**Imperial Visualisations**

****

*Suite Guide*

Welcome to the Imperial Visualisations developer team! It’s expected that a new developer has no previous experience in any web-based language; all we expect is knowledge of code and the enthusiasm to learn. This document will take you through what you need to think about when creating a new visualisation and the key things to consider when starting to make improvements on existing ones.

**Finding the overall goal**

When creating or improving a visualisation, the most important thing is to have clarity on what the visualisation is trying to achieve. Spend some time thinking about the key realisations you want your user to have, and how to place them in a well-ordered manner such that the learning process is as clear as possible. Think about what the user already knows, what knowledge do you assume? Do they need refreshing on some of the key concepts that'll be used later?

The title of a visualisation is very important. Be clear on actually what the content of a visualisation is. So, for example, instead of having “Fourier series” as a title, something like “Deconstruction and reconstruction of functions through Fourier series” would be much better.

**Sub goals/learning steps**

Early in the creation of a visualisation it’s important to think of what the key learning steps you want the user to have are. Think about how many steps you want to make, currently visualisations have between 4 and 6 pages, typically with one ‘learning step’ per page. For the content you will be demonstrating you may feel this is about right, but it’s possible that this is either too many or too few. Perhaps you only need 2 or 3 pages to fully explain the topic, or for complicated topics that build up through a lot, 9 or 10 pages might be a better idea.

A good way to list these steps and to decide how many to have is to write down sub-titles that you think each page deserves, where this sub-title describes what that page’s learning step will be in just two or three words. This makes it more obvious to the user the main result they need to take from that page as well as making the visualisation easier to navigate if they need to come back to anything. It also helps you keep track of the structure of the learning journey when creating a visualisation.

When creating a visualisation its important to consider the style of page Imperial Visualisations uses. Currently the left third of the page is a scrollable page which will explain the content, so any theory or derivations would go here. The middle and right third of the page will then be dedicated to the interactive part of the visualisation, anything that can be clicked, dragged or played with goes here. If this isn’t clear check out the Fourier series visualisation, this is a working model for the style we’re going for.

Note that you don’t want the user to have to scroll on the right-hand side of the page. The idea is that the user scrolls on the left-hand side of the page reading the content/explanations, and when they’ve reached a section where the next visualisation is more relevant, the right-hand side updates and shows new interactive material which they’ll go on and use. Done properly this gives the learning journey very good structure, as interactive material is revealed to them as they build up their knowledge of the topic at hand.

*Khan academy* and *Coursera* are great places to look for well thought out learning journeys, check these out!

**Creating new code**

When considering the experience that the user of a visualisation has, it’s inevitable that at some point you’ll need to create new content. When doing this, make sure to spend plenty of time thinking about what you’re actually trying to tell the user, talk with others who are working on it and make sure that the content you’re putting across is correct, makes sense and is worthwhile to the visualisation. It’s too easy to have an idea and get going straight away without properly considering what you’re trying to tell the user.

Once you’re happy with the content you’re trying to make, think about all the maths that you’ll need to do along with it. Make some sketches and write down everything relevant that’s going on. Be clear on what parameters a user will be inputting. Perhaps a user will be giving coordinates (x1,y1) and they’ll have an output (x2,y2), keeping in mind which variables are specified (and so known) and exactly what you want out. Sounds obvious but it’s easy to get lost! When working out the maths be careful with special cases that arise when you divide any two things. For example, if you wanted arctan(y1/x1) to find an angle, you’d need to create the special case for if x1=0, as JavaScript would have a problem with dividing by zero, even though you might not always consider this a problem. Algebraically we’d just expect pi/2, so for this you’d have to create the special case to return pi/2 for when x1 = 0.

When the functionality team later starts looking at our JS code, we want to make their life as easy as possible. Most importantly is to have the code structured into blocks that are **maths**, **interaction** and **calling**. We want this structure as it allows people who perhaps don’t understand the mathematics/theory we’re using to optimize the code without worry. It should work under the lines of “I have a set of functions I’ve defined that when I input the users input, I get the numerical values I need out” before doing any plotting. The interaction code is then everything that’s needed to get the JavaScript and HTML files to talk to one another, it finds takes the user input (which is found through HTML), inputs it into the maths block and takes the output and puts it into any plots you want to have. Finally the calling section updates everything you need, so if the user has for examples changed the value of some slider, the calling section of the code will respond to this change and call a single function which does everything you need it to in order to update the plots.

Having the code ready in this general structure makes the work of those optimizing the code and checking for redundancies much easier as it’s obvious from the beginning what everything is doing, and provided you’ve done the maths correctly, it allows those who don’t know the subject to do optimisation.

**Pro tips**

One of the most frustrating things to have happen is to lose track of good code. If ever significant changes are made to a file, make sure to push onto GitHub! Use the ordering: commit, pull,push on Github. Having a commit in the GitHub history when stuff is working is great, because if you accidentally make some mistakes in your code and can’t change them, then reverting to these commits (save points) are exactly what you need. Always keep this in mind!  
  
When multiple people are working on the same visualisation suite, make sure that there is only ever one person working on the same file. This stops merge conflicts happening when the people who have edited the same file try and get up to date with one another. It’s not terribly difficult to resolve a merge conflict, but it’s easier to just not have to do this!

**Creating a repository**

To create a repository, simply go to the Imperial Visualisations Github and click the green “new” button on the righthand side. This will take you to a new page where you can name the repository and give it a description. You want to make the repository public and initialize it with a readme. You can then create the repository.

**Step 1: suite structure**

The first and most important thing you need to do when creating a suite is to decide on the structure the suite is going to have. Once you’ve started writing code, the structure of the suite becomes difficult to change, and so make sure you know exactly what visualisations you want to go in the suite. The actual content of each visualisation is less important at this point, but you should have at least a good idea of what you want before you set off.

Once you’re happy with the structure you want your suite to have, you need to set up the file structure clearly. Start by naming carefully naming each section to your suite on paper. For example, “1Introduction”, “2Divergence\_free\_fields”. Each section requires three files, these files are: 1.) the HTML for the text on the left-hand side. 2.) The HTML for the interactive content on the right-hand side. 3.) The JavaScript for the interactive content on the right-hand side. Create these files for each section, for example “1Introduction\_textonly.html”, “1Introduction\_interactive.html”, “1Introduction.js”. The JavaScript files should go into their own “scripts” folder. You now have the basic file structure set up.  
  
On top of this, there are a lot of files that we use quite consistently throughout suites, with only slight modifications between the uses of each. In particular, we use MathJax, “object” and a folder we’ve called “styles”. These are easy to copy over. Clone a pre-existing repository (for example Fourier series) and open the files on your computer. Copy the “styles” folder over into your new repository, this has some CSS set up in it. Now enter the “scripts” folder of the pre-existing suite, you should see a “MathJax-Master” folder, copy this into your new “scripts” folder, this takes some time as it is a large folder. Finally, copy over the file named around the word “objects”, in different suites this is named differently (as changes to this file will most likely be made during the creation of your suite). This should be found in the scripts folder of the pre-existing visualisation, copy it over to yours, name it accordingly, and then you’re all done creating the file structure for the suite!

Push these changes onto GitHub. It’ll take a while to do this as MathJax-Master is such a big folder, but this should **all** be done first. Once these initial files have been made, you should create the branches “development” and “style”. Changes that are made during the suite creation should be made in the development branch, and once the suite is ready for passing on to the style team, they’ll be working in the style branch. Creating all the branches at the start doesn’t make any difference before the style team starts their work, but once they have, it makes it easy to see in the Git history which team has made what changes to the files.

It’s usually best if one person (the suite creator) does all of the above. Only once all of this is done should people start writing code for the suite.  
  
**Step 2: Starting to code**

Before writing anything, make sure you’re familiar with the content that already exists on the Imperial Visualisations website. If a pre-existing visualisation in a suite looks somewhat like what you want one of your visualisations to be, then having a look at the code for the pre-existing one will definitely give you a good help. Don’t be afraid to start by copying the content of pre-existing files into your currently empty files and having a play around with how the old code works. This is actually how a lot of visualisations start off!  
  
Once you’ve had a look at someone else’s code, copy over some interaction structure. It’s useful to understand how the JS and HTML files talk to one another, but if you’re stuck, taking other people’s files and just understanding how their files work is a big help. If you’ve done this, make sure that the names of the files referenced in the “brains” of the HTML (down the bottom) are the ones you want them to be (they need to be referencing the files locally to your computer), so just copy and pasting the content of their files won’t usually work, as the HTML will be unable to find the files that have (incorrectly) been asked for.  
  
Now that the interactions between files have correctly been set up, you should be able to get going.  
  
**Step 3: Creating visualizations**

At the time of writing of this document, the structure of Imperial visualisation suites are such that the majority of explanatory content goes in the left third of the screen, where scrolling occurs. The interactive part to the relevant part of the visualisation then goes on the right two thirds of the screen.   
  
When creating the visualisation, it’s important that you truly understand the content of what you’re trying to put across. You’ll most likely find that you go through many ‘realizations’ about the subject you’re talking about, some perhaps meaning that the content of what you’ve put in the visualisation is misleading or even incorrect.

The best way to avoid letting this becoming a problem is to nicely write up the content (left hand side) of your part of the visualisation. It should explain everything it needs to for the reader to understand the content. It should also explain how to correctly use and understand the visualisation. Doing this first allows you, the creator of the visualisation, to correctly deeply understand the content of what you’re letting the user to visualise and stops you wasting time having to change the content of the visualisation.  
  
At the same time as this however, it’s important that the right hand side takes up the attention of the user, is easy and intuitive to use. The concepts being displayed should be obvious for the user to at least anticipate before reading the text, and the changes that occur due to moving different sliders/buttons should easily be noticed right from the start. Key to this is picking sensible ‘default’ settings that the user sees upon reaching this visualization in the suite. For example, in Fourier series, if one of the key points is to do with how the number of terms changes our approximation to the function, then choosing an intermediate value of 3, which shows some wiggles, but not too many, is much better than choosing 0, where the user can’t see anything yet and has to interact in order to at least guess what’s going on, or 100, where the approximation is also perfect, and so the user will just get confused upon seeing it.

**Step 4: Reviewing the visualizations**

Any particular visualization must be reviewed before it is incorporated into a suite. This keeps up the quality of content! The best way to do this is to ask other members of your team to look at what you’ve produced. At this point it’s best if you can get someone with “clean eyes”, either someone who’s not seen the subject you’re trying to teach yet, or someone whose not seen your visualization trying to teach the content you’re doing yet. This gives you the feedback of how easy it is to understand what’s been said.

By now, lecturers for the course you’re teaching, or people who may be interested in showing this content to students should be contacted, asking if they’re interested in hearing your ideas on how you’re visualizing the content of the suites. Showing them what you have/what you’re planning to have at this sort of stage is crucial, as it stops you moving too far without any misconceptions. You might find it surprising how many incorrect ideas make it this far, where it takes a lecturer to catch out any inconsistencies!

**Step 5: Handover to style**

Once all individual visualisations have been created, both the text and interactive content (including anything for an intro or an outro you may want), it’s time to hand over to the style team. The style team will piece together all the individual pieces of visualizations and turn it into the full suite for your subject.  
  
At this stage, clarity is needed! Where will files they’re using be saved? Have you told them the structure you want to the suite? All of this needs to be translated to them. At the time of writing of this document, the convention is for all visualization creation to be done in a “development” branch, and then once handover occurs, the style team work in the “style” branch.  
  
Once this is done, you just have to wait a bit and the whole suite will come together!

**Step 6: Review of the suite**

Finally, once the whole suite has been pieced together. You need to go through and ask the question, does this flow nicely? Does it all work? Does everything do what I want it to? Is it easy to navigate etc etc. If you (and all the team responsible for the suite creation) are happy with this, then it’s time to send it the lecturers the content and ask for a review!  
  
Once you’ve found a lecturer who is going to review your suite, send them a link to the correct location on GitHub (that’s how it’s done at the moment, in the future it should be able to be done through the website), explaining how they should be able to view the content. Along with this message you should ask them to think of a set of questions. The main questions are:  
Are the learning objectives clear?  
Is the content correct?  
Are there any bugs?

Give the lecturer some time to think about these questions when they can play with the suite alone. Then set up a meeting with the lecturer (whenever is convenient for them). It’s nice to have about 3 people who contributed to the suite turn up in order to interview the lecturer about what they thought of the suite. Take a notepad and pen! Notes from these meetings are super important. Once the meeting is over, put the notes on Trello as soon as you can (if you don’t do it soon, you’ll never get around to it). This allows the whole group of people to act on the comments and produce a finalized version of the suite.

**Step 7: Celebrate!**

Congratulations, by now you have a suite that has the seal of approval of at least one lecturer. Well done, time for the next suite 😉