

COMP1811 Paradigms of Programming



COMP1811 - Introduction

*all about the module and
introduction to programming paradigms*

LP

Logic
Programming

$f(x)$

Functional
Programming



Object-oriented
Programming





On Today's Menu

- COMP1811 Housekeeping
- Introduction to COMP1811
- Principal Programming Paradigms





COMP1811

Overview and Learning Outcomes

- Understand the fundamental commonalities and distinctive features of different programming languages.
- Learn computational modes of thinking and think like a programmer.
- Master the art of computational problem solving.
- Write professional style programs.
- On successfully completing this module, you will be able to:
 - A. Understand the programming paradigms introduced and their applicability to real 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 in each paradigm.





COMP1811

Topics – Elements of Programming

- Data and knowledge representation with data types and structures.
- Features and constructs of the programming languages covered.
- Iteration and recursion as computational metaphors in programming.
- Abstraction of functionality and data types.
- Code design - organisation and modularisation of code and functions/methods.
- Problem solving for practical applications.
- Testing and error handling.



COMP1811 - Housekeeping



COMP1811 Teaching Team

contact information



Yasmine Arafa



Mohammad Al-Antary



Ralph Barthé



Christopher Beckwith



Cornelia Boldyreff



Mobolaji Orisatoki

Tuan Nguyen



Pushparajah Rajaguru



Máté Szabó



Rafael Torres



Andy Wicks



Adil Akram





Contacting your Tutor

- Preferably by email or during office hours.
- **Always use your Greenwich account,**
otherwise your emails may be caught by our spam-filter.
- **Put COMP1811 in the subject line,**
and your full name and student ID in the email body.
Otherwise, we don't know who you are or which module you're referring to!
- We will aim to reply within 48 hours during term time.
If you don't get a reply by then, send again!





COMP1811

module structure

- 1 hour weekly lecture
 - typically 50 minutes
 - giving you time to get to your lab sessions.
- 2 hour weekly labs (check your timetable for where they are)
 - coding exercises,
 - using pair programming,
 - on Mondays or Tuesdays.

Sorry if you've got a long gap!





COMP1811

assessment



1. Practical Coursework 1 (Python Project 40%) – Due 22/01/2023

- group work (pairs) – decide on a partner from your lab session
(you will be paired with a partner if you cannot find one or you are studying online),
- involves development of an application to given specification using the object-oriented programming paradigm (Python), 2 interim progress demos, a group report with an individual element and a demo,
- uploaded electronically on Moodle.

2. Practical Coursework 2 (Scheme Project 30%) – Due 18/03/2024

- group work (pairs),
- involves development of an application to given specification using the functional programming paradigm (Scheme), a group report with an individual element and a demo,
- uploaded electronically on Moodle.





COMP1811

assessment, *cntd.*

3. Logbook (30%) – 02/04/2024

- includes a number of tasks due at various stages in both terms 1 and 2,
- all logbook tasks are graded,
- tasks may include coursework milestones, practical hackathon activities, and online quizzes.,
- about programming skills and paradigm concepts rather than memory,
- quizzes are open-book tasks that are done during your scheduled lab session on campus and include multiple-choice and short coding questions covering all the topics covered in this module,
- schedule for the logbook tasks will be outlined throughout the terms.





COMP1811

module guidelines

- Attend all lectures and labs - attendance strongly correlates with marks.
- Sign the register each week (lecture and tutorial) - don't ask a friend to tap for you!!!
- Be considerate to others - chatting or *snoring* will upset those trying to listen.
- Feel free to email tutors to contact them outside their office hours or, better still, use the Moodle forum.
- Module related problems - raise them with Yasmine.
Don't wait until the end of the term or year!
- To do well, you need to work outside the three timetabled hours each week.
Practice, practice, and more practice...





How to *really* do well on COMP1811?

- COMP1811 is not about memorising keywords
- It is about
 - Practicing,
 - *lots of it*,
 - and understanding the concepts covered
- So, it is essential to
 - attend lectures,
 - work through the lab exercises each week,
 - develop a curiosity about how programs work and identify the similarities and differences between the languages covered.






Moodle Checklist









Monitor your Progress



- Moodle Checklist  is a very useful tool to monitor your progress.
- It lists all the activities that need to be completed in one place.
- Includes check boxes to tick off activities as you complete them.
- Progress bar shows overall completion.
- Can help with time management.

All items:



- ☒ 1.0 - Introduction and Programming Basics 
 - ☒ Learning Material
 - ☒ 1.0.1 Introduction 
 - ☒ 1.0.2 Programming Paradigms 
 - ☒ 1.0.3 Object-Oriented Programming - Why Python? 
 - ☒ 1.0.4 Python Basics - Hello World in Python 
 - ☒ Lab Exercises
 - ☒ 1.0.1a Getting Started with PyCharm 
 - ☒ 1.0.1b Examining Sample Code 
 - ☐ Induction Quiz 
- ☐ 1.1 - Python Basics: Variables, Data Types, Operators and Expressions



Working in the labs...

we are going to use pair programming

Wikipedia says "Pair programming is an agile software development technique in which two programmers work together at one workstation. One, the driver, types in code while the other, the observer (or navigator), reviews each line of code as it is typed in. The two programmers switch roles frequently."

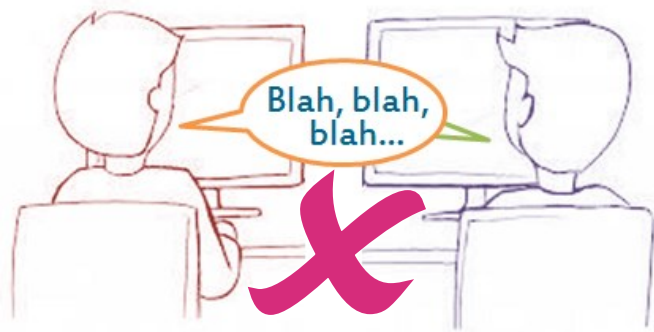
Industry aims - increased quality, reduced timescales, knowledge sharing

Educational aims - knowledge sharing, learning by explaining, greater confidence, help with problem solving, better use of tutor's time



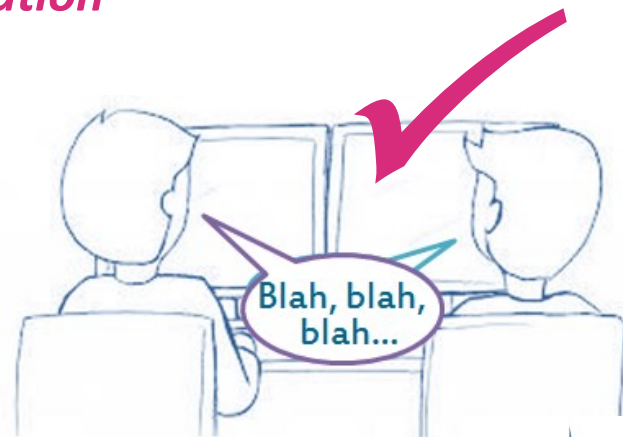
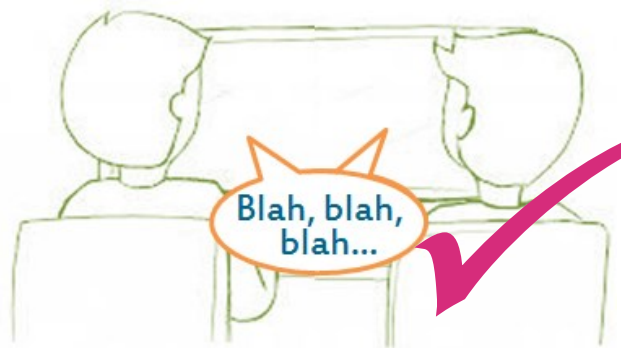


Pair Programming



*"Pair programming ... **two programmers** work together **at one workstation**"*

*Remember to
physically distance and
to share work at the
end of the session,*





Pair Programming

feedback from previous students...

- After initial doubts feedback was overwhelmingly positive...

It should be used on more courses as it helps us learn from each other

Works best if one partner is more confident and the other less rather than both at same level

It is a nice approach to help students develop their skills

It would be better if pairs were changed more frequently

Helped a lot and gave me more confidence





Avoiding Plagiarism

- Discussing the module and helping each other in the labs and with the coursework is great!

BUT!!!

- Copying from another student or allowing someone to copy your work, getting someone else to do your work, copying from the Web, books, etc. without clear and explicit acknowledgement...

IS PLAGIARISM!!

- We do check for it and penalties are severe :(
- Please don't put yourself **or your friends at risk.**
- If in doubt - ask the staff.





Essential Resources and Further Reading (term 1)

Python

- Moodle pages for [COMP1811](#).
- Further reading:
 - **A Concise Introduction to Programming in Python, 2nd Ed.** (available as eBook from the library)
Johnson, Mark, CRC Press, 2018
 - **Think Python: How to think like a Computer Scientist, 2nd Ed.** (available as eBook from the library)
Downey, Allen, O'Reilly, 2015. eCopy available [here](#).
 - **[Computational Thinking: A beginner's guide to problem-solving and programming](#)** Beecher, Karl,
BCS Learning & Development, 2017) (available as eBook from the library)
- Other useful ebooks:
 - [How to Code in Python](#) (pdf) — Aimed at complete beginners.
 - [pythonbooks.revolunet.com](#) — Links to a wide, varied collection of eBooks on Python and where it is used. Use this for future reference.





Essential Resources and Further Reading (term 1)

cntd.

- Online Python Tutorials
 - [LearnPython.org](https://www.learnpython.org/) — Interactive tutorials providing a comprehensive introduction to coding in Python. Suitable for complete beginners.
 - Python at [TutorialPoint](https://www.tutorialpoint.com/) — Another good tutorial starting from Python basics and leading onto more advanced topics.
- Beginner's [Python cheat sheet](#)





Any questions?



Programming Paradigms



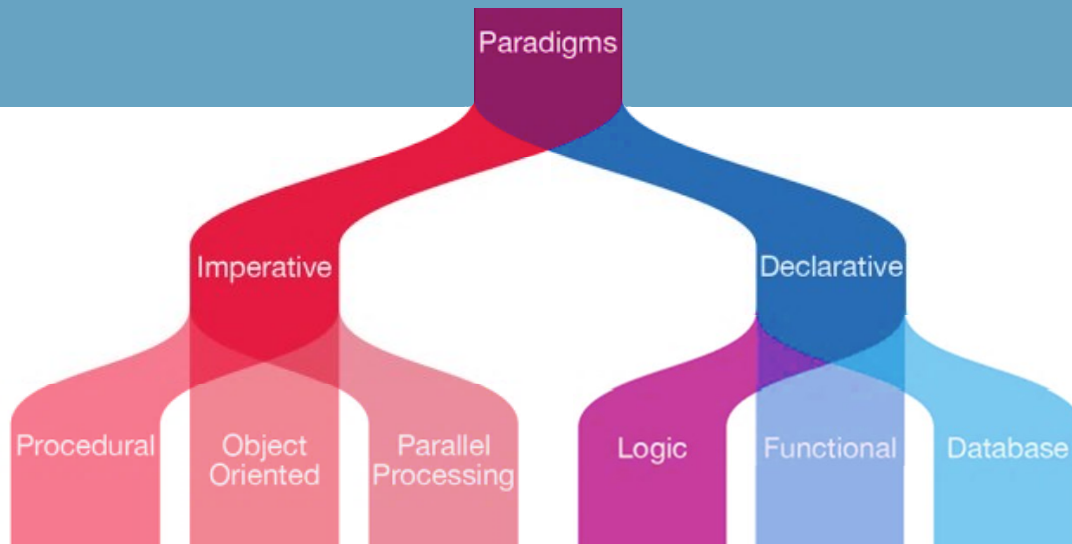
What are Programming Paradigms?

- A **programming paradigm** is a fundamental style of programming
 - it is an approach to programming,
 - defines the organising principle of a program, and
 - it is a way to classify programming languages based on their features.
- Programming paradigms are the result of people's ideas about how computer programs should be constructed
 - models that serve as a "school of thoughts" for computer programming.
- All programming languages can be categorised into programming paradigms.





Principal Programming Paradigms



- There are generally two main programming paradigms: Imperative and Declarative
 - each are further divided into closely related sub-paradigms
- Languages that strictly lie within these paradigms are called "*pure*"
 - in reality, *very few* languages are *pure*,
 - most combine features of different paradigms.





Imperative Programming

- **Computation is a series of steps (commands/instructions) executed in sequence:**
 - programs are a sequence of commands to direct the computer to perform some computation,
 - to control sequences, instructions can be run conditionally using **if**, and repeatedly using **while/for**,
 - statements can change a program's state, principally through assignment operations or side effects.
- **Features:**
 - easy to implement
 - complex problems difficult to solve
- **Disadvantages:**
 - code can become very long (Spaghetti Code)
 - difficult to scale
 - less efficient and less productive
 - parallel solutions cannot be implemented
- **Sub-paradigms**
 - Procedural Programming, Object-Oriented Programming.





Declarative Programming

- Computation is done by defining *what* should be done (computed) and not *how* it should done (how that computation should be implemented).
 - Sounds like magic! Not really. It is left to the compiler to figure out the *how*.
 - Declarative approaches rely on preconfigured functionality in the language to execute a task without explicit instructions on what steps to take to implement that task. It all predefined!
 - In practice, this approach requires a domain-specific language (DSL) for expressing *what* tasks a programmer wants, and shields them from the detail that defines *how* that task is implemented.
- Features:
 - short and efficient code,
 - programs have no state change, and
 - no side effects
- Sub-paradigms
 - Functional Programming, Logic Programming
- Disadvantages:
 - sometimes hard to understand (code based on unfamiliar conceptual model), and
 - despite potential for reusability, customising code may need significant rewrites





Which paradigm to use?

- Most important, the choice of paradigm (and therefore language) depends on:
 - the nature of the problem, and
 - how programmers best think about the problem.
- Other considerations:
 - efficiency;
 - compatibility with existing code;
 - availability of translators.
- There is NO "right" programming paradigm!
 - Use the best tool for the job!
 - The boundaries between paradigms are not sharp: many languages are a hybrid of different paradigms.





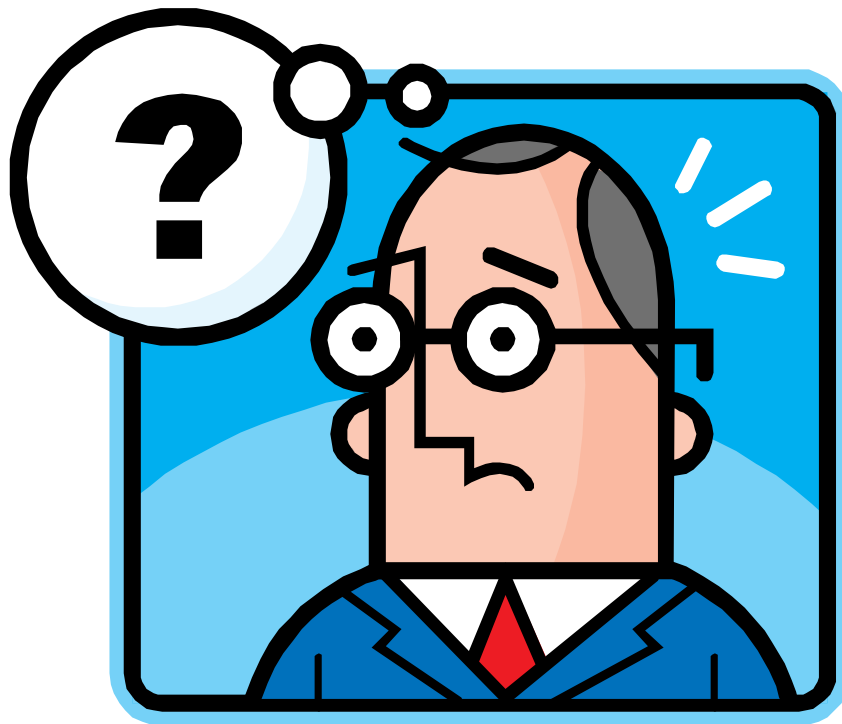
Common Concepts

- Unifying language concepts
 - Data types (both built-in and user-defined)
 - Expressions (e.g., arithmetic, boolean, strings)
 - Functions/procedures/methods
- COMP1811 covers how these concepts are defined syntactically, used semantically, and implemented pragmatically for:
 - OOP (Python),
 - FP (Scheme), and
 - briefly LP (Prolog)





Questions?





Summary

- Introduction to COMP1811
 - module structure and assessment
 - how to do well
 - pair programming
- Principle Programming Paradigms





In the lab today ...

- There is a list of tasks to complete – this week there are 2 parts.
 1. Getting started with the PyCharm IDE
 2. Examining sample code
- The lab sheet gives tips and points you to helpful resources.
 - Please follow these up. *They will help.*





Next week

- Object Oriented Programming paradigm.
- Python basics.
- Variables, Data Types and Operators
- Start coding in Python.

