# MsDS – Data Structures and Algorithms (Module 1)

## Expected Learning Outcomes

1. Recognise the significance of the 'Separation of Concerns' concept in structuring complex systems.

2. Recognise the differences between data abstraction and encapsulation, as well as how they affect modularity and data security.

3. Recognise the differences between Data Types and their Representations, taking into account the effects on memory and performance.

4. Understanding the difference between implementation and interface can help you see software usability and design more clearly.

## Separation of Concerns

• Separation of concerns (SoC) promotes a modular and organised approach to software design by partitioning a program into discrete portions that address specific issues or capabilities.

• Advantages of SoC include enhanced maintainability, improved scalability, efficient collaboration, reusability, and testability.

• SoC is implemented in various software development paradigms and tools, such as object-oriented programming, MVC architecture, microservices, and layered architectures.

## Data Abstraction and Data Encapsulation

• Data abstraction simplifies complex systems by highlighting key characteristics and reducing repetition.

• Data encapsulation enhances security by preventing direct manipulation of object data and promotes modularity and reusability.

• Practical applications of abstraction and encapsulation include OOP class design, API development, libraries and frameworks, and component-based development.

## Data Types vs. Representation

• Data types serve as a system for categorisation and provide guidelines for handling different data elements in computer science.

• Data representation involves the physical storage of data types in binary code and includes methods like floating-point representation and ASCII encoding.

• The interaction between data types and representation is crucial in software development, as appropriate representation ensures efficient allocation of resources and can impact computational efficiency and accuracy.

## Interface vs. Implementation

• The term "interface" refers to the contractual aspect of a software entity, providing defined interactions while hiding complexities.

• "Implementation" pertains to the core components of a system, including algorithms and code, responsible for actualising functionalities.

• The distinction between interface and implementation enables design flexibility, modularity, and encapsulation of complexity in software systems.

## Abstract Data Types

• Abstract Data Types (ADTs) bridge the gap between theoretical concepts and practical implementations in computer science.

• ADTs provide a formal mathematical construct that specifies operations on a set of data elements, allowing for flexible implementation approaches.

• ADTs promote modularity in software design, enabling separate development and replacement of components, improving maintainability and scalability.

## Important Terminologies

• Abstraction: The practise of concealing intricate intricacies while revealing just essential functionality.

• Separation of Concerns: The implementation of a system whereby certain portions is allocated to fulfil specific functionality or problems.

• Data Encapsulation: The integration of data and corresponding techniques inside a cohesive entity.

## Summary

❖ Abstraction is a strategic approach for handling complexities in software design, promoting cleanliness and organisation.

❖ Data abstraction and encapsulation obscure internal data mechanisms, enhancing data integrity and protecting against interruptions.

❖ Interface vs. implementation differentiation promotes flexibility and modularity in software design, allowing for updates without disrupting existing interfaces.

❖ Abstract Data Types (ADTs) provide conceptual frameworks for data structures, allowing developers to focus on higher-level logic and optimise implementations.

## Additional Important Terminologies

• Data Types: Data may be categorised based on its properties and the types of values it is capable of holding.

• Representation: The specific internal arrangement or organisation used for the purpose of data storage.

• Interface: A predetermined collection of interactions or functions, regardless of their inherent processes or implementations.