CS Project Ideas

Things to build alongside your Aston modules

How to use this

Each brief says what the project is, what to build and what you should get out of it. Pick one that fits what you know now, and finish the core requirements before you try the extra challenge. Each one lists what you should know first and roughly how long it takes, so you can fit it around your coursework.

The time estimates and levels are rough guides, not official module ratings. These are practice projects, not assessments, and the tools listed are only suggestions.

Rules for every project

- Use Git and commit as you go. A clear history is part of the work.
- Write a README that explains how to set it up and use it.
- Test the main logic, including edge cases and bad input.
- Write a short design note on the choices you made and the limits you know about.
- Set a time limit for the core. If you pass one and a half times the estimate, cut features instead of carrying on.
- Only use systems, networks and data you own or have permission to use.

Projects 06, 22 and 25 work well for a pair or small team.

Want feedback?

If you build one of these, I am happy to check it. Email me the link to your GitHub repo and ask me to take a look, and I will email back with feedback.

hello@hasnainamir.com

github.com/hasnain-amir

Contents

DG1IPD Introductory Programming for Digital Science

01	Student Budget Tracker	Beginner | 6-10 hours
02	Quiz Program	Beginner to Intermediate | 8-12 hours
03	Library Lending Simulator	Intermediate | 10-15 hours

DG1IAD Internet Applications and Databases

04	Club Event Booking Website	Intermediate | 20-30 hours
05	Study Resource Hub	Intermediate | 20-30 hours
06	Volunteer Shift Scheduler	Advanced | 30-40 hours

DG1CYF Cybersecurity Fundamentals

07	Password Strength Checker	Beginner | 6-10 hours
08	Encryption Demonstrator	Intermediate | 10-15 hours
09	Web App Threat Review	Intermediate | 12-20 hours

DG1FDS Fundamentals of Digital Systems

10	Logic Gate Simulator	Beginner to Intermediate | 8-12 hours
11	Microcontroller Reaction Timer	Intermediate | 12-18 hours
12	Binary Operations Simulator	Intermediate | 12-18 hours

DG1AID Foundations of AI and Data Science

13	Campus Route Finder	Intermediate | 12-18 hours
14	Dataset Analysis	Beginner to Intermediate | 8-14 hours
15	Exam Timetable Solver	Advanced | 25-35 hours

DG1MDS Introductory Mathematics for Digital Science

16	Finite-State Vending Machine	Beginner to Intermediate | 6-10 hours
17	Probability Simulator	Intermediate | 8-12 hours
18	Matrix Transformation Tool	Intermediate | 10-15 hours

DG2OOP Data Structures, Algorithms and Object Oriented Programming

19	Data Structure Visualiser	Intermediate | 15-25 hours
20	Delivery Route Planner	Advanced | 25-40 hours
21	Search and Sort Benchmarks	Intermediate | 12-18 hours

DG2OPS Secure Networks and Operating Systems

22	Secure Local Chat	Advanced | 25-40 hours
23	System Resource Monitor	Intermediate | 10-15 hours
24	Protocol Security Comparison	Intermediate to Advanced | 15-25 hours

DG2BUC Business and Enterprise Computing

25	Small Business Stock Tracker	Intermediate | 15-25 hours
26	Service Booking Business Case	Intermediate | 15-25 hours
27	Predictive Maintenance Prototype	Advanced | 20-30 hours

Further challenges

A1	Accessible University Society Site	Beginner to Intermediate | 8-12 hours
A2	Documentation Microsite	Beginner to Intermediate | 8-12 hours
A3	Responsive Design System	Intermediate | 10-15 hours
A4	Dashboard Layout Challenge	Intermediate | 10-15 hours
A5	Kanban Board	Intermediate | 12-18 hours
A6	API Explorer	Intermediate | 10-15 hours
A7	File Organisation Assistant	Beginner to Intermediate | 6-10 hours
A8	Mini Data Pipeline	Intermediate | 10-15 hours
A9	Concurrent Task Scheduler	Advanced | 15-25 hours
A10	REST Service with Persistence	Advanced | 20-30 hours

Part I: First-Year Modules

DG1IPD | Introductory Programming for Digital Science

Module focus: Variables, control flow, functions, lists, classes, input/output and error handling.

01 | Beginner | 6-10 hours

Student Budget Tracker

A tracker for your money. You enter what you earn and what you spend, and the program tells you what is left. It is a good first project because it uses most of what you learn early on: input, checks, lists, files and sums that have to come out right.

What to build

A command-line program that records income and spending, puts each item in a category and shows what is left of your budget.

Requirements

- Add, edit and delete transactions
- Show a monthly total for each category
- Check that amounts and dates make sense
- Save records to a file and load them when the program starts

What you should end up with

The program, a sample data file, and tests for the sums and for bad input.

If you finish early

Recurring costs and CSV export.

Before you start

Know first: Variables, loops, functions and reading and writing files.
Tools: Python.

---

02 | Beginner to Intermediate | 8-12 hours

Quiz Program

A quiz that reads its questions from a file. It sounds easy until someone types nonsense or the file has a mistake in it. You decide how the file is laid out, keep track of scores over several attempts and make sure the program never crashes.

What to build

A quiz program that loads questions from a file and tracks how a player does across several attempts.

Requirements

- A file of questions in a format you describe
- Questions asked in random order
- A score worked out at the end of each attempt
- Several attempts per player
- Bad input handled without crashing

What you should end up with

The program, a description of the question file format, and test cases.

If you finish early

Timed rounds and stats for each topic.

Before you start

Know first: Lists, dictionaries, functions and reading files.
Tools: Python or Java.

---

03 | Intermediate | 10-15 hours

Library Lending Simulator

A small library system. Books can be borrowed and returned, and there are rules: a copy can only be out once, a person can only borrow so many, and books can come back late. You turn those rules into classes and code so that a wrong state cannot happen.

What to build

A model of a small library with books, borrowers and lending rules.

Requirements

- Add books and users
- Lend and return books
- Due dates
- Stop the same copy being lent twice
- Search the book list

What you should end up with

The program, short notes on how the classes fit together, and tests for awkward cases, such as returning a book that was never lent or borrowing past the limit.

If you finish early

Save records between runs and charge fines for late books.

Before you start

Know first: Classes, collections and dates.
Tools: Python or Java.

DG1IAD | Internet Applications and Databases

Module focus: Client/server web development, relational design, SQL, security and deployment.

04 | Intermediate | 20-30 hours

Club Event Booking Website

A website where a club lists its events and members book a place. Like most web apps, it is forms, a database and some rules. The tricky rule is capacity: if two people book the last place at the same moment, only one of them should get it. You design the database, build the pages and make that rule work.

What to build

A responsive website that lists events and lets people book places, backed by a relational database.

Requirements

- Event listings
- Capacity checks that still work when two people book the last place at the same time
- Making and cancelling bookings
- Checks on the server for everything users enter

- Forms that work with a keyboard and a screen reader
- A properly designed database schema

What you should end up with

A working web app, an ER diagram, the schema and setup instructions.

If you finish early

User roles and waiting lists. After that, put it online and run the tests automatically on every push.

Before you start

Know first: Basic HTML, CSS and JavaScript, SQL joins and constraints, and one server-side framework.
Tools: HTML, CSS, JavaScript, a server-side framework and SQL.

---

05 | Intermediate | 20-30 hours

Study Resource Hub

A shared site for revision notes, links and past papers. Anyone can add something and anyone can search for it. It is a plain create, read, update and delete app, so it is a good place to get the basics right, especially search and handling what users type in.

What to build

A searchable catalogue where students add and organise revision material.

Requirements

- Create, view, edit and delete resources
- Subject tags
- Search and filter
- Checks on the server for all input
- Database queries that use parameters, not text pasted into the query

What you should end up with

The app, either online or easy to run locally, and notes on the database design.

If you finish early

A moderation queue and full-text search.

Before you start

Know first: Everything from project 04, plus parameterised queries.
Tools: HTML, CSS, JavaScript, SQL and any backend.

---

06 | Advanced | 30-40 hours

Volunteer Shift Scheduler

A sign-up system for a charity. Volunteers are free at different times and each shift needs a set number of people. Nobody should be booked into two shifts at once, and no shift should go over its limit. This is the hardest web project here because those rules still have to work when lots of people sign up at the same time.

What to build

A shift booking system for a small charity where volunteers have different availability.

Requirements

- Volunteer profiles
- A limit on people per shift
- Checks for clashing times

- A booking process that checks capacity and clashes inside a database transaction
- A test that sends many requests for the last place at once
- A layout that works on a phone

What you should end up with

The app, the data model, a security checklist and testing notes that include the many-requests test.

If you finish early

Notifications, and permissions based on role.

Before you start

Know first: Project 04 or similar experience, and database transactions.
Tools: A web stack and SQL.

Note

Works well for a pair or small team.

DG1CYF | Cybersecurity Fundamentals

Module focus: Authentication, access control, cryptography, network and web security.

07 | Beginner | 6-10 hours

Password Strength Checker

A tool that checks how good a password is. Most strength rules are weaker than they look, and this project shows why. The program gives honest feedback, and you write down what it cannot catch and what kind of attacker you have in mind.

What to build

A password checker for learning. It points out common weaknesses and never stores a real password.

Requirements

- Rules for length and common patterns
- Clear feedback for the user
- Safe example passwords only
- A short note on the kind of attacker the checker assumes

What you should end up with

The checker, a list of its limits, and test cases.

If you finish early

Compare passphrases with passwords and simulate rate limiting.

Before you start

Know first: Strings, functions and simple regular expressions.
Tools: Python or JavaScript.

---

08 | Intermediate | 10-15 hours

Encryption Demonstrator

Old ciphers like Caesar and substitution are easy to write and easy to break. You build one, then break it using letter frequencies. Doing both shows why modern encryption works the way it does.

What to build

A teaching tool that shows how simple substitution ciphers work and why they are not safe to use today.

Requirements

- Encrypt and decrypt
- Key handling
- Frequency analysis on sample text you supply
- A clear note that none of it is safe for real use

What you should end up with

The demo, worked examples, and a short write-up of what the ciphers do well and badly.

If you finish early

A separate demo using a well-known modern cryptography library.

Before you start

Know first: Strings, dictionaries and modular arithmetic.
Tools: Python.

---

09 | Intermediate | 12-20 hours

Web App Threat Review

A written project. Before you can fix security problems you have to find them and decide which matter most. You take an app that was built to be insecure, run it on your own machine and write a threat model like a security team would, ending with a plan to fix what you found.

What to build

A review of a deliberately vulnerable app, run in an isolated setup on your own machine, such as DVWA or OWASP Juice Shop.

Requirements

- A list of assets
- A list of threats
- Risks ranked by how serious they are
- Controls for login and input checking
- A plan to fix the problems

What you should end up with

The threat model, a risk table and recommended fixes.

If you finish early

Apply two of the fixes to your local copy and check that they work.

Before you start

Know first: Login, access control and injection, as covered in DG1CYF. You also need to be able to run Docker or a virtual machine.
Tools: A document for your write-up, and Docker or a virtual machine for the demo.

Note

Only test systems you own or have written permission to test.

DG1FDS | Fundamentals of Digital Systems

Module focus: Binary, gates, truth tables, sequential circuits, embedded I/O, interrupts and Unix/bash.

10 | Beginner to Intermediate | 8-12 hours

Logic Gate Simulator

Every digital system is built from gates. This simulator lets you join AND, OR, NOT and XOR gates into circuits, see their truth tables and check that a circuit does what you meant. It turns the logic from lectures into something you can run.

What to build

An interactive simulator for AND, OR, NOT and XOR gates and the circuits you can build from them.

Requirements

- True and false inputs
- Truth tables made automatically
- Joining gates into circuits
- Checking a circuit against the truth table you expect

What you should end up with

The simulator, truth tables and some example circuits.

If you finish early

Half and full adders, and simplifying circuits.

Before you start

Know first: Booleans, functions, and recursion or simple data structures.
Tools: Python or JavaScript.

---

11 | Intermediate | 12-18 hours

Microcontroller Reaction Timer

This one runs on real hardware. An LED lights up after a random wait, you press a button as fast as you can, and the board times you. Mechanical buttons bounce, which makes simple code give wrong times. Fixing that is the main thing you learn.

What to build

A low-voltage reaction game made from an LED, a button and a timer.

Requirements

- A random delay before the LED lights
- Reading the button
- Timing the response
- Showing the score
- Debouncing the button

What you should end up with

A circuit diagram, the code, a demonstration and a test log.

If you finish early

Rewrite it with interrupts and compare the timing with polling.

Before you start

Know first: Basic electronics (LED, resistor, button) and C or MicroPython.
Tools: A microcontroller board and the language it supports.

Note
You need hardware: a board such as an Arduino or Raspberry Pi Pico, an LED, a resistor, a push button and a breadboard. Get these before you start.

---

12 | Intermediate | 12-18 hours

Binary Operations Simulator

A tool that shows what a processor does at the lowest level. Numbers are fixed-size patterns of bits, and adding can overflow. It shows why 255 plus 1 can give 0, and how the CPU flags that it happened.

What to build
A teaching app that steps through simple machine operations on registers and binary numbers.

Requirements
- Convert between binary and hex
- Bit shifts
- Arithmetic with a fixed number of bits
- Carry and overflow flags

What you should end up with
A simulator you can run, and worked examples.

If you finish early
A simple ALU with condition flags.

Before you start
Know first: Binary, hexadecimal and bitwise operators.
Tools: Python or JavaScript.

DG1AID | Foundations of AI and Data Science

Module focus: Search, shortest paths, constraints, data collection, transformation and visualisation, introductory machine learning.

13 | Intermediate | 12-18 hours

Campus Route Finder

Finding the shortest route is the usual first lesson in search. You draw a campus as a graph, with buildings as points and paths as links that have lengths, then write the code that finds the best route. Trying both breadth-first search and Dijkstra's algorithm shows when the simpler one is enough.

What to build
A fictional campus stored as a graph, with a program that finds routes between buildings.

Requirements
- A graph of the campus
- Breadth-first search or Dijkstra's algorithm, whichever fits the data
- The path and its length as output
- A clear message when there is no route

What you should end up with
The program, a sample graph, a diagram that explains one route, and a short comparison of the algorithms.

If you finish early
Accessible routes, or more than one thing to optimise at once.

Before you start

Know first: Dictionaries, queues and the idea of a graph.
Tools: Python.

---

14 | Beginner to Intermediate | 8-14 hours

Dataset Analysis

Real data is messy. It has gaps, spellings that do not match and values that make no sense. You pick a public dataset, clean it and write an analysis that someone else can run from the raw file to get the same results. Your notes on what was wrong and what you did about it matter as much as the charts.

What to build

An analysis of a public dataset with real quality problems. Pick one that is properly messy, because cleaning it is the point.

Requirements

- Deal with missing and inconsistent values
- Summarise how the values are spread
- Clear charts
- A written list of every data problem you found and what you did about it

What you should end up with

A notebook or script that goes from raw data to results, the cleaned data, and a short report of what you found.

If you finish early

Look at bias, correlation versus cause, and sampling problems.

Before you start

Know first: The basics of pandas and matplotlib.
Tools: Python, pandas and matplotlib.

---

15 | Advanced | 25-35 hours

Exam Timetable Solver

Timetabling is a puzzle with rules. Exams share students, rooms hold only so many people, and some combinations cannot work. You write a solver that builds a valid timetable, or tells you why there is not one. It is the hardest project in first year.

What to build

A solver that schedules made-up exams so no student has two at the same time and no room is overfull.

Requirements

- Write the rules down in code
- Find clashes
- Build a valid timetable
- Say clearly when no valid timetable exists

What you should end up with

The solver, sample inputs, and a written reason for the algorithm you chose.

If you finish early

Reduce the gaps between a student's exams, or use rooms more efficiently.

Before you start

Know first: Recursion, backtracking and graph colouring.

Tools: Python.

DG1MDS | Introductory Mathematics for Digital Science

Module focus: Sets, numeral systems, finite-state automata, probability, matrices and reasoning.

16 | Beginner to Intermediate | 6-10 hours

Finite-State Vending Machine

A vending machine has only a few states, and strict rules about moving from one to the next. Drawing it as a finite-state machine makes its behaviour clear and easy to test. It is a direct use of the automata theory from the maths module.

What to build

A vending machine written as states and transitions, tested with valid and invalid sequences of events.

Requirements

- A fixed list of states
- Rules for moving between states
- Payment and change
- A defined result for an invalid move

What you should end up with

A state diagram, the code, and tests for the transitions.

If you finish early

Generate tests that cover every transition.

Before you start

Know first: Functions, and dictionaries or enums.
Tools: Python or Java.

---

17 | Intermediate | 8-12 hours

Probability Simulator

Probability is easier to believe once you have seen it happen. You roll virtual dice thousands of times, compare the results with the numbers you worked out, and watch the gap shrink as you roll more.

What to build

A program that compares worked-out probabilities with simulated results for dice, coins or cards.

Requirements

- Random trials
- Probabilities you calculated by hand
- Charts of how often each result came up
- Several different numbers of trials

What you should end up with

The code, the charts, and an explanation of why the results settle down as the number of trials grows.

If you finish early

Confidence intervals, and a fixed random seed so results can be repeated.

Before you start

Know first: Basic probability, random numbers and plotting.
Tools: Python.

18 | Intermediate | 10-15 hours

Matrix Transformation Tool

Graphics software uses matrices to move, turn and resize shapes. You build a tool where you enter some points and a transformation and see the result drawn. It makes matrix multiplication easier to picture, and shows why the order of transformations matters.

What to build

A tool that shows how matrices change points and shapes in two dimensions.

Requirements

- Enter coordinates
- Move, rotate and scale
- A before and after plot

What you should end up with

An interactive demo and worked examples with real numbers.

If you finish early

Combine transformations and show that the order changes the result.

Before you start

Know first: Matrix multiplication, and homogeneous coordinates, which you need to write a move as a matrix.
Tools: Python or JavaScript.

Part II: Second-Year Modules

DG2OOP | Data Structures, Algorithms and Object Oriented Programming

Module focus: Inheritance, interfaces, exceptions, abstract data types, searching and sorting, recursion and Big-O.

19 | Intermediate | 15-25 hours

Data Structure Visualiser

Data structures are easier to understand when you can watch them work. You build a stack, a queue, a linked list and a binary search tree yourself, each behind an interface, and make a view that shows every insert, remove and traversal one step at a time. The complexity table at the end links the code to Big-O.

What to build

Your own stack, queue, linked list and binary search tree, with a step-by-step view of how they work.

Requirements

- Insert, remove and traverse
- Each structure behind its own interface
- Checks on input
- A step-by-step view of each operation

What you should end up with

The source code, diagrams, unit tests and a table of time complexities.

If you finish early

Add a hash table and compare it with the others.

Before you start

Know first: Interfaces, generics, recursion and JUnit.
Tools: Java.

---

20 | Advanced | 25-40 hours

Delivery Route Planner

A delivery firm wants routes that cover a list of stops. You model the stops and drivers as objects, store the map as a graph and use pathfinding to plan routes. It uses object-oriented design and algorithms together, and you can see the program slow down as the map grows.

What to build

A model of deliveries and locations that uses graph search to find routes that work.

Requirements

- An object-oriented model of the problem
- The map as a graph
- Pathfinding
- Showing the route
- Error handling

What you should end up with

The application, a class diagram, tests and an analysis of the algorithm.

If you finish early

Compare different shortest-path methods on bigger and bigger graphs.

Before you start

Know first: The collections framework, graphs and shortest-path algorithms.
Tools: Java.

---

21 | Intermediate | 12-18 hours

Search and Sort Benchmarks

Big-O says how an algorithm slows down as the data grows. Here you check whether your measurements agree. You write several sorting and searching algorithms, run them on generated data and plot the results against what the theory predicts. Timing code needs care, or the numbers will mislead you.

What to build

Several sorting and searching algorithms, timed and compared with what Big-O predicts. In Java, run the code a few times before you start timing, repeat each test and use the median time, or your results will not be reliable.

Requirements

- Several algorithms
- Generated data of different sizes and orders
- Tests that check the output is correct
- Timing that gives similar results when repeated
- Plots

What you should end up with

The code, the tests and a short report that compares the results with the Big-O predictions.

If you finish early

Worst-case inputs and memory use.

Before you start

Know first: Sorting and searching algorithms, and Big-O.
Tools: Java.

DG2OPS | Secure Networks and Operating Systems

Module focus: Processes, memory, access control, network protocols and secure network programming.

22 | Advanced | 25-40 hours

Secure Local Chat

A chat server and client you could run on a home or lab network, with encrypted connections and logins. It brings networking, threads and security together. The main lesson is to use libraries that are known to work instead of inventing your own security.

What to build

A small chat system for a private network that you control, with protected connections. Use an established TLS library and do not write your own cryptography.

Requirements

- More than one client at a time
- Encrypted connections
- Login
- Safe handling of errors
- Event logs

What you should end up with

The server and client code, a deployment guide, tests, and a record of your security decisions and why you made them.

If you finish early

User roles and limits on connections.

Before you start

Know first: Sockets, threads, and the basics of TLS and certificates.
Tools: Python or Java.

Note

Works well for a pair or small team.

---

23 | Intermediate | 10-15 hours

System Resource Monitor

A program that watches CPU, memory and running processes and warns you when something goes over a limit. It shows what an operating system will tell you about the programs it runs, and you get to write a program that runs for a long time and shuts down cleanly.

What to build

A local monitor for CPU, memory and process figures that raises alerts at limits you can set.

Requirements

- Regular measurements
- A summary of processes
- Alerts when a limit is crossed
- Logging
- A clean shutdown

What you should end up with

The program, sample measurements, and a note on how it works on different operating systems.

If you finish early

Graphs on a dashboard, and data export.

Before you start

Know first: Processes, file input and output, and a library such as psutil.
Tools: Python.

---

24 | Intermediate to Advanced | 15-25 hours

Protocol Security Comparison

What changes when traffic goes from plain text to encrypted? You write small client and server programs both ways, capture your own traffic and see what someone watching could read in each case. Then you write up the controls that follow.

What to build

Small client and server programs on your own machine, used to measure how protocol and settings affect security.

Requirements

- Explain the TCP and naming assumptions involved
- Compare plain text with protected connections

- Capture only your own test traffic
- Suggest controls

What you should end up with

The local demo, a test plan, your findings and a table of fixes.

If you finish early

Simulate network faults and harden the service.

Before you start

Know first: Sockets, TCP basics, DNS and packet capture.
Tools: Python and a written report.

DG2BUC | Business and Enterprise Computing

Module focus: Business problems, information systems, metrics, innovation, AI and enterprise integration.

25 | Intermediate | 15-25 hours

Small Business Stock Tracker

Small shops lose money on stock they forgot to reorder and stock they ordered too much of. This project designs a system to stop that. In a business module the thinking counts as much as the code: who will use it, what they need, and how you would know it works.

What to build

A simple stock system. A made-up shop is fine, but a real small business, with the owner's agreement, makes a stronger project.

Requirements

- Stock entries
- Reorder warnings
- Simple reports
- Clear measures of success
- A list of what each user needs from the system

What you should end up with

A prototype, the requirements, a process diagram and a review of benefits and limits.

If you finish early

Estimate what it costs to adopt and write a plan for managing the change.

Before you start

Know first: A web stack or Python, and basic SQL or spreadsheet skills.
Tools: A web stack or Python.

Note

Works well for a pair or small team.

---

26 | Intermediate | 15-25 hours

Service Booking Business Case

Many small service businesses, such as barbers, tutors and trainers, still take bookings by message. You propose something better and argue for it. What problem does it fix, who uses it, what does it cost, what could stop people using it, and how would you measure whether it worked? The prototype backs up your argument.

What to build

A proposed booking system for an independent service business, backed by a business case.

Requirements

- A clear description of the problem
- Workflows
- User roles
- A prototype
- Reasons people might not adopt it
- The KPIs you would track

What you should end up with

A prototype, the business case, a risk register and how you would evaluate it.

If you finish early

Compare building it with buying one, using realistic costs.

Before you start

Know first: Basic prototyping in Figma or code, and spreadsheet skills.
Tools: Figma or a web prototype, and a spreadsheet.

---

27 | Advanced | 20-30 hours

Predictive Maintenance Prototype

Sensors on vehicles or machines can show wear before something breaks. You make up that data, write a simple rule or model that raises alerts, and count how many alerts are false. That count decides whether anyone would trust the system.

What to build

A prototype that uses sensor data to flag maintenance risks in a made-up logistics company.

Requirements

- A business goal
- Simulated measurements
- An alert rule or a simple baseline model
- An analysis of false alerts
- A plan for monitoring

What you should end up with

A proof of concept, a map of how it fits into the business process, and an assessment of the benefits.

If you finish early

Test a simple machine learning model against fixed thresholds.

Before you start

Know first: Handling data in Python, and basic statistics.
Tools: Python or diagrams, with made-up data.

Part III: Further Challenges

Optional projects beyond the module briefs, grouped by technology.

HTML

A1 | Beginner to Intermediate | 8-12 hours

Accessible University Society Site

A society website is a realistic first site with several pages. Done properly, with real headings, labels and landmarks, it works for someone using a screen reader or only a keyboard as well as for someone with a mouse.

What to build

A multi-page society website built with the right HTML elements and made accessible.

Requirements

- Proper heading order
- Navigation
- A form with labels that work
- Tables only where the content is a table
- Page metadata

What you should end up with

The site and a validation report for each page.

If you finish early

Full keyboard use, tested without a mouse.

Before you start

Know first: Core HTML.
Tools: HTML.

A2 | Beginner to Intermediate | 8-12 hours

Documentation Microsite

Documentation is mostly structure. You document a tool you know well and use HTML elements for what they are meant for, so people can find their way around and search engines can read the pages.

What to build

A documentation site for an API or a tool you know well.

Requirements

- Use article, section, nav and code elements properly
- At least five pages
- The same navigation on every page

What you should end up with

The site and a short note on the structure you chose.

If you finish early

A linked table of contents, and metadata that helps search engines.

Before you start

Know first: Core HTML.
Tools: HTML.

CSS

A3 | Intermediate | 10-15 hours

Responsive Design System

A design system is the set of parts a site is built from. You make buttons, cards, layouts and form styles once and control them with custom properties, so changing one colour or spacing value changes everything.

What to build

A small set of reusable interface parts.

Requirements

- Buttons, cards, layout helpers and form styles
- Custom properties for colour and spacing
- Grid and flexbox

What you should end up with

A stylesheet and a demo page that shows every part.

If you finish early

Support for small screens, clear focus states and reduced-motion settings.

Before you start

Know first: HTML and CSS layout.
Tools: CSS.

A4 | Intermediate | 10-15 hours

Dashboard Layout Challenge

The goal is a dashboard that stays easy to read from a phone to a wide monitor, using only CSS. It is mostly about spacing, hierarchy and the grid.

What to build

An analytics dashboard layout of your own design, made with CSS only.

Requirements

- A layout that works at three screen widths
- Spacing that scales
- Data that is easy to read

What you should end up with

The layout and a screenshot at each width.

If you finish early

Print styles and a dark mode.

Before you start

Know first: CSS grid and flexbox.
Tools: CSS.

JavaScript

A5 | Intermediate | 12-18 hours

Kanban Board

A task board makes you manage state in the browser: what is on the board, what is being dragged, and what has been saved. It is a good way to learn events without a framework.

What to build

A task board with drag and drop.

Requirements

- Add, edit and delete tasks
- Move tasks between columns
- Save the board in the browser

What you should end up with

A working app and a README.

If you finish early

Keyboard controls, filtering and undo.

Before you start

Know first: The DOM, events and local storage.
Tools: JavaScript.

A6 | Intermediate | 10-15 hours

API Explorer

Most front ends use someone else's API. You fetch live data and build an interface that copes with a slow connection, an error and a search with no results, as well as the case where everything works.

What to build

An interface that fetches data from a public API and lets you filter it.

Requirements

- A layout that fits any screen
- Separate loading, error and empty states
- Filtering

What you should end up with

A working app and a README that names the API and its usage limits.

If you finish early

Cache responses and retry failed requests with a growing delay.

Before you start

Know first: Fetch, and promises or async/await.
Tools: JavaScript.

Python

A7 | Beginner to Intermediate | 6-10 hours

File Organisation Assistant

A tidy-up script is only useful if you can trust it, so it has to show what it would do before it does anything. Making dry run the default is the whole point.

What to build

A script that sorts a folder by file type or date. It must do a dry run by default, which only reports what it would do.

Requirements

- Dry run by default
- Sort by extension or date
- A log of changes

What you should end up with

The script and a sample log.

If you finish early

A config file, undo and automated tests.

Before you start

Know first: File paths, and the os and pathlib modules.
Tools: Python.

Note

Only test it on copies of folders, never your real ones.

A8 | Intermediate | 10-15 hours

Mini Data Pipeline

Most data work is moving records around, checking them and reshaping them. A small pipeline that checks its input and gives the same report every time covers the basics.

What to build

A pipeline that imports CSV data, checks it, changes it and produces a report.

Requirements

- Checks with clear error messages
- Changes to the data
- Summary statistics
- An exported report

What you should end up with

The pipeline code with sample input and output.

If you finish early

Turn it into a command-line tool with unit tests and logging.

Before you start

Know first: CSV files, functions and basic testing.
Tools: Python.

Java

A9 | Advanced | 15-25 hours

Concurrent Task Scheduler

Programs that do several things at once fail in ways that are hard to repeat. You build a job queue with worker threads, then test it for the problems that only show up under load.

What to build

A program that runs made-up jobs using a fixed-size queue and a pool of worker threads.

Requirements

- A queue with a size limit
- Worker threads
- Tracking the status of each job
- A clean shutdown

What you should end up with

The source code, tests and a note on how you handled concurrency.

If you finish early

Tests for thread safety, cancelling jobs and throughput.

Before you start

Know first: Threads, synchronisation and java.util.concurrent.
Tools: Java.

A10 | Advanced | 20-30 hours

REST Service with Persistence

Many back ends are a REST API in front of a database. You build one for a small catalogue, with input checks and tests that call the real endpoints.

What to build

A small API for a resource catalogue.

Requirements

- Create, read, update and delete endpoints
- Input checks
- Saving to a database
- Integration tests

What you should end up with

The source code, tests and a description of the API.

If you finish early

Login, pagination and generated API docs.

Before you start

Know first: The basics of a Java web framework, and SQL.
Tools: Java.

Before you call it done

- Working code or a prototype you can demonstrate
- A README with setup and usage instructions
- Tests, including edge cases
- The reasons for your design and its known limits
- Notes on ethics, privacy and security where they apply
- Screenshots, diagrams or performance figures, if useful