**Coding for GUIs (JavaScript edition)
Final Project: A complete application with GUI**

*Please refer to the online lesson plan on the DT Hub to access all website links and additional resources.*

This is the final project in a series of lessons to incorporate Graphical User Interfaces (GUIs) into your General Purpose Programming. The series follows on from the Visual To Text Coding lesson series.

Drawing on skills from both previous courses, this lesson provides prompts, ideas and steps for designing and developing both the GUI and coded algorithm for an application that runs on a webpage with HTML, CSS and JavaScript.

The structure of this lesson utilises a simplified Design Thinking process, where students begin by choosing or defining a problem to solve, then proceed through identifying requirements, designing algorithms and user interface, and implementing code development. You may wish to augment this structure or apply a rigorous design process already used in your school.

# **Curriculum links**

Links with Digital Technologies Curriculum Area

| **Strand** | **Year** | **Content Description** |
| --- | --- | --- |
| Processes and Production Skills | 7-8 | Design the user experience of a digital system (AC9TDI8P07 )Generate, modify, communicate and evaluate alternative designs (AC9TDI8P08) Design algorithms involving nested control structures and represent them using flowcharts and pseudocode (AC9TDI8P05) Trace algorithms to predict output for a given input and to identify errors (AC9TDI8P06) Implement, modify and debug programs involving control structures and functions in a general-purpose programming language (AC9TDI8P09) |

# **Assessment**

In assessing code in languages like JavaScript, consider a rubric that brings in important skills for General Purpose Programming.

* Download a sample rubric in Word or PDF format.

# **Learning map and outcomes**

In this lesson, students will:

1. propose or choose a GUI application and identify requirements,
2. prepare a **GUI design** with a mockup,
3. prepare an **algorithm design** with a flowchart or pseudocode,
4. **develop** the application from scratch in HTML, CSS and JavaScript.

# **Starting point**

For this project, you'll start by choosing one of three options:

**Option 1: Household chore allocator.** Your older sibling has recently moved into a shared house with two other people. They need a way to fairly allocate chores to each household member.

The list of chores is already known. Some chores take a large amount of time, while others take a smaller amount of time. Some happen daily, while others happen weekly or fortnightly.

The three housemates want the chores to be allocated so that the overall work hours per person is as even as possible across a fortnight. Also, each household member should be able to specify one chore that they *won't* do.

*If you choose this option:*

* + *your problem has already been stated,*
	+ *you will be provided with a starter project (HTML, CSS and JS) containing just the raw data about chores: their names, frequencies and work time,*
	+ *your solution code will need to deal with* ***arrays*** *(lists).*

**Option 2: Shell game.** You will be creating a simplified digital version of the shell game.

In each round of this game, the player is presented with three upturned cups. A shell (or ball) is underneath one of them chosen at random. **The cups are not moved around as they would be in the traditional confidence trick.**

Using a GUI with images that change when clicked on, the player must try to guess the cup with the shell. A scoring system should reward different points if the player guesses first time or second time in each round, then give the total score after a fixed number of rounds.

*If you choose this option:*

* + *your problem has already been defined,*
	+ *you must start your HTML, CSS and JavaScript from scratch,*
	+ *you will need to source or create appropriate images.*

**Option 3: Your choice.** You must come up with your own problem or opportunity for a digital solution to address via a GUI application, and clearly define this problem before continuing.
Note that this project assumes a single-screen interface, not multiple webpages. To avoid being too ambitious, consider past examples in this course as well as Options A and B above.
 *If you choose this option:*

* + *you must first identify a problem or opportunity and define it,*
	+ *you must start your HTML, CSS and JavaScript from scratch.*

# **Identify requirements**

Carefully read over the problem / scenario you have chosen.

Now, write a list of **requirements** for your solution. This is a list of clear, concise statements that someone could use to begin designing and developing the solution, even if they did not have access to the original scenario blurb. It should be as complete as possible.

Consider:

* functional requirements – What are the main things the solution *has* to do?
* non-functional requirements – What should be included to make the user experience effective and efficient?

# **Design user experience**

Create an annotated mockup for the user interface of your solution. (Optionally, you might create 2 or 3 alternative designs.)

Remember:

* This is a single screen interface, not multiple webpages.
* We have not covered how to code advanced layouts through HTML or CSS. For simplicity, your interface can flow from top to bottom on the page, but you may wish to investigate how to place controls next to each other or in specific places on the page.
* Consider the most appropriate GUI controls available to you in HTML and JavaScript, including:
* text input fields
* spinners / updowns
* buttons
* radio buttons / option buttons
* checkboxes
* dropdown lists

SIDEBAR – GUI mockups

Mockups or wireframes are used during user interface design to visually plan or sketch the basic look and functionality.

Sometimes multiple "frames" will be drawn, to show how the interface changes when the user navigates and makes choices within it.



Importantly, mockups can be **annotated** with notes and arrows to explain details about the look and feel (eg. fonts, colours) or to explain functionality (eg. this option appears when this button is pressed).

Tools for wireframing range from free and simple to highly complex paid tools intended for user experience professionals.

Some free options for school students:

* Draw on paper. Use one colour for the interface elements and another colour for annotations. Sometimes grid paper can help.
* Free online tool diagrams.net (click **+ more shapes** to add GUI Mockup elements).
* Presentation software like Microsoft *PowerPoint* or Google *Slides* tends to allow more freedom of element placement than word processing software.

Next, create a **user experience flowchart** or **story**. Using flowchart symbols or just text, describe the steps the user will go through when interacting with your solution, from start to finish. (Note, this is not to be confused with an algorithm flowchart for describing an algorithm).

# **Design algorithm(s)**

As you have been working on your user experience design, you have probably been thinking about the algorithm(s) needed to make your solution work.

Now, use a flowchart or pseudocode to design the **main algorithm(s)** in your solution. This is only the core work of the program. You do not need to describe every short bit of code that might be needed to respond to every action of the user.

eg. For OPTION A, the main algorithm takes the pre-written chores data and the GUI input from the housemates about which chore they have selected not to do. It goes through the data and tries to fairly allocate chores to each housemate. Finally, it displays each housemate's list of allocated chores.

Consider:

* What triggers the algorithm to start? It will probably be triggered when the user presses a button or perhaps when the page first loads.
* What input data or variables are needed, if any?
* What is the output of the algorithm, and what will be done with it?
* What decisions are needed, if any?
* What loops are needed, if any?

**Algorithm flowcharts** and **pseudocode** were introduced in the Visual To Text Coding lesson series. See Lesson 2 for an example of each.

# **Development**

Now it's time to code your program. Open an appropriate environment for HTML, CSS and JS, such as the online repl.it environment used in this course.

If you chose OPTION A, use this starter project in the repl.it environment. Otherwise, start with a fresh project.

Here are some tips to remember:

* Take things step by step to get some results as you go. You don't have to make everything work in one go. The previous lesson in this series demonstrates coding a GUI application step by step.
* Test your code as you go.
* Use the **console.log** command to display the values of variables when you, the programmer, need to know them. Output for the user should be displayed on the webpage itself.

# **Resources**

* Online environments for creating webpages with HTML, CSS and JavaScript:
	+ JSFiddle – simple interface that hides linking HTML code, also used in Visual To Text Coding lesson series
	+ repl.it – shows complete HTML to reflect offline approach, and allows uploading of images and other files for use in webpages
* Introductory courses
	+ HTML tutorial – An introduction to HTML
	+ CSS tutorial – An introduction to CSS
	+ Visual To Text Coding lesson series – The predecessor to this learning sequence introduces JavaScript as well as Python.
* JavaScript CheatSheet (Tip: Press the little blue tabs to move Variables, Basics, Strings and Data Types to the top.)