COMPUTING
TEACHING AND LEARNING SYLLABUS

Secondary

G3

Syllabus 7155

Year of Implementation:
From 2024 with Secondary Three Cohort

Ministry of Education
SINGAPORE

Updated: April 2024

G3 COMPUTING
TEACHING AND LEARNING SYLLABUS

For implementation in 2024
First year of examination in 2025

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

CONTENTS

SECTION 1: INTRODUCTION

1. INTRODUCTION

Value of Computing

The fourth industrial revolution, driven by the rapid advancement of technology and growing capabilities of Artificial Intelligence (AI), has brought about significant changes in almost every business sector worldwide. To survive in this digital age, companies must undergo digital transformation and Singapore must maintain a robust pipeline of technology professionals. Computing subjects provide upstream support for this effort by providing students with opportunities to explore the field and satisfy their natural curiosity about the technology around them. Students who offer Computing develop Computational Thinking skills that will be relevant in whatever fields or careers they pursue and G3 Computing, with its coverage of text-based coding and basic AI algorithms, is tailored for talented or passionate students who may wish to pursue computing-related studies or careers in the future.

Curriculum Framework

The design of G3 Computing is guided by the Computing Curriculum Framework which was revised in 2017. See Figure 1. It consists of the following:

- Vision statement for computing education
- Dimensions of computing
- Core Concepts of computing
- Components of computational thinking (CT)
- Practices of computing practitioners and professionals

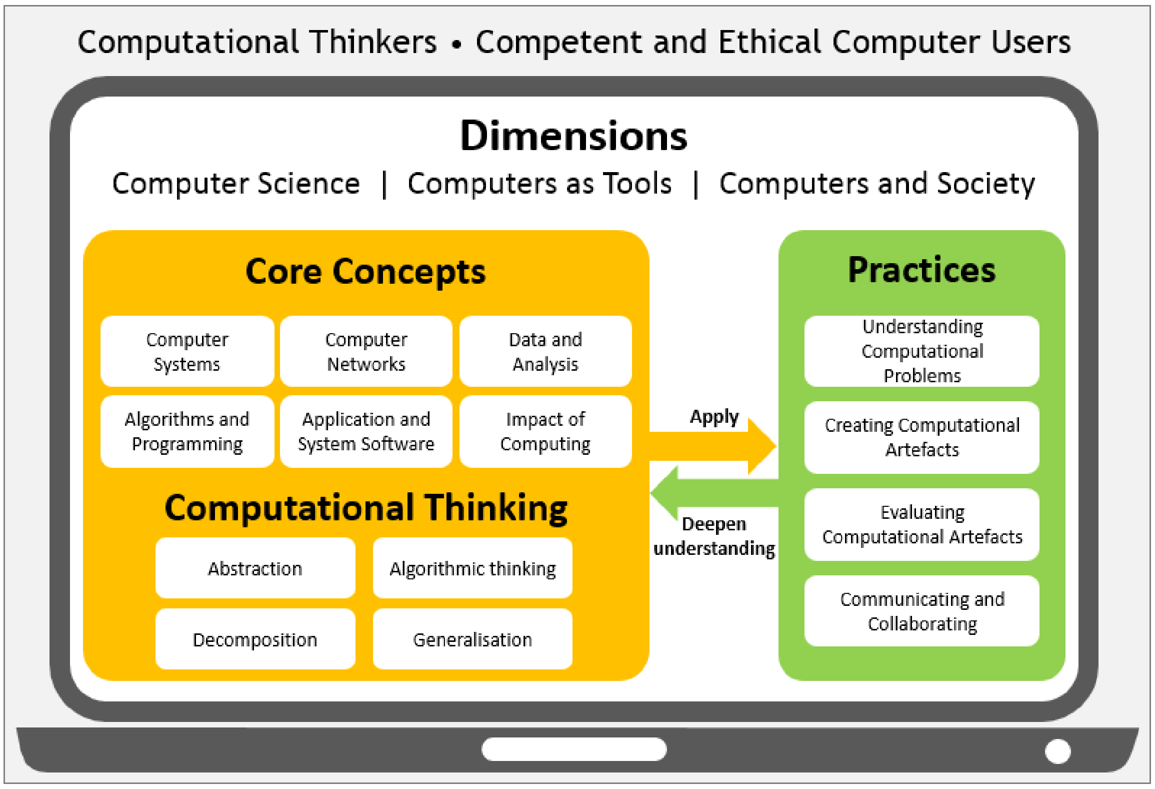

Computational Thinkers • Competent and Ethical Computer Users
Dimensions
Computer Science | Computers as Tools | Computers and Society
Core Concepts
Computer Systems
Computer Networks
Data and Analysis
Algorithms and Programming
Application and System Software
Impact of Computing
Computational Thinking
Abstraction
Algorithmic thinking
Decomposition
Generalisation
Apply
Deepen understanding
Practices
Understanding Computational Problems
Creating Computational Artefacts
Evaluating Computational Artefacts
Communicating and Collaborating

Figure 1: Computing Curriculum Framework (2017)

An important aspect of the framework is the relationship between the Core Concepts, Computation Thinking and Practices: Core Concepts and Computational Thinking are applied through the Practices, and the Practices will in turn deepen one's understanding of the Core Concepts.

Table 1 shows the alignment between G3 Computing topics and the Core Concepts in the Computing Curriculum Framework.

Table 1: Alignment of G3 Computing topics with the Core Concepts in the Framework

Core Concepts	**Topics in G3 Computing**
Computer Systems	Computer architecture, data representation and logic gates
Computer Networks	Types of networks, protocols, error detection, home networks and security
Data and Analysis	Data processing and analysis using spreadsheet software
Algorithms and Programming	Python programming, testing, debugging, algorithm design and software engineering
Application and System Software	Application software (spreadsheet and programming software)
Impact of Computing	Intellectual property, online falsehoods and emerging technologies (e.g., AI)

Table 2 shows the alignment between G3 Computing tasks and the Practices in the Computing Curriculum Framework.

Table 2: Alignment of tasks in G3 Computing with the Practices in the Framework

Practices	**Tasks in Computing**
Understanding Computational Problems	Students understand and identify key information about a complex problem.
Creating Computational Artefacts	Students design and create computational artefacts such as programs and spreadsheets.
Evaluating Computational Artefacts	Students test, evaluate and improve computational artefacts (incl. debugging and refining programs).
Communicating and Collaborating	Students work collaboratively in pairs or small groups to solve problems and describe / document their solutions.

Aims of the Syllabus

The G3 Computing 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. Specifically, the syllabus aims to enable students to:

1) Acquire knowledge and understanding of core areas in computing covering concepts of logic, algorithms, data analysis, data representation and networking;
2) Develop and apply computational thinking skills such as abstraction and decomposition to solve real-world problems by designing, writing, testing and debugging programs using a personal computer;
3) Develop an appreciation of computing as a dynamic and creative field including awareness of recent developments in computer systems;
4) Develop an understanding of the social, ethical, legal and economic implications of computing; and
5) Develop attitudes and 21CC needed to do well in computing such as inventive thinking, perseverance, collaboration, communication as well as striving for accuracy and thoroughness.

21st Century Competencies (21CC)

The Framework for 21st Century Competencies and Student Outcomes ("21CC Framework") in Figure 2 shows how Core Values, Social-Emotional Competencies, and Emerging 21st Century Competencies support the realisation of MOE's Desired Outcomes of Education.

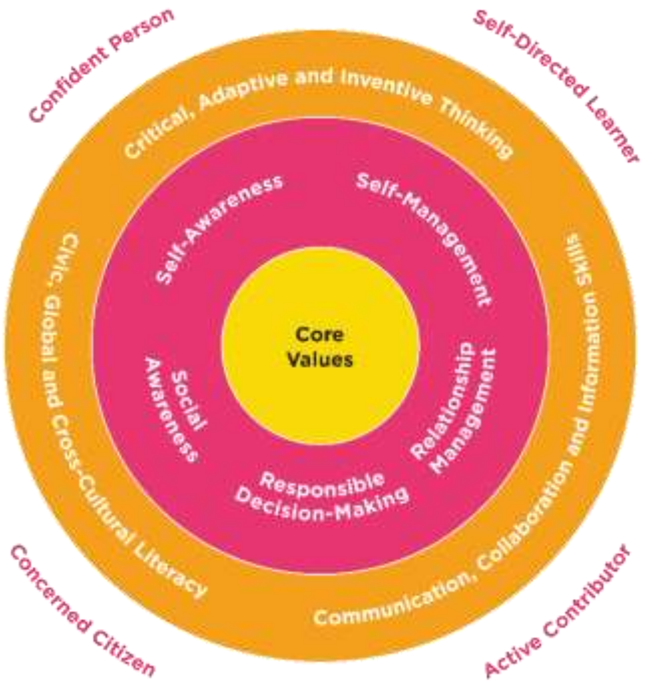

Confident Person
Self-Directed Learner
Concerned Citizen
Active Contributor
Critical, Adaptive and Inventive Thinking
Communication, Collaboration and Information Skills
Civic, Global and Cross-Cultural Literacy
Self-Awareness
Self-Management
Social Awareness
Relationship Management
Responsible Decision-Making
Core Values

Figure 2: Framework for 21CC and Student Outcomes

Table 3 illustrates how the G3 Computing curriculum is aligned with the Learning Goals and Developmental Milestones for Emerging 21CC. Core Values such as responsibility and resilience are fostered through the 'problem-driven' pedagogical approach (see Section 3). Social and Emotional Competencies are developed when students work collaboratively on different tasks.

Table 3: Alignment with Learning Goals and Developmental Milestones

Critical, Adaptive and Inventive Thinking (CAIT)		**G3 Computing Competencies and Attitudes**
Learning Goal	**21CC Developmental Milestone (By end of S4)**	**Understanding Computational Problems** **Creating and Evaluating Computational Artefacts**
CAIT 1: Exercises sound reasoning and decision-making	1.4: The student can use evidence and adopt different viewpoints to explain their reasoning and decisions, having considered the implications of the relationship among different viewpoints.	Ability to: • apply computational thinking and logical reasoning in the design and implementation of computational artefacts.
CAIT 2: Uses metacognition to enhance, monitor and regulate thinking	2.4: The student can plan, organise and evaluate their thinking strategies to monitor their learning. They suspend judgement, reassess conclusions and consider alternatives to refine their thoughts, attitudes, behaviour and actions.	Ability to: • persevere in creating computational artefacts despite challenges (e.g., not giving up when their computer programs do not work).
CAIT 3: Assesses different contexts and situations in order to make connections and draw new insights	3.4: The student can draw on the similarities and differences between different contexts or situations to extract new insights to inform their perspective or approach.	Ability to: • recognise when a new problem is similar to an existing problem that has been encountered before and adapt the corresponding solution to solve the new problem.
CAIT 4: Manages complexities and ambiguities by adjusting one's perspective and strategies	4.4: The student can draw on different perspectives and strategies to adjust their approach when required, adapting learnt knowledge and skills in new and unexpected contexts to solve complex and unexpected problems.	Ability to: • identify the key information about a complex task; and • analyse and break down a complex computational problem into manageable parts.
CAIT 5: Explores possibilities and generates novel and useful ideas with curiosity and openness	5.4: The student can generate ideas that are unique or modified substantially from existing ones and explore different pathways that lead to solutions.	Ability to brainstorm ideas to solve problems and explore different plausible solutions.

CAIT 6: Evaluates and refines ideas to formulate novel and useful situations	6.4: The student can evaluate and refine their ideas iteratively, using relevant strategies and based on a set of criteria that is appropriate for the task or context.	Ability to: • design test cases and evaluate solutions using them; and • debug and refine computer programs.
Communication, Collaboration and Information (CCI)		**G3 Computing Competencies and Attitudes**
Learning Goal	**21CC Developmental Milestone (By end of S4)**	**Communicating and Collaborating**
CCI 1: Effectively communicates information and co-constructs meaning	1.4: The student can convey and critically evaluate knowledge to co-construct new understandings and complex ideas persuasively and with impact, while considering the specific purpose and context of communication.	Ability to: • explain and justify the appropriateness of their computational designs and choices; and • describe the features and operation of their computational artefacts.
CCI 2: Engages empathetically with diverse perspectives	2.4: The student can respond with respect and empathy. The student is sensitive to the diverse backgrounds that influence the context of communication with others.	Ability to: • consider different perspectives when working with others to solve computational problems; and • reach compromises or consensus with stakeholders and/or other developers to pursue a shared goal as part of the software engineering process.
CCI 3: Interacts and works effectively in group settings to contribute to shared goals	3.4: The student can mediate conflict and disagreement, reaching compromises or consensus for collective decisions to meet shared goals.	
CCI 4: Collectively defines and negotiates the roles and tasks determined by the group to achieve its goals	4.4: The student can reflect on their working relationships with the group and adapt to contribute to the shared goals, as determined collectively by its members.	
CCI 5: Employs effective strategies to locate digital and non-digital information and resources, and exercises discernment by evaluating the accuracy, credibility, and	5.4 The student can refine search results, organise information systematically and manage information sensitively, and evaluate the accuracy, credibility and relevance of information	Ability to: • understand the concept of privacy and take measures to enforce the privacy of personal data; and • understand the factors that lead to online falsehoods as well as to identify and defend against them.

relevance of information		
CCI 6: Creates and shares digital and non-digital information ethically and responsibly, and maintains a positive online presence	6.4: The student can contribute to information and perspectives shared in constructive and ethical ways, and manage their online reputation and relationships responsibly.	Ability to: • understand copyright issues and intellectual property rights; and • understand threats to security and defend against them.
Civic, Global and Cross-Cultural Literacy (CGC)		**G3 Computing Competencies and Attitudes**
Learning Goal	**21CC Developmental Milestone (By end of S4)**	**Understanding Computational Problems** **Communicating and Collaborating**
CGC 1: Demonstrates understanding of values, ideals and issues of personal, community and national significance	1.4: The student can explain, analyse, evaluate, and construct new understandings of issues that affect the culture, social and economic development, governance, future and identity of Singapore. The student can manage tensions among multiple perspectives to work towards a common good.	Ability to recognise and state how the use of computers and ICT has impacted society and the way people live and work in Singapore.
CGC 2: Plays active and constructive roles to improve the school, community and nation	2.4: The student can independently identify appropriate and constructive steps to address issues, and initiate, plan and organise programmes with others that contribute to school, community and/or nation. The student can describe the rights and responsibilities of Singapore citizens and evaluate the complementary civic roles played by individuals, groups, organisations and government.	Ability to: • identify ways that computing and technology may address issues in the school, community and/or nation; and • prototype or communicate possible solutions.
CGC 3: Aware of global issues, interconnections, and trends, and forms informed perspectives on them	3.4: The student can actively find out about global issues, analyse them and discern their implications for Singapore and other countries.	Ability to show awareness of how the global issues of social media, AI and emerging technologies impact Singapore and other countries.
CGC 4: Interacts confidently with	4.4: The student can interact respectfully and confidently with	Ability to:

people from Singapore and beyond on different platforms, including digital ones, in response to global issues	people from Singapore and other countries on various platforms to discuss global issues and recommend appropriate actions to address them.	• explain Singapore's approach to enforcing privacy and defending against online falsehoods (e.g., PDPA, POFMA); and • appreciate how Singapore's approach is like or different from the approach taken by other countries and cultures
CGC 5: Aware of and appreciates the cultural background and identity of self and others	5.4: The student can appreciate the value of a diversity of cultural and religious communities' heritage, customs, perspectives, and contributions, and identify commonalities between one's own and other cultures.	
CGC 6 Shows sensitivity and openness in interactions with people from diverse social, cultural and religious communities to promote social cohesion	6.4: The student can demonstrate appropriate skills and behaviour to work together with people from a diverse range of social, cultural and religious backgrounds within and beyond Singapore, and contribute to promoting social cohesion.	

SECTION 2: CONTENT

2. CONTENT

Overview of Content

The syllabus consists of five modules as follows:

Module 1	Computing Fundamentals
Module 2	Algorithms and Programming
Module 3	Spreadsheets
Module 4	Networking
Module 5	Impact of Computing

The learning outcomes are shown in the following pages.

Module 1: Computing Fundamentals

1.1: Computer Architecture

Students should be able to:

1.1.1 Perform calculations using bits, bytes, kilobytes, kibibytes, megabytes, mebibytes, gigabytes, gibibytes, terabytes, tebibytes, petabytes and pebibytes.

1.1.2 Describe the function of key components of a computer system: its processor, main memory and secondary storage.

1.1.3 Describe the function of data and address buses in reading from and writing to memory.

1.1.4 Describe different input/output interfaces (USB, HDMI and PCI Express) in terms of typical applications, connectors and speed.

1.1.5 Describe the use of magnetic, optical and solid-state media for secondary storage in terms of durability, portability, typical capacities, cost and speed.

1.2: Data Representation

Students should be able to:

1.2.1 Represent positive whole numbers in binary form.

1.2.2 Convert positive whole numbers from one number system to another - binary, denary and hexadecimal; and describe the technique used.

1.2.3 Use two's complement for a fixed number of bits to represent both positive and negative whole numbers in binary.

1.2.4 Use the example of an 8-bit extended ASCII encoding for English text to explain how information can be represented as bits for storage or processing by a computer.

1.3: Logic Gates

Students should be able to:

1.3.1 Represent logic circuits using either logic circuit diagrams or Boolean statements and convert between the two representations.

1.3.2 Construct the truth table for a given logic circuit (maximum 3 inputs) and vice versa.

1.3.3 Draw symbols and construct truth tables for AND, OR, NOT, NAND, NOR and XOR logic gates.

1.3.4 Manipulate Boolean statements using the associative and distributive properties of certain logical operators and De Morgan's theorem.

1.3.5 Solve system problems using combinations of logic gates (maximum 3 inputs).

Module 2: Algorithms and Programming

2.1: Problem Analysis

Students should be able to:

2.1.1 For a given problem, identify and remove unnecessary details to specify the:
- inputs and the requirements for valid inputs
- outputs and the requirements for correct outputs.

2.2: Constructs

Students should be able to:

2.2.1 Interpret flowcharts to understand the sequence, selection and iteration constructs.

2.3: Python Code

Students should be able to:

2.3.1 Use variables to store and retrieve values.

2.3.2 Use literals to represent values directly in code without using a variable.

2.3.3 Use the built-in functions: input() and print(), to perform interactive input/output using the keyboard and screen.

2.3.4 Use the open() built-in function as well as the read(), readline(), write() and close() methods to perform non-interactive file input/output.

2.3.5 Use the import command to load and make additional variables and functions available for use.

2.3.6 Use Boolean values with the operators: or, and, not.

2.3.7 Use integer and floating-point values with appropriate operators and built-in functions (limited to those mentioned in the Quick Reference Guide) to perform:

- Addition, subtraction, multiplication, division, modulo and exponentiation
- Rounding (normal, up, down, towards zero)
- Calculation of square roots
- Generation of ranges
- Generation of random integers / floats
- Conversion to and from strings

2.3.8 Use string values with appropriate operators, built-in functions and methods (limited to those mentioned in the Quick Reference Guide) to perform:

- Concatenation and repetition
- Extraction of characters and substrings (i.e., indexing and slicing)
- Conversion to upper and/or lower case
- Conversion of single characters to and from ASCII
- Testing of whether characters are letters only, lower-case letters only, upper-case letters only, digits only, spaces only and/or alphanumeric
- Testing of whether the string contains a substring
- Testing of whether the string starts with and/or ends with a substring
- Searching for the location of a substring
- Splitting of string into list of substrings based on either whitespace or a given delimiter
- Calculation of length
- Output formatting

2.3.9 Use list values with appropriate operators, built-in functions and methods (limited to those mentioned in the Quick Reference Guide for Python) to perform:

- Concatenation and repetition
- Extraction of single items and subset of items (i.e., indexing and slicing)
- Testing of whether an item is in the list
- Calculation of length
- Calculation of sum, minimum value and maximum value (provided the list items are all integer or floating-point values)

2.3.10 Use dictionary values with appropriate operators to perform dictionary insertion, query, lookup and deletion.

2.3.11 Use the if, elif and else keywords to implement selection constructs.

2.3.12 Use the for and while keywords to implement iteration constructs.

2.3.13 Write and call user-defined functions that may accept parameters and/or provide a return value.

2.3.14 Distinguish between the purpose and use of local and global variables in a program that makes use of functions.

2.4: Testing and Debugging

Students should be able to:

2.4.1 Produce a trace table by performing a manual dry run and inspecting the value of variables at each step of a program.

2.4.2 Inspect the value of variables at selected steps of a program by inserting print statements.

2.4.3 Locate logic errors by backtracking from a point where unexpected behaviour is observed.

2.4.4 Test programs incrementally as small additions and changes are made during development.

2.4.5 Test small parts of a program by commenting out other parts of the program that are not needed for testing.

2.4.6 Justify the use of data validation and identify the appropriate action to take when invalid data is encountered: asking for input again (for interactive input) or exiting the program (for non-interactive input).

2.4.7 Validate input data for acceptance by performing:

- length check,
- range check,
- presence check,
- format check,
- existence check (i.e., checking for whether input data is already in the system), and/or
- calculation of a check digit

2.4.8 Understand and describe types of program errors: syntax, logic and run-time; and explain why they occur.

2.4.9 Design appropriate test cases to cover normal, error and boundary conditions and specify which type(s) of conditions is/are being tested for each test case.

2.5: Algorithm Design

Students should be able to:

2.5.1 Explain and use the algorithms for:

- Obtaining the minimum or maximum value(s) in a list without using the min() or max() functions
- Calculating the sum or average of values in a list without using the sum() function
- Searching for the location(s) of an item in a list or a character in a string without using the index() or find() methods
- Extracting items from a list or characters from a string based on a given criteria
- Splitting a string into a list based on a given delimiter without using the split() method

2.5.2 Solve problems by breaking them down into smaller and more manageable parts (modular approach).

2.5.3 Solve problems by solving a small version of the problem and gradually extending the solution to bigger versions of the problem (incremental approach).

2.5.4 Use the technique of solving many small instances of a problem manually to identify the generic steps that are needed to solve the problem in general.

2.5.5 Recognise when a new problem is similar to an existing problem that has been encountered before and adapt the corresponding solution to solve the new problem.

2.6: Software Engineering

Students should be able to:

2.6.1 Understand and describe the stages in developing a program: gather requirements, design solutions, write code, test and refine code, deploy code.

2.6.2 Recognise that the sequence of software development stages may not be linear and the use of iterative development may sometimes be more appropriate.

Module 3: Spreadsheets

3.1: Program Features

Students should be able to:

3.1.1 Use appropriate relative, absolute and mixed cell references in formulas so they give the correct results when copied to similar cells in a table.

3.1.2 Use the Goal Seek feature to determine the value needed in a cell for another cell to reach a specified target value.

3.1.3 Use the Conditional Formatting feature to automatically update cell formatting based on one or more rules.

3.2: Functions

Students should be able to:

3.2.1 Use logical functions to perform:
- Logical OR, AND or NOT
- Selection between two values based on a third logical value

3.2.2 Use mathematical and statistical operators and functions to perform:
- Addition, subtraction, multiplication, division, modulo or exponentiation
- Rounding (normal, up, down)
- Calculation of square roots
- Calculation of sums (normal, with condition)
- Calculation of average (normal, with condition)
- Calculation of median, mode, minimum value or maximum value
- Calculation of a value’s rank (ascending, descending)
- Calculation of n-th largest or smallest value
- Counting of values (numbers only, blank only, non-blank only, with condition)

- Generation of random numbers

3.2.3 Use text functions to perform:
- Extraction of characters from the left end, middle or right end of text
- Calculation of text length
- Concatenation of texts
- Calculation of the first position of one text within another text (case sensitive, case insensitive)

3.2.4 Use lookup functions to perform:
- Lookup of values from an unsorted vertical or horizontal table using exact matching
- Lookup of values from a sorted vertical or horizontal table using approximate matching
- Classification of values based on range using approximate matching and a secondary table
- Lookup of values at the intersection of a particular row and column of a cell range
- Calculation of the relative position of a value in a cell range

3.2.5 Use date functions to perform:
- Determination of the current date or the current date and time
- Calculation of the number of days between two dates

The examinable spreadsheet operators and functions are as follows.

Operator	**Meaning**
+	Addition
−	Subtraction or Negation
*	Multiplication
/	Division
%	Percent
^	Exponentiation
=	Equal to
>	More than
>=	More than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
&	Concatenates two values to produce one continuous text value

Category	**Function(s)**
Date and Time	DAYS, NOW, TODAY
Text	CONCAT, FIND, LEFT, LEN, MID, RIGHT, SEARCH
Logical	AND, IF, NOT, OR
Lookup	HLOOKUP, INDEX, MATCH, VLOOKUP
Mathematical	CEILING.MATH, FLOOR.MATH, MOD, POWER, QUOTIENT, RAND, RANDBETWEEN, ROUND, SQRT, SUM, SUMIF
Statistical	AVERAGE, AVERAGEIF, COUNT, COUNTA, COUNTBLANK, COUNTIF, LARGE, MAX, MEDIAN, MIN, MODE.SNGL, RANK.EQ, SMALL

Module 4: Networking

4.1: Concepts

Students should be able to:

4.1.1 Define computer networks as systems of two or more computers connected by a transmission medium for the exchange of data.

4.1.2 Describe the difference between wired and wireless transmission media and explain the factors that will determine the use of each medium.

4.1.3 Differentiate between Local Area Networks (LANs) and Wide Area Networks (WANs) based on their geographical scope.

4.1.4 Compare and contrast the client-server and peer-to-peer network architectures in terms of purpose, organisation and bandwidth.

4.1.5 Identify and state common applications of star and mesh topologies in a home network.

4.1.6 Define protocols as standards and rules that govern communication over a network.

4.1.7 Explain that LANs typically use protocols where data is transmitted as individual packets.

4.1.8 Explain the use of parity, checksums, echo checks and automatic repeat requests for detecting errors in packet transmission.

Home Networks and the Internet

Students should be able to:

4.1.9 Explain that home networks are examples of LANs and that the internet is an example of a WAN that is formed by connecting many different LANs from around the world together.

4.1.10 Explain that modems are used to provide internet access by converting from the protocols used by an Internet Service Provider (ISP) to the protocols used by LANs.

4.1.11 Explain that computers use network interface controllers to communicate via different transmission media.

4.1.12 Explain that Media Access Control (MAC) addresses are used by network interface controllers, network switches and wireless access points to direct data within the same LAN while Internet Protocol Version 4 (IPv4) and Internet Protocol Version 6 (IPv6) addresses are used by routers to direct data across different LANs.

4.1.13 Compare and contrast MAC, IPv4 and IPv6 addresses in terms of purpose, bit length, degree of permanence and typical representation formats.

4.1.14 Connect a router, a switch and a wireless access point to a modem correctly such that multiple computers form a LAN that can access the internet via wired and wireless transmission media.

4.2: Security and Privacy

Students should be able to:

4.2.1 Compare and contrast security and privacy in terms of what kind of data is being protected, what the data is being protected from and how that protection is enforced.

4.2.2 Explain how human actions threaten security and privacy by causing data corruption (through physical or non-physical means) or exposure of private data.

4.2.3 Explain how anti-malware programs enforce security and privacy by preventing malware from running and removing malware that may be present on a computer.

4.2.4 Explain how firewalls enforce security and privacy by using either hardware or software to monitor packets and decide which packets should be permitted or blocked based on a set of configurable rules.

4.2.5 Explain how encryption enforces security and privacy by making encrypted data appear meaningless without the corresponding secret key.

4.2.6 Explain how the Personal Data Protection Act (PDPA) enforces privacy by legally requiring that organisations do the following when collecting personal data:

- seek consent from the individual;
- disclose the purpose for collecting data when seeking consent; and
- retain the data for only as long as necessary to fulfil the stated purpose

4.2.7 Explain how adware threatens security and privacy by installing itself without the user's knowledge and displaying unwanted advertisements.

4.2.8 Explain how spyware threatens security and privacy by secretly collecting personal information and transmitting this information to attackers without the user's knowledge.

4.2.9 Explain how cookies are typically not malicious but can threaten privacy by tracking a user's browsing history across multiple web sites.

4.2.10 Explain how phishing threatens security and privacy by using emails and fake websites that appear to be from reputable companies to steal personal information.

4.2.11 Explain how pharming threatens security and privacy by intercepting requests to legitimate websites and redirecting them to fake websites while still appearing to use the same address as the legitimate website.

4.2.12 Describe good computing practices that can mitigate the threats posed by adware, spyware, cookies, phishing, pharming and human actions.

4.2.13 Analyse the effects of anti-malware programs, firewalls, encryption and the PDPA against the threats posed by adware, spyware, cookies, phishing, pharming and human actions.

Module 5: Impact of Computing

5.1: General

Students should be able to:

5.1.1 Give examples of the impact of computers in the following industries:
- Communication: ability to connect people and businesses over long distances
- Education: easy access to online classes and large amounts of information via the internet
- Transportation: widespread access to navigational services via Global Positioning System (GPS) and emergence of self-driving vehicles
- Retail: more reliable tracking of available stock and emergence of self-checkout counters

5.2: Intellectual Property

Students should be able to:

5.2.1 Define intellectual property as creations of the mind that have value but can exist purely as data.

5.2.2 Describe copyright as the legal right of owners to control the use and distribution of their intellectual property under the Copyright Act.

5.2.3 Explain that copyright owners can grant a license to authorise or forbid the use and distribution of their intellectual property under certain conditions.

5.2.4 Identify computer programs as an example of intellectual property and distinguish between proprietary software, freeware, shareware as well as free and open-source software (FOSS) based on their licenses.

5.2.5 Recognise software piracy as the illegal use and distribution of copyrighted computer programs in a manner that is forbidden by their license.

5.3: Communication

Students should be able to:

5.3.1 Explain why the promotion of social media posts based on engagement rate helps to deliver relevant content to users but can also lead to the proliferation of falsehoods.

5.3.2 Explain how the Protection from Online Falsehoods and Manipulation Act (POFMA) enables the government to tackle the spread of fake news by:

- establishing fines and/or prison terms for engaging in prohibited activities;
- optionally requiring offenders to put up a correction notice or to take down the falsehood; and
- identifying sites that repeatedly spread falsehoods.

<u>5.4: Emerging Technologies</u>

Students should be able to:

5.4.1 Describe Artificial Intelligence (AI) as the ability of a computer to perform complex tasks without constant human guidance and improve its performance as more data is collected.

5.4.2 Give examples of common personal and business tasks that can be performed well by AI: face recognition, voice recognition, image classification and spam filtering.

5.4.3 Define Machine Learning (ML) as a technique used in AI and explain the difference between ML and traditional programming.

5.4.4 Use the nearest neighbour method for a classification task with two quantitative features to demonstrate the basic principles behind ML.

5.4.5 Explain how unethical use of AI or using AI with biased data can lead to negative consequences.

5.4.6 Show an awareness of emerging technologies.[1]

[1] This is a non-examinable Learning Outcome.

Curriculum Time

The total curriculum time for G3 Computing is 46 weeks with 3 hours per week (138 hours) over 2 years. A summary of the estimated number of hours for the topics or content explications per module is provided in Table 4.

Table 4: Curriculum Time per Module

Module	Topics	Hours
1 Computing Fundamentals	1.1 Computer Architecture	3.5
	1.2 Data Representation	4
	1.3 Logic Gates	7
2 Algorithms and Programming	2.1 Problem Analysis	2
	2.2 Constructs	2
	2.3 Python code	32.5
	2.4 Testing and Debugging	11
	2.5 Algorithm Design	8
	2.6 Software Engineering	2
3 Spreadsheets		12
4 Networking	4.1 Concepts	12
	4.2 Home Networks and the Internet	3.5
	4.3 Security and Privacy	5
5 Impact of Computing	5.1 General	3
	5.2 Intellectual Property	2
	5.3 Communication	1.5
	5.4 Emerging Technologies	9
6 Programming Project		18
	Total	138

SECTION 3: PEDAGOGY

Pedagogical Approaches
Performance Tasks
Software

3. PEDAGOGY

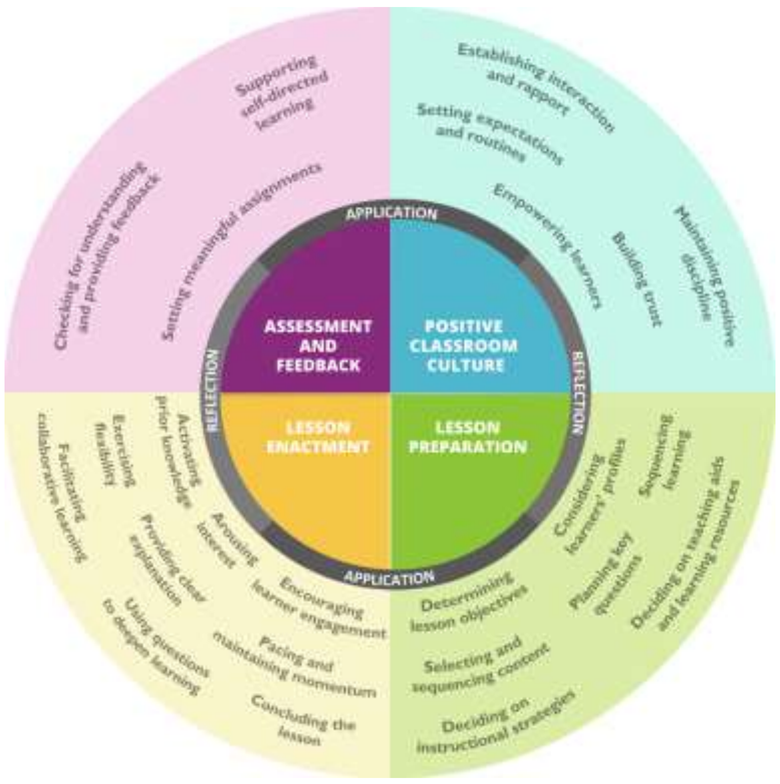
Pedagogical Approaches

The central pedagogical approaches adopted for G3 Computing are the ‘learning through doing’ and ‘problem-driven’ approaches. See Table 5 for the key features.

Table 5: Key features of ‘learning through doing’ and ‘problem-driven’ approaches

Learning through Doing	Problem-driven
Students design and create computational artefacts.	Students work on problems which are based on authentic contexts.
Students work collaboratively to design and generate solutions to tasks/problems.	Students understand and identify key information from the description of a computation problem.
Students examine computer programs (i.e. lines of codes) to identify bugs and correct them.	Students solve problems systematically by using decomposition and generalisation.

Table 6 provides examples of teaching actions/considerations for Computing and how they are aligned to the Singapore Teaching Practice (STP)[2].



Supporting self-directed learning
Checking for understanding and providing feedback
Setting meaningful assignments
Establishing interaction and rapport
Setting expectations and routines
Empowering learners
Building trust
Maintaining positive discipline
APPLICATION
REFLECTION
ASSESSMENT AND FEEDBACK
POSITIVE CLASSROOM CULTURE
LESSON ENACTMENT
LESSON PREPARATION
REFLECTION
APPLICATION
Activating prior knowledge
Exercising flexibility
Facilitating collaborative learning
Arousing interest
Providing clear explanation
Using questions to deepen learning
Encouraging learner engagement
Pacing and maintaining momentum
Concluding the lesson
Determining lesson objectives
Selecting and sequencing content
Deciding on instructional strategies
Considering learners' profiles
Planning key questions
Sequencing learning
Deciding on teaching aids and learning resources

Figure 2: Singapore Teaching Practice

[2] The STP covers different aspects of teaching which teachers can adopt for the teaching of G3 Computing. Visit https://go.gov.sg/stp for more information.

Table 6: Teaching Actions/Considerations applicable to the teaching of G3 Computing

Teaching Process: Lesson Preparation		
Teaching Area	**Teaching Actions/Considerations**	**Description/Examples**
Selecting and sequencing content	Put code reading before code writing	Reading helps students become familiar with basics of code and how it is structured before they attempt to write their own code. This helps to avoid common mistakes and develop better coding habits.
Sequencing learning	Give beginners a template	Beginners are often intimidated if they must start from scratch. Providing beginners with a template or using generative AI as a starting point can facilitate success and prepare students for the next phase.
Deciding on teaching aids and learning resources	Use unplugged or kinaesthetic activities	Unplugged or kinaesthetic activities can help students understand abstract concepts by making them more concrete and tangible. These activities can also be more engaging and fun, which can help students stay motivated and interested in the subject.
Deciding on instructional strategies	Practice live coding	Live coding can help students see how a more experienced programmer thinks and works through problems in real-time. This can also help students learn how to debug code and develop better problem-solving skills.
	Use pair programming	Pair programming can help students learn from each other and develop better communication and collaboration skills. It can also help students learn how to give and receive feedback.

Teaching Process: Lesson Enactment		
Teaching Area	**Teaching Actions/Considerations**	**Description/Examples**
Providing Clear Explanation	**Demonstration** Teachers demonstrate a 'walk through' of a new skill during which students learn by observing.	Teachers practice 'live coding' by explaining each step of code as it is written: • Students can see how teachers solve a problem. • Students get to see teachers make mistakes and see how they debug. • It slows teachers down thus giving time for students to process new knowledge.
	Model Thinking Aloud Teachers make thinking visible by verbalising and making explicit their thinking so that students can follow the teachers' thought processes.	• Teachers think out loud and verbalise their thought processes (e.g. "What are the inputs to this problem?", "What should happen if the string is empty?"). • Teachers step through a program line-by-line to explain how each line of code is executed (i.e., code tracing).
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.
Encouraging Learner Engagement	**Explore, Engage, Apply** Teachers design learning activities that are meaningful and relevant to students.	Use of kinaesthetic and unplugged activities to explain computer or programming concepts, e.g., Students can act out the actions indicated by the loops and instructions in a program.
	Engagement through Collaboration and Interactivity	In pair programming, the driver writes the program

	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. Students are to switch roles from time to time.	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.	Students consider possible solutions to a theory question (e.g., what are some negative consequences of using AI with biased data) and discuss answers with a partner followed by consolidation by teacher.
Instructional Strategies for Teaching Programming	**Predict and Compare** Students predict the outcome of a program/process and compare with the actual outcome.	Students predict the output of a piece of code before running it to assess their own understanding.
	Loksa's Six-Step Approach Students use Loksa's approach to solve programming problems.	Students use a framework to reinterpret a problem, then draw upon their experience to adapt a potential solution and evaluate its effectiveness.
	Making test cases Students create their own test cases (i.e., test data and expected outputs).	Students verify their understanding of the problem requirements using test cases before they start writing code.
	Incremental testing Students run and test their code incrementally during development.	Students run their code iteratively after each small part of the program is written to catch errors early instead of trying to debug a large amount of untested code at once.
	Debugging programs Teachers provide buggy programs for students to find and correct the errors.	• Students identify and correct the error(s) in a buggy program, e.g., the score in a game is not updated properly. • Students can use the \`Rubber Duck\` method of explaining their code line by line to themselves to identify the cause of error.

Table 6: Teaching Actions applicable to teaching of G3 Computing

Teaching Areas	**Teaching Actions**	**Examples of how it can be used in the classroom**
Providing Clear Explanation	**Demonstration** Teachers demonstrate a 'walk through' of a new skill during which students learn by observing.	• Teachers demonstrate a new skill (e.g., how to perform mail merge) to students.
	Model Thinking Aloud Teachers make thinking visible by verbalising and making explicit their thinking so that students can follow the teachers' thought processes.	• Teachers think out loud and verbalise their thought processes (e.g., "Which basic shapes are needed to create this drawing?", "Which backdrop should appear if the player wins the game?"). • Teachers step through a program line-by-line to explain how each line of code is executed (i.e., code tracing).
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.
Encouraging Learner Engagement	**Explore, Engage, Apply** Teachers design learning activities that are meaningful and relevant to students.	• Use of kinaesthetic and unplugged activities to explain computer or programming concepts. • Students can act out the actions indicated in by a program script. This is especially useful for scripts with motion blocks.
	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. Students are to switch roles from time to time.	• In pair drawing using a graphics software, the driver creates the drawing while the navigator provides feedback (e.g., accuracy, proportionality of the drawing objects). • In pair programming, 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	• Students consider possible solutions to a theory question (e.g., how to take proper care of computers) and discuss

Teaching Areas	Teaching Actions	Examples of how it can be used in the classroom
	followed by consolidation led by teacher with the whole class.	answers with a partner followed by consolidation by teacher.
Instructional Strategies for Teaching Programming	**Predict and Compare** Students predict the outcome of a program/process, and compare with the actual outcome.	• Students predict behaviour of a sprite based on given scripts.
	Debugging programs Teachers provide buggy scripts for students to find and correct the errors.	• Students identify and correct the error(s) in a buggy program, e.g., the score in a game is not updated properly.

Performance Tasks

To provide meaningful assignments to reinforce and extend students' learning, teachers could use the following:

(a) Quick Checks are provided at the end of each section of the textbook to reinforce concepts through short-answer questions. Teachers could use these to quickly check for students' understanding of that section.

(b) Worksheets are used to complement teaching. These could be used as classroom activities where teachers can do together with their students.

(c) Review Questions are provided at the end of the chapter of the textbook to reinforce and consolidate the concepts learnt in the chapter through structured questions. Teachers could assign these after the teaching of a chapter has been completed.

(d) Student Learning Space leverage on the affordances of ICT to enhance students' learning through simulations, games and videos.

(e) Each Mini Task consists of a few guided programming tasks with a real-world context. Teachers could assign these tasks after the teaching of Algorithms and Programming has been completed.

(f) Mini Projects are open-ended tasks carried out at the end of Secondary 3 over 6 weeks. The project will be based on authentic tasks founded on real-world problems. There are 2 proposed implementations for the Mini Project – hardware-based or software-based. In the hardware project, students program microcontrollers to perform real-world tasks such as monitoring temperature within an area. In the software project, students may program a game to move a sprite around to collect tokens. See Table 7.

Table 7: Possible Mini Projects

Software-based	
Pygame module	Creation of simple games
Turtle module	Creation of graphics and simple games
Numpy and Matplotlib modules	Analysis and visualisation of data
Hardware-based	
Raspberry Pi	Creation of simple maker projects (e.g., alarm clock)
	Creation of simple games using the Pygame module and Sense HAT (e.g., controlling game using rotation sensor of Sense HAT)
Micro:bit	Creation of simple maker projects (e.g., counter, dice, wireless messaging, pendulum oscillation counter)
	Creation of simple games (e.g., catch game)

Software

The required software to be used in G3 Computing are listed in Table 8.

Table 8: Software requirements for G3 Computing

Module	**Software**
Algorithms and Programming	• JupyterLab Desktop • Mu (for T&L only)
Spreadsheets	• Microsoft Excel
Networking	• Filius (for T&L only)

SECTION 4: ASSESSMENT

Assessment for Computing
School-based Assessment
National Examination

4. ASSESSMENT

Assessment for Computing

Assessment is integral to the learning process and helps students become self-directed learners. In this way, assessment is aligned to pedagogical approaches (as outlined in the previous section), curricular objectives and content. Both school-based assessment and national examinations play important and different roles in our education system.

Assessment is an important part of teaching and learning, and it is an ongoing process by which teachers gather information about students' learning to inform and support future teaching. Assessment can be categorised into the following types:

- Formative Assessment, which can be incorporated into skill-building tasks, problem sets and course projects, can be used to determine how students are progressing through certain learning outcomes during a series of learning activities. Formative assessment can be used to identify learning gaps and provide timely feedback to students on their learning as well as inform teachers on planning for future instruction. Teachers should also create opportunities for students to show that the feedback has enabled them to close their learning gaps.
- Summative Assessment, such as class tests, school and national examinations, are used at the end of a series of learning activities to determine the level of students' attainment of the desired learning outcomes. It is commonly used for placement and grading.

A balanced assessment system should have both formative and summative assessment.

School-based Assessment

School-based assessments provide opportunities for teachers to obtain information about students' level of competency and provide targeted feedback to their students. The adopted pedagogies and resources developed by CPDD provide such opportunities by getting the students to perform tasks that demonstrate their knowledge and skills.

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.

School-based summative assessment should consist of both written and practical components. The written paper may comprise multiple-choice and short-structured questions of variable marks. The practical paper may consist of a spreadsheet task and multiple programming tasks.

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 candidates'

AO1 Knowledge and understanding of core computing concepts, algorithms, techniques, tools and related ethics

AO2 Application of knowledge and understanding to analyse and solve computing problems

AO3 Design, development, testing and refinement of computing solutions

Students will handle and process data in computer systems and demonstrate their understanding on ethical issues when dealing with data. They will demonstrate problem-solving techniques through analysing and writing programming solutions for a range of computing problems in a variety of contexts. Students will also demonstrate computational thinking through the design and development of computing solutions.

Scheme of Assessment

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

Paper 1 (Written examination, 2 hours)
This paper will assess candidates' knowledge, understanding and application of concepts and skills in all the six modules. The questions consist of a mixture of:

- Multiple choice questions (single- and multiple-answer)
- Short-answer questions
- Matching questions
- Cloze passages
- Structured questions

Relevant formulae will be provided for candidates.

The paper carries 60% 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 with access to a spreadsheet, Python and JupyterLab software, will assess topics from the following modules:

- Spreadsheets
- Algorithms and Programming

A quick reference guide for Python will be provided for candidates.

Candidate will submit softcopies of the required work for marking. The allotted time includes time for saving the required work in the candidates' computer. This paper carries 40% of the total marks and is marked out of 70 marks.

Summary of details for each paper:

Paper	Mode	Duration	Weighting	Marks	Format	Modules Assessed
1	Written	2 h	60%	80	A mixture of • Multiple choice questions (single- and multiple-answer) • Short-answer questions • Matching questions • Cloze passages • Structured questions	All the five modules
2	Lab-based	2 h 30 m	40%	70	• One question on Spreadsheets • Four to five questions on Programming	Module 2: Algorithms and Programming Module 3: Spreadsheets

Specification Table

Assessment Objectives	Paper 1	Paper 2	Overall
AO1 Knowledge and Understanding	~30%	0%	**30%**
AO2 Application	~20%	~20%	**40%**
AO3 Development, testing and refinement	~10%	~20%	**30%**
TOTAL	**60%**	**40%**	**100%**

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 one shift on the day of the examination. Each candidate should have the sole use of a personal computer for the purpose of the examination. The candidates should be able to access spreadsheet and programming software (Python and JupyterLab).

THIS PAGE IS
INTENTIONALLY
LEFT BLANK