Revision Guide

for AS AQA Computer Science

Paper 1

zigzageducation.co.uk
POD 6072

Contents

TEACHER'S INTRODUCTION

This revision guide has been written to support the AQA AS Computer Science specification (first teaching from September 2015, first exams in June 2016).

It summarises the essential theory required for the AS Paper 1 examination; more specifically, topics 1–4 of the AS specification:

1. Fundamentals of programming
2. Fundamentals of data structures
3. Software development
4. Theory of computation

An equivalent resource is also available for the AS AQA Paper 2 examination (topics 5–9).

Note that part of the Paper 1 examination is based on pre-release material (including skeleton programming code) that is released annually by AQA. For details of resources supporting this pre-release, see the ZigZag Education website.

Each section includes student notes, examples, diagrams and examination-style questions. Example answers to all of these questions can be found at the back of the resource. *Note that credit should also be given for any valid responses that are not explicitly included in this resource.* There is also a revision progress grid which students may find useful in the lead up to their exams.

Programming concepts are exemplified throughout using pseudocode and a number of high-level programming languages including Java, C++, Visual Basic and Pascal.

P Chapman, January 2016

REVISION PROGRESS TRACKER: AS

Use the grid below to track your progress while revising for your exam. Start by ente the top, and working down the grid, give a rating of between 1 (you really don't know i

This should help you to focus your revision on the areas that require it the most, so tha comes up in the exam. Use the Notes column to record any actions.

Repeat this process until you feel you are confident enough in all areas and are ready f

Specification Topic	Confidence Level (1-5)				
	Date:	Date:	Date:	Date:	
1 – Fundamentals of programing					
Data Types					
User-defined data types					
Built-in data types					
Assignments					
Iteration					
Selection					
Arithmetic operations					
Relational operations					
Boolean operations					
Variables and constants					
String handling					
Random numbers					
Exception handling					
Subroutines					
Parameters and return values					
Structured programming					

Specification Topic	Confidence Level (1-5)				
2 – Fundamentals of data structures					
Arrays					
Text files					
Binary files					
3 – Software development					
Analysis					
Design					
Implementation					
Testing					
Evaluation					
Problem-solving					
Sequence					
Assignment					
Selection					
Iteration					
Hand-trace algorithms					
Pseudocode → high-level code					
Abstraction					
Information hiding					
Procedural abstraction					
Functional abstraction					
Data abstraction					
Problem abstraction / reduction					
Decomposition					
Composition					
Automation					
Finite state machines					
State transition diagrams & tables					

TOPIC 1 – PROGRAMMING

1.1 PROGRAMMING

DATA TYPES

Data types

ⓘ A **data type** is used to describe the type of data that a variable contains in a compute

Type	Description
Integer	An integer is a whole number that can be positive or negative.
Real/Float	A real number contains a decimal point; the position of the decimal point
Boolean	Boolean data types represent two logical states: 'true' (typically 1) or 'fals
Character	A character represents a single alphanumeric item of data, such as a num
String	A string contains one or more characters, plain text such as: 'Hello World'
Date/Time	The Date/Time data type contains details of an instant in time that is not
Records / or equivalent	Record data structures are made up of a list of elements where each reco fields with different data types – they are an available feature of Pascal a Visual Basic uses an equivalent of records, based on user-defined data ty records and C++ and C# use Structs as an equivalent of records.
Arrays	Array data structures are made up of a list of date elements that are the s Arrays can be one-dimensional (similar to a list) or two-dimensional (the After the array has been declared, the elements in an array can be initiali the program. Array elements are often assigned in repetitive program coc

Built-in data type

ⓘ A **built-in (or language defined) data type** is one where the programming language c
Typical built-in data types for commonly used programs are listed in the table below.

Built-in Types	Visual Basic	Pascal	Java	
Integer	Integer (4 bytes)	Integer (4 bytes)	int (4 bytes)	int (4
Byte	Byte (8 bits)	Byte (8 bits)	byte (8 bits)	byte
Boolean	Boolean (1 byte)	Boolean (1 byte)	boolean (1 byte)	bool
Real	Double (8 bytes) Decimal (16 bytes)	Real (8 bytes) Currency (8 bytes)	float (4 bytes) double (8 bytes)	doub deci
Character	Char (2 byte)	Char (1 byte)	char (2 bytes)	char
Strings (*) as required	String (*)	String (*)	String (*)	strin

1.1 – Progress Check

1. Describe with examples the following data types:
   (a) Real/Float (2 marks) (b) Character (2 marks) (c) Array (2 ma

User-defined data types

ⓘ Additional data types to those built in to the programming languages are often requi declare variables to meet their requirements. These are **user-defined data types**.

The examples below show the data structure declaration for a record in Pascal and an equ

Record Example Data Structure in Pascal	Struct Example Data Str

<pre>TYPE
    StudentRecord = RECORD
      StudentName   :STRING(20);
      MobileNumber   :INTEGER;
      EntryYear   :INTEGER;
      FeesOwing   :REAL;
    END;
VAR
      Student : Array[0..19] of
      StudentRecord;</pre>
	<pre>struct StudentRec
	{
	    string Stud
	    int Mobi
	    int Entr
	    float Fees
	} StudentRecord;
	
	struct StudentRec</pre>

PROGRAMMING CONCEPTS

Programming concepts – variable declarations

ⓘ **Variables** are used in programs to store data that may change when the program is exec require that the data types of the variables in the program are declared before they can b

C#	Java	Pascal

<pre>bool blogic;
int intVal;
char chVal;
double Sum;
string stVal;</pre>
	<pre>boolean blogic;
	int intVal;
	char chVal;
	float Sum;
	String stVal;</pre>
		<pre>var
		  blogic : boolean;
		  intVal : integer;
		  chVal : char;
		  Sum : real;
		  stVal : string;</pre>

In Python, variables are declared when assigned. For example, **Distance = 10.5** will be de

Programming concepts – constant declarations

ⓘ **Constants** are used where data used in program is preset and does not change. Progra types of the constants in the program are declared and their value initialised before th

C#	Java	Pascal

<pre>const double
Pi=3.14;
const int x = 12;</pre>
	<pre>static final float Pi=3.14;
	static final int x = 12;</pre>
		<pre>Const Pi=3.14;
		Const x = 12;</pre>

In Python, constants cannot be declared in, so capitalise the variable and don't change it,

Programming concepts – assignments

ⓘ An **assignment** in computer science is where a value is computed within a program ar for the variable is stored in the memory of the computer.

Pseudocode example with data declarations

```
# Area of a Circle Calculation
START
      Float Area, Radius, Pi;              // Data Declarations
      Pi ← 1.1416;
      Area ← Pi * (Radius * Radius)        // Area assignment usi
END
```

In a sequence structure the program performs each action or assignment statement in ord

Programming concepts – subroutines

ⓘ A **subroutine**, which is identified by name, is a set of instructions that perform a certa called many times within a program.
ⓘ A **procedure** is a subroutine that is called to perform a task. It may or may not return
ⓘ A **function** is a subroutine consisting of a series of instructions to perform a task. Whe returns a value.
Tasks that are repeated within a program are often defined as procedures and functions a within the program. They consist of a series of statements declared outside of the main pr main program or by other subroutines.

Programming concepts – iteration

ⓘ **Iteration** or repetition is where a program executes a statement or statements that a is satisfied. The number of iterations in a loop can be further distinguished into defin	
ⓘ **Definite** iteration is where the number of iterations that will take place is known before the start of the execution of the main body of the loop. A typical example is where a loop is set up to input n values and prints them all out.	ⓘ **Indefinite** iteration will take place is nc determined by whe depends on the ass
The simple pseudocode examples below indicate the range of iteration types available in	

FOR LOOP
This iterative control method is useful when the same instructions or calculations have to be carried out for a known number of iterations.
The example is based on definite iteration as the loop will be repeated a known number of times (10).
	`#Example FOR loop`
	`Total ← 0`
	`FOR X = 1 TO 10`
	`    Total ← T`
	`ENDFOR`

WHILE LOOP
In this iterative control method technique, the loop operates when a condition is satisfied at the start of the loop and stops when this is no longer true.
The example is based on definite iteration as the loop will be repeated a known number of times (10).
	`#Example WHILE loo`
	`F ← 1`
	`counter ← 10`
	`WHILE counter > 0`
	`    F ← F + 1`
	`    counter ← c`
	`ENDWHILE`

DO WHILE LOOP
In this iterative control method technique, the loop operates when a condition is satisfied at the start of the loop and stops when this is no longer true.
The example is based on indefinite iteration as the loop will continue an indefinite amount of times dependent upon the number input by the user.
	`#Example DO WHILE`
	`F ← 1`
	`OUTPUT "Enter a Nu`
	`counter ← INPUT`
	`WHILE counter > 0`
	`    F ← F * co`
	`    OUTPUT F`
	`    counter ←`
	`ENDWHILE`

REPEAT UNTIL LOOP
This iterative control method technique is primarily used in Pascal, the loop operates at least once and then the condition is tested at the end of loop and repeats until the condition is no longer true.
The example is based on indefinite iteration as the loop will continue an indefinite amount of times until the user inputs 10.
	`#Example REPEAT UN`
	`REPEAT`
	`    OUTPUT "Enter`
	`    N ← INPUT +`
	`    OUTPUT N`
	`UNTIL N = 11`

Programming concepts – selection

ⓘ A **selection** structure is where the program executes different actions or statements c the simple pseudocode examples below indicate the range of selection types availab

IF-THEN statements are used to execute one block of code when a Boolean condition is TRUE, there is no alternative branching when the Boolean condition is FALSE.

```
# IF-THEN Example
IF (X > MAX) THEN
      X ← MAX
END IF
```

IF-THEN-ELSE statements are used to execute one block of code when a Boolean condition is TRUE and an alternative when the Boolean condition is FALSE.

```
# IF-THEN-ELSE Exa
IF (Age >= 18) THE
      OUTPUT "You
ELSE
      OUTPUT "Not
END IF
```

Nested IF-THEN-ELSE statements are used if there is more than one expression to be tested.

In the example shown there are two IF statements required in a row to assign the output sign of a number.

Once an IF or ELSE IF expression is true the OUTPUT is assigned and the program moves onto the next statement.

```
# Nested IF-THEN-E
IF (Score > ) THEN
      OUTPUT "Pos
ELSE IF (Score = (
      OUTPUT "Zer
ELSE
      OUTPUT "Neg
END IF
```

CASE statements (termed switch statements in programs such as JAVA/C) are a variation of an IF-THEN-ELSE statement where several IFs are used in a row.

The Nested IF-THEN-ELSE approach outlined above operates in a similar way to the CASE statement.

```
# CASE or SWITCH E
CASE Weekdays
     1: THEN Day
     2: THEN Day
     3: THEN Day
     4: THEN Day
     5: THEN Day
END CASE
```

Programming concepts – identifiers

ⓘ **Identifiers** are symbolic names used for any variable, function or data definition in a (they are normally given meaningful names to make the program understandable.

Examples:

- StudentExamScore is more understandable than x
- TotalCost is more understandable that t

1.1 – Progress Check

2. Describe the following programming concepts:
   (a) Subroutine (4 marks)
   (b) Procedure (3 marks)
   (c) Function (4 marks)
   (d) Iteration (3 marks)
   (e) Selection structure (3 marks)
3. Explain the difference between definite and indefinite iteration (4 marks)

Arithmetic operations in a programming language

Arithmetic Operation	Operator	Explanatio
Addition	+	In addition the numbers are added either s So **a + b = 16** where a = 12 and b = 4.
Subtraction	-	In subtraction the number on the right sid subtracted from the number on the left. So **a - b = 8** where a = 12 and b = 4
Multiplication	*	In multiplication the numbers on either si So **a * b = 48** where a = 12 and b = 4.
Real/Float Division	/	The division operator performs floating pc operands is a floating point value. In the e returned from the calculation. So **a / b = 7/2 = 3.5.**
Integer Division and Remainder	/	The division operator performs integer div integers. In the example below an integer So **7 / 2 = 3** and the fractional part of the
	MOD or %	The remainder from an integer division op modulus operator. For example **7 MOD 2 = 1** can also be writ
	DIV	The DIV operator performs integer divisio and calculates the quotient and remainde So **7 DIV 2 = 3r1** (3 remainder 1).
Exponentiation	B**n	B^n the base number B is multiplied repeat exponent. For example: 12^4 = 12 * 12 * 12
Rounding	The process of rounding is to replace a number with an Some programming languages have built-in functions t round function is available to round a value to the neare So 34.5674 can be rounded up to 34.57 using the functi where 2 is the number of decimal digits required.	
Truncation	Truncation is the process of limiting the number of digit programming languages have built-in functions to trunc For example in Java the truncate library function is used number of decimal places. So 21.7546 is truncated to 21.75 using the function: **tru**	

1.1 – Progress Check

4. (a) Explain the difference between the truncation and rounding arithmeti
   (b) Calculate 11 MOD 3 (1 mark)
   (c) Calculate 3**2 (1 mark)

Relational operations in a programming language

ⓘ Relational or comparison operators use different symbols in some programming langu
result as shown in the examples below:

Operator	Java / C# / Python	Pascal / Visual Basic	True exa
equal to	==	=	7 == 7 or
not equal to	!=	<>	5 != 4 or
less than	<	<	6 < 1
greater than	>	>	15 >
less than or equal to	<=	<=	6 <=
greater than or equal to	>=	>=	10 >=

Boolean operations in a programming language

ⓘ The relational operations listed above can be used in conjunction with the Boolean lo
complete range of complex Boolean expressions needed by programmers.

<table>
<tr><th>Operator</th><th>Truth Tables</th><th>Pseudocode Examples</th><th></th></tr>
<tr><td>NOT</td><td><table><tr><th>A</th><th>NOT A</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table></td><td>NOT (Year > 2014)</td><td>The lo
Jav
Pyt
The ex
greate</td></tr>
<tr><td>AND</td><td><table><tr><th>A</th><th>B</th><th>A AND B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td><td>(Age > 5) AND (Age < 15)</td><td>The lo
Jav
Pyt
The ex
is olde</td></tr>
<tr><td>OR</td><td><table><tr><th>A</th><th>B</th><th>A OR B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td><td>(Time < 9) OR (Time >5)</td><td>The lo
Jav
Pyt
The ex
before</td></tr>
<tr><td>XOR</td><td><table><tr><th>A</th><th>B</th><th>A XOR B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td><td>(Code =1) XOR (Code = 7)</td><td>The lo
Jav
Pyt
The ex
code is</td></tr>
<tr><td colspan="4">Note that the programming languages Pascal, Delphi and Visual Basic use similar symbols</td></tr>
</table>

1.1 – Progress Check

5. Design a truth table for the exclusive-or function using variable A and B

Constants and variables in a programming language

A constant is data that uses a literal value, which doesn't change in the computer program. Also a constant can be given a meaningful identifier such as Pi, which would be set at 3.1416.

Variables are normally referenced by an identifier, such as: Total, Sum and Group. Unlike constants the data stored in variables can change as the program is executed. For example, the variable Total may be modified as further terms are added to it.

Advantages of using an identifier or name for a constant rather than a literal value:

- The program is more understandable when reading words or symbols than readinc
- If you need to change the accuracy of **Pi** you only need to change it once even if it
- Constants with well-chosen labels or identifiers are easy to change for business re time to time, so using a label makes it easy to change it only once in a program.

Extract from coding example using C#

```
const float Pi=3.1416;            //Initialise a constant value
Circumference = 2 * Pi * Radius;  //Calculation using Pi more m
```

Extract from coding example using Pascal

```
CONST Pi=3.1416;                  {Initialise a constant value
Circumference = 2 * Pi * Radius;  {Calculation using Pi more mea
```

1.1 – Progress Check

6. Explain the difference between a constant and a variable in a programm

String-handling operations in a programming language

The programming languages used on this course use a string datatype and have a series o information contained in a string. The syntax for string handling varies between programr handling operations are described in general terms below.

Operation	Description with examples
Length	The length of a string is a commonly available function such as: Len(s length of a string. For example **Len("Computer Science")** would return a value of 16 an
Position	Items in a string are numbered from 0 to the length of the string. Cha based on their position in the string using the [ ] in Python. For example where s ="Computer", **s[2]** extracts character 'o' and **s[:3**
Substring	A substring is a string contained within another string. The substring position range input. For example, in C#: **"computer".Substring(0,0);** returns 'c' and **"com**
Concatenation	The concatenation operation is used to join two strings together whe For example, in Pascal: **"Computer" + "Science";** returns 'Computer S
Character and Character Codes	A character code is a binary representation of the characters used in a characters which have a character code based on either ASCII or Unic Conversion from a **character code to a character** can be carried out u For example, in Java the conversion from Unicode to the character is: **var character = String.fromCharCode(88)**; which returns 'X'.

String Conversion Operation Functions

Conversion	Visual Basic	Pascal	Java	
String to integer	CInt(s)	StrToInt(s)	Integer.ValueOf(s)	(
String to float	CDbl(s)	StrToFloat(s)	Float.ValueOf(s)	C
Integer to string	CStr(n)	IntToStr(n)	Integer.ToString(n)	(
Float to string	CStr(r)	FloatToStr(r)	Double.ToString(r)	(
Date/time to string	CDate(d)	DateToStr(d)	dateformatJava.format(d)	(
String to date/time	CStr(s)	StrToDate(s)	SimpleDateFormat(s)	Co

The conversion operations listed are specific functions to carry out these string conversio could be used to convert 2 into a string '2'.

1.1 – Progress Check

7. Explain with an example, the string handling term 'concatenation' (2 ma

Random number generation in a programming language

Random numbers lack any sort of pattern and programming languages contain built- numbers within a range.

Program	Example to create a random number in the ra	

Visual Basic	`Dim rn As New Random`
	`Dim answer As Integer`
	`answer = rn.Next(1,10)`
		Declare a new object ty
		Declare an integer varia
		Next (1, 10) generates

Python	`import random` `answer = randrange(1,11)`	import random loads th **randrange(1,11)** genera endpoint not included i
Pascal	`Randomize ;` `answer := Random(10) +1 ;`	Call Randomize procedu **Random(10)** generates range 1 to 10
C#	`Random rn = new Random();` `int answer = rn.Next(1,10) ;`	Declare an object type R **Next (1, 10)** generates
Java	`Random rn = new Random();` `int answer = rn.nextInt(10) + 1;`	Declare an object type R **nextInt (10)** generates range 1 to 10

Exception handling

ⓘ **Exception handling** is a technique used by the programmer to deal with error conditi
code will allow the program execution to continue under error conditions.

The simple pseudocode example indicates the principle of exception handling which
programming language chosen by the teacher.

```
ConsoleInput ← USERINPUT                          // Variable to be
TRY
        R ← ConvertToReal(ConsoleInput);         // Variable conve
                                                 can continue with
CATCH
        OUTPUT "Input was not a real variable"   // Error caught ar
END TRY
```

Subroutines (procedures and functions)

ⓘ **Procedures** and **functions** are useful in helping to provide a structure to create logica
There is a similarity between procedures and functions, in that they both support the
calling the function or procedure many times, rather than continually writing out or c

This approach reduces the amount of code and creates a more readable solution; they cor
blocks that are executed by using an identifier in one or more places throughout the prog
the runtime version of the code at compile time.

ⓘ **Built-in functions** are used in computer programs to create efficient code to solve coi
functions have been provided by the computer program developers. Software docume
functions available with guidelines on how to use them.

A typical example is the square root function or specifically **sqrt(val)** which is availab
Python.

So sqrt(9) returns the square root of 9 which is 3

Programming languages also allow the user the option to create their own functions
square function:

```
{Define a Function to square a number}
FUNCTION Square (Val)
        S ← Val * Val;
        Return S
END FUNCTION
```

Advantages of using procedures and functions

1. The same code is re-used, resulting in less code being needed (this makes the prc
2. The solution easier to understand and update when maintaining or fixing errors i

Parameters of subroutines

ⓘ A **parameter** is a variable that is used as a data input or an argume

The definition for a subroutine normally includes a list of parameters, a
the arguments are assigned to the relevant parameters.

Returning a value / values from a subroutine

An example of a simple subroutine (VAT) is shown below; when the subroutine is called it based on the input parameter.

VAT example in C programming language	**Subroutine defined**
The function is named VAT and contains one input parameter named 'purchase_price'. The parameter is data type double (floating point) and the data type returned is also double. The function calculation details are within the {} brackets	`double VAT(double purchase_p` `{` `return 0.2 * purchase_pr` `}`
Once the function has been defined it can be called as shown on the right.	`total_price = purchase_price`
If the purchase price is 100 then **VAT** function returns 20.	`purchase_price = 100` `total_price = 100 +` `= 100 +`

Local variables in subroutines

ⓘ Where **local variables** are declared and used in a subroutine they are only in existenc and are only accessible or in scope within that subroutine.

It is good practice to use local variables rather than global variables in subroutines and fu content of a local variable in a large program and the subroutine retains modularity wher variables are passed to it for execution.

Local variables are more efficient than global variables as you free up memory each time

Global variables in a programming language

ⓘ Where **global variables** are declared at the beginning of a program they can be acces program in contract to local variables which are only accessible in the program block

Global variables should only be used if they are needed throughout the complete pro programming languages where variables are global unless declared in a function or s

Note this is not the case in Python where all variables are local unless declared expli

1.1 – Progress Check

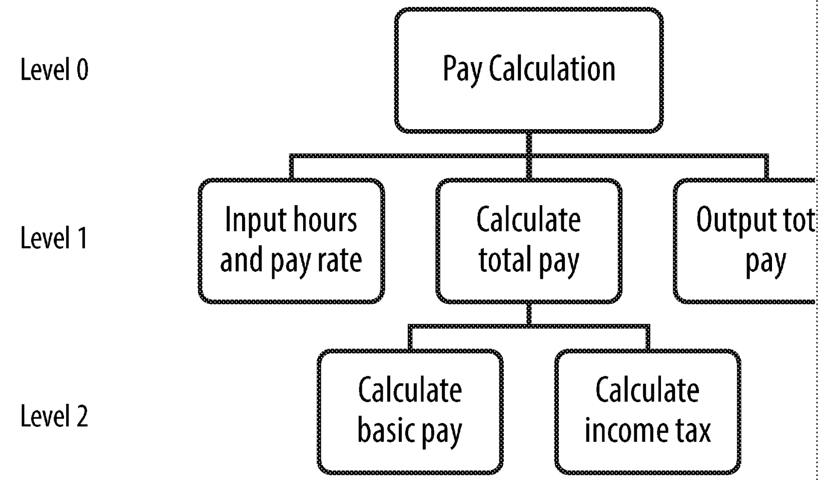
8. Explain why it is good practice to use local variables rather than global v

1.2 PROCEDURE-ORIENTED PROGRAMMING

Structured programming

ⓘ **Structured programming** is a type of procedural-oriented programming where the pr
help reduce development time and ensure that the program is easier to understand a

ⓘ **Hierarchy charts** are used to show the details of the modular structure in the progran
shows the modules involved in designing a simple pay calculation.



Advantages of structured approach

The structured approach to programming reduces the complexity of the task and has the f

1. Breaking down large programming tasks into manageable subtasks means that t programmers, which saves development time.
2. Program test and debug time is reduced, where modularity helps to reduce the t errors that may occur.
3. Programs are easier to understand and, therefore, easier to maintain; also a new introduction of an additional module.

1.1 – Progress Check

9. Describe the advantages of using a structured approach to programming

2.1 DATA STRUCTURES AND ABSTRACT DATA TYPES

Data structures

ⓘ A **data structure** is the format used to efficiently store and organise a collection of re
structures are chosen in computer programming for specific tasks; commonly used str

Arrays

Single-dimensional arrays

ⓘ **Array** data structures are made up of a list of data elements that are the same data type and size.

Arrays can be one-dimensional, similar to a list as shown below:

The 'Subject' one-dimensional array has 6 elements;
Note that for most programming languages the indexes are addressed between 0 and 5 rather than between 1 and 6:
So Subject[3] = "Computing"

Index	Subject
0	English
1	Maths
2	Physics
3	Computing
4	Chemistry
5	French

Array Declaration & Assignments:

```
string Subject[6]
Subject[0] = "English";
Subject[1] = "Maths";
Subject[2] = "Physics";
```

Pseudocode Basic Examples:

Output to printer (Physics):

```
Output (Subject[2]);
```

Clear all string data from the array:

```
For Index = 0 To 5
   Subject[Index] = "";
Next
```

Array elements are often assigned in repetitive program code.

Mult

Arrays can also be two-
a matrix).

The StudentTest array l
students (rows) and 4 s

Rows = Students

	C
0	3
1	1
2	6
3	4
4	5

Array Declaration:

```
int StudentTest
```

Pseudocode Basic Exa

Output to printer (56):

```
Output (Student
```

Set all elements of the

```
For Student = 0
    For Score =
        Student
    Next
Next
```

2.1 – Progress Check

1. Describe the term 'data structure' (2 marks)
2. Describe the array data structure (1 mark)
3. Explain the difference between a one-dimensional array and a two-dimen

Fields, records and files

ⓘ **Text files** store data in humanly-readable format (usually ASCII) using a text editor.

Read/write to text files Text files normally consist of a series of characters, separated by an end of line character, such as: \n. The basic commands used for reading and writing to text files are shown on the right. The commands shown are based on Python2, but different commands exist for different programming languages.	Create a file `file = c` Add a line of text `f` Close the file `file.clc` Open a file to read `file = c` Read 1st line of file `p` Read all lines of file `p` Append a file `file = c`

ⓘ **Binary files** are not humanly readable and must be interpreted by the computer program or hardware.

```
0000    FF D8
0010    08 00
```

Read/write to binary files The data returned when reading a binary file is presented in byte strings and not in readable text strings. Some basic commands used for reading and writing to binary files are shown on the right. The commands shown are based on Python2, but different commands exist for different programming languages.	Note the modes are slightly different t becomes 'rb', etc. Open a binary file to read `file` Open a binary file to write `file` Append a binary file `file` Write bytes into a binary file using hexadecimal `file.`

2.1 – Progress Check

4. Explain the difference between a text file and a binary file (2 marks)

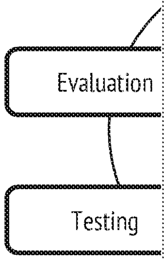
TOPIC 3 – SOFTWARE DEVELOPI

3.1 ASPECTS OF SOFTWARE DEVELOPMENT

The main aspects of software development are shown in the cyclical diagram on the right

There are various methods that can be used to develop software, such as Prototyping, Waterfall method or Spiral method. Whichever method is adopted, they will all contain the following phases:

1 Analysis
2 Design
3 Implementation
4 Testing
5 Review & Maintenance



The approach is termed the software development life cycle as all software, no matter how well created, will eventually be replaced or upgraded and the process will be repeated.

Analysis

The analysis phase is instigated when the organisation decides there is a need to change its system; this change is normally brought about by the need for new functions within the organisation or to make use of technical developments.

Part of the analysis phase is where the organisation decides whether or not it is feasible or possible to create the project. They consider the objectives of the new system, a range of alternative solutions and whether they have sufficient finance and exp

Once it has been decided that the new system is feasible, a detailed **list of requirements**

The information needed about the current system and the main issues involved can be ob

- **Interviews** with members of staff to determine what is problems with the current system.
- **Questionnaires** it is less time-consuming and easier to get t using questionnaires in preference to interv
- **Observation** the systems analyst will observe working pr can be improved.
- **Inspection of documentation** the systems analyst will examine current dc working knowledge of the current systems v requirements of the new system.

The data model is created based on the principles of abstraction and finally the **performa** evaluation part of the life cycle.

Design

During the analysis phase the file structure, **inputs**, **outputs** and **processing** needed by the system were defined, so in the design phase a detailed physical design needs to be created in preparation for the implementation phase.

This will normally include:

- Data structures for the data model
- Diagrams of the system
- Modular design method
- User interface – input and data capture methods
- Description of processing
- Design and format of s
- File structure
- Hardware and software
- System test plan

The design needs to be sufficiently detailed for the specialists who will implement the sy detailed in the original analysis.

Implementation

The implementation and coding phase is where a working version of the design is produc dependent upon the type of solution to be produced, but in all cases the data model and form of data structures and code; the code generation can begin once the software design

Code generation is relatively straightforward if software design has been completed to a

Software modules are broken down into logical units, which are stand-alone tested (unit t to ensure error-free code prior to integration with other software modules.

Testing

The test plan created in the design phase is used to ensure correct operation of the software stage is for the software modules that have been tested in a stand-alone way to be integrate

The results of the testing are printed out and delivered to the customer, to show that the

Testing should be documented so that there is some evidence that it worked correctly. If the software, the test plan can be used again to ensure that everything is still working sat

When testing, it is important to select data that works for erroneous as well as normal inp

- **Normal** data – chosen data values must be representative of normal inputs
- **Extreme** or **boundary** data – must be acceptable input data that is on the extreme lir
- **Erroneous** data – data that is outside of the normal input range or data that is in an i

Test Plan example – to ensure product price in the range of £10 to £500

Test	Test Title	Data	Expected Results
1	Price entry (normal)	£29.99	Accepted
2	Price entry (extreme data)	£499.99	Accepted
3	Price entry (extreme data)	£500.00	Accepted
4	Price entry (erroneous data)	£500.01	Rejected as outside range; displays error
5	Price entry (erroneous data)	£3.50	Rejected as outside range; displays error
6	Price entry (erroneous data)	"ABC"	Rejected as incorrect format; displays err

Review & Maintenance

Once the system has been installed and is fully operational, the performance criteria liste evaluate or measure system performance. The effectiveness of the solution will be determ users can operate the system.

There are many criteria that could be used when evaluating a system; for example:

- Requirements – does the system perform all of the requirements specified?
- Return on Investment – does the benefit of the system outweigh the investment c
- Usability – is the client able to use the system without too much support?
- Reliability – the time between failures and how often the system fails.
- Efficiency – does the system perform the task quickly?

Corrective maintenance involves any important improvements or error fixes that are re

In the future the software development cycle may be repeated as new features are ac new requirements will be catered for using **adaptive maintenance**.

3.1 – Progress Check

1. Explain the following terms used in the testing phase of the system life cy
   (a) Test plan (2 marks) (b) Normal data (2 marks) (c) Extreme data (2
2. Describe the types of software maintenance in the system life cycle (3 ma
3. State how the following methods can be used during the analysis phase
   (a) Interviews (1 mark) (b) Questionnaires (1 mark)
   (c) Observation (1 mark) (d) Inspection of documentation (1 mark)

4.1 ABSTRACTION AND AUTOMATION

Problem-solving

Problem-solving involves reaching a desired outcome from an initial situation.

The first stage in problem-solving is to gain a good understanding of the problem that is to be solved.

The next stage is to create a definition of the problem, which involves the following components:

- The initial situation
- The desired outcome
- The resources available to solve the prob
- Responsibility for planning and implemer

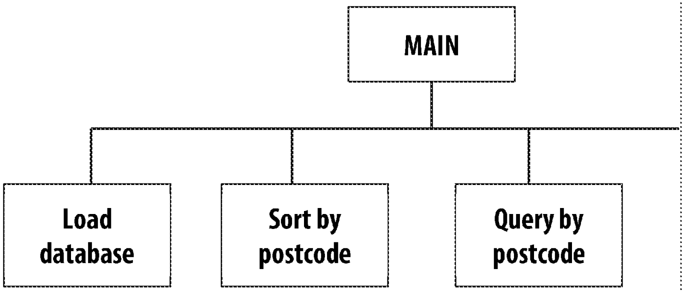
Planning a solution involves deciding upon a plan of action or strategy to solve the probler solve the problem on paper using a range of assumptions to simplify the problem. Most pr series of smaller problems or sub-problems that are easier to solve, which is known as a to

ⓘ **Top-down design** or stepwise refinement is used to plan a solution based on a top-do

Using this approach a complex problem can be solved by breaking it down into a series of : down further into even smaller steps. The smaller the steps the easier it is to both underst and eventually the whole problem.

Example: Addressing labels
A travel agent wishes to send a brochure to each of their customers in a certain area, basec

Solution: Load the database and create a query for the customer file based on a selected p with the relevant address. The hierarchy chart below shows the top level of the design.



Each of the steps can then be broken down further to add more detail for the solution, so f function includes additional detail:

1. Switch on printer
2. Load envelopes into printer tray
3. Run postcode query on database
4. Select print query from database

The other functions: 'load database', 'sort by postcode' and query by postcode' can also be l
The top-down design should include enough steps for the designer to create the algor then further steps need to be added, hence the term 'stepwise refinement'.

There are several **advantages of top-down design** as listed below:

- Breaking the problem into parts helps to clarify exactly what needs to be achieved
- Each stage of the refinement process creates smaller sub-problems that are easier t
- Some functions or parts of the solution might be reusable
- Breaking a design into parts allows more than one person to work on the solution.

4.1 – Progress Check

1. Explain the term 'problem-solving' (2 marks)
2. Describe top-down design (4 marks)
3. Describe the advantages of using top-down design (4 marks)

Following and writing algorithms

ⓘ An **algorithm** is a step-by-step approach to solving a problem. It is normally written i any particular programming language.

ⓘ An algorithm can be expressed using **pseudocode**; this is a written list of steps that a connected to any particular programming language.

Note that pseudocode can be written in many styles, but it should be in sufficient detail t based on it. In the examples below, comments are made using the '**#**' Symbol; for exampl

The solutions to simple problems can be written in pseudocode using one or more of thes selection, and iterations (see Topic 1 for information and examples).

4.1 – Progress Check

4. Describe the term 'algorithm' (2 marks)
5. Create algorithms in pseudocode to solve the following problems:
   (a) Calculate the area of a square from a value entered by the user (3 ma
   (b) Calculate the area of all the squares with values 2, 3, 4 and 5 (5 mark

Hand-trace algorithms

Hand tracing is a form of dry run testing where a program is tested and variables are reco table can be used which contains columns for the expected answers. The variables used i error it can be detected and the code corrected.

The simple example below demonstrates the dry run process to calculate the mileage charge for a car hire company.

```
# Luxury Car Hire Mileage Solution – the car hire firm makes
a charge for hiring the car and also charges for the mileage
driven by the hirer.

START
   # Data declarations
   INTEGER Mileage
   FLOAT Cost

   # Input / Output
   OUTPUT "Enter Total Mileage used"
   Mileage ← USERINPUT

   # Car hire mileage cost calculation
   IF (Mileage < 50) THEN
      # First 50 miles charged at 5p per mile
      Cost ← Mileage * 0.05
   ELSE IF (Mileage < 200) THEN
      # Next 150 miles charged at 25p per mile
      Cost ← (Mileage – 50) * 0.25 + (50 * 0.05)
   ELSE
      # Mileage above 200 charged at 40p per mile
      Cost ← (Mileage – 200) * 0.40 + (150 * 0.25) + (50 * 0.05)
   END IF
      OUTPUT "Mileage Cost for car hire = £" Cost
END
```

Test No.	Mileage	Cost
1	0	
2	45	= 4
3	160	= 50 *0.05 + 90
4	280	= 50 *0.05 + 150 *0.25 + 80

Final comments

It can be seen from the test results that the program above works as expected.

The program code is efficient making use of a nested IF-THEN-ELSE statement; it is easy t indentation and a range of comments.

Test data was chosen logically to ensure that all parts of the code were accessed.

User feedback would be necessary if the program had been created for a real client.

4.1 – Progress Check

6. Hand-trace the following algorithm using the data in the array (4 marks):

```
Total ← 0
FOR X = 1 TO 5
      Total ← Total + Array[X]
      Average ← Total / X
      Output "X", "Average"
ENDFOR
```

Element	Data
1	9
2	7
3	1
4	5
5	6
6	2
7	6
8	1
9	1
10	4

Pseudocode to program code

Pseudocode is not program-language-specific and so cannot be understood by a programr code written to aid understanding of a problem.

Well-written pseudocode can be converted into the program language of your choice; for produced in any of the following languages: C#, Java, Pascal / Delphi, Python, VB6 or VB.N

It is suggested that pseudocode should include the following to ensure it is straightforwa programming languages.

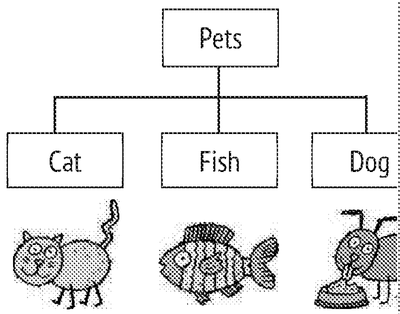
- The use of meaningful variable names
- Naming procedures and functions using understandable titles
- Structure pseudocode making use of white space and indentations to aid understa

Abstraction

ⓘ **Abstraction** is the process of including only the important features when solving a problem; this reduces the complexity of the system by removing unnecessary details.

The benefits of using abstraction techniques are that it is easier for the programmer or user to view, to modify and to maintain the solution as they are not distracted by excessive detail which is hidden from them.

The hierarchy diagram below is an **abstra by generalisation** for pets



Information hiding

ⓘ **Information hiding** is the principle that the details of the implementation of a class o accessible by the user; the user simply needs the essential details of how to initialise

Example – car manufacturers break the process down into a series of modules. Some elec model as well as the luxury model. Teams work on particular modules and the detail of th is hidden from them.

So the type of entertainment system installed (basic or luxury) is hidden from the team re (speakers). This approach creates flexibility where the car manufacturer is able to use mar cars it produces.

Procedural abstraction

ⓘ **Procedural abstraction** is based on programming where large programs are written b programs; this approach applies to any programming language although the units are methods in Java and functions in C and many other languages.

The procedure is a named block of code, where the actual data and values used in the cor use of local variables declared within a procedure helps to ensure that the block of code c be used by the operator without an understanding of its process.

Note that the actual computational method used is not hidden with procedural abstractio aid understanding.

Functional abstraction

ⓘ In **function abstraction** the exact computational method used is hidden, unlike proce

The use of built-in or library functions is an example of functional abstraction, the user si returned with no knowledge of the internal code within the function.

Example – using a built-in square root function: **SQRT(16)** will return the value 4, so **x** =

Data abstraction

ⓘ **Data abstraction** is where the details of how a compound data object is constructed a used.

Therefore, the primitive data objects that make up a user-defined data structure are hidde user-defined data structures, such as Records in Pascal and Structs in Java.

Problem abstraction/reduction

ⓘ **Problem abstraction** is where the details of the problem are successively removed or represented in a way that is straightforward to solve.

Once the unnecessary details in the problem are removed, then the problem can be more possible that the problem has already been solved or a similar problem has been solved b simplification approach.

Decomposition

ⓘ **Procedural decomposition** is the process of breaking down a problem into a series of s where each sub-problem achieves an identifiable task. In some cases the sub-problem further subdivided into smaller identifiable tasks.

Problems that are not decomposed are more difficult to solve since dealing with several s at the same time is more challenging than decomposing the problem and solving one sub-problem at a time.

Composition

ⓘ **Composition** is the opposite of decomposition; it is the process of creating a system by combining the tasks identified in the decomposition abstraction.

The process involves:

- Writing procedures for each of the tasks and sub-tasks identified in the decomposition
- Linking these procedures to create compound procedures
- Creating the necessary data structures to support the compound procedures

Automation

This automation process is based on the following:

- **Creation of algorithms** – which includes the breaking up of the problem into sub-problems and the listing of the steps needed to solve each sub-problem.
- **Implementing the algorithms in program code** – which includes conversion betw pseudocode algorithm and the instructions of the programming language chosen.
- **Implementing the models in data structures** – chosen data structures should be s for specific model; commonly used data structures include arrays, files and record/
- **Executing the code** – once the code has been created it should be executed to en runs and then tested/debugged to ensure it operates as expected.

It is essential that sufficient detail is included from the abstraction process to create a mo that can solve the problem to the required level of accuracy.

4.1 – Progress Check

7. Explain the difference between procedural and functional abstraction (6 m

4.2 FINITE STATE MACHINES (FSM)

Finite state machines (FSMs)

ⓘ A **finite state machine (FSM)** is an abstract machine that can be in any one of a finite one state at a time and a transition to a new state is triggered by an event.

FSMs are useful in that they can recognise logical sequences and they are used to model a lights, combination locks, electronic design automation and lifts.

ⓘ A finite state machine with no output is called **finite state automata (FSA);** it does no sequence for the final (or goal) state.

ⓘ **State transition diagrams** are used to describe an FSM in a graphical format, where e transition is connected to a circle by an arrowed line with a description of the input tl

In a finite state machine with no output, the final (or goal) state is indicated by a double c

ⓘ **State transition tables** are a method used to record all the states and transitions pos:

The state transition diagram for a table lamp push button switch is shown on the right.

Where:

- S_0 = state 0 = Lamp off
- S_1 = state 1 = Lamp illuminated

The user simply presses the switch to turn the light from on to off or vice versa.

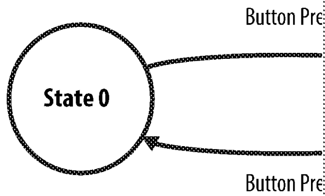


Table lamp push button st

State transition table for a table lamp push button switch is shown on the right.
The current state to the next state toggles the lamp between illuminated and off.

Input	Currei
Table lamp switch pressed	Lamp ill
Table lamp switch pressed	Lan

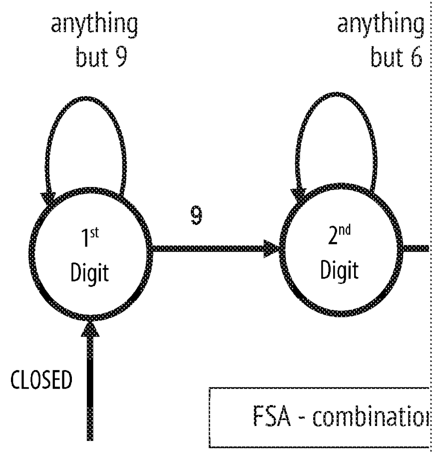
Combination lock – FSA (FSM with no output) example:

The code for combination lock is the sequence 963.

The initial state is shown as closed (or locked) and indicated by the lined arrow on the left.

To open (or unlock) the combination, the sequence 9 must be entered into the first digit, then 6 into the second digit and finally 3 into the third digit. The combination lock opens only when all three digits have been correctly entered.

The final or goal state is indicated by the double circle on the right side of the diagram.



FSA - combinatio

Decision table for combination lock

ⓘ **Decision tables** can be used to model logic sequences in a compact way.

As shown in the state transition diagram above, the combination lock is only open when all three digits have been correctly entered for the combination code 963.

Condition	Condit	
1st Digit = 9	Y	Y
2nd Digit = 6	Y	Y
3rd Digit = 3	Y	N
Final / Goal State	Lock Open	

4.2 – Progress Check

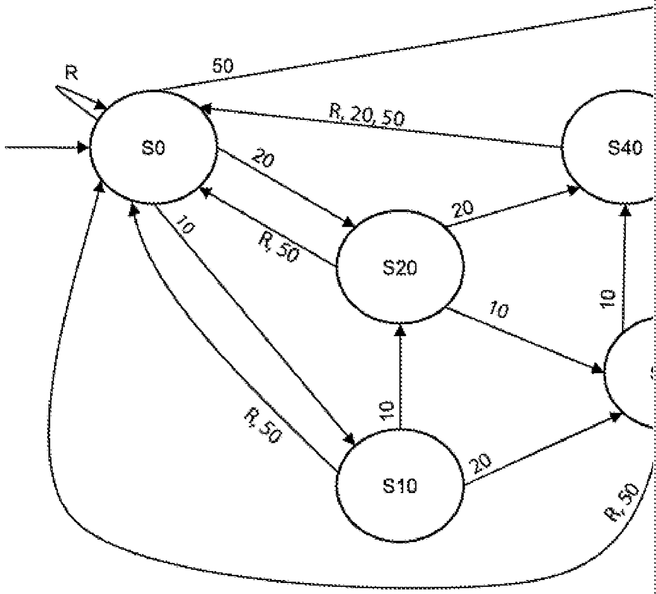
8. Explain the terms 'finite state machine' and 'state transition diagrams'
9. The state transition diagram of a finite state machine (FSM) used to c

   The vending machine dispenses a drink when a customer has inserted

   A transaction is cancelled and coins returned to the customer if more button (R) is pressed.

   The vending machine accepts 10, 20 and 50 pence coins. Only one ty

   The only acceptable inputs for the FSM are 10, 20, 50 and R.



   (a) Complete the state transition table for this finite state machine (

Original state	Input	New state
S0	10	S10
S0		
S0		
S0		

   (b) There are different ways that a customer can provide exactly thre machine dispensing a drink. Three possible permutations are '20,

   List four other possible permutations of exactly three inputs that above (4 marks)

ANSWERS

Topic 1 – Programming

1.1 (a) Real/Float – a real number contains a decimal point, for example 65.25 (1); th
be varied, hence the term 'float' (1)

(b) Character – a character represents a single alphanumeric item of data, for exa
number, letter or other character (1).

(c) Array – Array data structures are made up of list of date elements that are the
example: Char Array[6] = {'C', 'O', 'D', 'I', 'N', 'G'} (1);
Note: accept examples in a range of formats.

1.2 (a) Subroutine is a set of instructions (1) to perform a certain task (1). It can be st
times within a computer program (1).

(b) Procedure is a subroutine (1) that is called to perform a task (1). It may or may

(c) Function is a series of instructions (1) to perform a task (1). When called it exe
returns a value (1).

(d) Iteration or repetition is where a program executes a statement or statement
some logical condition is satisfied (1).

(e) A selection structure is where the program executes different actions (1) or st
of a comparison (1).

1.3. Definite iteration is where the number of iterations that will take place is known be
main body of the loop (1). A typical example of definite iteration is where a loop is
them all out (1). Indefinite iteration is where the number of iterations that will take
the loop, but is determined by when a logical condition becomes true (1); this depe
loop (1).

1.4 (a) The process of rounding is to replace a number with an approximate value usi
be rounded to 34.57 which is rounded to two decimal places (1). Truncation is
of digits to the right of the decimal point (1). So 34.5674 can be truncated to 3
decimal places (1).

(b) 11 MOD 3 = 2 (1 mark)

(c) 3**2 = 9 (1 mark)

1.5 Truth table for the exclusive-or function using variable A and B (1 mark)

A	B	A XOR B
0	0	0
1	0	1
0	1	1
1	1	0

1.6 Constants are used where data that is used in a computer program is preset and dc
are used in computer programs to store data that may change when the program is

1.7 The concatenation operation is used to join two strings together where the combin
example, "Computer " + "Science"; returns 'Computer Science' (1).

1.8 Where local variables are declared and used in a subroutine they are only in existen
executed and are only accessible or in scope within that subroutine (1). It is good p
than global variables in subroutines and functions as it is easier to trace the conten
program and the subroutine retains modularity where only parameters and not glo
execution (1).

1.9 The advantages of using a structured approach to programming are:

1. Breaking down large programming tasks into manageable subtasks means tha
several programmers, which saves development time (1).
2. Program test and debug time is reduced where modularity helps to reduce the
correcting the errors that may occur (1).
3. Programs are easier to understand and, therefore, easier to maintain; also a n
introduction of an additional module (1).

Topic 2 – Data structures

2.1 A data structure is the format used to efficiently store (1) and organise a collection

2.2 Array data structures are made up of a list of date elements that are the same data

2.3 A one-dimensional array is a list of data elements (1) whereas a two-dimensional ai elements (1).

2.4 Text files store ACSII data and so they are humanly readable using a text editor (1) \ records data that contains unprintable characters so cannot be read using a text ed

Topic 3 – Software development

3.1 (a) A test plan is a table of tests (1) that are to be carried out with the test data tl
(b) Normal data is chosen data values (1), must be representative of normal inpu
(c) Extreme data must be acceptable input data (1) that is on the extreme limits c
(d) Erroneous data is data that is outside of the normal input range (1) or data th expected (1).

3.2 Corrective maintenance is used to fix any errors that occur in the software (1). New using perfective maintenance (1) and new requirements will be catered for using a

3.3 (a) Interviews – with members of staff to determine what is required from the ne current system (1).
(b) Questionnaires – it is less time-consuming and easier to get the input from m questionnaires in preference to interviewing them all (1).
(c) Observation – the systems analyst will observe working practices to determin (1).
(d) Inspection of documentation – the systems analyst will examine current docu working knowledge of the current systems which can help to dictate the requi

Topic 4 – Theory of computation

4.1 Problem-solving involves reaching a desired outcome (1) from an initial situation (1

4.2 Top-down design or stepwise refinement is used to plan a solution based on a top- complex problem can be solved by breaking it down into a series of small steps (1), further into even smaller steps (1). The smaller the steps the easier it is to both unc sub-problems (1) and eventually the whole problem.

4.3 The advantages of using top-down design are:
(a) Breaking the problem into parts helps to clarify exactly what needs to be achi
(b) Each stage of the refinement process creates smaller sub-problems that are e
(c) Some functions or parts of the solution might be reusable (1).
(d) Breaking design into parts allows more than one person to work on the soluti

4.4 An algorithm is a step-by-step approach to solving a problem (1). It is normally writ independent of any particular programming language (1).

4.5 (a) Algorithm to calculate the area of a square from a value entered by the user (

```
INPUT Length
Area ← Length * Length
OUTPUT "Square"; Area
```

(b) Calculate the area of all the squares with values 2, 3, 4 and 5 (5 marks)

```
FOR Length = 2 TO 5
      Area ←Length * Length
      OUTPUT "Square"; Area
ENDFOR
```

4.6 Hand-tracing (4 marks)

Element	Data	X	Total	Average
1	9	1	9	9
2	7	2	16	8
3	2	3	18	6
4	10	4	28	7
5	2	5	30	6
6	2			
7	6			
8	1			
9	1			
10	4			

4.7 Procedural abstraction is based on programming where large programs are written programs (1). The procedure is a named block of code, where the actual data and v method are abstracted (1). The use of local variables declared within a procedure h can be treated as a 'black box' that can be used by the operator without an underst the actual computational method used is not hidden with procedural abstraction, b aid understanding (1).

In the case of functional abstraction the exact computational method used is hidde The use of built-in or library functions is an example of functional abstraction, the u value is returned with no knowledge of the internal code within the function (1).

4.8 A finite state machine (FSM) is an abstract machine that can be in any one of a finit only be in one state at a time and a transition to a new state is triggered by an ever

State transition diagrams are used to describe an FSM in a graphical format (1), wher each transition is connected to a circle by an arrowed line with a description of the in

4.9 (a) Complete the state transition table for this finite state machine (3 marks):

Original state	Input	New state
S0	10	S10
S0	20	S20
S0	50	S50
S0	R	S0

(b) Any four from the following permutations (4 marks)
`20,20,10;      R,R,50;      10,20,20;      20,50,50;      20,R,50`