Supplementary note JS2-MAR23

Notes on asynchronous programming

JS can't be blocked in the browser

It is:

- Single threaded
- Non-blocking

Async tasks can be started but JS will carry on

Ask yourself what will be done with the data rather than assign to global var

Do what needs to be done in the async callback

Pass in alternate functions to the callback if you want to swap out different renders / filters

axios removes need for parsing response into json

async await - caveat:

Code ***looks*** synchronous but is not as it always returns a promise

Case study / async / CLI steps

node -v
v18.14.1 // to run fetch in node needs to be 18>

npm init -y

mkdir src

touch src/index.html

! **+ TAB** to populate HTML via Emmet

touch src/index.js

Link to script in head of landing page - include **type="module"** attribute

npm i -D tailwindcss postcss

//////////////
CREATE
npx tailwindcss init
//////////////
Add
```
  content: [
    "./src/**/*.{html,js,ts,jsx,tsx}",
  ],
```
//////////////
CREATE
.postcssrc
//////////////
Add
```
{
  "plugins": {
    "tailwindcss": {}
  }
}
```

//////////////
CREATE
touch src/index.css
//////////////
Add
@tailwind base;
@tailwind components;
@tailwind utilities;
Link in head of landing page

package.json

Replace
```
  "main": "index.js",
```

With
"source": "src/index.html",

Add
"scripts": {
"start": "parcel",
"build": "parcel build **—public-url ./**"
},

Test with static html and a simple console.log in index.js
npm start

Fixes build paths to bundled files (only need to run before a deploy):

npm run build

WHY DO WE USE FUNCTION EXPRESSIONS?

FUNCTION DECLARATION

function myFunc(){
//code to execute
}

FUNCTION EXPRESSION

var myFuncExpression = function(){
//code to execute
}

- unlike function declarations they are not loaded into memory during the first pass of the code
- so they will not be "hoisted" - they have to be declared nearer to (above) where they are used in the code
- they may be assigned to **const** and so made final
- they may be assigned to the **prototype** property of a function

constructor

clear online blog:

https://gomakethings.com/function-expressions-vs-function-declarations/

for the enthusiastic:

https://frontendmasters.com/courses/advanced-javascript/function-declarations-function-expressions-and-block-scope/
[course preview, paywall]

for grabs:
https://github.com/getify/Functional-Light-JS/blob/master/manuscript/ch2.md/#chapter-2-the-nature-of-functions

this is not the whole story - also concerns binding the context of 'this', but ES6 arrow functions have largely taken over - see Movies example code in 04-scope/04-bind...

scope and closures

"Closure is when a function remembers and accesses variables from outside of its own scope, even when that function is executed in a different scope." **Kyle Simpson**, from Functional-Light JS

---

links (tooling):

https://www.npmjs.com/package/json-server

http://dontpanic.website/tools

FURTHER READING AND RESOURCES

Kyle Simpson You Don't Know JS series
https://github.com/getify/You-Dont-Know-JS

Dan Abramov Just JavaScript (STRONG recommend)
https://justjavascript.com/

---

references - deep dives

divs - separation of concerns example code
https://codepen.io/lavenderlens/pen/mdXPvVG

Symbol, the new primitive
https://www.freecodecamp.org/news/how-did-i-miss-javascript-symbols-c1f1c0e1874a/

global symbol registry - fictitious concept
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Symbol#shared_symbols_in_the_global_symbol_registry

recent article going in to BIG detail on JS prototypes
https://levelup.gitconnected.com/prototypal-inheritance-the-big-secret-behind-classes-in-javascript-e7368e76e92a

Recent article - big detail - JS memory
https://www.zhenghao.io/posts/javascript-memory

Module exports and live bindings
https://jakearchibald.com/2021/export-default-thing-vs-thing-as-default/