O-LEVEL COMPUTING SYLLABUS

Upper Secondary

Express Course

Normal (Academic) Course

Syllabus 7155

(Revised)

Year of Implementation: from 2017 with Secondary Three

Ministry of Education
SINGAPORE

RESTRICTED
FOR USE BY EDUCATION OFFICERS ONLY

O-LEVEL COMPUTING SYLLABUS
For implementation in 2017
First year of examination in 2018

Computer Education Unit
Sciences Branch
Curriculum Planning and Development Division
Ministry of Education
Singapore

First published 2016
Second version 2020

CONTENTS

SECTION 1: INTRODUCTION

General Overview
Importance of Subject
Computer Education Framework
Big Ideas in Computing
Rationale for Change
Design Elements

1. INTRODUCTION

General Overview

Over the past decades, advancement in computer technology has been the key driving force behind much of the changes witnessed in many aspects of our lives. Beyond this impact on our daily lives as users of technology, Computing has become so entrenched in the fields of Science, Technologies, Engineering and Mathematics (STEM) that it has transformed the very practices of those disciplines.

Data analytics are transforming the future of business and research in diverse areas. We will need stronger computational capacity and people with considerable computational thinking skills to automate and streamline processes, work with data (e.g. text and images) to discern trends and draw inferences, develop hypotheses and innovate in fields like medicine, science and engineering to solve complex problems.

Professionals in finance, retail and manufacturing will require computational thinking skills to be competitive in their fields. For example, it will be an asset to acquire the ability to use a computer to analyse data to predict market and business trends, understand economic conditions and build participatory relationships with clients in the digital environment. Students who study O-Level Computing would acquire a stronger foundation in computational thinking that would be useful to their further studies and careers.

Purpose and Philosophy

The global computing education landscape is experiencing a renewed emphasis on programming (coding) and the fundamental computational skillsets that will prepare students to thrive in a more digitally-connected world and work place. The plethora of applications on mobile and computing devices have changed the way we live, learn and work. The impact is felt across all age groups. To narrow the digital divide, various government schemes and initiatives were launched to reach out to the population so that more people can harness technology effectively to improve their lives and contribute to society.

While the current baseline ICT programme equips all students with basic computer literacy skills to be informed users of technology, the O-Level Computing curriculum aims to grow students' interest and competency in more advanced concepts and skills. This will equip them with the necessary foundation to continue with post-secondary computing courses. A secondary aim is to encourage students to consider careers in computing technology and systems or as skilled programmers, system developers and software engineers. Our aspiration is for this group of students with the passion for Computing to eventually harness their talent to solve complex problems or create new value propositions in society through technology.

Importance of Subject

The value of computing arises from the fact that it has the ability to integrate the use of software and hardware to create new artefacts, solve/address real-world problems and perform tasks. Computing is grounded in (i) computational thinking and (ii) systems thinking; and enables (iii) exploration and innovation. Thus, *Computational Thinking* and *Systems Thinking* form the two arms in the dimension of Computer as a Science in our computer education framework (shown in Figure 1.1). The other two dimensions on Computer as a Tool and Computer in Society also cover important areas in Computing and are taught so that students are able to develop their ideas and creativity through the use of information and communication technology (ICT) to contribute to society.

Computational Thinking

Computational thinking is a thought process that involves formal reasoning, logical and algorithmic thinking, and the reformulation of a problem so that a computer-based solution is viable.

In the O-Level Computing subject, students are taught how to go through a systematic process of thinking when solving problems (abstraction), formulating steps for solutions (algorithmic thinking) and writing computer programs (programming/coding) to produce the solutions. Hence, students develop computational thinking skills when analysing problem situations and reformulating them into problems with computational solutions. They also develop computational thinking when coding effective and efficient programming solutions. Students with computational thinking skills will be able to apply their skills across other subject disciplines and be better enabled to solve real-world problems in those disciplines. Armed with computational thinking skills, our students will have a competitive edge in the increasingly digital landscape.

The promotion of computational thinking in students also featured predominantly in the syllabuses offered in the countries scanned (e.g. UK, US, Australia and Hong Kong). Computational thinking and the study of Computing allows the students to understand the digital world in a deeper way; just as physics allows students to understand the physical world better and the study of biology allows a better understanding of the biological world.

Computer Education Framework

The computer education framework comprises three dimensions:

- Computer as a Science
- Computer as a Tool
- Computer in Society

Computer as a Science

The dimension of Computer as a Science looks into the scientific aspect of computer science, focusing on the core components of computational and systems thinking.

Computational thinking develops students' skills in problem solving through algorithmic thinking and design. Acquisition of programming language skills is usually a part of this area of learning. Computational thinking, as defined by Jeannette M. Wing[1], is a way people solve problems and that it is not about trying to get people to think like computers[2]. This often involves thinking and problem-solving processes to reformulate a seemingly difficult task into one we know how to solve. Thus, computational thinking, in her opinion, is a fundamental skill for everyone, not just computer scientists.

Systems thinking develop students in the design and creation of systems and solutions through processes in problem definition, system analysis, and systems design.

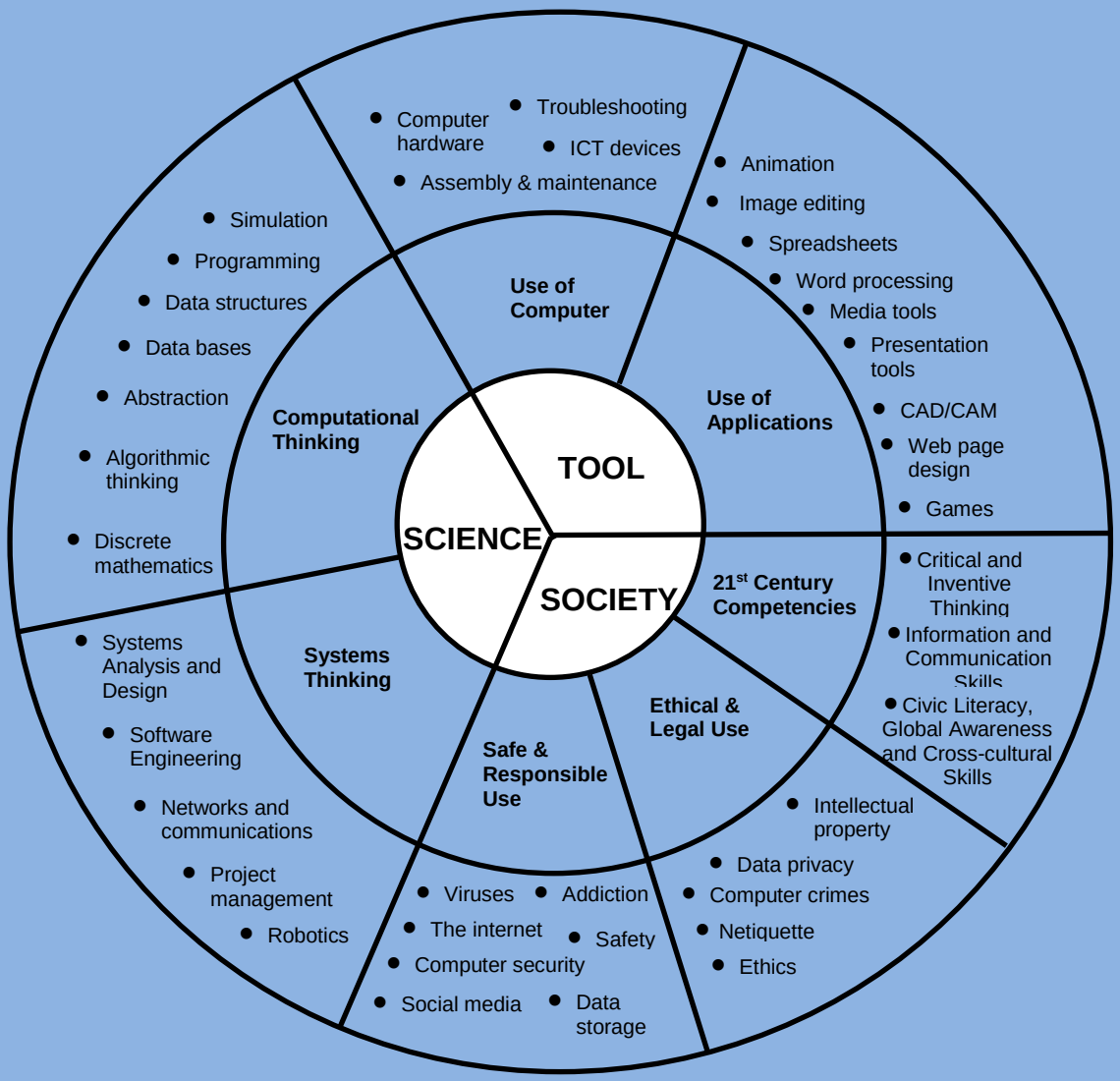


Figure 1.1: Computer Education Framework (The outer ring are examples of topics that could be studied for the three dimensions shown in the inner ring. The middle ring shows the components of each dimension.)

[1] Jeannette M. Wing is the President's Professor of Computer Science and head of the Computer Science Department at Carnegie Mellon University.
[2] J. M. Wing. Computational Thinking. Communications of the Association for Computing Machinery (ACM), March 2006, Vol. 49(3).

Computer as a Tool

The dimension of Computer as a Tool looks mainly at the utilitarian aspect of computing and ICT. At the heart of it are the use of the computer and the use of computer applications. Use of computer exposes students to the hardware, the technology and related devices and peripherals that open up ways for work, play and living.

Use of computer applications focuses on the mastery of productivity, communications and creative tools to complete tasks for specific purposes. Common examples include word processing, spreadsheets, graphics, emails, animation, and web design.

Computer in Society

This dimension focuses mainly on the ethical, legal and security issues relating to the use of computers and ICT in society. Issues commonly associated with this dimension include internet security, intellectual property, computer addiction, and data privacy.

The inclusion of the 21st Century Competencies component reflects the impact of technology on the kind of skills needed at the workplace of the future. 21st Century Competencies relevant to the ICT area include the ability to work collaboratively, produce creative work and be self-directed in learning.

The Computer Curriculum Framework was updated in September 2017.
For the full details, please refer to the MOE OPAL Computer Education website.

Big Ideas in Computing

The US College Board and the National Science Foundation (2010)[3] focused on practices of computational thinking for high school CS curricula based on seven 'big ideas' of Computing. The 'big ideas' are:

(a) Computing is a creative human activity.
(b) Abstraction reduces information and detail to focus on concepts relevant to understanding and solving problems.
(c) Data and information facilitates the creation of knowledge.
(d) Algorithms are tools for developing and expressing solutions to computational problems.
(e) Programming is a creative process that produces computational artefacts.
(f) Digital devices, systems, and the networks that interconnect them enable and foster computational approaches to solving problems.
(g) Computing enables innovation in other fields, including science, social science, humanities, arts, medicine, engineering, and business.

[3] AP Computer Science: Principles – Big Ideas, Key Concepts and Supporting Concepts (2010, The College Board and National Science Foundation).

In addition, Stanford University's[4] undergraduate program on 'Ways of Thinking/Ways of Doing' describes formal reasoning as a way of thinking that involves rigorous deductive (logical and analytical) thinking. This way of thinking underpins decision making and analysis skills that are applied in many fields, including Computing.

Rationale for Change

The core focus of the OSIE Computer Studies Syllabus was on ICT. An environmental scan on related overseas syllabi revealed the need to shift the focus of the new O-Level Computing syllabus to emphasise computer science concepts and skills such as programming which are critical for the 21st century.

Table 1.1: ICT and Computer Science Modules in the Two Syllabuses

Subject	O-Level Computer Studies	O-Level Computing
Emphasis	ICT	Computer science
ICT modules	• Applications of computer and their social and economic implications • Generic software and organisation of data • Hardware, systems and communications	A unit of study on Data management with spreadsheet software application to support the learning of computer science concepts such as functions and conditional statements.
Computer science modules	• Analysis of the system • Problem solution, including design and programming concepts (does not require the use of programming language)	• Data and Information • Systems and Communications • Abstraction and Algorithms • Programming (require the extensive use of a programming language to code problem solutions)

Table 1.1 shows the ICT and computer science modules in the two syllabuses. The ICT component in the O-Level Computing subject will leverage on the use of the internet and spreadsheet application software to support students' learning of computer science concepts. Students will learn how to use selected mathematical, statistical and financial functions; as well as conditional statements or expressions in the spreadsheet application software to execute tasks. They are expected to have, as baseline ICT skills, the knowledge of different data types and the ability to organise data into a table, plot charts, and print data tables and charts.

A textual programming language is used intensively in the computer science modules; especially for the modules on Abstraction and Algorithms, and Programming. Students will learn to code computer solutions for various real-world problems so that they could develop computational thinking skills, as well as apply concepts and skills taught. They will also

4 The study of undergraduate program at Stanford: Ways of Thinking/Ways of Doing, a capacity-based approach to fostering breadth in General Education Electives.

develop systems thinking skills, particularly, through the study of the Systems and Communications module.

Design Elements

Core Areas

The big ideas in computing that we have distilled for syllabus design are classified as core areas of study. The core areas and definitions are:

Core Area	Definition
• Data and Information –	Ways of managing and processing data so that these data can be turned into meaningful and useful information to serve its purpose.
• Systems and Communications –	Ways of getting the individual parts of a computer to work together to form a computer system.
• Abstraction –	Ways of thinking about a problem.
• Algorithms –	Ways of planning the steps to solve a problem.
• Programming –	Ways of giving instructions to computers to execute a solution.

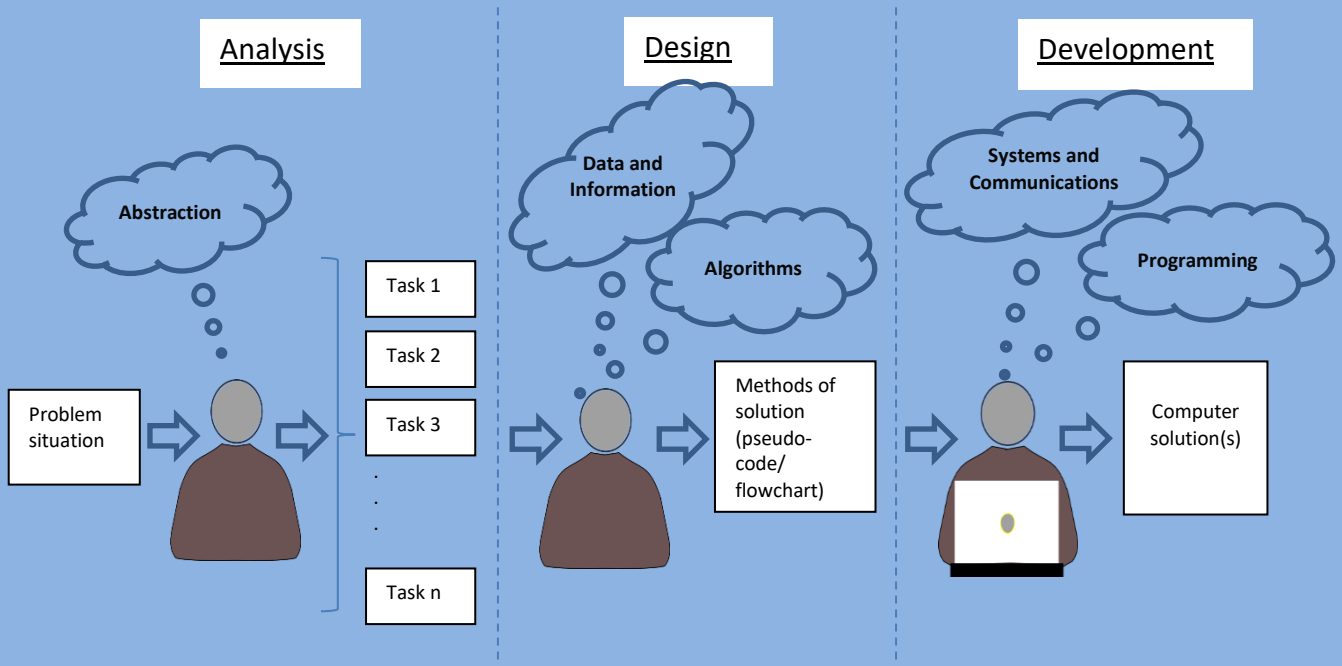


Figure 1.2: Using knowledge and skills from the five core areas in the problem-solving process

These five core areas are selected such that students combine the knowledge and skills gained from these areas to produce computational solutions for medium- to large-scale or complex problems. As illustrated in Figure 1.2, when presented with a problem situation, the student considers the kind of data and information required for the problem (**Data and Information**), decides how the problem could be solved by removing unnecessary complexity (**Abstraction**) and designs the method of solution in the form of pseudo-code/flowchart (**Algorithms**). The student also determines the type of system set-up needed to implement the solution(s) keeping in mind the relevant data required for the solutions (**Systems and Communications**). The student finally applies the knowledge of using a programming language to create the solutions for the various tasks (**Programming**) that will be executed and tested in computer(s) or computing device(s) before combining them to form the overall solution required for the problem.

Curriculum Objectives

Based on the big ideas, curriculum objectives are formulated for the study of the subject at various levels; from upper primary to lower secondary to upper secondary and junior college, and covering the five core areas in varying degrees. Curriculum objectives are enacted through the syllabus aims, content and learning outcomes for students.

Broadly, the curriculum objectives at upper secondary level are:

1. Understand algorithms that reflect computational thinking;
2. Use logical reasoning to compare the utility and feasibility of algorithms and programming solutions in attaining the desired outcomes;
3. Use a textual programming language to solve a variety of computational problems; making appropriate use of data structures (e.g. list, table, array), including the use of functions and procedures;
4. Understand that data and instructions are represented and stored in the form of binary digits (bits);
5. Understand the hardware and software components of a computer network system and how they communicate with one another and with other systems;
6. Use a range of computer technology safely and responsibly, and to create useful products and services.

Syllabus Aims

The syllabus aims to provide students with the foundation to continue with further studies in computing and skills to participate in a rapidly changing technological environment so that the concepts and skills learnt would also be applicable in other fields that require computing. The two-year course at upper secondary level is to enable students to:

1. Apply logical reasoning and algorithmic thinking[5] in analysing problem situations and developing solutions;
2. Develop simple programs through the use of appropriate programming language(s);
3. Understand how and where information communications technology (ICT)[6] is used in daily life;
4. Understand and explain the ethical, social and economic issues associated with the use of ICT.

Students will be taught how data can be managed and processed by using functions and formulas in a spreadsheet application to produce information and inform decision making; how digital systems work and how this knowledge can be used to create safe homes and secured networks; and how to use the computer efficiently and effectively to code solutions for problem situations. Details of what the students will be taught are in the following section.

[5] Algorithmic thinking is a way of getting to a solution through clear definition of the steps. Instead of a single answer, a set of instructions or rules is developed, that if followed precisely leads to answers.

[6] Information and Communications Technology (ICT) refers broadly to technology involving computing devices, software and other hardware.

SECTION 2: CONTENT

Syllabus Overview
Module I: Data and Information
Module II: Systems and Communications
Module III: Abstraction and Algorithms
Module IV: Programming
Curriculum Time

2. CONTENT

Syllabus Overview

This syllabus comprises four modules of study to cover five common areas of computer science concepts and skills. The study is undertaken at the upper secondary levels for two years. The four modules and the units of study for each module are as listed with details in subsequent pages. Schools are encouraged to have in place, at lower secondary levels, a computer science programme that gives students a firm foundation to undertake the subject at upper secondary levels.

The four modules are:

Module I – Data and Information

- Data Management
- Data Representation
- Ethical, Social and Economic Issues

Module II – Systems and Communications

- Computer Architecture
- Data Communications

Module III – Abstraction and Algorithms

- Problem Analysis
- Algorithm Design

Module IV – Programming

- Program Development
- Program Testing

Although the modules are presented as individual modules emphasising software or hardware, the value of computing lies in the integrative use of software and hardware to create new artefacts and solve or address real-world problems. Thus, teaching and learning approaches should leverage on the inter-connectedness of these modules and skilful integration of the learning objectives.

Module I: Data and Information

This module is about the handling and processing of data in computer systems, as well as the need to be ethical when dealing with data. Students should be able to identify different types of data, understand and explain what the data is for, and explain how the data is represented or organised for processing and output, with reference to a given problem. Students will be more aware of ethical issues with respect to data, including privacy of data. There are three units of study in this module:

1.1 Data Management
1.2 Data Representation
1.3 Ethical, Social and Economic Issues

1.1 Data Management

This unit of study emphasises logical thinking and reasoning through data analysis, data processing and visual representations of data. Students will use the spread sheet application software in hands-on activities to enhance learning of functions and formulae to compute and process data.

Ref	Learning outcome	Notes for teacher
	Students should be able to	
1.1.1	Tabulate data under appropriate column headings (i.e. data field names) and data types (e.g. numeric, text and date). *Exclude: plotting of charts*	Organise and categorise data by putting them in columns with meaningful headings. Explain that the columns represent the structure of the data table. Highlight the need to format (cells in) columns accordingly for the different data types used.
1.1.2	(a) Use mathematical operators, functions and what-if data analysis (goal seeking) to prepare spreadsheets and solve real-world problems such as - find the total of a list of numbers, average of a list of numbers, min/max of a list of numbers, square root of a number, simple interest, remainder after division of numbers; - round values; - randomise values; - convert data from one type to another (e.g. get integer values from decimal values); and - count number of data items.	Give examples of problem situations where these functions are applied. Explain and illustrate the difference between absolute and relative cell referencing.

<table>
<tr><th>Ref</th><th>Learning outcome</th><th>Notes for teacher</th></tr>
<tr><td></td><td colspan="2">Students should be able to</td></tr>
<tr><td></td><td>(b) Understand and use conditional statements (simple and nested): COUNTIF and IF with relational and Boolean operators such as AND, NOT and OR.

<u>Include:</u> conditional formatting

(c) Use functions effectively to look up data in rows or columns (horizontal and vertical table lookups) in a list or table for data processing.

The list of 33 examinable functions are:
<table>
<tr><th>Area</th><th>Function</th></tr>
<tr><td>Date and Time</td><td>TODAY</td></tr>
<tr><td>Text</td><td>LEFT, LEN, MID, RIGHT</td></tr>
<tr><td>Logical</td><td>AND, IF, NOT, OR</td></tr>
<tr><td>Lookup</td><td>HLOOKUP, VLOOKUP</td></tr>
<tr><td>Mathematical</td><td>CEILING.MATH, FLOOR.MATH, MOD, POWER, QUOTIENT, RAND, RANDBETWEEN, ROUND, SQRT, SUM, SUMIF</td></tr>
<tr><td>Statistical</td><td>AVERAGE, COUNT, COUNTA, COUNTBLANK, COUNTIF, LARGE, MAX, MEDIAN, MIN, MODE.SNGL, SMALL</td></tr>
</table>
Students should be able to use absolute and relative cell referencing.

Students should not be penalised for using equivalent functions not listed.</td><td></td></tr>
</table>

1.2 Data Representation

This unit of study explains how data is represented internally as binary numbers, how the conversion of data from one number system to another is done, and where these number systems are used. The number systems covered here are binary number system, decimal number system and hexadecimal number system.

Ref	Learning outcome	Notes for teacher
	Students should be able to	
1.2.1	Represent positive whole numbers in binary form.	Bit = binary digit and is 0 or 1.
1.2.2	Convert positive whole numbers from one number system to another - binary, denary and hexadecimal; and describe the technique used.	The data is represented in not more than 16 bits, and the description of the conversion techniques should be in short prose.

Ref	Learning outcome	Notes for teacher
	Students should be able to	
1.2.3	Describe situations in which the number systems are used such as ASCII codes, IP (Internet Protocol) and Media Access Control (MAC) addresses, and RGB codes.	Explain how the number system is being used in each of the specified areas.

1.3 Ethical, Social and Economic Issues

This unit of study covers ethical use and security of data in desktop applications or applications on networks like the internet. Social and economic issues arising from the use of computing technologies in workplace settings are discussed.

Ref	Learning outcome	Notes for teacher
	Students should be able to	
1.3.1	Understand how data can be kept safe from accidental damage due to data corruption or human errors and malicious actions such as unauthorised viewing, deleting, copying and corrupting or malware.	Understand and explain measures that could be taken such as: • Backup data • Safe and secure storage of data • Use of passwords and/or biometrics • Different levels of access rights for different levels of use
1.3.2	Understand the effects of threats to privacy and security of data from spam, spyware, cookies, phishing, pharming and unauthorised access as well as defensive measures employed such as the use of appropriate hardware and software.	There is a need to keep abreast with the trends from time to time.
1.3.3	Describe ethical issues that relate to the use of computers, public/private networks, freeware, shareware and open courseware; and the sharing of information.	Good to know what “Creative Commons Licensing” is and how people should behave when accessing resources on private and public networks.
1.3.4	Compare the positive and negative social and economic impacts of technology in education, communication (e.g. via mobile apps and social media), finance, medicine/healthcare, transportation and entertainment; and explain plagiarism and software piracy.	The different careers in computing and the use of computing technology will provide contexts for discussion of ethical, social and economic issues. Give examples of situations where plagiarism and software piracy could occur and why.

Module II: Systems and Communications

This module is about systems involving computer technology and computing devices. Students will learn the inter-relationships between parts and whole of a system; as well as the functions of parts of systems in enabling communications between human and computing device (machine), machine and machine, and within a machine. There are two units of study:

2.1 Computer Architecture
2.2 Data Communications

2.1 Computer Architecture

This unit of study covers the basic computer architecture and components of a computer network. Students will understand the purpose of hardware and software required in a computer system or network but they are not required to know how these work technically. Students will have the opportunity to work with hardware components and understand the integrative use of hardware and software through learning by doing.

Ref	Learning outcome	Notes for teacher
	Students should be able to	
2.1.1	Describe basic computer architecture with reference to - Computer processor - Memory - Data and address buses - Input (e.g. data and instructions) and output (e.g. intermediate and final results of processing) - Storage media *Exclude: Fetch and execute cycle*	The focus is on the "internals" of a computer. No technical details are needed.
2.1.2	Recognise a logic gate from its truth table and evaluate Boolean statements by means of truth tables.	Know and understand the truth table for each of the following logic gates: AND, OR, NOT, NAND, and NOR. Source for simple DIY kits as a teaching aid to enhance learning through a hands-on approach.
2.1.3	Construct the truth tables for given logic circuits (maximum 3 inputs).	
2.1.4	Design and construct simple logic circuits using AND, OR, NOT, NAND and NOR.	

2.2 Data Communications

This unit of study uses networks as the context for understanding sharing of resources and data communications. Students should have a general understanding of how data is transmitted and the need to check for data accuracy and data security in transmissions.

Ref	Learning outcome	Notes for teacher
	Students should be able to	
2.2.1	Identify and explain the function of different network hardware: modem, network interface controller, hub, switch and router.	Name each device and explain its function but no technical details about how each device operates are needed. It may be necessary to explain what a modem (modulator-demodulator) is. It is assumed that students know the following input and output devices: • Monitor • Keyboard • Mouse • Printer/plotter • Scanner • Web camera
2.2.2	Describe the difference between wired and wireless networks and explain the factors that will determine the use of each type of network. *Include: descriptions of LAN/MAN/WAN*	
2.2.3	Describe the components for a simple home network and design a simple home network.	The components could be all or part of those mentioned in 2.2.1. Related concepts include IP (internet protocol) address, MAC (media access control) address, ports, SSID (service set identifier) and access points. As part of hands-on activities to enhance learning, students may engage in the setting up of a simple network.
2.2.4	Compare and contrast client-server and peer-to-peer network strategies with emphasis on: - Purpose - Function - Organisation - Bandwidth *Include: topology principles of bus, ring and star networks*	Function refers to how the network operates. Organisation refers to how the hardware components are laid out or connected. Speed can be mentioned under bandwidth. It is not required to know technical details about the industry standards or materials used to manufacture any of the items in a network.
2.2.5	Explain the use of parity and checksums in data transmission.	Use 8 – 16 bit patterns.

Module III: Abstraction and Algorithms

This module is about problem solving and how a problem may be solved by breaking it into smaller, manageable parts and solving all the smaller parts. An algorithm describes a solution for the problem that is independent of a programming language and may be presented in pseudo-code (where program structures will be more pronounced) or diagrammatically (flowchart). Students should be able to know the difference between pseudo-code and flowchart. There are two units of study:

3.1 Problem Analysis
3.2 Algorithm Design

3.1 Problem Analysis

This unit of study covers problem interpretation and analysis. Students will learn how to approach problem solving in a systematic manner. They will learn problem-solving strategies like breaking a problem into its parts and solving the problem by first solving the individual parts. They will have opportunities to reinforce their understanding through hands-on activities in solving simple real-world problems.

Ref	Learning outcome	Notes for teacher
	Students should be able to	
3.1.1	For a given problem, specify the • inputs and the requirements for valid inputs • outputs and the requirements for correct outputs Examples: • Business: produce an itemised list of items purchased, cost of each item, total cost payable (like a receipt) or calculate interest on mortgages and print instalments over a period of time • Education: find and print the student with the top score in each subject or check user inputs against expected input (like test scoring); find the mean subject grade (MSG) for a class • Scientific/Mathematics: determine whether an input number is odd or even and whether a number is divisible by another number • Entertainment: a number guessing game or any game involving text manipulation	Students should be able to express a given problem as a computational problem and be able to write the solutions for these problems in the form of algorithms.

Ref	Learning outcome	Notes for teacher
	Students should be able to	
3.1.2	Solve problems by decomposing[7] them into smaller and manageable parts.	Students should be able to identify the parts and create solutions for the parts. They will then test these partial solutions and the total solution where the partial solutions are put together.
3.1.3	Identify common elements across similar problems and state generalisations[8].	

3.2 Algorithm Design

This unit of study is about interpreting and understanding algorithms; correcting and writing algorithms for given problems; as well as refinement of algorithms. Students' learning will be enhanced through written exercises, classroom discussions and presentations.

Ref	Learning outcome	Notes for teacher
	Students should be able to	
3.2.1	Perform a dry run of a set of steps to determine its purpose and/or output.	This method could be used to check for logic errors or mistakes in faulty algorithms.
3.2.2	Produce trace tables to show values of variables at each stage in a set of steps.	
3.2.3	Locate logic errors in an algorithm, and correct or modify an algorithm to remove the errors or for changes in task specification.	Find and identify logic errors in algorithms with the intent of correcting or modifying the algorithms.

[7] Decomposition is a way of thinking about problems, algorithms, artefacts, processes and systems in terms of their parts. The parts can then be understood, solved, developed and evaluated separately. This approach makes it easier to solve complex problems and design large systems.

[8] Generalisation is a way of quickly solving new problems based on previous problems that had been solved. That is, the algorithm that works for a previous specific problem is adapted so that it solves a whole class of similar problems through the identification of patterns and commonalities in problems, processes, solutions, or data.

3.2.4	Produce an algorithm to solve a problem, either as a flowchart or in pseudo-code. The following pseudo-code keywords and constructs should be used: • INPUT/ OUTPUT • IF… THEN…ELSEIF… ELSE… ENDIF • WHILE… ENDWHILE • FOR… NEXT	To make it easier for students to switch between pseudo-code and Python code, teachers can present pseudo-code to students in the following style: • list indices should start from 0 • assignment operations should be represented by "=" • equality should be represented by "==" • inequality should be represented by "!=" Teachers should be aware that students are not required to follow this style in their examination answers. By definition, pseudo-code is not a programming language with a defined, mandatory syntax. Any pseudo-code presented by students will be assessed for the logic of the solution presented – where the logic is understood, and correctly solves the problem addressed, the student should be given credit regardless of whether the student has followed the style above. Using a recommended style will, however, enable the student to communicate their solution more effectively. [extracted from Pseudocode Guide for Teachers Cambridge International AS & A Level] For flowcharts, please refer to Annex B (Quick Reference for Flowcharting).

Module IV: Programming

This module is about application and development of logical thinking and reasoning, as well as problem-solving skills through the design and development of software solutions using programming language(s). An algorithm describes a solution independent of a programming language while a programming language depicts the solution that is workable on a computing device. Students will get to test whether their algorithms work as planned when they run the programming solutions. There are two units of study:

4.1 Program Development
4.2 Program Testing

4.1 Program Development

This unit of study covers the development of programming solutions (coding) for simple problem situations. Students will reinforce their understanding through practical work. Python is the programming language for this syllabus.

Ref	Learning outcome	Notes for teacher
	Students should be able to	
4.1.1	Understand and describe the stages in developing a program: gather requirements, plan solutions, write code, test and refine code, deploy code.	Revisit the tools that can be used for planning. Run through all the stages in the formulation of a programming solution for a problem situation so that it becomes a thinking routine.
4.1.2	Understand and use sequence, selection and iteration constructs to create a program.	Python is the programming language for this syllabus. The style of the programming language must be adhered to and good practices emphasised. For example, • variable names should be lowercase, with words separated by underscores to improve readability • constants are usually written in all capital letters with underscores separating words.
4.1.3	Use and justify the use of variables, constants and simple lists in different problem contexts.	

4.1.4	Understand and use different data types (integers, floating-point numbers, strings, Booleans, lists) and built-in functions.	Refer to Annex A for a list of built-in functions that should be covered. Generally, built-in functions help to simplify code and can be used in programming solutions. However, the use of built-in functions should not trivialise a problem's main task. Problems should specify any restrictions they may have on the use of built-in functions. However, this may not always be possible given the large number of functions in Python's built-in library. For example, where the main task is to find: (a) the smallest number in a given list, students should know and be able to write out the steps for the process. They should not be allowed to use the built-in function, min(). (b) which month has the lowest rainfall for the past year given daily rainfall, students should be allowed to use the built-in function, min(), after rainfall values for each month are calculated and stored in a list.
4.1.5	Produce programming solution for a given problem to • find min/max value in a list • find average of a list of numeric values • search for an item in a list and report the result of the search • find check digits • find the length of a string of characters • extract required characters from a string of characters	This may include but not limited to the following skills: - Compare two items - Interchange two items - Use of mathematical formulae A string may contain letters, digits or symbols (e.g. + and :).
4.1.6	Write and use user-defined functions.	Keyword arguments are excluded.

4.2 Program Testing

This unit of study covers the testing and refinement of programs based on test results. Students will learn the appropriate use of test cases and what type of testing is necessary and/or sufficient. Students will reinforce their learning and understanding through hands-on practical work. Python is the programming language for this syllabus.

Ref	Learning outcome	Notes for teacher
	Students should be able to	
4.2.1	Identify and justify the use of data validation techniques.	*Exclude: Validation of data type (type check).* This means that students do not have to write programs to check for data types. Type check is excluded as error messages will usually be generated by the programming language interpreter.
4.2.2	Validate input data for acceptance by performing • length check, • range check, • presence check, and • format check	
4.2.3	Design appropriate test cases to cover normal, error and boundary conditions, and specify what is being tested for each test case.	Boundary – lower and upper limits.
4.2.4	Understand and describe types of program errors: syntax, logic and run-time; and explain why they occur.	
4.2.5	Explain how translators are used to detect syntax errors and state the difference between interpreter and compiler translators.	
4.2.6	Understand and apply debugging techniques to isolate/identify and debug program errors: using intermittent print statements, walking through or testing a program in small chunks or by parts.	

Curriculum Time

The total curriculum time for O-Level Computing is 46 weeks with 3 hours per week (144 hours) over 2 years. A summary of the estimated number of weeks and hours for the topics or content explications per module is provided in Table 2.1.

Table 2.1: Curriculum Time per Module

Content Details	Estimated Curriculum Time	
	~ No. of Weeks	~ No. of Hours
Module I: Data and Information	**9**	**27**
1.1: Data Management		
a. Data Tabulation, Conditional Statements and Operators	4	12
b. Mathematical and Statistical Functions		
c. Data Lookup, Boolean, Date and String Functions		
1.2: Data Representation		
a. Number Systems	2	6
1.3: Ethical, Social and Economic Issues		
a. Data Safety and Access	1	3
b. Social and Economic Impacts	1	3
c. Ethical Issues	1	3
Module II: Systems and Communications	**8**	**24**
2.1: Computer Architecture		
a. Basic Hardware Components	1	3
b. Boolean Logic and Logic Gates	1	3
c. Logic Circuits and Truth Tables	1	3
2.2: Data Communications		
a. Introduction to Networks	2	6
b. Wired and Wireless Networks		
c. Home Network	2	6
d. Client-Server vs Peer-to-Peer Network		
e. Parity and Checksums	1	3
Module 3: Abstraction and Algorithms	**8**	**24**
3.1: Problem Analysis		
a. Problem Statements	1	3
b. Modularity and Generalisation	1	3
3.2: Algorithm Design		
a. Pseudo-code	1	3
b. Program Flowchart and Flowchart Symbols	2	6

c. Dry Runs and Trace Tables	1	3
d. Logic Errors	2	6
Module 4: Programming	**15**	**45**
4.1: Program Development		
a. Concept of a program and Program Constructs	1	3
b. Python Syntax, Constants and Variables	2	6
c. Arithmetic and Relational Operators		
d. Numeric Data Types	2	6
e. Data Types - Strings and Console Input/output		
f. Selection Statements	2	6
g. Loops	2	6
h. Lists	2	6
i. User-defined functions	1	3
4.2: Program Testing		
a. Test Cases	1	3
b. Program Errors & Translators	1	3
c. Debugging Techniques		
d. Data Validation	1	3
Python Mini Project	**6**	**18**

SECTION 3: PEDAGOGY

Applied Learning
Learn by Doing
Teaching Actions
Resources
Hardware and Software

3. PEDAGOGY

Applied Learning

As an applied subject, there is greater emphasis on learning by doing in the real-world context. The pedagogies and activities used must thus be suitable for applied learning. In addition, it is intended that students be provided with opportunities to acquire 21st Century Competencies through applied learning.

21st Century Competencies (21CC)

Table 3.1 illustrates how the O-Level Computing curriculum is aligned with the Standards and Benchmarks for 21CC. To illustrate, students will develop critical and inventive thinking skills (2.1d) when they are tasked to design a program to compute point-of-sales receipts. The students will need to assess information and process requirements, and analyse costs and benefits before applying sound reasoning and decision-making to propose a feasible solution with respect to the hardware and software to be acquired.

Table 3.1: Alignment with 21CC Standards and Benchmarks

O-Level Computing Competencies and Attitudes	**21st Century Competencies Benchmarks (By end of S4/S5)**
Computer as a Science	**Critical and Inventive Thinking (CIT)**
Ability to brainstorm ideas for problem solutions.	1.1d: The student is able to generate ideas and explore different pathways that lead to solutions.
Ability to apply computational thinking by: • synthesizing knowledge and skills from the five core Computer Science areas; and • applying formal reasoning and systems thinking in the analysis, design and implementation of computer solutions.	2.1d: The student is able to use evidence and adopt different viewpoints to explain his/ her reasoning and decisions, having considered the implications of the relationship among different viewpoints.
Ability to debug and refine computer programs (computational thinking).	2.2d: The student is able to suspend judgement, reassess conclusions and consider alternatives to refine his/ her thoughts, attitudes, behaviour and actions.

Ability to: • analyse and simplify problems into manageable tasks (analytical thinking); • persist in developing computer solutions for the problems (resilience); and • evaluate solutions using test cases (evaluative thinking).	3.1d: The student is able to identify essential elements of complex tasks, stay focused on them, take on diverse roles and persevere when they encounter difficulties and unexpected challenges.
Computer as a Tool	**Information and Communication Skills (ICS)**
Ability to explain and communicate programming solutions.	1.2d: The student is able to use information and ideas developed collectively with others to create new information, products and/ or solutions.
Ability to be resourceful in searching for pertinent information required to solve the problem.	2.1d: The student is able to integrate information from a variety of sources to complete a task.
Computer in Society	**Civic Literacy, Global Awareness and Cross-cultural Skills (CGC)**
Ability to understand data protection, copyright issues and intellectual property rights.	1.1d: The student is able to discuss issues that affect the culture, socio-economic development, governance, future and identity of Singapore and use evidence to support their viewpoints.
Adopt ethical practices in cyberspace and social media.	

Authentic Situations

Real case studies on data protection, for example, will be used for students to inquire the need for data privacy and integrity, and to compare/contrast the different measures of data protection. Students will realise the dangers and understand the mistakes made by others through online transactions or sharing of information via the internet; and will learn to be discerning and careful with their personal information while adopting ethical practices in cyberspace.

Learn by Doing

A wide range of pedagogies is recommended for the teaching of O-Level Computing. For example, using a flipped classroom approach, the students may watch a video (at their own time) on the design considerations and actual setup of a simple wireless network in a home or business setting. The students will then set up a simple network physically in the computer classroom or use a web-based similar available in the Student Learning Space and learn by doing. While the pedagogies applied are dependent on the nature of the topic, a problem-driven approach could be used.

Problem Tasks

A set of carefully designed tasks will be developed for teachers to take the students through the learning objectives of the O-Level Computing syllabus over the two years of study. These tasks and the experiential learning through individual, paired or group work coupled with discussions and reflections will allow students to acquire and develop the 21st Century competencies.

Details of lesson plans and lesson ideas will be provided in the teaching and learning guide. The use of code is an essential part of testing the algorithms developed and a manifestation of algorithmic thinking (Syllabus Aim 1) and hence, computational thinking. Thus, the teacher must be aware of the possible lab-based activities that could support or enhance student's learning. The following examples are provided to illustrate possible problem tasks for O-Level Computing.

Example 1

A snack shop sells the following items:

Item	Selling price
Bun	$0.80
Coffee	$1.20
Cake	$1.50
Mango pudding	$1.80
Fruit salad	$3.20

Write a program which

(a) captures the items sold each day,
(b) ends the input of items when "XXX" is entered as item name,
(c) finds the total number of items sold,
(d) finds the total amount of money collected for the day,
(e) finds the item that sold the most for the day,
(f) displays the total amount of money collected for the day, and
(g) displays the item that sold the most for the day.

Example 2

Write a program which

(a) reads in from the keyboard a positive whole number,
(b) determines the number of digits in the positive whole number,
(c) displays the positive whole number and the number of digits,
(d) reverses the order of the digits in the whole number and add this new whole number to the original number, and
(e) displays the sum of these two whole numbers.

Example 3

Write a program which

(a) reads a word or phrase from the keyboard,
(b) counts the frequency of each vowel in the word or phrase,
(c) finds the vowel that appears the most,
(d) displays each vowel and its frequency, and
(e) displays the vowel with the highest frequency.

Programming Project

Students will also be involved in a school-based programming project over 6 weeks where they will have to apply computational thinking to create programming solutions for one or more tasks. Students may work independently, in pairs or as a group depending on the project. They may find their own projects to work on.

A blended learning approach will be recommended for the programming project as the specific purpose is to find out how well students have learnt the programming and computing concepts and their ability to perform the various tasks expected of them; as well as to rectify gaps in pedagogical scaffolding when students learn by doing and self-direct their own learning. The students can consult the teachers or any other resource persons. They may also learn from the resources available on the internet.

The programming project also provides an opportunity for students to showcase their innovative skills while developing a practical computational solution. These skills will be evident in the design and algorithmic thinking as well as rigorous testing of the developed product.

Examples of Python programming projects:

- Monitor temperature: You will construct a device to monitor the temperature of anything, e.g. fish tank, bath water or fridge; using a Raspberry Pi and a waterproof temperature sensor as the main components. The project can be extended to turn the setup into an Internet of Things (IoT) device that would text you when the temperature gets too hot. (Reference: Simon Monk, learn.adafruit.com)

- Minecraft Pi: You can explore the virtual world of Minecraft Pi, the special edition of Minecraft made for Raspberry Pi. You will manually build blocks and use the Python interface to manipulate the world around you, create a new world and post text to the chat window. (Reference: raspberrypi.org)

There are also several other Raspberry Pi projects. For example,

- Donald Norris, Raspberry Pi Projects for the Evil Genius
- Simon Monk, Raspberry Pi Cookbook

Besides the Raspberry Pi projects, there are projects with the Arduino board that students could engage in but Python may not be the programming language used. For example,

- John Boxall, Arduino Workshop – A hands-on introduction with 65 projects
- Simon Monk, 30 Arduino Projects for the Evil Genius

Student Learning Space

Students' programming assignments could be graded by teachers or through an online grading system. Feedback on students' work could be provided by teachers or the online system. Students could check and clarify their understanding with the teacher during contact time.

Teaching Actions

Table 3.2 below provides examples of how Teaching Actions from Singapore Teaching Practices (STP)[9] can be enacted during Computing lessons.

Table 3.2: Teaching Actions applicable to teaching of Computing

Teaching Areas	Teaching Actions	Examples of how it can be used in the classroom
Arousing Interest	**Discrepant Events** Teachers use a discrepant event to introduce to a surprising outcome that students do not expect thus arousing interest.	• Teachers get students to predict the outcome of a program and compare with the actual outcome. The students can then embark on further discussion on why they are different. • Teachers give students a program with errors to get them to identify and correct the errors.
Providing Clear Explanation	**Demonstration** Teachers demonstrate a 'walk through' of a new skill during which students learn by observing.	• Teachers demonstrate solving a programming task from scratch to show the thinking processes and techniques involved when developing a program.
Using Questions to Deepen Learning	**Initiate-Response-Feedback Chains** Teachers use questions to elicit, probe and scaffold students' thinking.	• Teachers ask questions such as "What makes you say that?" after students have given a response to help students identify the basis for their thinking as they elaborate on the reasoning behind their responses.

[9] Visit https://opal.moe.edu.sg/stp for more information on STP.

Table 3.2: Teaching Actions applicable to teaching of Computing (continued)

Encouraging Learner Engagement	**Explore, Engage, Apply** Teachers design learning activities that are meaningful and relevant to students.	• Teachers prepare cards (in powers of 2) to demonstrate conversion from denary to binary.[10] • Teachers introduce programming concept of looping through the use of dance moves (e.g. Macarena).
	Engagement through Collaboration and Interactivity Teachers assign students to work collaboratively in pairs. One student (the driver) has control of the keyboard and mouse, while the other (the navigator) looks at the big picture and provides comments. The students are to switch roles from time to time.	• Teachers carry out pair programming by pairing students of similar ability, assigning one to be the 'driver' and the other the 'navigator'. The driver writes the program while the navigator considers the requirements and checks for errors.
Facilitating Collaborative Learning	**Think-pair-share** Students first think through a problem alone and then discuss in pairs. This is followed by consolidation led by teacher with the whole class.	• Teachers use think-pair-share to identify and generate test cases for a programming task.

Resources

There will be affordances for ICT and self-directed learning through the use of resources from the online learning portal for teachers and students. There will be instructional materials in the form of textbook, task sheets and worksheets to support teaching and learning. A quick reference on Python programming language constructs and elements (see Annex A) is provided for students' use in lab-based activities during curriculum study or examination.

[10] Visit https://classic.csunplugged.org/binary-numbers/ for more information.

Hardware and Software

The list of recommended software is:

- Microsoft Office Productivity Suite
- Python Programming Language

Schools may, at their own discretion, use other software to aid in the teaching and learning of computing concepts.

Schools can also use appropriate hardware to allow students to apply what they have learned or to have a better understanding of computing concepts. For example, schools can use the Raspberry Pi, microcontrollers and programmable gadgets to teach computer concepts like computer architecture and simple computer or network system. Students can develop creativity and inventive skills by programming these devices to create useful innovative products with real-world applications.

SECTION 4: ASSESSMENT

School-based Assessment
Objectives of National Examination
Scheme of National Examination

4. ASSESSMENT

School-based Assessment

Assessment for Learning

Students are provided with opportunities for deep and enriched learning experience through the use of online learning environments in the Student Learning Space or the internet. They can monitor and self-regulate their pace and progress of learning. They receive immediate feedback from these learning environments on errors in their programming solutions and whether the outcomes of their programming solutions are as intended.

The six weeks set aside for the programming project will enable students to collaborate and learn from one another. Besides assessment for learning, assessment of learning also takes place when students apply programming concepts and skills that they have learnt to the project. Computational thinking would also be developed and if the project is done on a group basis, students would also learn to explore different pathways to solutions and enhance communication skills.

Assessment of Learning

The Student Learning Space will also allow students to assess their learning as well as learn from the assessment tasks. School-based assessment can also consist of timed written and practical assessment of students' understanding of the four modules in the syllabus. The questions should test more on students' understanding and application of concepts and skills learnt, and less on recall of information. The written paper may comprise short-structured questions of variable mark values.

A sample lab-based paper may consist of a spreadsheet task and one or two programming tasks. Alternatively, the paper may consist of a series of planning and progressively developed tasks culminating in a final programming solution.

The format of the assessment papers may also be modelled after the format of the national examinations. The marks for school-based assessment may be used for reporting students' performance at end of Semesters.

National Examination

Assessment Objectives
The examination will assess:

a) Knowledge and understanding of basic computing technology and systems, concepts, algorithms, techniques and tools;

b) Application of knowledge and understanding to analyse and solve computing problems;

c) Development, testing and refinement of solutions using appropriate software application(s) and/or programming language(s).

The weighting for each assessment objective is shown in Table 4.1.

Table 4.1 Specification Grid

Ref	Assessment Objective	Weighting in Paper 1	Weighting in Paper 2	Overall Weighting
(a)	Knowledge and understanding	~ 35%	~ 5%	40%
(b)	Application	~ 25%	~ 5%	30%
(c)	Development, testing and refinement	~10%	~20%	30%
	TOTAL	70%	30%	100%

Students can handle and process data in computer systems, as well as the need to be ethical when dealing with data. They will demonstrate problem-solving techniques through analysing and writing programming solutions for a range of computing problems in business, education, mathematics and science. Students will be able to demonstrate computational thinking through the design and development of programming solutions.

Scheme of National Examination

Mode of Examination

All candidates will offer Paper 1 and Paper 2. All questions are compulsory in both papers.

Paper 1 (Written examination, 2 hours)
This paper tests knowledge, understanding and application of concepts and skills in all the four modules. The questions consist of a mixture of

- short answer questions
- matching questions
- cloze passage
- structured questions

This paper carries 70% of the total marks, and is marked out of 80 marks.

Paper 2 (Lab-Based Examination, 2 hours 30 minutes)

This paper, taken with the use of a computer, spreadsheet and programming[11] software, tests Module 1 (Unit 1.1: Data Management) and Module 4 (Programming). Four structured questions will be set based on the following topics:

- Use of spreadsheet functions and features
- Refinement of program
- Debugging of program
- Development of program with no more than 40 lines

Development of program will carry 20 marks. The remaining three questions average 10 marks.

A quick reference for Python will be provided for candidates.

Candidates will submit soft copies of the required work for marking. The allotted time includes time for saving the required work in the candidates' computer. This paper carries 30% of the total marks, and is marked out of 50 marks.

Table 4.2 summarises the formats of the two compulsory papers.

Table 4.2: Format of Papers

Paper	Mode	Duration	Weighting	Marks	Format	Modules Assessed
1	Written	2 hours	70%	80	A mixture of • Short-answer questions • Matching questions • Cloze passage • Structured questions	All the four modules
2	Lab-based practical exam	2 hours 30 minutes	30%	50	4 compulsory structured questions • Use of spreadsheet functions and features • Refinement of programme • Debugging of programme • Development of programme with no more than 40 lines of code	Unit 1.1 Data Management from module 1 Module 4: Programming

[11] *The centre will be required to declare the programming language(s) to be used for the examination before the centre begins teaching the course for the cohort taking the examination.*

					Development of programme will carry 20 marks. The remaining three questions will carry an average of 10 marks per question. A quick reference for Python will be provided for candidates.	

Use of Calculator

An approved calculator may be used in Paper 1 and Paper 2.

Centre Infrastructure for Lab-based Examination

The Centre will ensure adequate hardware and software facilities to support the examination of its candidates for Paper 2, which will be administered over at most two shifts on the day of the examination. Each candidate should have the sole use of a personal computer for the purpose of the examination. Candidates should be able to access Spreadsheet application software and programming language software. The Centre will be required to declare the name and version number of the software to be used for the cohort sitting for the examination at least two years before the cohort sits for the examination.

Quick Reference for Python

This quick reference shows some examples of the Python language constructs. The complete Python language is not limited to these examples.

1. Identifiers

When naming variables, functions and modules, the following rules must be observed:

- Names should begin with character 'a' - 'z' or 'A' - 'Z' or '_' and followed by alphanumeric characters or '_'.
- Reserved words should not be used.
- User-defined identifiers are case sensitive.

2. Comments and Documentation Strings

```
# This is a comment

"""
   This is a documentation string
   over multiple lines
"""
```

3. Input/Output

```
print ("This is a string")

s = input ("Instructions to prompt for data entry.")
```

4. Import

import <module>

e.g. **import** math

5. Data Type

Data Type	Notes
int	integer
float	real number
bool	boolean
str	string (immutable)
list	series of values

6. Assignment

Assignment Statement	Notes
a = 1	integer
b = c	variable
d = "This is a string"	string
mylist = [1, 2, 3, 4, 5]	list

7. Arithmetic Operators

Operator	Notes
+ -	plus, subtract
* /	multiply, divide
%	remainder or modulus
**	exponential or power
//	quotient of the floor division

8. Relational Operators

Operator	Notes
==	equality
!=	not equal to
> >=	greater than, greater than or equal to
< <=	less than, less than or equal to

9. Boolean Expression

Boolean Expression	Notes
a and b	logical and
a or b	logical or
not a	logical not

10. Iteration

while loop
while condition(s): <statement(s)>

for loop
for i **in** range(n): <statement(s)>
for record **in** records: <statement(s)>

11. Selection

Type 1	Type 2	Type 3
if condition(s)**:** <statement(s)>	**if** condition(s)**:** <statement(s)> **else:** <statement(s)>	**if** condition(s)**:** <statement(s)> **elif** condition(s)**:** <statement(s)> **else:** <statement(s)>

12. Functions

```
# Function definitions
def <function name> (<parameters>):
    <function body>
    return <return value>

# Function calls
<function name>(<arguments>)
```

13. Built-in Functions

(a) Basic functions

abs()	chr()	float()	input()	int()
len()	max()	min()	ord()	print()
range()	round()	str()	<str>.endswith()	<str>.find()
<str>.format()	<str>.isalnum()	<str>.isalpha()	<str>.isdigit()	<str>.islower()
<str>.isspace()	<str>.isupper()	<str>.lower()	<str>.split()	<str>.startswith()
<str>.upper()				

(b) Math module

ceil()	floor()	pow()	sqrt()	trunc()

(c) Random module

randint()

14. Reserved Words

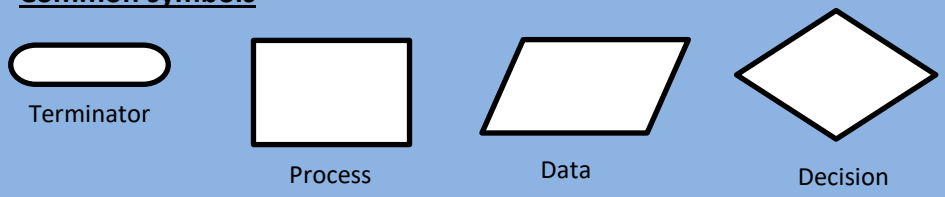
Reserved words are part of the syntax of the language. They cannot be used as identifiers.

False	None	True	and	as
assert	break	class	continue	def
del	elif	else	except	finally
for	from	global	if	import
in	is	lambda	nonlocal	not
or	pass	raise	return	try
while	with	yield		

Quick Reference for Flowcharting

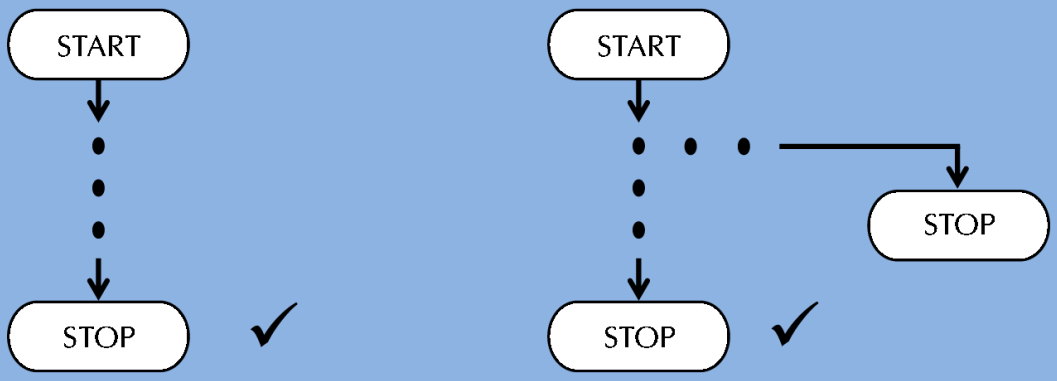
This quick reference shows four common symbols used in program flowcharts, and provide a standard guide for constructing program flowcharts.

1. Common symbols

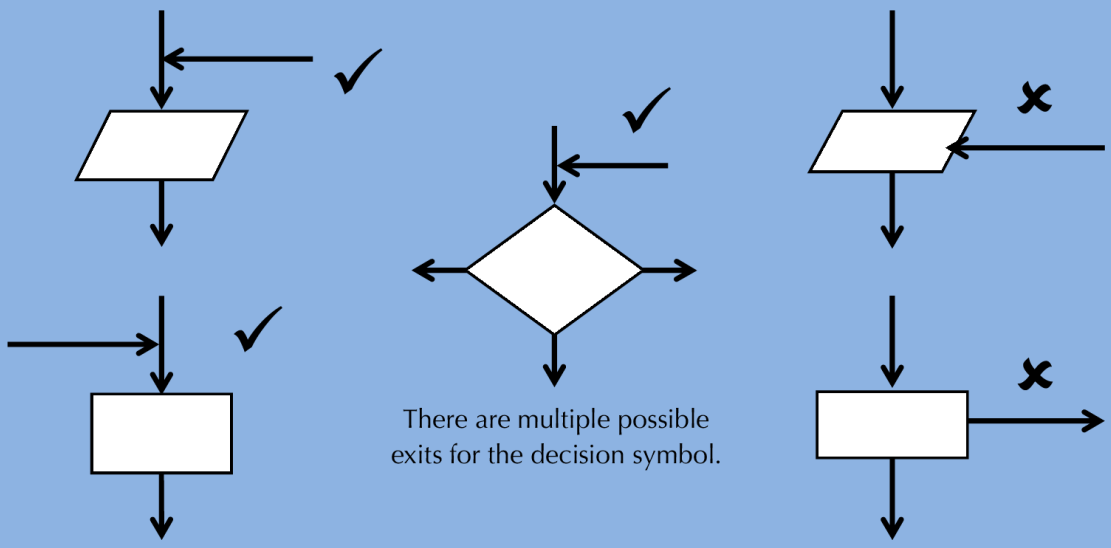


2. Constructing flowcharts

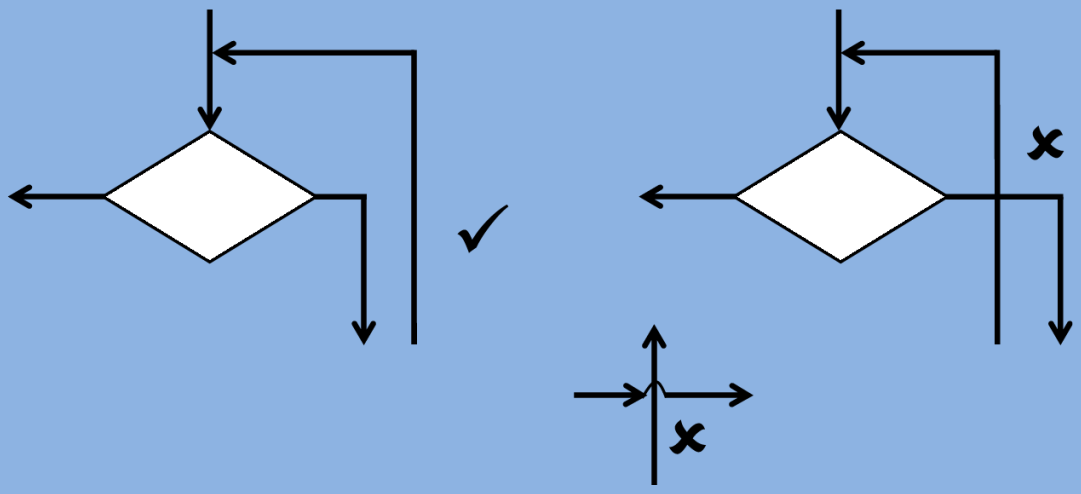
(a) Start with ONE terminator symbol



(b) Have a single entry and a single exit (except for the terminator and decision symbols)



(c) Flow lines should not cross one another



Some people used a kink in the flow line but this is not recommended for our students.

Glossary of Terms

Students should be familiar with the following terms in the learning outcomes, and these terms would be similarly used in the examination questions.

Term	Learning outcomes	Definition
Apply	4.2.6	Carry out or use common techniques or procedures.
Compare	1.3.4	Give an account of positive and negative aspects, or advantages and disadvantages of two items.
Compare and contrast	2.2.4	Map, match. Give an account of similarities and differences of two items.
Construct	2.1.3, 2.1.4	Produce, create. *Put elements together to form a coherent whole; reorganise into a new pattern or structure.*
Convert	1.2.2	Transform, express in another way or manner.
Correct	3.2.3	Rectify, make changes for improvement.
Describe	1.2.2, 1.2.3, 1.3.3, 2.1.1, 2.2.2, 2.2.3, 4.1.1, 4.2.4	Give a detailed account or give more information on the "what" and "how".
Design	2.1.4, 2.2.3, 4.2.3	Plan, visible thinking, evidence of ideas e.g. an artefact, a computer program, an outcome, a diagram, a drawing or a model.
Explain	1.3.4, 2.2.1, 2.2.5, 4.2.4, 4.2.5	Construct models. Includes elements of analysis. Give a detailed account or give more information on the "where", "how" and "why".
Evaluate	2.1.2	*Make judgements based on criteria and standards.*
Identify	2.2.1, 3.1.3, 4.2.1	Recognise, name. Ability to differentiate and discriminate.
Justify	4.1.3, 4.2.1	Give reasons or evidence to support an action or decision.
Locate	3.2.3	Find.
Modify	3.2.3	Make changes.
Perform	3.2.1	Do, carry out.

Term	Learning outcomes	Definition
Produce	3.2.2, 3.2.4, 4.1.5	Construct, create.
Recognise	2.1.2	Identify.
Represent	1.2.1	Present, express, show as.
Specify	3.1.1, 4.2.3	State.
State	3.1.3, 4.2.5	Say or write what something is about.
Solve	3.1.2	Take action to arrive at an outcome or decision.
Tabulate	1.1.1	Put in the form of a table.
Understand	1.1.2, 1.3.1, 1.3.2, 4.1.1, 4.1.2, 4.1.4, 4.2.4, 4.2.6	*Construct meaning from instructional messages, including oral, written, and graphic communication.*
Use	1.1.2, 4.1.2, 4.1.3, 4.1.4	Practise, implement, create, construct.
Validate	4.2.2	Check accuracy of.

The definitions in italics are based on the Revised Bloom's Taxonomy (Anderson and Krathwohl, 2001)[12] in a hand-out by the Iowa State University, Centre for Excellence in Learning and Teaching:

- Remember – retrieve relevant knowledge from long-term memory
- Understand – construct meaning from instructional messages, including oral, written, and graphic communication
- Apply – carry out or use a procedure in a given situation
- Analyse – break material into constituent parts and determine how parts relate to one another and to an overall structure or purpose
- Evaluate – make judgements based on criteria and standards
- Create – put elements together to form a coherent whole; reorganise into a new pattern or structure

[12] Anderson, L.W. (Ed.), Krathwohl, D.R. (Ed.), Airasian, P.W., Cruikshank, K.A., Maye, R.E., Pintrich, P.R., Raths, J., & Wittrock, M.C. (2001). A t*axonomy for learning, teaching and assessing: A revision of Bloom's Taxonomy of Educational Objectives (Complete edition). New York: Longman.*