GCSE

Computer Science

J277/02: Computational thinking, algorithms and programming

General Certificate of Secondary Education

Mark Scheme for June 2025

OCR (Oxford Cambridge and RSA) is a leading UK awarding body, providing a wide range of qualifications to meet the needs of candidates of all ages and abilities. OCR qualifications include AS/A Levels, Diplomas, GCSEs, Cambridge Nationals, Cambridge Technicals, Functional Skills, Key Skills, Entry Level qualifications, NVQs and vocational qualifications in areas such as IT, business, languages, teaching/training, administration and secretarial skills.

It is also responsible for developing new specifications to meet national requirements and the needs of students and teachers. OCR is a not-for-profit organisation; any surplus made is invested back into the establishment to help towards the development of qualifications and support, which keep pace with the changing needs of today's society.

This mark scheme is published as an aid to teachers and students, to indicate the requirements of the examination. It shows the basis on which marks were awarded by examiners. It does not indicate the details of the discussions which took place at an examiners' meeting before marking commenced.

All examiners are instructed that alternative correct answers and unexpected approaches in candidates' scripts must be given marks that fairly reflect the relevant knowledge and skills demonstrated.

Mark schemes should be read in conjunction with the published question papers and the report on the examination.

MARKING INSTRUCTIONS

PREPARATION FOR MARKING
RM ASSESSOR

1. Make sure that you have accessed and completed the relevant training packages for on-screen marking: *RM Assessor Online Training: OCR Essential Guide to Marking*.

2. Make sure that you have read and understood the mark scheme and the question paper for this unit. These are available in RM Assessor

3. Log-in to RM Assessor and mark the **required number** of practice responses ("scripts") and the **required number** of standardisation responses.

MARKING

1. Mark strictly to the mark scheme.

2. Marks awarded must relate directly to the marking criteria.

3. The schedule of dates is very important. It is essential that you meet the RM Assessor 50% and 100% (traditional 40% Batch 1 and 100% Batch 2) deadlines. If you experience problems, you must contact your Team Leader (Supervisor) without delay.

4. If you are in any doubt about applying the mark scheme, consult your Team Leader by telephone or the RM Assessor messaging system, or by email.

5. **Crossed Out Responses**
   Where a candidate has crossed out a response and provided a clear alternative then the crossed out response is not marked. Where no alternative response has been provided, examiners may give candidates the benefit of the doubt and mark the crossed out response where legible.

   Multiple Choice Question Responses
   When a multiple choice question has only a single, correct response and a candidate provides two responses (even if one of these responses is correct), then no mark should be awarded (as it is not possible to determine which was the first response selected by the candidate).
   When a question requires candidates to select more than one option/multiple options, then local marking arrangements need to ensure consistency of approach.

Contradictory Responses
When a candidate provides contradictory responses, then no mark should be awarded, even if one of the answers is correct.

Short Answer Questions (requiring only a list by way of a response, usually worth only **one mark per response**)
Where candidates are required to provide a set number of short answer responses then only the set number of responses should be marked. The response space should be marked from left to right on each line and then line by line until the required number of responses have been considered. The remaining responses should not then be marked. Examiners will have to apply judgement as to whether a 'second response' on a line is a development of the 'first response', rather than a separate, discrete response. *(The underlying assumption is that the candidate is attempting to hedge their bets and therefore getting undue benefit rather than engaging with the question and giving the most relevant/correct responses.)*

Short Answer Questions (requiring a more developed response, worth **two or more marks**)
If the candidates are required to provide a description of, say, three items or factors and four items or factors are provided, then mark on a similar basis – that is downwards (as it is unlikely in this situation that a candidate will provide more than one response in each section of the response space.)

Longer Answer Questions (requiring a developed response)
Where candidates have provided two (or more) responses to a medium or high tariff question which only required a single (developed) response and not crossed out the first response, then only the first response should be marked. Examiners will need to apply professional judgement as to whether the second (or a subsequent) response is a 'new start' or simply a poorly expressed continuation of the first response.

6. Always check the pages (and additional objects if present) at the end of the response in case any answers have been continued there. If the candidate has continued an answer there, then add a tick to confirm that the work has been seen.

7. There is a NR (**No Response**) option. Award NR (No Response):

   - if there is nothing written at all in the answer space
   - OR if there is a comment which does not in any way relate to the question (e.g., 'can't do', 'don't know')
   - OR if there is a mark (e.g., a dash, a question mark) which is not an attempt at the question.

   Note: Award 0 marks – for an attempt that earns no credit (including copying out the question).

8. The RM Assessor **comments box** is used by your team leader to explain the marking of the practice responses. Please refer to these comments when checking your practice responses. **Do not use the comments box for any other reason.**
   If you have any questions or comments for your team leader, use the phone, the RM Assessor messaging system, or e-mail.

9. Assistant Examiners will send a brief report on the performance of candidates to their Team Leader (Supervisor) via email by the end of the marking period. The report should contain notes on particular strengths displayed as well as common errors or weaknesses. Constructive criticism of the question paper/mark scheme is also appreciated.

10. **Annotations**

Annotation	Meaning
	Omission mark
	Benefit of doubt (must be accompanied with a tick)
	Cross
	Follow through (must be accompanied with a tick)
	Not answered question
	Benefit of doubt not given
	Repeat
	Tick
	Too vague
	Blank pages, pages with no annotation, no attempt to answer the question, page seen on QER

11. **Subject Specific Marking Instructions**

COMPONENT 2 SECTION B SYNTAX GUIDANCE

In Section B, certain questions require candidates to answer in either the OCR Exam Reference Language or the high-level programming language they are familiar with. The information in this section provides generic guidelines in relation to the marking of these questions. Where a response requires an answer in OCR Exam Reference Language or a high-level programming language, a candidate's level of precision will be assessed. These questions are designed to test both a candidate's programming logic and understanding of core programming structures.

Marks will be given for correctly using syntax to represent core programming constructs which are common across all programming languages. The construct must be present in a recognisable format in a candidate's answer.

Where the response requires a candidate to respond using the OCR Exam Reference Language or a high-level programming language, answers written in pseudocode, natural English or bullet points must **not** be awarded marks.

The guidance below covers the elements of each core construct. As guidance, several examples are provided for each. These examples are not exclusive but do present a variety of acceptable ways taken from a range of different languages.

Concept		Examiner Guidance – what is required
Commenting		

`//`	`//This function squares a number`
	`function squared(number)`
	`squared = number^2`
	`return squared`
	`endfunction`
	`//End of function`
		• Other examples allowable, e.g.:
		○ # this is a comment
		○ /* this is another comment */

Variables		

`=`
`const`
`global`
	`x = 3`
	`name = "Louise"`
	`const vat = 0.2`
	`global userID = "Cust001"`
		• Variables and constants are assigned using the = operator
		• Constants are assigned using the `const` keyword (or similar)
		• Identifiers should **not** have clear spaces within them or start with numbers
		• String values must use quotation marks (or equivalent)
		• Assignment must use =, :=, ← (or a suitable alternative)
		• variable identifier must be on the left and the value to be assigned on the right
		• Variables and constants are declared the first time a value is assigned. They assume the data type of the value they are given
		• Variables and constants that are declared inside a function or procedure are local to that subroutine
		• Variables in the main program can be made global with the keyword global

Input/Output		

`input(…)`
 	
`print(…)`
	`myName = input("Please enter a name")`
	
	`print("My name is Noni")`
		• For input, a suitable command word for input and a variable identifier to assign data to (if required)
		e.g.
		`INPUT identifier`
		`identifier = INPUT`

	`print(myArray[2,3])`	• For output, a command word for output (e.g. output, print, cout) • Data to be output. If this is a string then quotation marks (or equivalent) are required • If multiple items are to output, a suitable symbol for concatenation (such as +, & or a dot) or a separator (such as a comma) is required.
Casting		

`str()`
 	
`int()`
 	
`float()`
 	
`real()`
 	
`bool()`
	`str(345)`
	
	`int("3")`
	
	`float("4.52")`
	
	`real("4.52")`
	
	`bool("True")`
		• Variables can be typecast using the int str and float functions

Iteration		

`for … to …`
 	
`next …`
 	
`for … to … step …`
 	
`next …`
	`for i=0 to 9`
	`  print("Loop")`
	`next i`
	
	`for i=2 to 10 step 2`
	`  print(i)`
	`next i`
	
	`for i=10 to 0 step -1`
	`  print(i)`
	`next i`
		• `for` keyword
		• …with counter variable
		• Identification of number of times to iterate
		• Clear identification of which section of code will be repeated (e.g. using indentation, next keyword or equivalent, {braces})

`while …`
 	
`endwhile`
 	
`do`
 	
`until …`
	`while answer != "Correct"`
	`  answer = input("New answer")`
	`endwhile`
	
	`do`
	`  answer = input("New answer")`
	`until answer == "Correct"`
		• `While / do..until` key words or equivalent
		• …with logical comparison
		• identification of which section of code will be repeated (e.g. using indentation, endwhile/until keyword or equivalent, braces).

Selection		

`if … then`
`  elseif … then`
`  else`
`  endif`
	`if answer == "Yes" then`
	`  print("Correct")`
	`elseif answer == "No" then`
	`  print("Wrong")`
	`else`
	`  print("Error")`
	`endif`
		• `if` key word followed by logical comparison
		• key word for `elseif` or equivalent followed by logical comparison
		• key word for `else` or equivalent with no comparison
		• clear identification of which section of code will be executed depending upon decision

`switch … :`
`  case … :`
`  case … :`
`  default:`
`endswitch`
	`switch day :`
	`  case "Sat":`
	`  print("Saturday")`
	`  case "Sun":`
	`  print("Sunday")`
	`  default:`
	`  print("Weekday")`
	`endswitch`
		• May be referred to differently in some languages. The format to the left will be used in all questions
		• `switch/select` key word or equivalent followed by variable/ value being checked
		• key word for each case followed by variable/ value to compare to
		• key word for default case (last option)
		• clear identification of which section of code will be executed depending upon decision

String handling/operations		
`.length`	`subject = "ComputerScience"` `subject.length` **gives** `the value 15`	• Suitable key word to indicate length and string identifier e.g. **`len(string)`**

`.substring(x , i)`
`.left(i)`
`.right(i)`
	`subject.substring(3,5)` **returns** `"puter"`
	`subject.left(4)` **returns** `"Comp"`
	`subject.right(3)` **returns** `"nce"`
		• Suitable string and characters required identified
		• Use of key words such as **`left`**, **`right`**, **`mid`**, etc, are all acceptable as long as these are precise
		• Treating a string as an array of characters is acceptable

`+` (concatenation)	`print(stringA + stringB)` `print("Hello, your name is: " + name)`	• Alternate symbol used indicate two strings or values are being concatenated is acceptable e.g. `stringA & stringB` or `stringA.stringB` • Use of comma e.g. `print(stringA, stringB)` is acceptable to output multiple values but examiners should be aware that this is not concatenation.
`.upper` `.lower`	`subject.upper` **gives** `"COMPUTERSCIENCE"` `subject.lower` **gives** `"computerscience"`	• Suitable key word to indicate string to be converted and whether this is to be converted to upper or lower case e.g. **`lower(stringname)`**
`ASC(…)` `CHR(…)`	`ASC ('A')` **returns** `65` (numerical) `CHR(97)` **returns** `'a'` (char)	• Suitable keyword to indicate conversion and whether this is to or from ASCII. Where converting from ASCII, an integer value must be given and where converting to ASCII, a single character must be given.

File handling		
`open(…)` `.close()`	`myFile = open("sample.txt")` `myFile.close()`	• `open` keyword (or equivalent) • `read` or `write` clearly identified • `write` or read keyword (or equivalent) • `close` file keyword (or equivalent) • `newFile` keyword (or equivalent)
`.readLine()`	`myFile.readLine()` **returns** the next line in the file	
`.writeLine(…)`	`myFile.writeLine("Add new line")`	

`.endOfFile()`	`while NOT myFile.endOfFile()`
	`  print(myFile.readLine())`
	`endwhile`	

`newFile()`	`newFile("myText.txt")`	

Arrays		
`array colours[…]`	`array colours[5]`	• Array identifier • Index number to be accessed in square brackets, rounded brackets or curly braces (all acceptable)
	`array colours = ["Blue", "Pink", "Green", "Yellow", "Red"]`	• Array identifier assigned to initial values in one step

`array gameboard[…,…] = …`
`names[…] = …`
`gameboard[…,…] = …`
	`array gameboard[8,8]`
	`names[3] = "Noni"`
	`gameboard[1,0] = "Pawn"`
		• For 2D arrays, the two indices should be given in one bracket separated by a comma **or** in two separate brackets, e.g.
		`gameboard[4,6]`
		`gameboard[4][6]`
		Where 2D arrays are represented by tables in a question, candidates are expected to use the **same** row/column or column/row format as given in the question. **This will always be given.**

Sub programs		

`procedure` *`name`* `(…)`
 	
`endprocedure`
	`procedure agePass()`
	`  print("You are old enough to ride")`
	`endprocedure`
	
	`procedure printName(name)`
	`  print(name)`
	`endprocedure`
	
	`procedure multiply(num1, num2)`
	`  print(num1 * num2)`
	`endprocedure`
		• `function` or `procedure` key word (or equivalent such as `def`)
		• … followed by identifier
		• Any parameters passed in are contained within brackets and come after identifier name
		• Clear identification of which section of code is contained within the subroutine (e.g. indentation, **`endsub`** key word, braces)

`procedure(`*`parameters`*`)`	`agePass()`
	
	`printName(parameter)`
	
	`multiply(parameter1, parameter2)`
		• **functions only**: a suitable method of returning a value (e.g. `return` keyword or assignment of value to function identifier as used in languages such as VB)
		
		e.g.
		`def newfunction(x,y)`
		`  total = x + y`
		`  newfunction = total`

`function` *`name`* `(…)`
`  …`
`  return …`
`endfunction`
	`function squared(number)`
	`  squared = number^2`
	`  return squared`
	`endfunction`	

`function(`*`parameters`*`)`	`print(squared(4))`
	
	`newValue = squared(4)`	

Random numbers		

`random(…,…)`	`myVariable = random(1,6)`
	
	`myVariable = random(-1.0,10.0)`
		• **`random`** key word (or equivalent)
		• identification of either smallest and largest number to be chosen **or** just largest number
		
		e.g.
		`randnumber(10)`
		`rand(1,6)`

Comparison operators			
`==`	Equal to	`<=`	Less than or equal to
`!=`	Not equal to	`>`	Greater than
`<`	Less than	`>=`	Greater than or equal to

Boolean operators	
`AND`	Logical AND
`OR`	Logical OR
`NOT`	Logical NOT

Arithmetic operators	
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`^`	Exponent
`/`	Division
`MOD`	Modulo
`DIV`	Quotient

- = or == are both acceptable for equal to.
- <> is acceptable for not equal to.
- Care must be taken by candidates to ensure that > and < are not mixed up.
- Candidates must understand that < and > are non-inclusive, so that <9 does not include 9. This is different than <=9 which is inclusive and therefore does include 9.
- Alternative symbols for arithmetic operators are acceptable where these appear in other high-level languages (such as % for MOD or ** for exponentiation).

- 6 x 5 is not an acceptable alternative for multiplication.
- Alternative logical operators are acceptable where these appear in other high-level languages (such as **`&&`** for **`AND`**).
- Alternative Arithmetic Operators may be used as well (such as % for modulus).
- Candidates must be aware that logical operators must be used correctly:

`if x > 0 AND x < 10` is logically correct.
`if x > 0 AND < 10` is **not** logically correct.

Question			Answer	Mark	Guidance
1	(a)		• Normal • Invalid / erroneous • Invalid / erroneous • Boundary // boundary **and** normal	4	Accept either boundary by itself for last row **or** <u>both</u> boundary and normal.
1	(b)	(i)	• `>= 1 // > 0` • **`num`** `<=100 //` **`num`** `< 101` • `else`	3	Allow check either way around (e.g. could check higher boundary first) for BP1 and BP2. BP2 must include reference to **`num`** variable. BP3 Do not allow `elseif / elif` unless full correct logical comparison included (e.g `elseif num <1 or num > 100`) Section A so allow “less than or equal to” etc.
1	(b)	(ii)	• `num`	1	Do not allow any other code around variable identifier. Ignore capitalisation. Ignore minor spelling errors.
1	(b)	(iii)	• `AND`	1	Do not allow any other code around operator. Ignore capitalisation. Ignore minor spelling errors.

Question			Answer	Mark	Guidance
2	(a)		• 2 • 6	2	
2	(b)	(i)	2 marks per construct max • FOR • Count controlled • repeats a specified number of times • WHILE • Condition controlled • repeats **while** a condition is **true** / **until** a condition is **false** // condition tested before code runs • DO UNTIL / REPEAT UNTIL • Condition controlled • repeats **until** a condition is **true** / **while** a condition is **false** // condition tested after code runs	4	Marks are independent, allow mark for description even if example/type of loop not given. Allow any 2 per construct. Allow suitable examples from other high-level languages. If WHILE and DO UNTIL given, allow "condition controlled" as description of both. Do not give "condition controlled" twice if no other differentiation between types of iteration. Do not allow "repeats infinitely" – this is only true if the condition is never met. Do not allow mismatched answers (e.g. "a while loop repeats until a condition is true")
2	(b)	(ii)	• Sequence • Selection	2	Allow slight alternatives (e.g. sequencing) Do not allow examples by themselves (e.g. IF) Allow branching as an alternative to selection.

Question			Answer	Mark	Guidance
2	(c)		1 mark for tool 1 mark for matching description e.g. • Text/code editor… • …allows program code to be **written** / **entered / changed** • …allows errors to be fixed • Debugger / error reporting / diagnostics… • …identifies / finds / checks for errors • …shows location / detail of errors • …suggests fixes • Run-time environment / output window… • …allows **program / code** to be run / executed • …shows output of the **program / code** • Translator / compiler / interpreter … • … convert to **low-level/machine code** • …allow program to be executed / run • …produce executable file (only for compiler) • …stops execution when error found (interpreter only) • Stepping … • … execute/run the program line by line	4	Allow other sensible names for features. Description must add more than is given in the identification of the feature to be awarded. For example, “keyword highlighting, highlights keywords” is 1 mark for the feature only. If compiler and interpreter given as two distinct features, allow both (with suitable descriptions). Do not allow translator and compiler/interpreter. **Description must match tool. If tool name wrong / missing, do not give description.** Allow sensible references to AI where appropriate. Allow other sensible features of an IDE (e.g. line numbering, **auto** indent, collapsed blocks, etc) with suitable description. Description must add (slightly) more than just repeating the feature name. Do not allow features of programming languages (comments, casting, etc). For text editor / error diagnostics / debugger, allow other sensible features listed as features in the mark scheme as description (e.g. “text editor does pretty printing”, “debugger does stepping”)

Question			Answer	Mark	Guidance
			• Variable watch… • … see the **contents/data** held in variables / see how (contents of) variables **change** • Break points … • … will allow the program to stop **at a chosen / set position** • Pretty printing // keyword highlighting… • … allows keywords / variables to be coloured / identified • Keyword completion // autocomplete // autocorrect // syntax suggestion… • …suggests/corrects code/syntax (when first part entered). • GUI (builder)… • …allows creation of user interface / buttons / by example		

Question			Answer	Mark	Guidance
3	(a)	(i)	• Split up into individual lists each of **one item** (divide stage) • **[2,7] [1,3] [6,9] [4,5]** (sorted lists of 2) • **[1, 2, 3, 7] [4, 5, 6, 9]** (sorted lists of 4) • **[1, 2, 3, 4, 5, 6, 7, 9]** after reasonable attempt at merging from **4 to 8**	4	**Do not allow merging out of order and <u>then</u> sorting for BP 3 or 4.** Splitting can be done in multiple stages or all at once. If no attempt at splitting up, assume initial list is already split up and mark BP2, 3 and 4. If splitting done incorrectly, do not give BP1 but allow access to further points. Do not accept sorted list for BP4 if no steps shown or wrong algorithm used. Brackets not needed around numbers – must be reasonably clear numbers are in 2s, 4s, etc. If candidate uses numbers other than the ones given in the question or sorts into descending order, penalise once then FT.
3	(b)		• Uses an array of numbers with a sorted part and unsorted part **(top box)** • Swaps them if they are in the incorrect order **(bottom box)** • Is declared with a fixed length **(top box)**	3	No mark if more than one box ticked per section.
3	(c)	(i)	• Linear search	1	Accept ”linear” by itself Do not allow “linear **sort**”

Question			Answer	Mark	Guidance
3	(c)	(ii)	• End of list reached // gets to last value (and not found) // checked every value (and value not found)	1	Do not accept "value not found / value not in list" by itself. Must have idea of getting to the end or having checked the entire list.

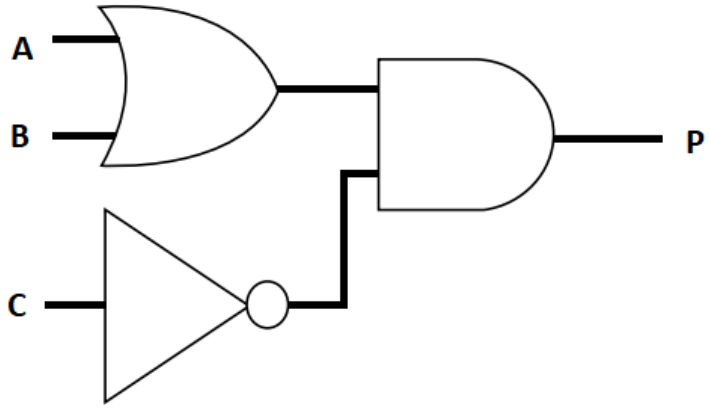
Question			Answer	Mark	Guidance

4	(a)		`myFile = open("data.txt")`
			
			`while NOT myFile.endOfFile()`
			
			`temp = myFile.readLine()`
			
			`if int(temp) != 0`
			
			`print(temp)`
			`endif`
			
			`endwhile`
			
			`myFile.close()`
				5	Accept `readLine` for BP2 / `close` for BP5
					Ignore spelling errors / case / minor differences as long as clear which option has been selected.
					
					Accept **`MyFile.readLine()`** as 3rd missing option (instead of `temp`) **only** if not used on line above.
					
					Only allow one choice per gap. Mark incorrect if multiple options chosen, e.g. `myFile.readLine(x)`

4	(b)		• Convert data to an **integer / int**	1	Do not allow generic “change to different data type”

<table>
<tr><th colspan="3">Question</th><th>Answer</th><th>Mark</th><th>Guidance</th></tr>
<tr><td>5</td><td>(a)</td><td></td><td>

Any use of an OR gate with 2 inputs
NOT C
Correctly joined by AND gate



</td><td>3</td><td>Gate diagrams must be correct. NOT must include circle; AND / OR must not include circle.

FT for BP3 if appropriate (e.g. missing/wrong NOT gate, wrong gate used instead of OR)

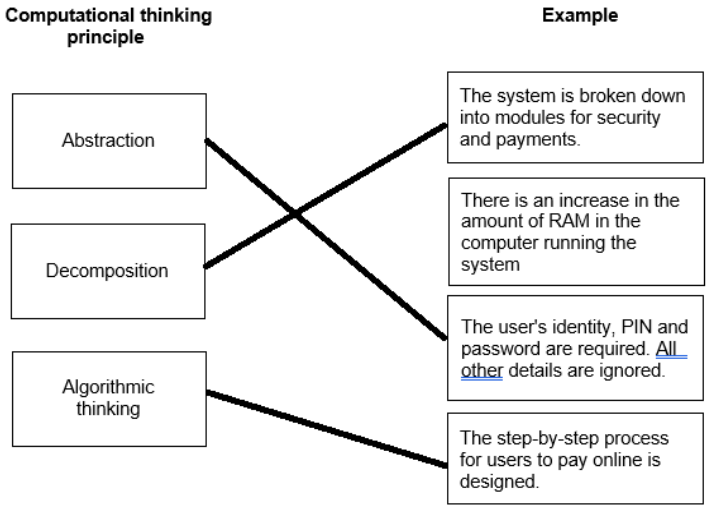
Max 2 if extra gates or system is not logically correct.</td></tr>
<tr><td>5</td><td>(b)</td><td></td><td>

All input possibilities for A, B (00, 01, 10, 11), does not have to be in order…
…Correct output for P (only 1 when A and B are both 1, 0 otherwise).

</td><td>2</td><td>P = A AND B

<table>
<tr><th>A</th><th>B</th><th>P</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>

Accept T/F, Yes/No, ticks/crosses, etc.</td></tr>
</table>

Question			Answer	Mark	Guidance
5	(c)		• Correct link for abstraction to third example • Correct link for decomposition to first example • Correct link for algorithmic thinking to bottom example	3	No mark for more than one connection from a box. 

Question			Answer	Mark	Guidance

5	(d)		• Input **and** storage/use of PIN
			• Attempt at selection / condition controlled loop…
			• …Correctly checks for 4 digits
			• …Correctly checks for **both** 1234, 4321
			• Correct output of "INVALID PIN" after sensible attempt at <u>all three</u> checks
			• Correct output of "VALID PIN" after sensible attempt at <u>all three</u> checks, **only** when PIN is fully valid and no contradictory output (e.g. printing "invalid PIN" as well)
			
			If diagram/flowchart given, ignore shapes if incorrect and mark text inside as algorithm.
				6	**<u>Example code</u>**
					`pin = input("Enter PIN")`
					`if pin.length == 4 and pin != "1234" and pin != "4321" then`
					`  print("VALID PIN")`
					`else:`
					`  print("INVALID PIN")`
					
					Note - | or || means OR in some languages. AND is &.
					
					BOD input and use/comparison of PIN as integer. No need for quotation marks around 1234 / 4321. Allow equivalent outputs.
					
					BP3 and 4 can be combined or as separate checks. **If combined, must logically work**. BP3 and 4 can be checks for positive or negative. Check must refer to variable for **both** comparisons.
					
					BP5 given if **invalid** output (once or multiple times) for all invalid PINs even if valid also output. BP6 only given if **valid** is output as the only output.
					
					If sensible attempt at BP3 and BP4 (but incorrect), FT to BP5 and 6.
					
					Section A so candidates can respond in pseudocode, structured English, flowcharts, etc. However, must not simply be a repeat of the question.

<table>
<tr><th colspan="3">Question</th><th>Answer</th><th>Mark</th><th>Guidance</th></tr>
<tr><td>6</td><td>(a)</td><td>(i)</td><td>• 3rd box ticked</td><td>1</td><td></td></tr>
<tr><td>6</td><td>(a)</td><td>(ii)</td><td>• Checks if <code>passA</code> and <code>passB</code> are equal…
• …Correct output if True
• …Correct output if False

Design so answer can be flowchart, etc. If flowchart, decision (diamond) box needed for BP1, True / False labels essential for BP2 and 3. Output (parallelogram) box needed for BP2 and 3.</td><td>3</td><td><u>Example code</u>
<pre>if passA == passB:
    print("passwords match")
else
    print("passwords do not match")
end if</pre>
Allow alternative messages.

Ignore superfluous code (including inputs or overwriting variables) as long as it does not affect outcome.

Could be done as separate <code>if</code> statements. Do not allow <code>elseif/elif</code> etc by itself for BP1 unless complete logical comparison (e.g. <code>elif passA != passB</code>). FT to BP2 and 3.

BP2 and 3 require a sensible attempt at comparing variables in BP1 to be given.

<code>passA</code> and <code>passB</code> must be correct with no spaces. Ignore case. Penalise spaces only if clear and obvious.</td></tr>
</table>

Question			Answer	Mark	Guidance
6	(b)	(i)	One mark per refinement • `SELECT ActID , Stage` • `FROM TblMusic` • `WHERE` • `Day = "Saturday"`	4	Allow descriptions of change. Allow quote marks around field and table names. Allow == for BP4. Spellings / spaces in field/table names must be accurate. Penalise spaces only if clear and obvious. Ignore case. Max 3 if final SQL is invalid / in wrong order / has extra superfluous code. For same error repeated (e.g. **:** included), penalise once then FT.
6	(b)	(ii)	• Any example of **one** complete record / row from the table given • Any example of **one** complete record / row that could potentially be added to the table	1	Accept any sensible data for new record as long as 4 distinct items of data given. Allow field names with data. Ignore order of data. Ignore case / spelling. Ignore data types Do not allow definition of a record.
6	(b)	(iii)	• Stage: **String** • SetLength: **Real** // **Float**	2	Accept equivalent data types, including those that may be common in databases such as : • String = text / varchar / char etc • Real / float = decimal / single / double / numeric etc

Question			Answer	Mark	Guidance

6	(c)	(i)	• Function **definition** for `calculatePrice()`
			• … with parameter that is not overwritten
			• Calculate price based on multiplying parameter (if present) by 60 …
			• …always return calculated price
				4	Section B so must be high level code. Do not accept x for multiplication. Do not accept spaces in function, variable or parameter names if clearly obvious.
					
					Example code
					`function calculatePrice(num)`
					`  price = num * 60`
					`  return price`
					`endfunction`
					
					All code must be within function definition (if present).
					
					BP3 – if no parameter present, allow multiplying by sensible value (e.g. input value). FT overwriting parameter.
					
					BP4 must be **return**. Do not credit output/print. Value returned can be with or without £ sign and other text. FT for BP4 if sensible attempt made at calculation.
					
					Accept multiple parameters passed in. Ignore superfluous code
					
					Accept returning by assigning to function name (e.g. `calculatePrice = price`), some languages such as VB do this.

Question			Answer	Mark	Guidance
6	(c)	(ii)	• `CalculatePrice(3)` called… • … return value assigned to `totalPrice`	2	Example code `totalPrice = calculatePrice(3)` Do not accept == for assignment Allow FT to BP2 only if sensible attempt at BP1 (e.g. missing/wrong parameter) Allow multiple values passed in as long as 3 is one of the values. Do not accept definitions of a function – must be calling the existing function. Penalise spaces in variable / function name only if clearly obvious. Ignore superfluous code. Ignore case.
6	(d)	(i)	• `Count` **initially** set to 0 • `count` increases to 10, 24, 30 and no other changes	4	Example

<table>
<tr><th colspan="3">Question</th><th>Answer</th><th>Mark</th><th>Guidance</th></tr>
<tr><td></td><td></td><td></td><td>

- `x` goes from 0, 1, 2 (, 3) and no other changes.
- 30 output and no other output.

</td><td></td><td>

<table>
<tr><th>Line number</th><th>count</th><th>x</th><th>Output</th></tr>
<tr><td>1</td><td>0</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>0</td><td></td></tr>
<tr><td>3</td><td>10</td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>1</td><td></td></tr>
<tr><td>3</td><td>24</td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>2</td><td></td></tr>
<tr><td>3</td><td>30</td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td></tr>
<tr><td>5</td><td></td><td></td><td>30</td></tr>
</table>

Ignore line numbers - not required. Mark order of values.

Values can be repeated in `count` and `x` columns, look at changes.

FT for BP4 if incorrect answer calculated but output. Allow **last** value of count calculated to be output for the mark.

Give BOD for "-", "nothing", etc as extra in output column. Do not accept `print(30)` or equivalent for output.

</td></tr>
</table>

Question			Answer	Mark	Guidance
6	(d)	(ii)	• Line 02 • `For x = 0 to 4`	2	Must use `x` as counter variable. Accept rewrites in other languages BOD `0 to 5` / `range(5)` etc Accept other changes that implement the change successfully e.g. using length of `actNumbers`
6	(d)	(iii)	• `print("The total act count is " + count)`	1	Allow other concatenation operators e.g. & (VB), dot (PHP, Rust), double dot (Lua), etc. Allow comma. Must have some form of separator. Ignore spacing. Ignore case. Ignore casting. Allow other commands that are equivalent to `print` (e.g. `echo`, `output`, etc) Allow multiple print statements. Allow multiple lines of code that achieve the outcome. Allow string interpolation or language specific examples that will achieve this (e.g. f strings in Python, use of $ before variables inside strings in PHP). Both of the following and examples like this are OK : • `print("The total act count is {count}")` • `print("The total act count is $count")`

Question			Answer	Mark	Guidance

6	(e)		• input **and store/use** number of tickets needed
			• Check if enough `tickets` left…
			• …sensible confirmation message output correctly for **both** booked and no tickets available
			• …`tickets` variable updated correctly
			• Attempt at iteration…
			• …repeat **all above that has been attempted** while tickets > 0 // while tickets != 0 // until tickets <=0 // until tickets = 0
				6	Allow any suitable confirmation message for BP3 (booked / not enough) but must be right way around.
					
					BP2 must allow tickets to be booked when exact number remain (e.g. 5 tickets needed and 5 remaining). Check comparison operator does not exclude this.
					
					BP3 and 4 dependent on BP2. If BP2 incorrect / not given but reasonable attempt (e.g. checking if **`tickets`** bigger than 0) allow FT for BP3/4.
					
					Any attempt at loop means giving BP5
					
					BP6 must not repeatedly reset tickets to 500. BP6 must include repeated input (if present).
					
					Ignore superfluous code.
					Example code
					`tickets = 500`
					`while tickets > 0`
					`  num = input("enter num tickets")`
					`  if tickets >= num then`
					`  print("tickets booked")`
					`  tickets = tickets - num`
					`  else`
					`  print("not enough tickets")`
					`  endif`
					`endwhile`

Need to get in touch?

If you ever have any questions about OCR qualifications or services (including administration, logistics and teaching) please feel free to get in touch with our customer support centre.

Call us on

01223 553998

Alternatively, you can email us on

support@ocr.org.uk

For more information visit

ocr.org.uk/qualifications/resource-finder

ocr.org.uk

Twitter/ocrexams

/ocrexams

/company/ocr

/ocrexams

CAMBRIDGE
UNIVERSITY PRESS & ASSESSMENT

OCR is part of Cambridge University Press & Assessment, a department of the University of Cambridge.

OCR operates academic and vocational qualifications regulated by Ofqual, Qualifications Wales and CCEA as listed in their qualifications registers including A Levels, GCSEs, Cambridge Technicals and Cambridge Nationals.

OCR provides resources to help you deliver our qualifications. These resources do not represent any particular teaching method we expect you to use. We update our resources regularly and aim to make sure content is accurate but please check the OCR website so that you have the most up-to-date version. OCR cannot be held responsible for any errors or omissions in these resources.

Though we make every effort to check our resources, there may be contradictions between published support and the specification, so it is important that you always use information in the latest specification. We indicate any specification changes within the document itself, change the version number and provide a summary of the changes. If you do notice a discrepancy between the specification and a resource, please contact us.

Whether you already offer OCR qualifications, are new to OCR or are thinking about switching, you can request more information using our Expression of Interest form.

Please get in touch if you want to discuss the accessibility of resources we offer to support you in delivering our qualifications.