FAQ

Module-2	
Question 1.	**What is the fundamental difference between linear and non-linear optimization problems?**
Answer	Linear optimization problems are those where the objective function and the constraints are linear functions of the decision variables. This means that each term in these functions is either a constant or a product of a constant and a single decision variable. Graphically, this leads to straight lines or flat planes. Non-linear optimization problems, on the other hand, involve at least one non-linear function, either in the objective or the constraints. The relationships in non-linear problems can be polynomial, exponential, logarithmic, etc. Graphically, they can take on a variety of shapes that aren't straight lines or flat planes. The distinction is crucial because linear problems have well-defined solution methods like the Simplex method and can guarantee global optima, while non-linear problems often require more complex solution methods and might lead to local optima.

Question 2.	**How does the Warehouse Problem differ from the Assignment Problem?**
Answer	The Warehouse Problem typically concerns the optimal location and operations of warehouses in a supply chain to minimise costs or maximise efficiency. Key considerations might include warehouse storage capacities, transportation costs, demand at different locations, and so forth. The Assignment Problem, however, is about optimally assigning tasks to agents (or resources to jobs) in such a way to minimise total cost or maximise total efficiency. It's often visualised as a bipartite graph, where one set of nodes represents tasks and the other set represents agents. While both problems involve optimization and allocation, their objectives, constraints, and real-world implications differ. The Warehouse Problem is more spatial and logistical, whereas the Assignment Problem is about optimal matching.
Question 3.	**Why is the Knapsack Problem considered NP-hard and why does it matter?**
Answer	The Knapsack Problem is considered NP-hard because, as of our current understanding, there's no polynomial-time algorithm (i.e., an algorithm that runs in time proportional

	to a polynomial in the size of the input) to solve all instances of the problem optimally. The complexity of the problem grows exponentially with the number of items, making it computationally infeasible to solve larger instances using brute force. Its NP-hard designation matters because it indicates a fundamental difficulty in achieving efficient solutions. In practice, we might resort to approximation algorithms or heuristics for larger instances, which provide "good enough" solutions without guaranteeing optimality.
Question 4.	**How can we apply optimization techniques in Excel? Isn't it a spreadsheet tool?**
Answer	Excel, while primarily known as a spreadsheet tool, has built-in functionalities for optimization, most notably the Excel Solver add-in. Solvers can handle linear, non-linear, and integer programming problems. To use Excel for optimization: 1. Define your decision variables in cells. 2. Construct the objective function and constraints using standard Excel formulas. 3. Use the Solver add-in to specify the objective cell, decision variable cells, and constraints. 4. Run Solver to find the optimal solution. While Excel is user-friendly and excellent for smaller

	optimization problems, it might not be suitable for very large or complex problems due to computational limitations.
Question 5.	**Why should one learn to formulate optimization problems in Python?**
Answer	Python is one of the most versatile programming languages and has rich libraries and tools for optimization, such as PuLP, SciPy, and Gurobi. By learning to formulate optimization problems in Python, one can: 1. Handle larger and more complex problems than with tools like Excel. 2. Integrate optimization into broader data processing, analysis, and visualisation pipelines. 3. Leverage advanced algorithms and heuristics available in specialised Python libraries. 4. Automate repetitive optimization tasks. 5. Benefit from the broader Python ecosystem, including tools for machine learning, data visualisation, and web applications.