Curriculum Plans: Year 10 (Computer Science)

	Topic	Knowledge: By the end of the unit students will know:	Skills: What skills will students have developed by the end of this unit?	Key terms: What new key terms and vocabulary will be learnt in this unit?	Summative Assessment: How will pupils be assessed in this unit?
	Inputs, outputs and structure	• Introduction to basic Python syntax, including the use of print statements and input functions. • Understanding how variables store data inputted by the user. • Importance of data types (e.g., strings, integers, floats) and how they affect program behavior. • The structure of a Python program, including sequence and simple decision-making (if statements). • The concept of program flow and how code executes in a structured sequence. • Basic error handling (e.g., dealing with invalid user input). • The role of comments in improving program readability. • Introduction to Python's built-in functions and how to call them in a structured program.	• Writing simple Python programs that use print() to display messages and input() to capture user input. • Defining and using variables to store and manipulate data captured via input(). • Converting user input data types using functions like int(), float(), and str(). • Organizing code using sequence and basic conditionals to control program flow. • Using if, elif, and else to introduce basic logic into programs. • Implementing simple error-handling strategies such as validating user input for correct data types. • Writing clear comments to explain code structure and logic.	• Syntax, Print Statement, Input Function • Variable, Assignment • Data Types, Type Casting • Sequence, Conditional Statements • Program Flow, Conditional Logic • Error Handling, Validation • Comment • Built-in Functions, Function Call	

Curriculum Plans: Year 10 (Computer Science)

			• Using built-in functions like len() or type() to enhance program functionality.		
	Selections	• Understanding the concept of selection in programming and how it is used to make decisions. • Introduction to relational operators (<, >, ==, !=, <=, >=) and their role in comparisons. • The use of Boolean expressions and how they evaluate to True or False. • Understanding nested selection and how multiple conditions can be checked within one another. • The importance of indentation in Python to ensure proper code structure and execution. • The concept of logical operators (and, or, not) and how they combine multiple conditions. • Handling multiple selection cases efficiently using elif statements.	• Writing Python programs that use if, elif, and else statements to implement decision-making logic. • Applying relational operators to compare values and control the flow of a program based on conditions. • Writing Boolean expressions to evaluate conditions and make decisions within a program. • Implementing nested if statements to handle more complex decision-making scenarios. • Structuring code using correct indentation to avoid syntax errors and improve readability. • Using logical operators to combine multiple conditions and control program flow.	• Selection • Conditional Statement • Relational Operators • Boolean Expression • True • False • Nested Selection • Nested If • Indentation • Syntax • Logical Operators • And, Or, Not • Elif Statement	

Curriculum Plans: Year 10 (Computer Science)

			• Simplifying decision-making logic using elif to avoid unnecessary nesting.		
	Condition controlled iterations	• Understanding the concept of condition-controlled loops (while loops) in programming. • Knowing how to initialize and update variables within a loop to prevent infinite loops. • The difference between entry-controlled and exit-controlled loops. • Using break and continue statements to manage loop execution. • Understanding the importance of loop conditions and how they are evaluated to True or False. • Using nested while loops to handle more complex, repeated tasks. • The role of condition-controlled loops in input validation (e.g., repeatedly asking for valid input). • How to optimize loops by minimizing unnecessary iterations and conditions. • Understanding when to use while loops instead of other	• Writing Python programs that use while loops to repeatedly execute code based on a condition. • Properly initializing and updating loop control variables to ensure the program terminates as expected. • Implementing entry-controlled loops (while) and explaining how they differ from other loop types (e.g., do-while in other languages). • Controlling loop flow with break to exit early and continue to skip iterations based on conditions. • Writing Boolean conditions that control the loop execution until a specified condition is met.	• Condition-Controlled Loop • While Loop • Infinite Loop • Loop Control Variable • Entry-Controlled Loop • Exit-Controlled Loop • Break Statement • Continue Statement • Boolean Condition, True, False • Nested Loop • Input Validation • Repetition • Loop Optimisation • Efficiency • For Loop, • While Loop • Conditional Iteration	**Practical Task –** Programming

		types of iteration, such as for loops.	• Implementing nested loops to perform operations within another loop (e.g., iterating over multi-dimensional arrays). • Using condition-controlled loops to implement input validation, ensuring the user provides valid data. • Writing efficient loops by reducing redundant checks and iterations. • Deciding when a while loop is more appropriate based on whether the number of iterations is unknown or depends on a condition.		
	Number operations and libraries	• Understanding basic number operations in Python (addition, subtraction, multiplication, division). • The use of modulus (%) and floor division (//) for integer operations. • Introduction to Python's mathematical functions from the math library (e.g., sqrt(), pow()).	• Writing Python programs to perform mathematical operations using arithmetic operators. • Implementing modulus and floor division to handle integer division and remainder calculations.	• Arithmetic Operators, Addition, Subtraction, Multiplication, Division • Modulus, Floor Division, Remainder	

		• Understanding how to import and use Python libraries for enhanced functionality. • The importance of floating-point arithmetic and precision issues. • Applying formatting to output numbers, including specifying decimal places. • Using the random library for number generation in simulations and games. • Writing functions that accept numbers as arguments and return calculated results.	• Utilizing Python's built-in math library for more advanced mathematical operations like square roots and powers. • Importing external libraries and calling their functions to perform specific tasks in a Python program. • Handling floating-point numbers and understanding potential precision errors in calculations. • Formatting numbers for output using Python's format() function or f-strings. • Generating random numbers using the random library for tasks like simulations or games. • Creating reusable functions that perform mathematical operations and return results.	• Math Library, Square Root, Power Function • Import Statement, Library • Floating-Point Number, Precision • Number Formatting, F-String • Random Library, Random Number Generation • NumPy, Arrays, Numerical Computation • Function, Return Value	
	One-dimension arrays and lists	• Understanding the concept of lists (arrays) in Python and their role in storing multiple values.	• Creating and manipulating lists in Python using indexing	• List • Array • Indexing • Static Array	**Practical Task –** Programming

		• Difference between static arrays and dynamic lists in Python. • How to access, modify, and delete elements in a list using indices. • Iterating over a list using loops (e.g., for loop) to access each element. • Built-in functions for lists such as len(), max(), min(), and sum(). • List slicing to retrieve subsets of data from a list. • How to concatenate and repeat lists. • Sorting and reversing lists using sort() and reverse() methods.	and list operations (e.g., append, remove). • Using dynamic lists to store data that can change in size during program execution. • Accessing and modifying list elements by index, and removing elements using list methods (e.g., pop(), del). • Writing loops to traverse lists and perform operations on each element (e.g., summing elements). • Utilizing Python's built-in list functions to calculate properties of lists (e.g., length, maximum value). • Using list slicing to extract specific parts of a list (e.g., list[1:4]). • Concatenating multiple lists and repeating elements using + and * operators. • Sorting lists in ascending or descending order and	• Dynamic List • Index • Append • Remove • Pop • For Loop • Iteration • Len() • Max() • Min() • Sum() • List Slicing • Concatenation • Repetition • Sort() • Reverse()	

			reversing the order of elements.		
	Count controlled iterations and scope	• Understanding the concept of count-controlled loops (for loops) and their purpose. • Knowing how to define and control the iteration count using a range of values. • The difference between while loops (condition-controlled) and for loops (count-controlled). • How to nest for loops and apply them to solve complex problems. • Understanding variable scope: local vs. global variables. • The concept of loop control using break, continue, and pass. • Importance of scope in relation to loop variables and how variables declared inside a loop are scoped locally. • Use of functions within loops and how scope affects variable accessibility. • Efficient use of loops for handling repetitive tasks and how to optimize loops to reduce unnecessary iterations.	• Writing Python programs that use for loops to execute code a specific number of times. • Using the range() function to generate sequences of numbers for controlling iterations in for loops. • Choosing the appropriate loop structure (while or for) based on whether the number of iterations is known or dependent on a condition. • Implementing nested for loops to handle multi-dimensional arrays or more complex iteration tasks. • Using local and global variables appropriately in functions and loops to avoid conflicts. • Managing loop flow with break to exit a loop early, continue to skip an iteration, and	• Count-Controlled Loop • For Loop • Iteration • Condition-Controlled Loop, • For Loop • While Loop • Nested Loop Iteration • Variable Scope • Local Variable • Global Variable • Break, Continue, Pass • Local Scope • Global Scope • Function • Scope • Return Value • Loop Optimisation • Efficiency	**Practical Task –** Programming

			pass to ignore a condition. • Demonstrating how loop variables are scoped within the loop and explaining variable lifetime. • Writing programs that use functions inside loops while managing the scope of loop variables. • - Writing efficient loops by minimizing unnecessary operations within iterations.		
	String operations	• Understanding the concept of strings as sequences of characters in Python. • Common string methods such as len(), upper(), lower(), strip(), replace(), and find(). • How to concatenate strings and use escape characters (\n, \t, \\). • Understanding and using string formatting with f-strings and .format(). • How to iterate through strings using loops (e.g., for loop) to access individual characters.	• Creating and manipulating strings in Python using indexing and slicing. • Using built-in string methods to modify, format, and search within strings. • Concatenating multiple strings and using escape sequences to format text output. • Implementing string formatting techniques to inject variables and control the display of data.	• String • Indexing • Slicing • String Methods • len() • upper() • lower() • strip() • replace() • find() • Concatenation • Escape Characters (\n, \t, \\) • F-String • .format() Method • For Loop	

Curriculum Plans: Year 10 (Computer Science)

		• - Comparing strings using relational operators (==, !=, <, >).	• Writing loops to traverse strings character by character for tasks like counting vowels or letters. • Writing conditions that compare strings for equality or relative ordering.	• Iteration • Character • Relational Operators (==, !=, <, >)	
	Files and exceptions	• Understanding how to work with files in Python (opening, reading, writing, and closing files). • Differentiating between file modes ('r', 'w', 'a', 'rb', 'wb'). • Using with statements to handle file operations and ensure proper file closure. • Understanding how exceptions are used to handle runtime errors. • Common file-related exceptions such as FileNotFoundError, IOError, and EOFError. • - How to read different file formats (e.g., text, CSV) and handle binary files.	• Opening files using Python's open() function and properly reading from and writing to text files. • Using file modes to control file access, such as reading ('r'), writing ('w'), and appending ('a'). • Implementing with statements for safe file handling to automatically close files after operations. • Using try, except, and finally blocks to handle exceptions and manage program errors gracefully. • Catching and handling specific file-related exceptions to ensure robust file operations.	• Open • Read • Write • Close • File Modes ('r', 'w', 'a', 'rb', 'wb') • With Statement • File Context Manager • Exception Handling • Try, Except • FileNotFoundError, • IOError • EOFError • CSV • Binary Files	

			• Reading and writing structured data like CSV files using Python's csv module and working with binary files.		
	Two-dimension arrays and lists	• Understanding the concept of two-dimensional arrays (lists of lists) in Python. • How to access, modify, and iterate over elements in a two-dimensional array. • The structure and usage of nested loops for traversing two-dimensional arrays. • The difference between row-major and column-major order in data storage. • Performing common operations on two-dimensional arrays: insertion, deletion, and updating. • How to use two-dimensional arrays to represent grids, tables, and matrices in practical problems.	• Creating two-dimensional arrays and initializing them with values. • Accessing and modifying elements in a 2D array using nested indexing (array[row][col]). • Using nested for loops to iterate through rows and columns of a 2D array. • Traversing arrays in both row-major and column-major order, depending on the problem requirements. • Solving real-world problems (e.g., representing game boards, spreadsheets) using 2D arrays. • Understanding how Python stores two-dimensional arrays	• Two-Dimensional Array • Nested List • Indexing • Nested Loops • Iteration • Row-Major Order • Column-Major Order • Insert • Delete • Update	