Use #id for IDs

All available styles are combined as the page loads. Final appearance of elements determined b “specificity”, or last one applied last.

|  |  |  |
| --- | --- | --- |
| IDs | Classes | Tags |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 0 | 1 |
| 1 | 0 | 2 |
| 1 | 1 | 1 |

(task from css powerpoint)

CSS uses absolute and relative units.

Absolute e.g. px, pt, mm

Relative e.g. %, em, ex

CSS allows use of rgb or hex codes for colours

Pseudoclasses for :hover etc

CSS Box Model – every page element represented as a box

* Border – outline of box – width, style (solid, dashed etc), colour
* Padding – space between edge of the box and the outside edge of the contents within the box – width
* Margin – space outside of the border of the box to create space between boxed – width

Shorthand e.g. padding: 20px 40px 10px 30px -> sets to top/right/bottom/left

Background images can be set for elements, and can be moved using background-position and set to repeat using background-repeat.

Web browsers parse HTML and CSS to decide which elements to draw first.

Position property – static, relative, absolute, fixed

Absolute – removes element from page flow and positioned in a given location on the screen

Fixed – removes element from page flow and fixes in place – rest of page can scroll under it but it remains fixed.

JavaScript

-Object oriented language, client and server side

-often used for cross platform app development (web apps etc), and desktop apps.

-Good JS engines in mobile browsers

-Interpreted not compiled

-console.log() = print statement 😊

-Scripts referenced in html using <script> tags e.g. <script src = “link.com”></script>, or directly in the script tag (not recommended)

-alert() function creates a popup

Syntax:

* Insensitive to white space
* Case sensitive
* Block structured like Java
* Semicolons at the end of each line (like Java)
* Use require for modules in nodejs

Variables n types:

* Use = for assignment
* +, -, \*, /, % operators same as python
* ++ increment, -- decrement
* ==, != equal to, not equal to
* === same value *and* same type

e.g. 3 == “3” returns True, 3 === “3” returns false

Variables must be declared in strict mode

* var – declaration hoisted to the top, i.e. variable initialised at the top of code block even before line it is declared on is run. (Javascript is weird), var statements run before other code). Var declarations are executed in order tho. Redeclaring a variable without assigning a new value means it keeps the old value.
* let – not hoisted, declaration must be done before variable is referenced (like normal). Let does not allow for redeclaration of variables
* Advice – use Let when possible, makes code nicer 😊
* const – basically Let for constants

Data Types:

Primitive

* Boolean
* Null
* Undefined
* Number (no int)
* String
* Symbol

Non-primitive

* Objects
* Functions

Get a thing’s type using typeof() function, e.g. typeof(4) = “number”

Values have types, but declarations do not unless a value is assigned – dynamic typing (i.e. a variable’s type is determined by what is assigned to it)

Function parameters do not have types

Control structures:

* If, else
* Do, while, for, switch
* Throw, try, catch
* Ternary

Undefined = variable declared but no value, null = object does not exist

If(condition){

Code

}else{

Code2

}

While(condition){

Code

}

Do{

Code

} while condition

A do while statement will always execute at least once regardless of condition as condition is checked at the end

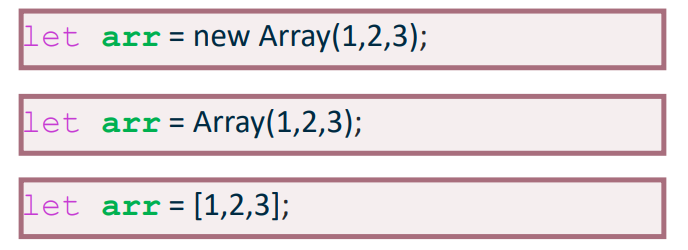
For(let i = 0; i<10; i++){

Code

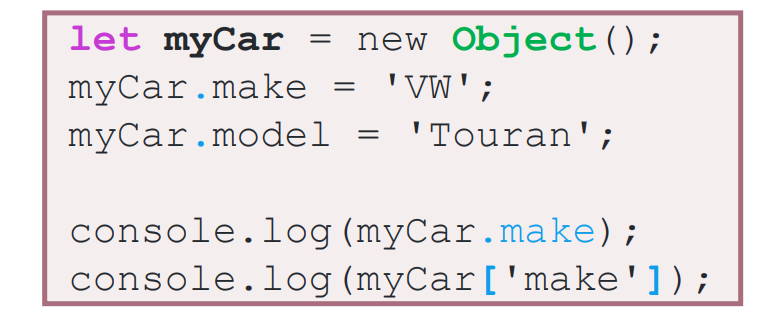
}

Variables declared using Let within an if/for/while statement have local scope only within the block it is defined within. Use var for variables you want to use outside of the block.

Arrays:

* Multiple/dynamic variable type – can have multiple data types in an array
* Use square brackets like python 😊
* ^ top one defines a new array object, others define an array variable

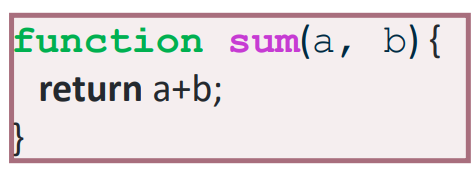
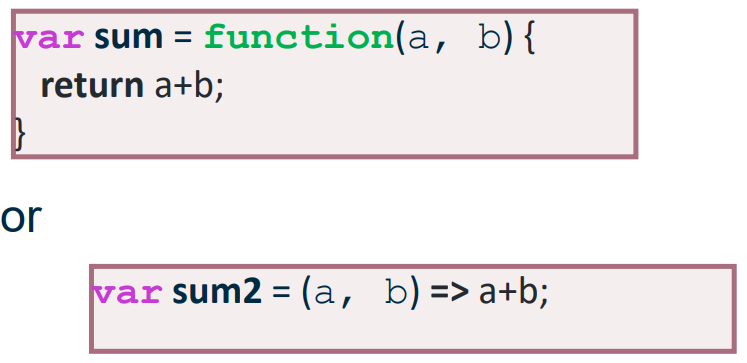
Objects

* Have named properties
* Can have any type, including object and function
* Create with Object constructor or literal syntax
* Access with dot or bracket
* Inheritance through prototypes
* (similar to classes in python?)

^both console.log() statements equivalent

(will be covered more next week)

Functions

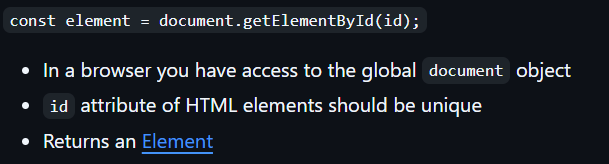
* Are first-class objects, and can be assigned to variables and passed as parameters
* Used for defining event callbacks
* Do not have to be associated with objects but can be
* Function parameters are typeless, use typeof() for error handling to check input data is the correct type.

Document Object Model (DOM)

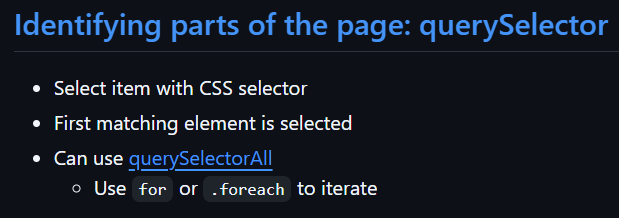
* Internal representation of a browser page
* Read contents with Javascript – JS is a part of the DOM and can be used to read the contents of the DOM
* JS can update contents of the DOM to change page content
* Historically problematic due to lack of universal standards between browsers – libraries such as jQuery had to be used instead to provide a common interface.

Event handling with HTML:

Identifying parts of the page:

Using getElementById:

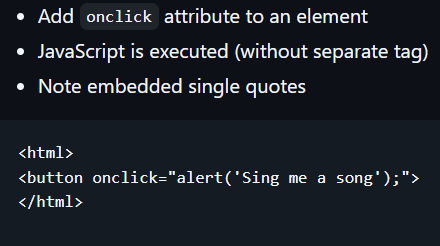
Using QuerySelector:

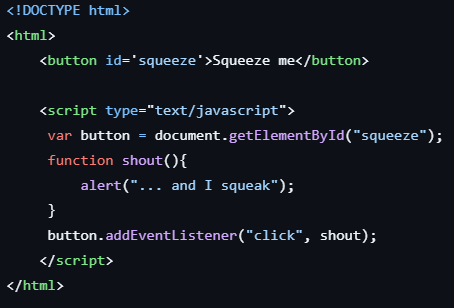
(<https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector>)

Listening for events:

Browsers produce events when users interact with elements on a page, e.g. click, mouseover

Use element.addEventListener to attach a function to an event, e.g.:



Separating JS and HTML is best.

Waiting for the DOM to be loaded:

* JS should be separated from HTML, and as such JS files are often referenced in the <head>, however this means DOM elements are not defined when code is executed. Event listeners are only added for elements that are defined when the JS is run, which is usually before the page has loaded, meaning elements are often not defined. Use

document.addEventListener("DOMContentLoaded",function(){\*JS code goes here\*})

to wait for the DOM to be loaded before executing the code.

function() is an anonymous function as it is not named – you can do that in JS

Manipulating the DOM:

* The contents of an element can be changed using the innerHTML property
* New elements can be created using createElement
* Elements can be added and removed using append and remove
* Attributes such as class and style can be changed using setAttribute
* Update classes by using classList.add() or classList.remove()

(see mdn web docs for info on these)

To iterate through a list in JS, use:

for(let x of list\_name){

}

HTTP

* Underlines many aspects of the web.
* Based around sockets – usually port 80 for web pages.
* Is fairly stable
* HTTP2 approved in 2015 included compression, push, pipelining and multiplexing.
* Not just for HTML

Overview:

* Client/server – usually no response without request.
* Requests and responses have similar formats:

Request/status line includes HTTP version and status codes for response

Headers including the host, allowing multiple sites on same IP

Blank line

Request types include:

* GET
* POST (for forms)
* HEAD to check if a page exists
* PUT – replace (rare outside web services)
* PATCH – update (rare outside web services)
* DELETE – rarely used outside web services

Headers can include cookie values, URLs can include GET-encoded variables.

Response codes:

* 100-199 Informational, client should repond
* 200-299 Successful
* 300-399 – file has oved temporarily or permanently
* 400-499 – client error, e.g 404 not found
* 500-599 – server error

Introduction to Nodejs

Server-side scripting in Javascript (alternative to php, RoR etc)

Single-thread execution: non-blocking. – cannot handle multiple requests simultaneously.

Not designed for compute-heavy applications.

Package manager npm – largest ecosystem of open source libraries in the world

Also yarn package manager.

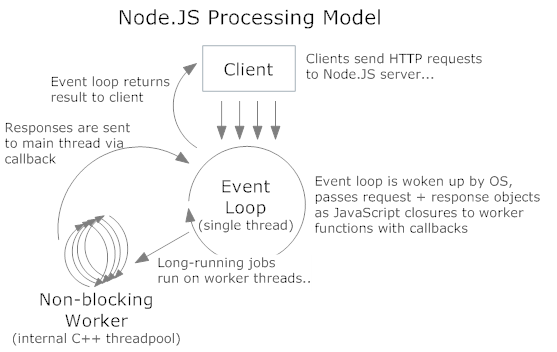
Not as widely available as php, but more than most other alternatives.

console.log(“Hello world”) <- same code as with client side but cannot be read from browser console as it is server-side. Uses terminal instead.

Event Loop

Node runs on an event loop. Callbacks are associated with events. Programs should be non-blocking so that callbacks are provided for things like blocks of data arriving from files or REST requests, or data arriving from a database request.

Callbacks can trigger new events; the next instruction is executed once all callbacks are complete.



Routing requests – express:

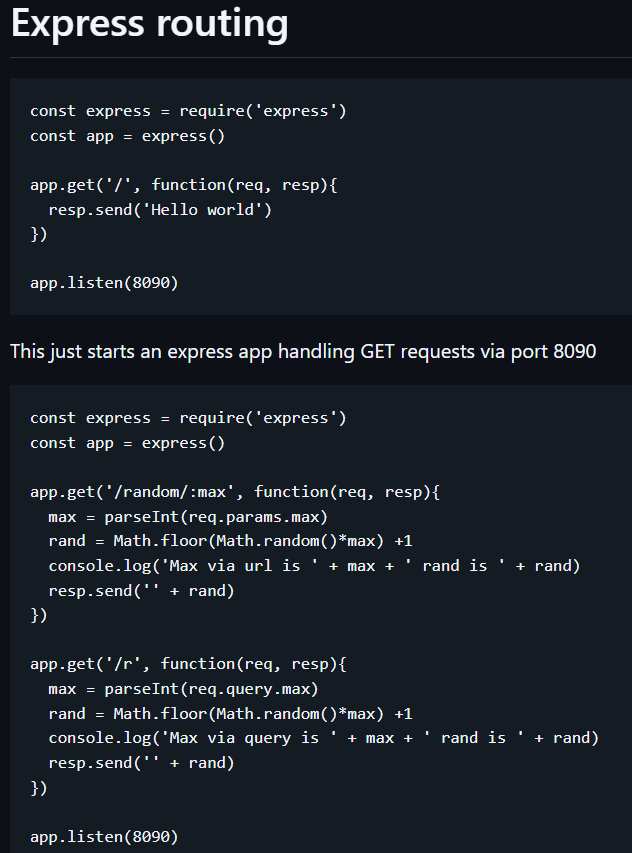
Variables can be GET encoded in URLs, and in REST APIs they are often included in the URL directly (e.g. Django urls can contain variables 😊). We want to be able to yoink these variables out of the URLs to use them in nodejs. This is done with a package called express.

Npm packages: npm is good 😊. Npm uses semantic versioning, and allows using different versions of packages for different projects. Package.json is a file for a project which should contain all dependencies for a project, and their version number.

Version numbering – X.Y.Z

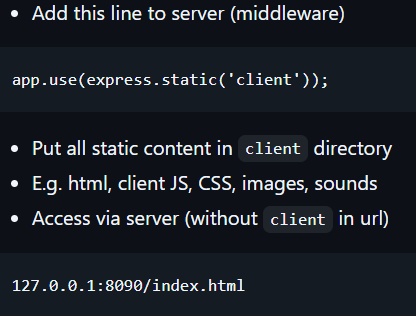
* Update Z when fixing bugs but not adding functionality
* Update Y when introducing functionality (and reset to 0) – only if backwards compatible, i.e. doesn’t break any code written using the previous version.
* Update X when changes not backwards compatible – major version.
* NEVER change code within a version once released – release any changes as a new version.
* Use X = 0 for pre-release versions, i.e. not entirely functional, “alpha”.

Package.json file stores info on all packages and dependencies needed for the node project. This is created by npm init, and all packages installed through npm will be added to this json file along with their version number.

The :max in the path of the first example is a variable that can be accessed using the parseInt(req.**params**.max) function.

The second example obtains the data from /r?max=<value> using req.**query**.max to obtain the value of max from the url.

To send client-side content through node:



Nodejs provides server side JS. NPM is package manager and servers can run locally or remotely. Use express 😊

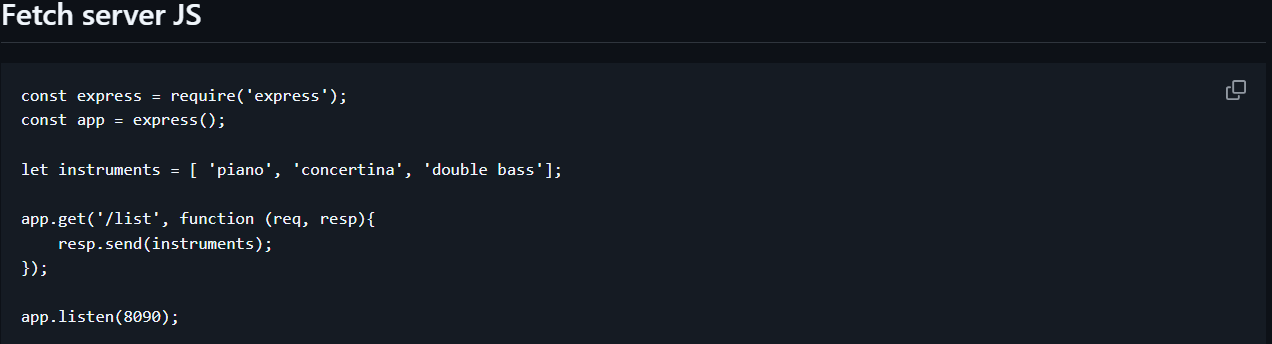
Javascript Client-Server Interaction

AJAX – Asynchronous JavaScript and XML – used to update part of a HTML page with the result of a request. (we mostly use JSON instead of XML nowadays)

* Client side JS sends request
* Page needs to work while waiting
* An event is triggered when a response arrives
* Content of response is used to update page

(single page app)

Fetch API – does AJAX natively.

^.then() function applied to the result of the preceding function, i.e. will only execute when the previous function returns a response. It then takes this response as a parameter.

Same-origin policy – a security feature used by browsers

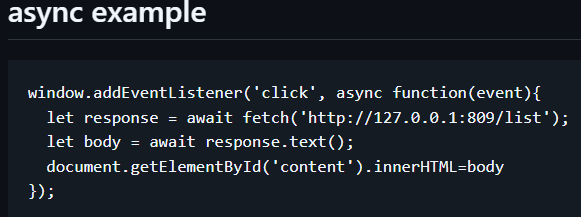
* Opening the HTML file directly (i.e. from the file system and not accessing it via http) and trying to make a request to the localhost: IP address returns an error.
* The security feature prevents javascript injection attacks, i.e. where people can embed malicious javascript code into their page
* Only allows asynchronous requests to source of page.
* To get around – serve everything from the same place.

Promises – provide successful and unsuccessful handlers for fetch().

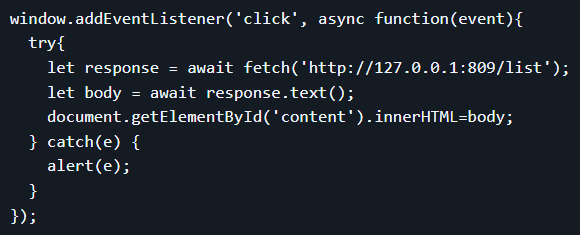
* .then() is for successful callbacks
* .catch() is for failure callbacks
* Promise objects are returned by .then() functions.
* A 404 is still counted as a success – returning a 404 is still returning something 😊
* **Use .catch() in coursework assignment for robustness**

Async and await:

* Different syntax to make it easier to write – doesn’t do anything special
* Use “async” keyword before “function” keyword

^this does the same as the previous example – better practice to write like this.

For exceptions just dump it in a try{} catch{} block 😊

Set a variable = require(“./path/file.json”) to yoink a json file.

REST

REST = Representational State Transfer

* Architechtural style for web services
* Provides interoperability – is language independent
* Uses HTTP methods – GET, POST etc
* Like a function call – parameters provided in URL (GET) or body (POST), results provided in body
* Published as an API.

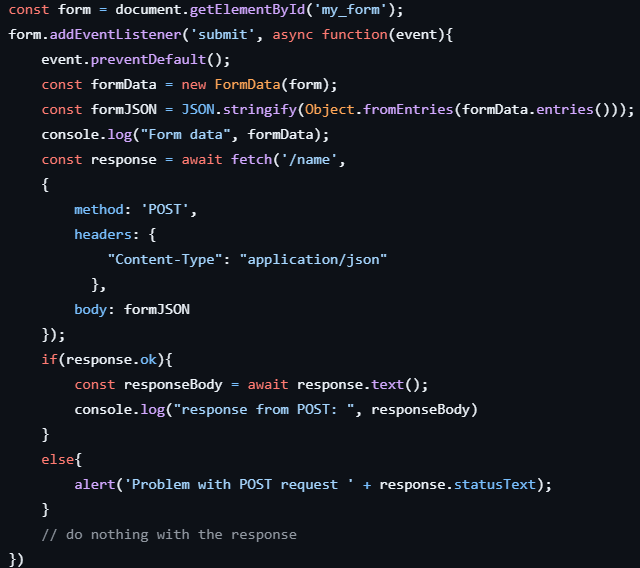
Implementing a REST API

* GET methods for listing/searching and detail
* POST methods for adding and updating
* Setup http response codes, e.g. 200, 404 etc
* Extract parameters, send response.
* For GET methods, parameters can be passed in the URL directly, or in the query string.
* For POST methods, a body parser is needed for url encoding and JSON. If using JSON (normal), app.use(express.json());

REST and Single Page App

* In a single page app, we reduce traffic by updating content rather than reloading the whole page with css, js etc.
* Access REST methods directly from the client
* Problem: default form submission loads the “action” page. Solution: event.preventDefault(); or event.stopPropagation();

Sending POST data from a form

* Use a [FormData](https://developer.mozilla.org/en-US/docs/Web/API/FormData) object to extract all inputs
* Turn this into an object with [Object.fromEntries](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/fromEntries)
* Turn this into a string with [JSON.stringify](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify)
* Make sure the headers in your fetch set the content type to be application/json
* Make sure your server is using the [express.json middleware](https://expressjs.com/en/api.html#express.json)
* See the [example server](https://github.com/stevenaeola/progblack_lectures/blob/main/nodejs_rest/server.js) and [example client](https://github.com/stevenaeola/progblack_lectures/blob/main/nodejs_rest/client)

Testing Node.js REST

Why testing?

* Testing does not prove the absence of bugs (cannot test every single possible input), best you can do is identify existing bugs.
* Can avoid introducing new bugs as new functionality is added -> repeated testing needed in agile environment
* Testing should be automated as far as possible.
* (it’s in the assignment ☹ )

Testing approaches:

* Manually using browser
* Using Postman (partly automatic)
* Using test scripts (automatic)

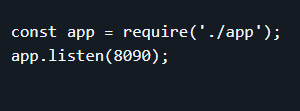
Browser-based testing:

* GET easy – just type in the URL
* POST slightly less easy – need to build a HTML form
* Other HTTP methods more difficult.

Postman:

* A web/desktop app
* Can define a set of requests (GET, POST or other) in a collection
* Include body parameters for POST, or use raw JSON.
* Can specify expected responses.

Automated Test Scripts:

* Recommend using Jest testing module
* Npm install –save-dev jest
* Tests are .js programs, name files name.test.js or in a separate tests directory – note name = name of js file that is being tested.
* Add test script to package.json, run nmp test to run test.
* Need to separate app from server to prevent tests from running the server – all server side code goes into app.js, server.js is just going to use:
* Making sure to use module.exports = app in app.js so it can be called by the require() in server.js
* See example test files for how to set up test file 😊
* Testing needs to have as much coverage as possible (coverage = how much functionality has been tested)
* Could measure this in terms of the space of possible inputs (black box) or the code that has been executed (white box), i.e. what % of lines of code are covered by the tests.
* Testing should combine the two and consider valid and invalid inputs

Code quality:

* ESLint – checks coding standards are applied
* Install with npm install eslint
* Uses a particular set of rules, e.g. standard