Student Details

Name: Jack Guan

Deakin Student ID: 221393284

Tutor: Dr.Nayyar Zaidi , Dr. Ziwei Hou , Mr Xiangwane Yang

Class: Online

Intended Grade: Distinction

Instructions: Please fill in the module name, along with submission and discussion deadlines from Ontrack website.

I will adhere to following timetable for submitting tasks, and will come to class for task discussion with my tutor.

SIT320 - Time Table

	Module		Tasks	Submission Deadline	Discussion Deadline
1	Introduction	Must	Module 1 Task	20 July	27 July
2	Advanced Trees	Must		20 July	27July
3	Advanced Hashing Sorting	Must		3 Aug	10 Aug
4	Advanced Algorithmic Complexity	Must		17 Aug	24 Aug
5	Graphs II	Must		17 Aug	24 Aug
6	Dynamic Programming	Must		17 Aug	24 Aug
7	Greedy Algorithms	Must		24 Aug	31 Aug
8	Linear Programming	Must		31 Aug	7 Sep
9	Network Flow Algorithms	Must		7 Sep	14 Sep
10	AI Algorithms	Must		14 Sep	21 Sep
11	Task 11 Advanced Algorithms			21 Sep	28 Sep

Discussed with your Tutor (Circle one): Yes / No

Signatures: Jack Guan

Task 1: Lesson Review

Write a one-page reflection summarising what you have learned in this module.

This unit reveals how algorithms are more broadly problem solving. This blends bottom-up and top-down perspectives — using tools like pseudocode and Python. Here are the key takeaways:

Algorithm Design and Problem Solving

The bottom-up/top-down distinction. Bottom-up views algorithms as input-output processes; top-down embeds them in the Polya cycle (understand → plan → solve → check). Techniques like dynamic programming are derived from mathematics and applied to real-world problems. As Siggi points out, experience will hone this in areas like data science, where data structures support rather than dictate solutions.

Pseudocode: A Strategic Bridge

Pseudocode avoids syntax distractions and enables language-agnostic design. Good form prioritizes readability, modularity, and ease of conversion; it serves a dual purpose: team collaboration and upfront debugging. Writing the logic first can catch bugs that the compiler misses. Adopting Python-style pseudocode will streamline my workflow, ensuring modular decomposition and efficient peer review.

Python and Jupyter: The Power of Prototyping

Its pseudocode-like syntax makes the transition from design to code more fluid. Jupyter's cell-based execution supports iterative Polya loops: test steps, visualize results, and

optimize - all in one space. Use it to tackle data science challenges and merge prototyping, visualization, and documentation.

The main content of this module is: top-down problem framing guides algorithm selection, pseudocode clarifies logic, and Python/Jupyter accelerates iteration.

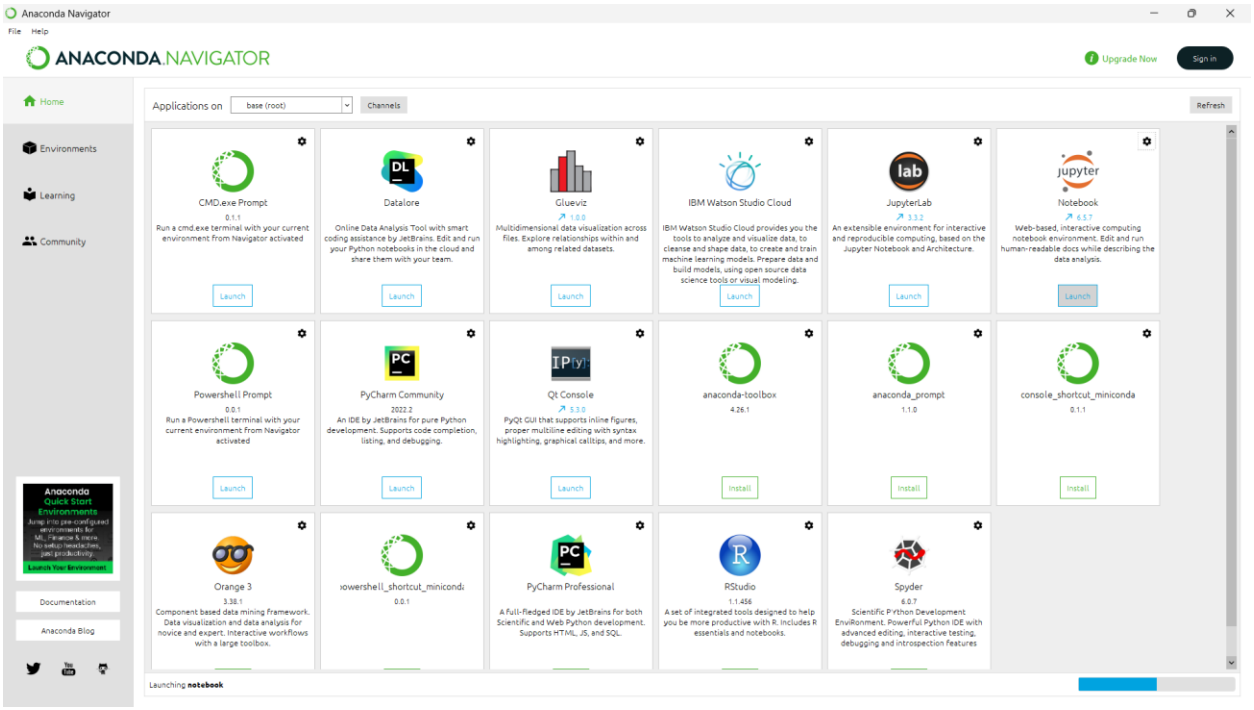
Task 4: Python Environment Setup

You must set up your Python environment and demonstrate basic programming ability.

- (2a) Provide evidence that Anaconda Navigator is installed on your system.
- (2b) Show that you have created a new environment named SIT320.
- (2c) Demonstrate execution of a basic Python script using while and if statements.

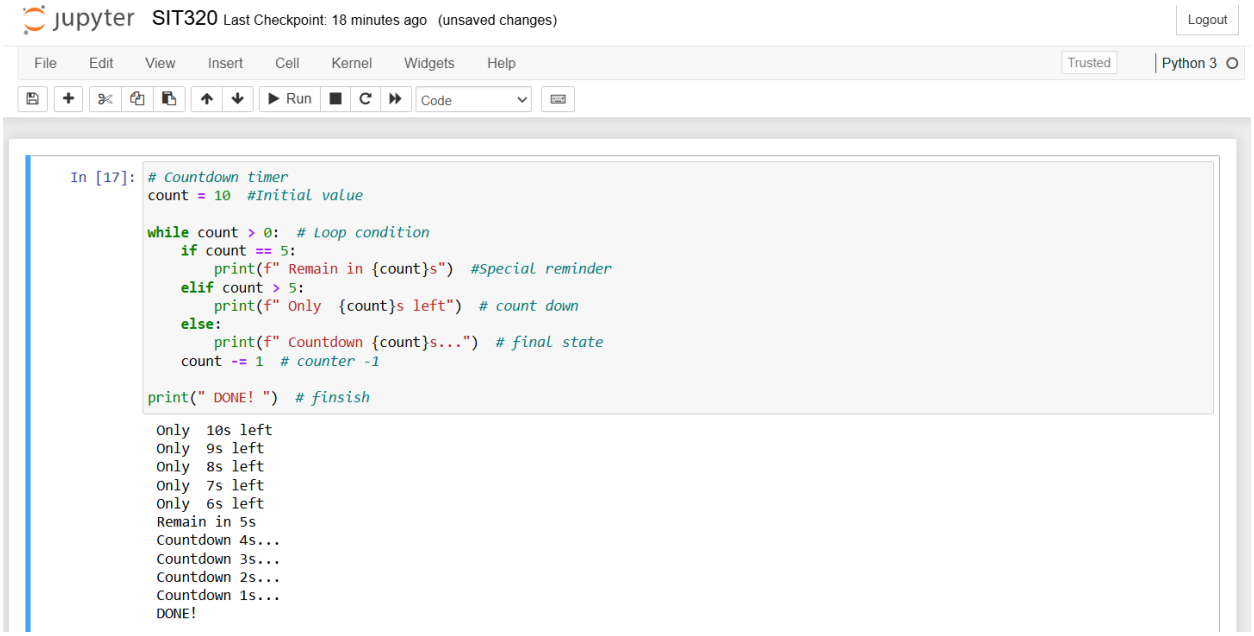
Screenshots or screen recordings are acceptable.

2a)



2b)

2c)



While loop:

Repeat the indented code block when count > 0

Execute count -= 1 after each iteration (counter)

If loop:

if/elif/else conditional branch

count == 5: Special reminder

count > 5: Count down

else: Final state

When value is < 0 , end loop.

Task 5: Complexity Analysis of Your Algorithm

Analyze the time complexity of your Tic-Tac-Toe algorithm. Is it a polynomial-time algorithm (in P), or does it fall under NP?

SIT320 — Advanced Algorithms

Your response should reflect a clear understanding of algorithmic complexity and the difference between P and NP problems. Your tutor will discuss this with you.

Algorithm complexity:

Time complexity: O(9) (worst case)

In practical applications, the amount of search can be reduced by pruning.

Space complexity: O(9)

The recursive depth is 9.

Tic-tac-toe is a Polynomial-time algorithm problem because its solution process can be completed through a deterministic polynomial time algorithm. The verification and search for the optimal solution can be carried out within a reasonable time frame. It cannot be other NP-complete problems.

Task 7: [Optional – For Distinction and High Distinction Students] Advanced Exploration

Analyze your algorithm that you designed for Tic-Tac-Toe, and critically analyze your strategy. Can you use your approach to design solution to games like Chess, backgammon? If not, what modifications will you make? This task will prepare you for you D and HD tasks, when we design a Reinforcement Learning based solution to Tic-tac-toe.

The core strategy of the pseudocode is to use the Minimax algorithm to simulate all subsequent games for each possible move, calculate the result score, backtrack and select the most favorable step for the current player.

The time complexity is constant (for a fixed 3x3 chessboard)

But it also has certain limitations, because it is not scalable and cannot be expanded to larger chessboards or complex games.

At the same time, the Minimax algorithm cannot automatically adapt to games with different rules and is less efficient.

It is unlikely to be achieved for chess because it has a huge state space and a deep game tree, and Minimax cannot be fully searched.

If certain changes are to be made, more powerful optimization techniques and modeling methods need to be introduced.

For example: set a search depth limit; use a heuristic evaluation function instead of a final judgment, etc.