Programming — Chapter study guide

Outcome 1 — Programming · 14 key-knowledge points · exam overview, links & practice

The big picture

Respond to teacher-provided solution requirements and designs to develop working software modules using an object-oriented programming language.

Unit 3 Outcome 1 is about **building and documenting working modules** in a programming language. The exam rewards precise definitions (data types, OOP, errors, validation), the ability to read/write pseudocode and IPO charts, and clear testing using test data and testing tables.

Think of the flow: understand requirements → design with tools → choose data types/structures/sources → write code using language features and naming conventions → validate input → document internally → debug and test until expected = actual.

WHAT THE EXAM REWARDS

- Definitions are easy marks — memorise the four OOP principles, the three error types, the three validation checks and the data types.
- Pseudocode and IPO questions appear often — practise writing them with correct sequence, selection and iteration.
- Testing questions want valid, boundary AND invalid test data, plus an expected-vs-actual testing table.
- Binary search's sorted-data precondition and the linear/binary comparison recur regularly.

Key knowledge at a glance

KK01	**Emerging trends in programming using artificial intelligence** How AI tools assist code generation, debugging, optimisation — and using them responsibly.
KK02	**Characteristics of functional and non-functional requirements, constraints and scope** Functional vs non-functional requirements, plus constraints and scope.
KK03	**Design tools for representing modules** Five design tools that represent modules: data dictionaries, mock-ups, object descriptions, IPO charts, pseudocode.
KK04	**Characteristics of data types** Characteristics of text, numeric and Boolean data types.
KK05	**Characteristics of data structures** Characteristics of 1-D arrays, 2-D arrays and records.
KK06	**Characteristics of data sources (TXT, delimited CSV and XML files)** Structure of, and reasons to use, TXT, CSV and XML data sources.
KK07	**Principles of OOP** The four OOP principles: abstraction, encapsulation, generalisation, inheritance.
KK08	**Features of a programming language** Core features of a programming language: variables/constants, data types, control structures, operators, GUIs, functions/methods, classes/objects.
KK09	**Purposes and features of naming conventions for solution elements** Purposes and features of naming conventions: Hungarian notation, camelCase, snake_case.
KK10	**Validation techniques for data** Validation techniques: existence, type and range checking.

KK11	**Purposes of internal documentation** Purposes of internal documentation: explaining/justifying structures, maintenance, and stub comments.
KK12	**Algorithms for sorting and searching** Sorting (selection, quick) and searching (linear, binary) algorithms.
KK13	**Types of errors** Error types: syntax, logic, and runtime (overflow, index out of range, type mismatch, divide by zero).
KK14	**Debugging and testing techniques for checking modules function correctly** Debugging/testing techniques: breakpoints, debugging statements, test data, and testing tables.

Must-know glossary

Functional requirement	A task the solution must perform.
Non-functional requirement	A quality of how well the solution performs.
Abstraction	Hiding detail, exposing only essential features.
Encapsulation	Bundling data and methods together, restricting direct access to data.
Validation	Checking input is reasonable (existence, type, range) before processing.
Logic error	Code runs but produces incorrect output.
Testing table	A table comparing expected and actual output for each test.

Before the exam — revision checklist

- ■ I can define functional/non-functional requirements, constraints and scope.
- ■ I can write pseudocode and complete an IPO chart for a module.
- ■ I can choose data types, structures (1-D/2-D arrays, records) and sources (TXT/CSV/XML).
- ■ I can define the four OOP principles with examples.
- ■ I can implement existence, type and range validation.
- ■ I can classify syntax, logic and runtime errors and the four runtime errors.
- ■ I can build test data (valid/boundary/invalid) and a testing table.
- ■ I can compare linear and binary search and state binary's precondition.

Exam practice — questions & model answers

Q1. A module reads student marks from a CSV file, stores them, and reports the highest mark and whether each student passed (≥50). Identify a suitable data structure for the marks and one validation check that should be applied.

Apply · 3 marks

Model answer

• A one-dimensional array (or array of records if names are kept) for the marks (1).
• Type check — each mark must be numeric (1).
• Range check — each mark must be 0–100 (1).

Q2. The module above runs but always reports the highest mark as 0. Name the error type and suggest a likely cause.

Name/Suggest · 2 marks

Model answer

• Logic error (1).
• Likely the comparison/initialisation is wrong, e.g. the ‘highest’ starts too high or the comparison operator is reversed (1).

Q3. Explain how internal documentation and a naming convention would improve this module’s maintainability.

Explain · 3 marks

Model answer

• Internal comments explain/justify the logic so another developer understands it (1).
• Meaningful, consistently-named variables (e.g. camelCase) make the code self-explanatory (1),
• together making future maintenance faster and less error-prone (1).