COMP1811 Scheme Project Coursework Specification

COMP1811 (2024/25)	Paradigms of Programming	**Contribution:** 30% of module
Module Leader/Moderator Yasmine Arafa/Andy Wicks	Practical Coursework 2: Scheme Project	**Deadline Date** Monday, 10/03/25
This coursework should take an average student who is up-to-date with tutorial/lab work approximately 35 hours. Feedback and grades are normally made available within 17 working days of the coursework deadline.		
Learning Outcomes: A. Understand the programming paradigms introduced and their applicability to practical problems. B. Apply appropriate programming constructs in each programming paradigm. C. Design, implement, and test small-scale applications in each programming paradigm. D. Use appropriate tools to design, edit, and debug programs for each paradigm.		

Plagiarism is presenting somebody else's work as your own. It includes copying information directly from the Web or books without referencing the material, submitting joint coursework as an individual effort, copying another student's coursework, stealing coursework from another student, and submitting it as your own work. Suspected plagiarism will be investigated and if found to have occurred will be dealt with according to the procedures set down by the University. Please see your student handbook for further details of what is/isn't plagiarism.

All material copied or amended from any source (e.g. internet, books) must be referenced correctly according to the reference style you are using. Code snippets from open-source resources or YouTube and any code generated by generative AI (e.g. ChatGPT) must be acknowledged appropriately.

Your work will be submitted for electronic plagiarism checking. Any attempt to bypass our plagiarism detection systems will be treated as a severe Assessment Offence.

Coursework Submission Requirements

- An **electronic copy** of your work for this coursework must be fully **uploaded by 23:30 on Monday, 10/03/25.** Submissions must be made using the Scheme Project Upload link under "Coursework Specification and Submission" on the Moodle page for COMP1811. Your grade for the coursework will be capped at 40% if you fail to submit by the deadline unless you have an accepted EC.
- You must upload **two SEPARATE files** for this coursework: **a)** a **zip file containing** the .rkt file needed to run **your Scheme code in**, a **screencast** explaining your work and showing the code running, AND **b) a PDF version of your report**. In general, any text in the document MUST NOT be an image (i.e. must not be scanned) and would normally be generated from other documents (e.g. MS Office using "Save As ... PDF"). A penalty will apply for the use of screenshots of code in place of code text, or if two separate files are not uploaded or named correctly. **The maximum size for each upload is 2GB**.
- Make sure that any files you upload are virus-free and not password protected or corrupt, otherwise, they will be treated as null submissions.
- All coursework must be submitted as above. Under no circumstances can they be accepted by academic staff.

The University website has details of the current Academic Regulations, including details of penalties for late submission, procedures for Extenuating Circumstances, and penalties for Assessment Offences. See https://gre.ac.uk/policies/undergraduate-and-postgraduate-taught

Coursework Regulations

- If you have been granted Extenuating Circumstances (ECs), you may submit your coursework up to 10 working days after the published deadline without penalty. However, this is subject to your claim being accepted by the Faculty Extenuating Circumstances Panel.
- Late submissions will be dealt with in accordance with University Regulations.
- Coursework submitted more than two weeks late may be given feedback but will be recorded as a non-submission regardless of any extenuating circumstances. The only exception to this is where your EC claim outcome has granted you a deferral.
- **Do not ask lecturers for extensions to published deadlines - they are not authorised to award an extension**.

Please refer to the University Portal for further detail regarding the University Academic Regulations concerning Extenuating Circumstances claims.

Coursework Specification

Please read the entire coursework specification before starting work.

This is a group development coursework. **You are expected to work in pairs to implement your project** but should **work individually to write the required report**.

Scenario

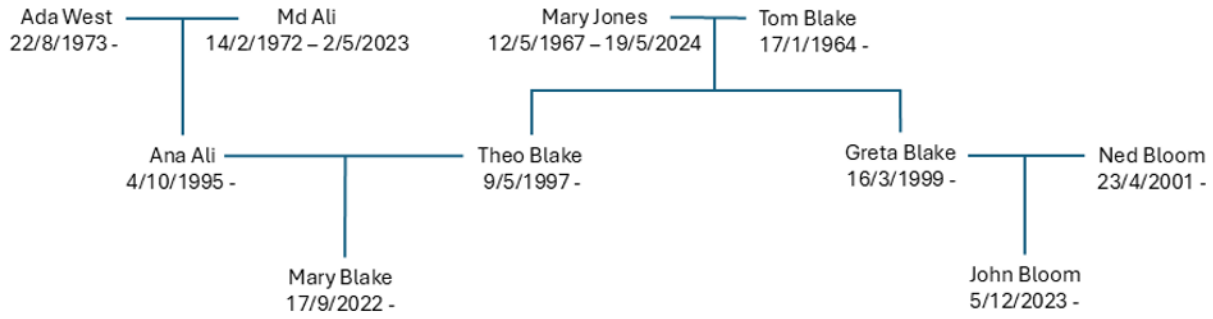
This coursework follows the same scenario used in Coursework 1: to create and handle a digital family tree. The objective of your development is to create a family tree given the maternal and paternal branches of the family tree below and implement the specified features to handle and retrieve information about the family.

The coursework requires the functional programming paradigm to be applied in the analysis of a family tree using the Scheme programming language. You are required to work in pairs formed from within your own lab group and each of you will develop a set of functions to process the family tree.

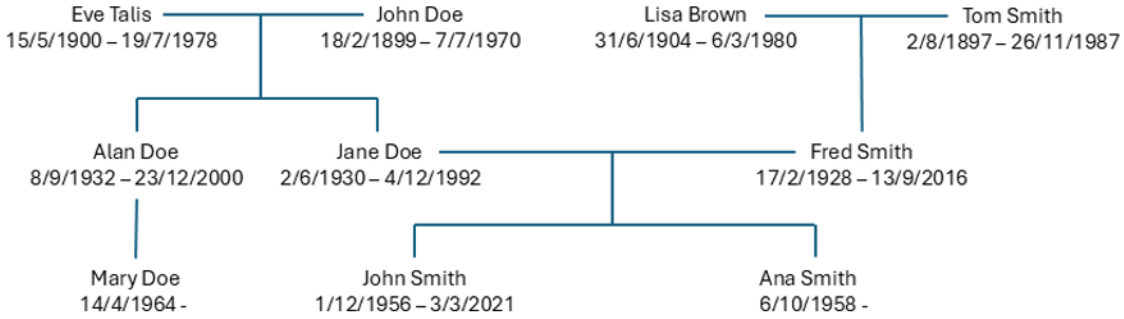
Stub functions and the data for the maternal and paternal branches of the family tree are provided in a template program file on Moodle. Use this template to fill in all the functions relevant to your feature (either A or B). Once you have completed and tested the functions you developed, you should then use the data (the family tree branch) for your partner's part to test your functions. You should document these tests using a professional testing method (test plan) in your report. This will usually be in the form of a table showing what is being tested, what the expected outcome would be, a screenshot showing the actual outcome and whether the test was passed or failed. For testing purposes, you should redefine your family tree and include the new tree in your report as a list.

The family tree consists of a maternal branch and a paternal branch as shown in the diagram below. The code template defines the two branches that you are required to analyse. Partner A should work on the maternal branch and Partner B on the paternal branch. Some functions will require you to process the whole family tree.

Maternal Family Tree



Paternal Family Tree



System Features and Requirements

Feature 1 **Both partners should work together on this task.**

C1. This function should return a list of all members in the maternal branch.

C2. This function should return a list of all members in the paternal branch.

C3. This function should return a list of all members of both branches

Make sure you understand each of these functions and the lists that they produce as you will need them when developing the following functions.

Feature 2 **Partner A should develop the functions A1 to A8 and Partner B should develop the functions B1 to B8.**

Partner A

A1. This function should return a list of all the parents in the branch. (function *parents*)

A2. This function should return all living members of the branch. (function *living-members*)

A3. This function should return a list of the current age of all living members. (function *current-age*).

A4. This function should return a list of all members who have birthdays in the same month as your birthday. (function *same-birthday-month*)

A5. This function should return a sorted list of all members in your branch. Sort the members by their last names. (function *sort-by-last*)

A6. This function should return a new family tree changing the name of any member of the family with the first name of John to Juan. (function *change-name-to-Juan*)

Partner B

B1. This function should return a list of all the children in the branch. (function *children*)

B2. This function should return the oldest living member to date of your branch. (function *oldest-living-member*)

B3. This function should return the average age on death of members of your branch. (function *average-age-on-death*)

B4. This function should return a list of all members who have birthdays in the same month as your birthday. (function *birthday-month-same*)

B5. This function should return a sorted list of all members in your branch. Sort the members by their first names. (function *sort-by-first*)

B6. This function should return a new family tree changing the name of any member of the family with the first name of Mary to Maria. (function *change-name-to-Maria*)

Getting Started

Your first step should be to re-read this document carefully. Many marks are lost because of simple mistakes in handing in something that does not meet the requirements.

Download the code template from Moodle. You use this to build up your code. For example, if you are Partner A then you will enter the code to find the parents in the section highlighted below:-

```
;; A1
(define (parents lst)
  ())
```

To continue to A2, you then move on to:-

```
;; A2
(define (living-members lst)
  ())
```

And so on.

Deliverables on Completion

Be careful to ensure that you attach **all** the parts required. **Failure to do so can result in a score of zero**.

The required parts are:-

- **Report in PDF format**. Use the coursework template provided on Moodle. Ensure that you get both your full name and Greenwich ID number for you and your partner correct in the heading section. Please complete all sections (even if some are less well worked out). Keep the reader in mind. It is up to you to convince us that you understand and not up to us to hunt for what you may have meant. For example, all screenshots should be annotated to explain what they show. This principle holds true for each section of your report.
- The **code file** you amended (see above) along with a **screencast** which includes an audible and understandable commentary. **These should be zipped into a single file**. The code file MUST start with the lines for your name, student ID and date of birth completed. The screencast will be used to assess whether the work is yours. Therefore, the commentary should clearly explain how each function works and show what happens when the code is run. Again, it is up to you to convince us that you understand. **You FAIL this assessment if you do not submit a screencast**.

Please note that we recommend you upload well in advance of closing time. Our servers get busy at these times and being held up will NEVER be a valid excuse for lateness.

Marking Scheme

Partner A	Partner B	Out of	My estimated score
Parents	Children	3	
Living members	Oldest member	4	
Current age	Average age on death	5	
Same birthday	Same birthday	8	
Sort by last name	Sort by first name	10	
Change a name	Change a name	10	
Appropriate use of lambda functions		5	
Appropriate use of recursion		7	
Appropriate use of higher order functions		8	
Quality of code		10	
Screencast		10	
Testing		10	
Documentation		10	
TOTAL		**100**	

Grading Criteria

Grading Criteria Judgement for the items in the marking scheme will be made based on the following criteria. To be eligible for the mark in the left-hand column you should at least achieve what is listed in the right-hand column. Note that you may be awarded a lower mark if you do not achieve all the criteria listed. For example, if you achieve all the criteria listed for a 2:1 mark but have a poor report then your mark might be in the 2:2 range or lower.

> 80% *Exceptional*	• Completed functionality implemented and tested to an outstanding standard or above (all required features implemented, no obvious bugs, outstanding code quality, Scheme language features accurately applied). • Able to show outstanding, thorough knowledge and systematic understanding of functional programming concepts and Scheme language features in the code and the report. • Group work: Pair worked well together to achieve objectives. Extensive commitment to the team, ensuring team engagement and feature integration. Well organised and made a significant contribution to system design and development. • Overall report – outstanding (required sections completed accurately and clearly, easy to read, structured and coherent and insightful arguments for design justification and use of Scheme language).
70-79% *Excellent*	• Completed functionality implemented and tested to an excellent standard (required features implemented, no obvious bugs, excellent code quality, Scheme language features accurately applied). • Able to show excellent, detailed knowledge and systematic understanding of functional programming concepts and Scheme language features in the code and in the report. • Group work: Pair worked well together to achieve objectives. Excellent commitment to the team and facilitating team engagement and system feature integration. Well organised and made significant contribution to system design and development. • Overall report – excellent (required sections completed accurately and clearly, easy to read, well justified design decisions and clear argument, comparative reasoning).

60-69% *Very Good*	• Completed functionality implemented and tested to a very good standard (required features implemented, few if any bugs, very good code quality, very good attempt using Scheme language features, may contain some design flaws). • Able to show very good knowledge and systematic understanding of functional programming concepts and Scheme language features in the code and in the report. • Group work: Group worked well together most of the time, with only a few occurrences of communication breakdown or failure to collaborate when appropriate. A good number of group meetings organised and attended to achieve objectives. Made significant contribution to system design and development. • Overall report – very good (required sections completed accurately and clearly, good argument for design justification and use of Scheme).
50-59% *Good*	• Completed functionality implemented and tested to a good standard (required features implemented, few if any bugs, good attempt using Scheme language features, contains some design flaws). • Able to show good knowledge and mostly accurate systematic understanding of functional programming concepts and Scheme language features in the code and in the report. • Group work: Pair worked well together most of the time, with only a few occurrences of communication breakdown or failure to collaborate when appropriate. An adequate number of team meetings held or attended to achieve objectives. Made partial contribution to system design and development. Sub-features implemented are fully integrated with overall system. • Overall report – good (required sections completed accurately, mostly clear, some reasonably balanced argument for design justification and use of Scheme).
45-49% *Satisfactory*	• Completed functionality implemented and tested to a good standard (some features implemented, few if any bugs, acceptable attempt using Scheme language features, contains some design flaws). • Able to show satisfactory knowledge of functional programming concepts and Scheme language features in the code and in the report. • Group work: Pair worked together some of the time with frequent occurrences of communication breakdown or failure to collaborate when appropriate. Few team meetings organised or attended. Features implemented may be only partially integrated with system or not working accurately. • Overall report – acceptable (required sections completed, mostly accurate and clear, superficial justification for design choices and use of Scheme).
40-45% *Satisfactory*	• Completed functionality implemented and tested to a reasonable standard (at least 2 required features implemented with a few minor bugs, acceptable attempt using Scheme language features, contains some design flaws). • Able to show fairly satisfactory knowledge of functional programming concepts and Scheme language features in the code and in the report. • Group work: Pair did not collaborate or communicate well. Members often worked independently with weak regard to objectives or priorities. Very few team meetings organised or attended. Features implemented may be only partially integrated or not working accurately.

	• Overall report – acceptable (required sections completed, mostly accurate, clumsy language, descriptive account of design choices and use of Scheme).
35-39% *Fail*	• Good attempt at some features although maybe buggy with some testing. • Confused knowledge of functional programming concepts and Scheme language features in the code and in the report. • Group work: Pair did not collaborate or communicate well. Members mostly worked independently, without regard to objectives or priorities. Very few team meetings organised or attended. Features implemented not integrated or not working accurately. • Overall report – mostly completed to an acceptable standard. Some sections are confused.
<35% *Fail*	• Little or no attempt at features or very buggy with little or no testing. • Not able to show satisfactory knowledge functional programming concepts and Scheme language features in the code and in the report. • Group work: Pair did not collaborate or communicate. Members worked independently, without regard to objectives or priorities. No meetings attended. Very little or no attempt at integration. • Overall report – mostly in-completed, at an un-acceptable standard or missing.