**Computing Fundamentals**

1. **Syllabus 9597 - Learning Objectives**
   1. Fundamental algorithms

Students will apply standard algorithms in the creation of programming solutions for a range of problem scenarios. Students should know and understand:

* + 1. concept of a program and flowcharting
    2. search algorithms such as linear/sequential search, binary search, and hash table search, and comparison of efficiencies
    3. sort algorithms such as bubble sort, insertion sort, quick sort, and comparisons of their efficiencies
    4. modulo operation and weighted-modulus method of computation, and random number generation
    5. binary tree traversals: pre-order, in-order and post-order, and hence binary tree sort and binary tree search
  1. Abstraction

Students will use data structures in their programming solutions, and are able to explain or justify the use of a particular data structure. Students will understand how data may be represented/encoded, and the capabilities and limitations of computers and computation, as well as explain programming processes. Students should know and understand:

* + 1. data types such as integer, real, char, string, Boolean
    2. ASCII codes for character representation and representation of positive integers in binary, octal and hexadecimal forms
    3. the Unicode encoding system (that is, the Unicode Standard)
    4. data structures such as array, stack, queue, list and binary tree, and their associated algorithms
    5. concept of recursion
    6. limitations of the computation, problem situation and/or computer
  1. Modularity

Students will use design, debugging and testing techniques to assure a quality software product. They will apply top-down design methodology to develop an algorithm to solve a problem and be able to follow an algorithm. Students should know and understand:

* + 1. types of program errors: semantic, syntax, logic and arithmetic, and why they occur
    2. appropriate test cases (normal, abnormal, erroneous and boundary data) for testing algorithms and debugging techniques
    3. data validation techniques: range and type checks, and the difference between data validation and data verification
    4. the importance of defining a problem precisely
    5. techniques to formulate and represent a computer solution (e.g. data specification, top-down design, modular design, data flow diagrams, decision table/tree, pseudocode and programming language code) and step-wise refinement where applicable
    6. that clarity of programming solution may be enhanced through comments, indentation, white space and meaningful names
  1. Programming

Students will design and develop computer programming solutions for problem tasks. They will understand programming constructs and general object-oriented concepts through the use of a programming language. Good programming practices will be emphasised in the coding of programming solutions. Students should know and understand:

* + 1. input, output, sequence, selection and iteration constructs in programming
    2. serial and sequential text files
    3. classes and objects
    4. encapsulation and how classes support information hiding and implementation independence
    5. inheritance and how it promotes software reuse
    6. polymorphism and how it enables code generalisation

**Reading List**

* **Cambridge International AS and A Level Computer Science Coursebook (9608)**
  + Chapter 11: Algorithm design and problem solving ……………………………… 125-154
  + Chapter 12: Stepwise refinement and structure charts …………….…………… 155-175
  + Chapter 13: Programming and data representation ...…………………………... 176-211
  + Chapter 14: Structured programming ………………………………………...…… 212-227
  + Chapter 23: Computational thinking and problem-solving ..…………………….. 317-336
  + Chapter 24: Algorithm design methods ………………………………………….... 337-346
  + Chapter 25: Recursion ………………………………………..…………………….. 347-355
  + Chapter 26: Further programming ……………………………….…...……….…... 356-367
  + Chapter 27: Object-oriented programming (OOP) …………………..…………... 368-393
  + Chapter l: Information representation ……………………………………….…….. 2-17
  + Chapter 16: Data representation …………………………………………………... 246-257
* **Cambridge International AS and A Level Computer Science Revision Guide (9608)**
  + Chapter 9: Algorithm design and problem-solving ……………………………….. 88-96
  + Chapter 10: Data Representation ………………………………………………….. 97-105
  + Chapter 11: Programming ……………………………………………………..……. 106-125
  + Chapter 19: Computational thinking and problem solving …………………...….. 209-235
  + Chapter 20: Algorithm design methods ………………………………………….... 236-244
  + Chapter 21: Further programming ………………………………………………..... 245-273
  + Chapter 1: Information Representation …………………………………………..... 2-11
  + Chapter 13: Data Representation …………………………………………………... 133-145

**Contents**

**Part A - Fundamental skills**

1. Data Representation

Number systems

Coding of integers

Coding of text

1. Algorithms and representations

What are algorithms

Basic Constructs

Flowcharts

Pseudo-code

1. Refinement and modularity

Stepwise Refinement

Modularity

Structured Charts

**Part B - Programming In Python**

1. Variables
   1. Variables
   2. Data types
   3. Assignment
   4. Lists
   5. Dictionaries
2. Collections
   1. Strings
   2. Lists
   3. Tuples
   4. Dictionaries
3. Basic Constructs
   1. For loops
   2. While loops
   3. If statements
4. File Handling
   1. Opening and closing files
   2. Reading files
   3. Writing and appending to files
5. Types of Errors
   1. Semantic
   2. Syntax
   3. Logic
   4. Arithmetic
6. Exception Handling
   1. Exceptions
      1. ValueError
      2. FileNotFoundError
      3. NameError
      4. KeyboardInterrupt
      5. NotImplemented
   2. Try Statements
   3. Except Statements
   4. Else Statements
   5. Finally Statements
7. Good Programming Practices

**Part C - Functional Programming**

1. Functions
2. Iteration
   1. General Structure
3. Recursion
   1. Base Cases
   2. Recursion Calls

**Part D - Common Algorithms**

1. Complexity
   1. Need for efficiency
   2. Big-O notation
   3. Worst Case Complexity
2. Searching
   1. Linear Search
   2. Binary Search
3. Sorting
   1. Bubble sort
   2. Insertion sort
   3. Merge sort
   4. Quick sort

**Part E - Object Oriented Programming (OOP)**

1. Concepts
   1. Encapsulation and Modularity
   2. Inheritance and Polymorphism
2. Constructs
   1. Classes, Objects and Instances
   2. Attributes
   3. Methods
3. Design and Representation
   1. UML and Class Diagrams

**Part F - Abstract Data Types (ADT)**

1. Linked Lists
2. Stacks
3. Queues
4. Binary Search Trees (BST)

**Part A - Fundamental Skills**

1. Data Representations

1. What are the different forms of representing data?
   1. Name the different representations
2. When do we use the different forms (applications of each representation)?
   1. What does the application require, and what does the representation have that enables it provide what is necessary?
3. Why is there a need for this representation?
   1. What are the problems when we use another representation?
   2. What is lacking, and how does this representation solve it?
4. Why does this representation suit the application?
   1. What are the functionalities that this representation provides?
   2. What makes it easy to be used for the application?
5. What are its specific advantages over other representations?
   1. What makes it better than another representation?
   2. What added functionality does this provide?
6. What are its limitations?
   1. What are some possible problems that may arise from the use of it?
   2. What are some functionalities that may not be present if we use it?
7. How do we convert between the various representations?
   1. What are the methods we can use to convert?
8. How are these representations stored on the computer?
   1. How does the computer put them in storage?
   2. How does the computer access them?
9. How are these manipulated to achieve a desired outcome (performing operations)?
   1. What are the operations we can do on them?
   2. How are these operations carried out by the computer?
10. What are the technical terms associated with the representation?
11. How do we implement the use of such representations on Python?

2. Algorithms and representations

What are algorithms

Basic (or social) Constructs

Flowcharts

Pseudo-code

3. Refinement and Modularity

1. Stepwise Refinement
2. Structure charts

**Part B -Programming In Python**

1. Variable

<TEXT>

2. Collections

<TEXT>

3. Basic Contructs

<TEXT>

4. File Handling

<TEXT>

5. Types of Errors

<TEXT>

6. Exception Handling

<TEXT>

7. Good Programming Practices

<TEXT>

**Part C - Functional Programming**

1. Functions

<TEXT>

2. Iteration

<TEXT>

3. Recursion

<TEXT>

**Part D - Common Algorithms**

1. Complexity

<TEXT>

2. Searches

<TEXT>

3. Sorts

<TEXT>

**Part E - Object Oriented Programming (OOP)**

1. Object Oriented

<TEXT>

2. Classes

<TEXT>

3. Inheritance

<TEXT>

**Part F - Abstract Data Types (ADT)**

1. Linked Lists

<TEXT>

2. Stacks

<TEXT>

3. Queues

<TEXT>

4. Binary Search Trees(BST)

<TEXT>