Any references I make to folders or files in the code base can be followed through in my work-in-progress GitHub repository here:

https://github.com/breakingco/functional-visualiser

Problem analysis

In order to successfully implement this prototypal visual representation and comparison of functional versus imperative programs, through my initial research I have broken down the problem into what I believe are the individual components which all **must** be separately addressed in order to achieve minimum functionality. These are written here outside of technology considerations needed for actual implementation, and can be broken down into three areas:

Infrastructure

- writing a simple back-end server to serve the initial webpage of the single-page application (SPA);
- creating a build system / workflow for development:
    - on the back-end, server simulation for development and automatic restarting of the server simulator when server code changes; and
    - on the front-end, automatic transpilation of code that is not browser native, and refreshing of the browser, on code change;

Parsing and data creation for visual content generation

- choosing a language and parser library to create an abstract syntax tree (AST) of the functional/imperative code samples which are to be displayed, and then:
- deciding whether to use the resulting syntax tree as-is on the front end and plucking certain values from the structure to dictate what visual components display, or evaluating each step and then transforming each into a simpler subset that creates a data structure with a more direct description of visual components, if necessary;
    - determining how to traverse the resulting data structure in a sequential order;
    - determining how to dynamically evaluate the code at each step in a flexible manner to create the representations for visual display (i.e., if possible not have to separately code a process for every type of instruction in the code, which would still break as soon as new features were added to the code's language);
- the security concerns of evaluating this code: deciding whether outside code beyond the simple written examples should be loadable, and whether this parsing should be implemented on the frontend or backend; and
- writing sample code in both functional and imperative styles for the purpose of providing content to the parser.

Visual display of data

- Designing a skeleton interface for the app;

- Designing the visualization and deciding how each step of a program should be shown in a layout that best represents the flow of data, and also in a way that highlights the difference between functionally and imperatively 'guided' data flow
- Creating an engine for displaying the elements of the program (e.g., a function, a variable, an 'if' statement, the passing of a parameter from one function to another), in a way that creates a base whereby each new element can be added one by one as a 'building block' component to gradually build up the functionality of the visualization;
- Creating and styling a component and animation for each 'building block';
- Deciding whether any other features should be present (e.g., giving the ability for a user to upload their own code for evaluation, changing the speed of the code stepping through, changing ways in which the code could be viewed).

Technology survey

At this stage, my time has been mainly spent investigating, evaluating and learning the various technologies that I have decided to use for this project, and completing the build system aspect. I have spent the past year working exclusively using the AngularJS framework daily for my year-long internship working in front-end development, but did not want to use this framework for the project. Rather, I wanted to take this opportunity away from work to learn new tools and frameworks that could expose me to a greater range of workflows, with the possibility to then recommend and eventually use these back in my professional work.

Language choice

My plan has been quite ambitious, with my scope of new learning needing to have been reduced somewhat given the time allowed. I have decided to switch from using Clojure on the backend to JavaScript. Although I would like to learn to code in a purely functional style using Clojure, it is not technically necessary for this task, so long as the language used gives me the ability to create code samples in a functional style so that it can be compared to the same code written in an imperative style.

JavaScript still has higher-order functions and has many powerful built-in functional features, so is suitable for this task. The first prototypal example of a functional/imperative example can be found here.

This is just a sum function and very basic, but I think ultimately these will need to be fairly simple as the limiting factor will be how well the UI can express the differences between the two kinds of functions - in this work I potentially don't anticipate writing more complex functions, rather just piping some array through a series of these simple transformations and then spending more time working on the interface to actually display these.

Stack

I have chosen to use an isomorphic JavaScript setup, whereby the frontend and backend share the same codebase and are written in the same language. Using NodeJS on the backend and Browserify on the frontend to compile library and written JavaScript into a single file for serving to the browser, I can use Node's included and popular package manager, npm, to manage a single set of library dependencies usable by both the frontend and backend.

I have also chosen to use the newest version of Javascript, ES6, on both the frontend and backend. This is mainly to take advantage of its new module system. Although it has added more time to my initial research stage getting to grips with it, the idea was that it should give me a clean way to organise and package the code for each new visual 'building block' as mentioned above as I create and add them to the project.

Other choices

Besides these major choices, below is a summary of the technologies I have researched and chosen, along with notes on how completely they are investigated or integrated into my project.

- SASS, a CSS pre-processor which enables me to more neatly package the related style class information up with each component as I create them.
- BrowserSync, for live-reloading of the browser;
- Gulp, a build tool . This automates the watching of my codebase for changes, and accordingly re-runs associated necessary preparation tasks (to create the content in `/public/build`. The code in `gulpfile.js` was written from scratch so I now can fully use this tool.
- D3, for taking the AST data steps and calculating the layout for each display component.
- React (with JSX for allowing HTML-like syntax in Javascript), for creating the base interface - it should have a fairly minimal involvement in this project given that D3 will be doing most of the work.
- acorn, for generating an estree-spec AST. I investigated and tried the newer Shift-AST format, but there is less documentation and tools available for it so I think I should go with the established JS AST creator for now.

Proposed methodology

So far I have learned how to use ES6, enough Node to set up the basic server and put the build system and folder structure in place. I am now further breaking down the tasks from 'Proposed Analysis' using trello.com, a simple project management app. A rough summary of the steps to follow are:

- visually design the interface layout on paper;
- decide how D3 should manage this layout using pre-provided functionality wherever possible.
    - The D3 force-layout graph would seem like a good choice for representing the AST of a whole program (and was what I originally hoped to present as a prototype here), but it does not lend itself to displaying the actual flow of data through a time-based sequence. However, none of the baked-in graphs supplied with D3 seem to be good for this, so I think I will have to create this myself, somehow.

- create visual function components by:
  - designing the first component (representing a function) in the frontend, and displaying it using D3 (this is currently most basically implemented without any style/design yet in `/public/modules/components/functionBlock`.
  - parse and examine the AST from my simplest function `sumDemo` examples (in `/modules/examples`) and decide how to map the AST information given to that particular frontend-created component;
  - add that information to the D3 engine so as to adjust the layout to accommodate it or that component.

I would then iterate through the above for each new component, adding functionality to the D3 engine as I went. I don't think it will become apparent as to how best to step through the actual code, evaluate it and animate the changes in the D3 graph until I have a few of these components to play with.

Risk factors

Minimum functionality

One of the hard things here is choosing some bare minimum of functionality, and then trying to build upon it to the stage where a program can be represented. I don't think it would be possible to create visual representations for every aspect of a Javascript program for this project, but I will need some engine underneath that can accept new components (basically, an HTML/CSS/JS 'block' that represents a single aspect of the program) and can incorporate it into the layout in a more generic fashion.

Visual design

How best to represent the passage of data in a functional manner, in a layout that highlights how functional programming is 'functional' seems to be a fairly unique proposition. At the core I see a comparison and contrast, whereby representations of the same program coded functionally and imperatively are shown side by side, with the visualization for the imperative one somehow appearing more 'messy' or 'displeasing' to highlight the comparative elegance of the functional version.

Ideally, a number of prototypes would be built and tweaked to evaluate the best, but I will probably need to settle upon one way of attempting this shortly and just go with it.

In the time available, I plan to first very simply represent the AST of the same program, one coded functionally and one imperatively, using two D3 force-directed graphs side by side. Then, I can try as a first step to perhaps dynamically show the clean passing of variables as parameters between functions in the functional version, whilst the global variables in the imperative version could perhaps be 'floating' outside the functions, and could perhaps flash red if modified by a function with a different scope to show undesired side effects. Even to get to this stage though will still take some time due to the functionality that must be in place underneath, so I am concerned about not ending up with anything near a complete representation in the time period.

Evaluating and representing data flow through AST representation

Whilst I think that mapping out the display of a single AST in D3 is more a question of time than complexity (just creating a visual component for each function, a variable, etc, and the relevant part of the AST that causes it to appear), trying to determine how to actually evaluate and step through the tree in a sequential fashion mimicking the actual operation of a program ('meta programming') and displaying an update to the graph seems to be quite challenging.