The New Senior Secondary Curriculum for Sierra Leone

Subject Syllabus for Applications of Mathematics for Coding
Subject stream: Mathematics and Numeracy

This subject syllabus is based on the National Curriculum Framework for Senior Secondary Education. It was prepared by national curriculum specialists and subject experts.

Curriculum elements for Applications of Mathematics (Coding) – an applied subject

Subject Description

The Applications of Mathematics (Coding) subject aims to provide students with the opportunity to combine the study of aspects of mathematics with computer programming. It is designed to offer students with practical experience in using computer programming techniques and skills to solve problems that can be set up as mathematical models.

Rationale for the Inclusion of Applications of Computer Mathematics (Coding) in the Senior Secondary School Curriculum

Applications of Computer Mathematics (Coding) is an innovative subject at the forefront of emerging trends in technology such as Artificial Intelligence (AI), the Internet of Things (IoT), and Big Data and Blockchains. These trends - many of which are yet less known about, will make a huge impact on digital societies in general and digital economies in particular. To keep up with these trends, the curriculum of Sierra Leone needs to be positioned amongst the best in the world to ensure that Sierra Leonean youth can develop the skills to participate and succeed in the 21st century.
With this course providing strong foundations through problem solving in building computational thinking such as decomposition, abstraction and algorithm design, it takes its place in the senior secondary school curriculum as a strong contributor both to personal as well as national progress and development. Students will acquire critical thinking skills which will help them to function in a technological world, and easily adapt to any challenges they face.

Applications of Computer Mathematics (Coding) is taught through a combination of mathematical rigour and experiential learning. It offers many advantages to students who participate and successfully complete the course, and therefore deserves its place in the senior secondary school curriculum.

General Learning Outcomes

At the end of the course, students will be able to:

- identify the characteristics and functions of a computer
- identify the components and functions of a computer system
- outline the safe use of computers and recognize its role in society
- develop confidence in using the Windows Operating System
- competently use the internet for research
- use a specified programming language and programming environment
- given a problem, solve it using the Problem-Solving method
- understand decomposition and abstraction as applied to programming
- define a problem and understand what is required to solve it
- develop an algorithm and represent it using pseudocode and a flowchart
- develop code by using the programming concepts of the specified programming language

- write programs using control structures, e.g. iteration, to control program flow
- write programs using data structures, such as arrays, to store data in an organised and efficient manner
- write programs using subroutines (procedures/functions) to improve the organisation and readability of a program
- use searching and sorting techniques in solve problems
- test and debug programs by identifying the cause of errors and exceptions, and correcting them
- demonstrate an understanding of how computer programs are implemented and maintained
- demonstrate an understanding of the skills acquired by completing an extended project

Subject Content Outline by Broad Themes & Specific Topics

SSS 1	SSS 2	SSS 3
Computer systems • Characteristics and Functions of Computers • Components of a Computer System • Safe Use of Computers • Role of Computers in Society **Essential computer skills** • Introduction to the Windows Operating System • Introduction to Computer Applications Software • Basic Keyboard and Mouse Skills • Introduction to Using the Internet for Research **Introduction to programming** • Computer Programs • Programming Languages and the Programming Environment • Syntax and Syntax errors • Hello World! • **Problem solving and programs** • Problem Solving Method • Algorithms	**Programming concepts II** • Data Types • Operators • Order of Operations • Strings • Control Structures • File Handling **Data structures and arrays** • Data Structure Concepts • Static and Dynamic Data Structures • Arrays **Programming mini-projects: Using the problem-solving method** • Defining • Designing • Coding • Testing and Debugging **Subroutines: procedures and functions** • Subroutine Concepts • Difference Between Procedures and Functions	**Programming the extended project: using the problem-solving method** • Defining • Designing • Coding • Testing and Debugging • Implementing

• Control Structures • Decomposition • Abstraction **Programming concepts I** • Keywords • Variables and Constants • Identifiers • Assignment • Comments • Data Types • Operators • Order of Operations • Strings • Data Structures • Exception Handling • Input/Output **Programming mini-projects: Using the problem-solving method** • Defining • Designing • Coding **Flow of program: control structures** • Introduction to Program Flow • Sequence • Selection • Iteration **Programming mini-projects: Using the problem-Solving method** • Defining • Designing • Coding • Testing and debugging	• Advantages of Using Subroutines • Using Parameters in Subroutine • Subroutine Operations • Built-in and User-Defined Subroutines **Programming mini-projects: Using the problem-solving method** • Defining • Designing • Coding • Testing and Debugging **Searching** • Searching Concepts • Linear Search • Binary Search • Compare Searching Algorithms **Sorting** • Sorting Concepts • Bubble Sort • Selection Sort • Insertion Sort • Compare Sorting Algorithms **Programming mini-projects: Using the problem-solving method** • Defining • Designing • Coding • Testing and De-bugging	

Structure of the Syllabus Over the Three Year Senior Secondary School Cycle

	SSS 1	SSS 2	SSS 3
Term 1	**Computer systems** • Characteristics and Functions of computers • Components of a computer system • Safe Use of Computers • Role of Computers in Society **Essential computer skills** • Introduction to the Windows operating system • Introduction to computer applications software • Basic keyboard and mouse skills • Introduction to using the internet for research **Introduction to programming** • Computer programmes • Programming languages and the programming environment • Syntax and syntax errors • Hello World!	**Programming concepts II** • Data Types • Operators • Order of Operations • Strings • Control Structures • File Handling **Data structures and arrays** • Data Structure Concepts • Static and Dynamic Data Structures • Arrays **Programming mini-projects: Using the problem-solving method** • Defining • Designing • Coding • Testing and debugging	**Programming the extended project: Using the problem-solving method** • Defining • Designing • Coding
Term 2	**Problem solving and programs** • Problem solving method • Algorithms • Control structures • Decomposition • Abstraction **Programming concepts I** • Keywords • Variables and constants	**Subroutines: procedures and functions** • Subroutine Concepts • Difference Between Procedures and Functions • Advantages of Using Subroutines • Using Parameters in Subroutine • Subroutine Operations • Built-in and User-Defined Subroutines **Programming mini-projects:**	**Programming the extended project: Using the problem-solving method** • Testing and debugging • Implementing

	• Identifiers • Assignment • Comments • Data Types • Operators • Order of operations • Strings • Data structures • Exception handling • Input/Output **Programming mini-projects: Using the problem-solving method** • Defining • Designing • Coding	**Using the problem-solving method** • Defining • Designing • Coding • Testing and debugging	
Term 3	**Flow of program: control structures** • Introduction to Program Flow • Sequence • Selection • Iteration **Programming mini-projects: Using the problem-solving method** • Defining • Designing • Coding • Testing and debugging	**Searching** • Searching Concepts • Linear Search • Binary Search • Compare Searching Algorithms **Sorting** • Sorting Concepts • Bubble Sort • Selection Sort • Insertion Sort • Compare Sorting Algorithms **Programming mini-projects: Using the problem-solving method** • Defining • Designing • Coding • Testing and debugging	

Teaching Syllabus

Topic/Theme/Unit	Expected learning outcomes	Recommended teaching methods	Suggested resources	Assessment of learning outcomes
Year 1/Term 1				
COMPUTER SYSTEMS • Characteristics and Functions of Computers • Components of a Computer System • Safe Use of Computers • Role of Computers in Society	Students will be able to: • State and explain characteristics and functions of computers • outline the architecture of the central processing unit (CPU) • describe the functions of the arithmetic logic unit (ALU), the control unit (CU) and the registers within the CPU • identify and describe the functions of input and output devices • describe primary memory • describe secondary memory • define and explain the terms: hardware, software, network • understand and apply the safe use of computers • explain the role of computers in society	Ask students to write down at least four things they know about computers, their characteristics (e.g., speed, accuracy, versatility, reliability, etc.) and functions (e.g., input, processing, output, storage, etc.) Discuss the characteristics and functions of computers with students Illustrate computer architecture using a block diagram showing the relationship between the elements of the CPU, input and output, and storage. Discuss the functions of each constituent part of the CPU Explain briefly how input and output devices convert data into acceptable form within the computer system.	Textbooks Computers Internet devices and/or pictures showing hardware, software, and input/output devices	Students are able to: • explain to someone who has never used a computer what it is and how it works • write a report / make a mind-map of the characteristics and functions of a computer • reproduce a block diagram showing the architecture of the central processing unit (CPU) • describe the functions of the ALU, CU and registers • identify input and output devices and describe their functions • distinguish between RAM and ROM, and their use in primary memory • list and describe the different devices providing secondary memory • explain the advantage and disadvantages of each type of memory • write statements explaining the given terms • explain to an absent friend the risks and remedies of computer use

		Discuss and show (pictures of) input devices (e.g. keyboard, mouse, scanner, touch screen) and output devices (e.g. monitor, projector, printer, headphone, speaker, printer, etc.) Explain the difference between random access memory (RAM) and read-only memory (ROM), and their use in primary memory Discuss with students why secondary memory might be needed. Show the need for persistent storage and the different devices which have been, and are currently used for storage Briefly discuss the terms and show how they fit within a computer system Discuss the risks related to the use of computer equipment (e.g., to the back wrists and eyes) Discuss some simple strategies for minimising these risks		• take part in group discussion about the role of computers in society. Reflect on the good/bad aspects of computer use

		Discuss the merits and demerits of computers in society, e.g., although they are used to improve productivity and solve problems in all fields of study and work, e.g., education, health, business, manufacturing, etc., they are also used for cyberbullying and cybercrimes		
ESSENTIAL COMPUTER SKILLS • Introduction to the Windows Operating System • Introduction to Computer Applications Software • Basic Keyboard and Mouse Skills • Introduction to Using the Internet for Research	Students will be able to: • understand and apply the basic principles of using the Windows operating system: • start and shut down the computer • change computer settings (e.g., display, colour, accessibility options etc.) • resize, move and scroll windows • create, name and manage files and folders • manage hardware such as printers, scanners, mouse, digital cameras etc.	Demonstrate and guide students to use the various components of Windows Demonstrate and guide students to use the features of a computer application software such as a word processing application Demonstrate and guide students to use the keyboard and mouse to develop acceptable speed and accuracy. Demonstrate and guide students to access the internet and find information on a topic Guide students to evaluate the website	Computers (with Windows OS and typical software applications) Internet Printers Textbooks Activity sheets	Students are able to: • demonstrate they can follow the basic principles of using the Windows operating system in practice • demonstrate they can create, edit, save, and print documents of computer applications • demonstrate they can use the keyboard and mouse effectively • demonstrate they can access the internet safely, and validate the information found

	• create, edit, save, and print documents of a computer applications software • develop basic keyboarding and mouse use • access the internet and find and evaluate information	information, e.g., by checking the domain name (reputable extensions include .edu, .ac, .gov etc.), or cross-checking with another site, to assess whether the source of the information can be trusted		
INTRODUCTION TO PROGRAMMING • Computer programs • Programming languages and the programming environment • Syntax and syntax errors • Hello World!	Students will be able to: • describe a computer program, its uses, and features • describe programming languages and outline their main features • describe the features of low-level and high-level languages • describe programming language environments • develop familiarity with the programming environment of the specified programming language • identify basic syntax of the specified language • use basic syntax to write a first program in the	Show a short computer program on the whiteboard, e.g., a program which shows a message indicating the result of a calculation on the screen Use the program to explain the uses of a computer program to solve a perceived problem (e.g., in education, communication, sports, entertainment, business, etc.) Discuss the features of a familiar computer program, e.g. a mobile app or social media site Explain what a programming language is and discuss their main	• Computers • Operating system (usually Windows) • IDE of specified programming language OR • Text editor and specified language implementation • Internet access • Printers • Textbooks • Activity sheets	Students are able to • research on the internet the uses of computer programs • choose and outline the features of a computer program they use in their studies (e.g., an office application such as MS Word etc.) • describe and explain features of a programming language • create a mind map of low-level and high-level programming languages, giving examples, features and advantages and disadvantages of each type • answer questions such as: "What is the language environment needed to program in …?"

	specified programming language • Identify and correct common syntax errors in computer program	features, e.g., simplicity, abstraction, efficiency, etc. Introduce the two types of languages. Guide students to research on the internet the features of both types of languages, giving the advantages and disadvantages of each type Discuss the different programming environment whether a text editor and the specified language implementation or an integrated programming environment (IDE) Demonstrate the features of the programming environment of the specified language Use statements to demonstrate some of the basic syntax of the given language, e.g., for input and output (I/O) Guide students to use the programming learning environment to write a simple program (written using Python, for		"What is the difference between a text editor and an IDE?" "What else is needed with a text editor for it to run?" • Answer questions such as: What is syntax used for?" • identify and write programs using basic syntax • correct any syntax error in their programs

		example) to print "Hello World!" on the computer screen Use deliberate errors to demonstrate how the specified programming language deals with them		
		Year 1/Term 2		
PROBLEM SOLVING AND PROGRAMS • Problem Solving Method • Algorithms • Control Structures • Decomposition • Abstraction	Students will be able to: • understand the concept of problem solving in program development: ◦ define the problem ◦ design an algorithm to solve the problem ◦ code the program ◦ test and debug the program ◦ document the program code ◦ implement and maintain the program • state and explain the characteristics and uses of algorithms • describe the three program control structures of sequence, selection, and iteration • understand the concept and need for decomposition in program development	Discuss with students the problem-solving method as applied to program development Write the steps (or programming process) on the board. Divide the class into groups of 4/5 students Give each group one of the steps to research on the internet and present their findings Ask the class to select the most informative of each step and write up as notes. Explain what an algorithm is and the need for one in problem solving Discuss the characteristics of algorithms and show simple everyday	• Computers • Internet access • Textbooks • Activity sheets	Students are able to: • list and describe each of the steps involved in writing a computer program • create a mind map of the problem solving (programming) process • describe algorithms and its use during problem solving in computer programming • use given pseudocode and flowchart to identify control structures • write a short report on decomposition and why it is needed in program development • explain and give examples on abstraction in everyday life • give reasons for abstraction in program development

	• understand the concept and need for abstraction in program development	examples using pseudocode and flowchart Briefly explain the input, process, and output sections of the algorithm Use simple everyday examples of pseudocode and related flowcharts to discuss the three basic control structures and how they control the flow of an algorithm Show and discuss how different algorithms and resulting programs can be used to address the same problem Use everyday examples (e.g. planning a party or holiday) to discuss a problem is 'decomposed' or broken down into smaller sub problem Discuss how decomposition helps each sub problem to be solved independently from other parts of the program Explain abstraction as the process of hiding complex details from a		

		user Give everyday examples, e.g. talking to your friend on a phone without knowing how it connects you to your friend Discuss why abstraction is needed in program development		
PROGRAMMING CONCEPTS I • Keywords • Variables and Constants • Identifiers • Assignment • Comments • Data Types • Operators • Order of Operations • Strings • Data Structures • Exception Handling • Input/Output	Students will be able to: • identify the keywords of the specified programming language • declare variables and constants • use standard naming conventions to create identifiers • write assignment statements for variables • use comments in programs for clarity • understand and use the basic data types ◦ integer ◦ real (fixed and scientific notation) ◦ character ◦ string ◦ Boolean	Explain keywords in the specified programming language as reserved words used only for the purpose for which it was defined – they cannot be used for any other purpose Provide a table of the reserved words, or keywords of the language Explain the difference between a variable and a constant. Explain declaration of variables and constants Guide students to follow the conventions of the specified programming language (e.g., Python) to create identifiers or variable names	• Computers • Operating system (usually Windows) • IDE of specified programming language OR • Text editor and specified language implementation • Internet access • Printers • Textbooks • Activity sheets	Students are able to: • make appropriate use of the keywords of the specified language • translate mathematical expressions into programming expressions by declaring variables and constants, assigning appropriate values, and using standard naming conventions • add comments to programs to aid clarity • use given program, to identify the appropriate data types • use given program, to identify the correct operators (arithmetic, relational or Boolean) • use given pseudocode and flowcharts, to create a program that uses the correct operators

	• understand and use the arithmetic operators: • addition • subtraction • multiplication • real / float division • integer division, including remainder • understand and use the relational operators: • equal to (=) • not equal to (≠,) • less than (<,) • greater than (>,) • less than or equal to (≤,) • greater than or equal to (≥) • understand and use the Boolean operators: • NOT • AND • OR • use order of operations in evaluating mathematical and Boolean expressions • use basic string handling techniques • understand the concept	Demonstrate how to use the correct syntax to assign values to variables Show how comments are non-executable statements added to a program to make it easy to understand and maintain Explain the different variable data types including how they are declared using the syntax of the specified language Discuss with examples what each data type represents, e.g., string represents a sequence (or string) of characters, "Hello World!" Briefly explain the different memory space occupied by each variable type Demonstrate the various operators, noting that a calculation such as 11/2 would generate the following values: Integer division: the integer quotient of 11 divided by 2 (11 DIV 2) = 5 Remainder: the		(arithmetic, relational or Boolean) • work in pairs to solve mathematical and Boolean expressions using order of operations • recognise and use basic string handling techniques in a given program • recognise and explain the use of data structures in a given program • use, access and amend the one-dimensional array data structure in a program • identify and interpret exceptions in a given program • code input and output statements • list the expected output from chosen input values

	of data structures • use a one-dimensional array in a program • understand basic exception handling • obtain user input from the keyboard • output data and information from a program to the computer display	remainder when 11 is divided by 2 (11 MOD 2) = 1 Demonstrate using pseudocode how the relational and Boolean operators are usually used together Provide a written program and ask students to identify different operators used in the program Demonstrate how to evaluate mathematical and Boolean expressions using the standard order for arithmetic calculations (BODMAS) and the order NOT, AND, OR for Boolean. Guide students to also use brackets to change priority of Boolean operations Demonstrate and guide students to manipulate a string using its length, position and by extracting a substring. Demonstrate and guide students to get the character code for a character and vice versa		

		Demonstrate and guide students to perform concatenation and string conversion operations Limit string conversions to: string to integer, string to real, integer to string and real to string. Introduce data structures, e.g., one- or multi-dimensional arrays and explain how they used in programs Demonstrate and guide students how to access, add, delete, and loop over the elements of a one-dimensional array Discuss the concept of exceptions and demonstrate using examples how exceptions are handled by the specified programming language Demonstrate and guide students to code for input from the user and output to the screen		
PROGRAMMING MINI PROJECTS:	Students will be able to: • use the concept of	Guide students to write a program to, for example,	• Computers • Operating system	Students are able to • use appropriate problem-

USING THE PROBLEM-SOLVING METHOD Defining Designing Coding	problem solving in program development to: • define the problem • design an algorithm to solve the problem • code the program	add two numbers Guide students to define the problem by stating what they are required to do Guide students to design a solution using an algorithm both in pseudocode and flowchart. They ask for input from the user, process the information and output a message giving the result of the addition. Guide students to code the program using their algorithm, making sure they correct syntax errors and handle exceptions arising from their code	(usually Windows) • IDE of specified programming language OR • Text editor and specified language implementation • Internet access • Printers • Textbooks • Activity sheets	solving concepts to write a program to solve a given problem. Problems include: • adding three numbers • finding the average of two or three numbers • calculating perimeter, areas and volumes of shapes etc.
Year 1/Term 3				
FLOW OF PROGRAM: CONTROL STRUCTURES • Introduction to programme flow • Sequence • Selection • Iteration	Students will be able to: • understand the concept of program flow • describe and use the three control structures of sequence, selection, and iteration ◦ sequence: ▪ a linear execution of statements	Demonstrate using pseudocode and flowcharts how more complex algorithms and programs are written using the three control structures Discuss sequential control as the default means by which a program is executed as used in the basic	• Computers • Operating system (usually Windows) • IDE of specified programming language OR • Text editor and specified language implementation • Internet access • Printers	Students are able to: • use given pseudocode, flowchart, and program source code to identify and interpret control structures • use given pseudocode and flowcharts, to write code that uses sequential, selection and iteration control structures

	◦ selection (conditionals): ▪ if/then ▪ if/then/else ▪ case ▪ Boolean logic ◦ iteration (loops): ▪ while ▪ for ▪ do/while ▪ repeat/until	programs to date Explain how selection control is used to execute one or more statements if a given condition is met Guide students to research the internet to find out what iteration control works (repeats a statement a certain number of times, or while a condition is fulfilled)	• Textbooks • Activity sheets	
PROGRAMMING MINI PROJECTS: **USING THE PROBLEM-SOLVING METHOD** • Defining • Designing • Coding • Testing and debugging	Students will be able to: • use the concept of problem solving in program development to: • define the problem • design an algorithm to solve the problem • code the program • test and debug the program	Guide students to work independently to write a program, for example to find the larger of two numbers. Guide students to define the problem by stating what they are required to do Guide students to design a solution using an algorithm both in pseudocode and flowchart. They ask for input from the use, process the information and output a message giving the result of the addition. Guide students to code the program using their	• Computers • Operating system (usually Windows) • IDE of specified programming language OR • Text editor and specified language implementation • Internet access • Printers • Textbooks • Activity sheets	Students are able to: • use appropriate problem-solving concepts to write a program to solve a given problem. Problems include: • find the largest / smallest number among three numbers • generate the 5 times tables from 1x to 12x • output the count of all even numbers between a user defined range of numbers • check whether a number is prime or not (composite) • write error message when input number is not 5 or 6 • etc. • answer questions such as:

		algorithm, making sure they correct syntax errors and handle exceptions arising from their code Discuss with students how their programs can be improved Choose one or two programs and compare the code Ask students to improve their own programs in the light of the discussion Guide students to improve the previously written program by writing code to validate input into the program		• "How can we check that the correct input has been made by the user?" • "What sorts of error message can be given to alert the user they have given incorrect input? • write new algorithms for input validation of the previously written programs • add input validation code to their previously written programs

Year 2/Term 1				
PROGRAMMING CONCEPTS II • Data Types • Operators • Order of operations • Strings • Control structures • File handling	Students will be able to: • recall and extend use of data types to include • date/time • records (or equivalent) • define and use user-defined data types based on the specified language's built-in data types. • recall and extend use of the arithmetic operators to include:	Guide students to recall the integer, real (fixed and scientific notation), character, string, Boolean data types. Introduce and explain new data types for the specified programming language Demonstrate how user-defined data types can be created by using the built-in data types	• Computers • Operating system (usually Windows) • IDE of specified programming language OR • text editor and specified language implementation • Internet access • Printers • Textbooks • Activity sheets	Students are able to: • identify the new programming techniques from given pseudocode, flowcharts and source code • create pseudocode, flowcharts and source code to demonstrate understanding of the advanced programming techniques

	• exponentiation • rounding • truncation • recall and extend use of the relational operators: • =, ≠, <, >, ≤, ≥ • recall and extend use of the Boolean operators to include: • XOR • recall and extend use of order of operations in evaluating mathematical and Boolean expressions to include new operators • recall and extend use of string handling techniques to include: • string to float • float to string • date/time to string • string to date/time. • recall and extend control structures to include: • definite and indefinite iteration • nested selection • nested iteration • read from/write to a text file • read from/write to a	Guide students to create their own data types Review the various operators: (+, –, , real/ float division, integer division Demonstrate using pseudocode how the relational and Boolean operators are usually used together Review the NOT, AND OR Boolean operators Demonstrate and guide students to use arithmetic, relational and Boolean operators, alone and in combinations within algorithms and code containing selection and iteration structures. Review string handling techniques including string manipulation and how to perform concatenation and string conversion operations Review the following conversions: string to integer and vice versa, string to real and vice		

	binary (non-text) file.	versa Demonstrate and guide students to perform the new string handling techniques Review basic selection and iteration control structures Use algorithms and code to discuss how far and wide loops are used for definite and indefinite iteration respectively Demonstrate and guide students to recognise nested structures using pseudocode and flowcharts Guide students to write algorithms to show multiple layers of nested selection and iteration constructs Demonstrate and guide students to input and output via a text file Demonstrate and guide students to input and output via a binary file		

DATA STRUCTURES AND ARRAYS • Data Structure Concepts • Static and Dynamic Data Structures • Arrays	Students will be able to: • improve understanding of data structures • differentiate between static and dynamic data structures • understand and use one- and multi-dimensional arrays as examples of data structures	Review the concept of data structures Give examples and uses of the most commonly used data structures, e.g., arrays, linked lists, stacks, queues, etc. Discuss static and dynamic data structures. Guide the students to work in pairs to give the differences between static and dynamic data structures Demonstrate using algorithms and programmes some basic array operations, e.g., traverse, search, and update Demonstrate the procedure required to insert into and delete elements from arrays Guide students to write algorithms and code for operations in one- and multi-dimensional arrays		Students are able to: • explain to a friend, the most commonly used data structures • draw up a table showing the differences between static and dynamic data structures • write algorithms and code to traverse, search and update one- and multi-dimensional arrays • write algorithms and code to insert into and delete elements from one- and multi-dimensional arrays
PROGRAMMING MINI PROJECTS: USING THE PROBLEM-SOLVING METHOD	Students will be able to: • use the concept of problem solving in program development to:	Guide students to work independently or in pairs to write a program, for example to find the larger	• Computers • Operating system (usually Windows) • IDE of specified	Students are able to: • use appropriate problem-solving concepts to write a program to solve a given

• Defining • Designing • Coding • Testing and debugging	• define the problem • design an algorithm to solve the problem • code the program • test and debug the program	of two numbers. Guide students to define the problem by stating what they are required to do. Guide students to design a solution using an algorithm both in pseudocode and flowchart. They ask for input from the use, process the information and output a message giving the result of the addition. Guide students to code the program using their algorithm, making sure they correct syntax errors and handle exceptions arising from their code. Discuss with students how their programs can be improved. Choose one or two programs and compare the code. Ask students to improve their own programmes in the light of the discussion. Guide students to improve the previously written program by writing	programming language OR • Text editor and specified language implementation • Internet access • Printers • Textbooks • Activity sheets	problem. Problems include: • find the roots of the quadratic equations $ax^2 + bx + c$ (real roots only) • swap two variables' values without using third variable (Hint: use the XOR operator) • find the sum of individual digits of a given positive integer • generate the first *n* terms of the Fibonacci, (Hint: use one-dimensional array to store the series) • answer questions such as: "How can we check that the correct input has been made by the user?" "What sorts of error message can be given to alert the user they have given incorrect input? • write new algorithms for input validation of the previously written programs • add input validation code to their previously written programs

		code to validate input into the program.		
Year 2/Term 2				
SUBROUTINES: PROCEDURES AND FUNCTIONS • Subroutine concepts • Difference between procedures and functions • Advantages of using subroutines • Using parameters in subroutine • Subroutine operations • Built-in and user-defined subroutines	Students will be able to: • describe a subroutine and its uses • explain the difference between procedures and functions • list and describe the advantages of subroutines • describe the purpose and functions of parameters in subroutines • understand and apply defining and calling subroutines using parameters • understand and apply passing data into subroutine • understand and apply subroutines which return a value • understand and use both built-in and user-defined subroutines	Discuss how a subroutine is a block of code that is called from different places from within a main program or other subroutines, and sometimes returns a value to the calling code Explain how they are used to simplify the complexity of a program (i.e. for decomposition) Illustrate using examples of algorithms and programs Explain the difference between procedures and functions and how they are used in the specified programming language Discuss the advantages of subroutines in a program (e.g., improved organisation of the programme, increased usability as the subroutine can be called from another sub-routine programme, increased readability for other users of the code, etc.)		Students are able to: • explain what a subroutine is and what it is used for • answer questions to explain the differences between procedures and functions • explain the advantages of using subroutines • write a brief report on the purpose and functions of parameters in subroutines • write algorithms and program to define and call subroutines, using meaningful identifiers, passing data to and using any return value from the subroutine as appropriate • write algorithms and programs to use both built-in and user-defined subroutines

		Guide the students to research and make notes from the internet the purpose and functions of parameters in sub-routines Demonstrate and guide students to use algorithms and programs to define and call subroutines. Show how meaningful identifiers are used which give an indication of what the subroutine is set up to do. Ensure students understand how to call a sub-routine from the main run of the program using parameters Demonstrate and guide students to use algorithms and programs to pass data into a subroutine. Limit this to passing data by value only (call by value) Demonstrate and guide students to use algorithms and programs which return data from a subroutine		

		Discuss how the calling code uses the return value in the main program Explain how they can also be used, without being declared as variables, to call other subroutines Demonstrate and guide students to use algorithms and programs to use the built-in subroutines of the programming language Guide students to code their own user-defined subroutines to improve the organisation and readability of their programmes		
• **Year 2**	Students will be able to: • use the concept of problem solving in program development to: • define the problem • design an algorithm to solve the problem • code the programme • test and debug the program	Follow previous guidelines to guide students to work independently to write a programme to solve a given problem	• Computers • Operating system (usually Windows) • IDE of specified programming language OR • text editor and specified language implementation • Internet access • Printers	Students are able to: • use appropriate problem-solving concepts to write a program to solve a given problem, by calling appropriate built-in or user-defined subroutines Problems include: • find the length of a string input by user • find both the largest and smallest number in a list

			• Textbooks • Activity sheets	of integers • swap the values of two variables using a call by value • perform addition of two matrices (*provide method to add matrices, if needed*) • perform multiplication of two matrices (*provide method to multiply matrices, if needed*) • follow previous guidelines to improve the quality of written programs
Year 2/Term 3				
SEARCHING • Searching Concepts • Linear Search • Binary Search • Compare Searching Algorithms	Students will be able to: • describe searching and its uses in computer programming • understand and explain how the linear search algorithm works • understand and explain how the binary search algorithm works • compare and contrast linear and binary search algorithms.	Explain searching as the action of locating an element, called key, in a set of objects. The result of searching returns both the presence (or absence) and location of the object Use several versions of the algorithms to identify best, worst, and average cases of searching techniques in the programming language Guide students to research the advantages and disadvantages of	• Computers • Operating system (usually Windows) • IDE of specified programming language OR • text editor and specified language implementation • Internet access • Printers • Textbooks • Activity sheets	Students are able to: • give practical applications of the search algorithms • write algorithms and code to locate the requested object from a list of objects

		both algorithms Use the research to discuss how the two algorithms compare with each other		
SORTING • Sorting concepts • Bubble sort • Selection sort • Insertion sort • Compare sorting algorithms	• describe sorting and its uses in computer programming • understand and explain how the bubble sort algorithm works • understand and explain how the selection sort algorithm works • understand and explain how the insertion sort algorithm works • compare and contrast bubble, insertion and selection sort algorithms.	Explain sorting as the action of ordering a given set of elements in a particular order Guide students to research the advantages and disadvantages of the algorithms Use the research to discuss how the algorithms compare with each other		Students are able to: • give practical applications of the sort algorithms • write algorithms and code to order a given set of elements
PROGRAMMING MINI PROJECTS: **USING THE PROBLEM-SOLVING METHOD** • Defining • Designing • Coding • Testing and De-bugging	Students will be able to: • use the concept of problem solving in program development to: • define the problem • design an algorithm to solve the problem • code the program • test and debug the program	Follow previous guidelines to guide students to work independently to write a programme to solve a given problem	• Computers • Operating system (usually Windows) • IDE of specified programming language OR • text editor and specified language implementation • Internet access	Students are able to: • use appropriate problem-solving concepts to write a program to solve a given problem. Problems include: • find the positions of 5 out of a given list of 10 elements (linear search) • given a list of elements and a key value, find i)

			• Printers • Textbooks • Activity sheets	whether the key is present ii) return the position of key, or any other appropriate message (linear, binary search). Which is more efficient? • given a list of 10 elements, display the partially sorted list after three complete passes (Bubble sort) • identify the number of swaps required for sorting a given list using selection sort and bubble sort, and identify which is the better sorting technique giving reasons *The appropriate algorithm is given in brackets* • Follow previous guidelines to improve the quality of written programs

Year 3/Terms 1 and 2				
PROGRAMMING THE EXTENDED-PROJECT: USING THE PROBLEM SOLVING METHOD • Defining • Designing • Coding • Testing and Debugging • Implementing • Maintaining	Students will be able to: • define a simple programming problem • design an algorithm to solve a problem • code the programme • test and debug the programme • document the programme code • implement the programme • maintain the programme	Guide students to identify what is known about the problem and what the desired result is Guide students to produce a written agreement (specification) that specifies the kind of input, processing, and output required Guide students to design the solution to the problem by writing an algorithm: both pseudocode and a flowchart Guide students to translate the logic from the pseudocode and flowchart to the specified programming language Guide students to plan their test data to ensure all parts of the program are tested Guide students to trace, or check, the logic of the program to ascertain that it is error-free and workable Guide students to correct bug or mistakes		Students are able to complete a programming project to the required standard. The project can be either assigned by the teacher or be a problem proposed by students based on their daily experiences which can be solved through programming

		Ensure students follow good programming practice by documenting both within the program code (comments) as well as documentation for stakeholders Guide students to share their program for others to run and critique Guide students to make improvements to their programs bearing in mind the critique received Discuss how programs can be maintained in general. Guide students to research for more information on the internet and write a brief report on program maintenance		

RESOURCES

Textbooks
Computers (with Windows Operating System and typical software applications)
Internet
Devices and/or pictures showing hardware, software, and input/output devices
Computers
Printers
Activity sheets
Integrated Development Environment (IDE) of specified programming language OR Text editor and specified language implementation