Full Stack Syllabus

CFGdegree Full-stack stream

Please read the Syllabus. Please.

Document purpose:

This is a brief document is intended for:

- Tools you'll need for the stream
- Setting expectations for the next few weeks and the course goals
- Outlining the learning objectives (and why these were set in relation to your career) and how you may be assessed
- Recommendations for assessments and advised reading list
- Lay out a few important announcements (Plagiarism, etc)
- Anything else relevant

Recommended Tools

For this course, please:

- Download and setup an IDE like Atom or Visual Studio Basic - even Notepad works! We'll be writing .html, .css, or .js based files - something like PyCharm isn't as suitable as a result as it's intended for .py files instead.

Course Goals

The point of this stream is to equip you with front-end development skills: it's named 'Full Stack' as you'll finish with knowledge necessary for back-end development (and practice them during the stream), though the first half of the course will focus on front-end development. The goal is that you essentially have the skills needed to specialise in any area of development you'd like afterwards due to exposure to a full-stack setup.

For the course, you'll be learning the following:

1. Standard web development technologies (HTML, CSS, JavaScript)
2. Brief overview of NodeJS (specifically via ExpressJS)
3. React (and related libraries)
4. Angular, Linux and React Testing (all very lightly, and in theory only; not enough time for anything deep)
5. Big O Time & Space Complexity
6. Data Structures
7. Common algorithms (Search, Sort, etc)

Learning Objectives and how you may be assessed

To be able to:

- Develop your own simple website (via Point 1 and 2 above)
- Create a React application, after extending what we learnt from standard web development (Point 3)
- Have some knowledge of other topics that may be come up in your career (Point 4)
- Possess critical Computer Science theory (Point 5 to 7)
    - This is a particularly important area; technical interviews, particularly at FAANG (Facebook, Amazon, Apple, Netflix, Google) or other tech-focused companies, usually require knowledge of these areas
    - For example, you may be given a piece of code and asked to assess its performance (which needs Big O understanding) or to write a algorithm that accomplishes xyz objectives (which requires data structures and algorithms knowledge)

As such, the first three bullet points will teach you the knowledge needed to build applications whilst the last bullet point intends to teach you (some of) the sheer theoretical knowledge that underpins the discipline.

You'll be assessed through homework (to test your independent research skills and knowledge of algorithms), assessment and group project. The group project will primarily be used to assess your React knowledge (hence why it is weighted highly) whilst the assessment will be used for both React + other CS areas (e.g. data structures knowledge, ability to write an algorithm in response to a question, etc). The reason it's structured this way is because it is **difficult** to ask you to write a React application in a 2-hour assessment, hence why the bulk of it is assessed through the group project.

Recommendations for assessment/s and advised reading list

For the assessment, recommendation is:

- Use pen & paper, or a whiteboard (e.g. a digital one online) wherever you can! When looking at a problem, try to sketch it and **solve it in your head manually** (if you didn't write any code, but had to describe the solution, how would you do it?). Once you've done that, try to see if you can recreate this mental solution into a step-by-step coding solution; this is effectively how all coding answers are usually written (they're just a replication of some mental solution you've thought of in your brain!).

Reading list wise, if you have the time try:

- Any material on OOP

- Any material on Data Structures and Big O Complexity
    - Recommend MIT's 'Introduction to Algorithms' on this - simply pick out the lecture videos that you'd like to visit. Its scope is slightly past us due to its complexity, but the lecturer is **brilliant** and goes into an amazing level of depth onto the topics.
    - Link: https://ocw.mit.edu/courses/6-006-introduction-to-algorithms-spring-2020/

- Clean Code by Robert C. Martin
    - Outlines how code should be written - haven't personally read it (I as the course creator) but I've had a ridiculous number of senior engineers recommend it: on my reading list, I recommend you check it out too!

Any other important announcements: Plagiarism

This is a very quick PSA: unless made explicit, **please do not copy from other sources - your work should be of your own.** If in doubt, please reference and quickly notify your instructor so that they can advise you whether they'll allow it or not for your submission.

Please note that we will deduct heavily for plagiarism - academic integrity is prioritised highly for this stream and violations of it are unfortunately punished. **It is ridiculously easy to detect plagiarism within this discipline - please do not do it as it complicates marking for us (we do not like deducting! Plus academic integrity > random mark).**