Question			Answer/Indicative content	Marks	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. <u>Examiner's Comments</u> This question assessed candidates' understanding of different types of test data: normal, boundary, and invalid/erroneous. The vast majority of candidates performed very well on this question, demonstrating a secure understanding of the definitions and application of each test data category. It was clear that this topic had been well taught across centres, with many candidates achieving full marks. Use of Invalid/erroneous test data Although the specification defines invalid data and erroneous data separately, questions in examinations will always group these together as data that a system would reject, as shown in this question.
	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. <u>Examiner's Comments</u> Most candidates were able to correctly identify the purpose of the IF statement and therefore complete the third gap (**`else`**) with ease. Many candidates also correctly completed the first gap as a simple comparison with the variable **`num`** that was already given. However, the second gap proved more challenging. A common mistake was to write **`<= 100`** without referencing the variable a second time, which is not

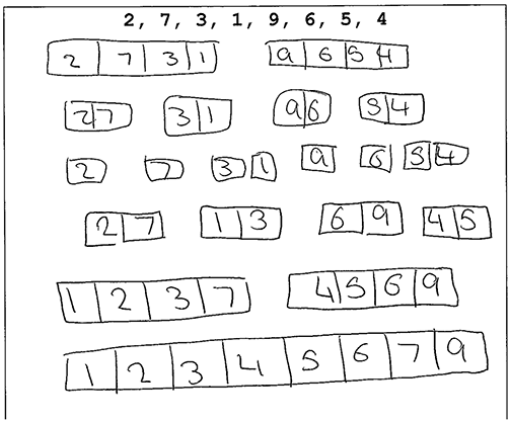
Question			Answer/Indicative content	Marks	Guidance
					syntactically valid. Another frequent error involved misunderstanding the inclusivity of the boundaries. Some candidates used **`num > 1 AND num < 100`**, which would **`exclude`** valid inputs of 1 and 100. This suggests a need for further emphasis on the difference between inclusive and exclusive comparisons when validating input ranges.  Misconception Selection statements using two conditions joined together with a Boolean condition such as AND must make sure that both conditions are logically valid comparisons. In many cases, this will mean referencing a variable twice, once in each condition. For example, **`if num >= 1 AND <= 100`** is incorrect because the second condition is **`not`** a valid comparison - the value num must be included in the second part of the comparison. The correct statement would be **`if num >= 1 AND num <= 100.`**
		ii	• `num`	1	Do not allow any other code around variable identifier. Ignore capitalisation. Ignore minor spelling errors.
		iii	• `AND`	1	Do not allow any other code around operator. Ignore capitalisation. Ignore minor spelling errors.
			Total	9	

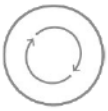
Question			Answer/Indicative content	Marks	Guidance
2	a		• 2 • 6	2	Examiner's Comments This question involved following a flowchart to determine the outcome produced. Most students did this successfully for the input value of 3, determining correctly that OK would be output twice. However, fewer candidates were able to do this for the input value of 10. This may be due to confusion over the use of the 'less than' operator when the value held in variable **x** was set to zero.
	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") Examiner's Comments Iteration is commonly referred to as loops, or repetition. Candidates were confidently able to name examples of keywords that are used in a high-level language (such as **`for, while`** or **`do until`**) and describe the use of these well. Many candidates used subject-specific terminology such as **condition controlled** loops or **count controlled** loops when doing this, as well as describing these in more depth. Some candidates lacked precision with their description of a **`while`** loop, stating that this repeats **until** a condition is **`True`** which is incorrect.

Question			Answer/Indicative content	Marks	Guidance
					 Misconception A **`while`** loop is condition controlled. It has a condition that is evaluated before each iteration begins; if the condition evaluates to be **`True`**, the loop repeats one more time before the condition is evaluated again. For example, **`while x > 7`** : The condition here that is evaluated is whether the contents of the variable **`x`** is larger than 7. As long as this is the case, the loop will continue to repeat. Many candidates suggested that the while loop should repeat **until** the condition is **`True`**, that is, **`while x > 7`** should repeat for values of **`x`** that are not larger than 7 and only stop repeating when **`x`** is larger than 7. This is clearly incorrect. Centres and candidates must be aware that their answers should be precise and technically correct if they are to achieve highly.
		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. <u>Examiner's Comments</u> Many candidates were able to list sequence and selection as two other programming constructs which are listed in the specification document. We accepted sensible alternatives (such as branching).
	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…	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

Question			Answer/Indicative content	Marks	Guidance
			• …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 • 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		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”) <u>Examiner’s Comments</u> Candidates were mostly able to name and describe suitable tools contained within an Integrated Development Environment, such as an editor, error diagnostics, run-time environment or translator. Many other sensible tools and sub-tools such as a variable watch window, stepping, keyword completion or Artificial Intelligence (AI) tools were all also credited. A small number of candidates mistakenly described tools contained within programming languages, such as commenting or use of variables. Descriptions were generously given where they added to the tool name, but repetition (such as “a code editor / edits code”) was not given both mark parts.
			Total	12	

Question			Answer/Indicative content	Marks	Guidance
3	a		• 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. <u>Examiner's Comments</u> The majority of candidates showed clearly that they understood the steps a merge sort takes to sort values into ascending order. The strongest responses clearly showed both the divide stage (splitting values up into individual lists) and then the conquer stage (merging lists together in order). Only a minority of candidates attempted a different sorting algorithm or showed a lack of understanding of the abstract idea behind the algorithm; this is very pleasing to see. However, several candidates still merge lists together into an unsorted list and then take a second step to sort the list. This has been commented on in previous Examiner's Reports but remains a common misconception. Where this was the case in candidate answers, the mark scheme instructed examiners not to give the relevant mark.  **Misconception** A merge sort merges smaller lists together to form a larger sorted list. At each step of

Question	Answer/Indicative content	Marks	Guidance
			the merge process, **all** lists are **always** in order; it is the process of merging together that performs the sort. A very common misconception is for candidates to merge lists together by simply appending one to the other (to form an unsorted list) and then sorting this list. Of course, this begs the recursive question - what algorithm is being used to carry out the sort? **Exemplar 1**  In this example, the candidate has successfully achieved full marks. The list is repeatedly divided into smaller and smaller lists until lists of size one are present. These are then merged together into lists of size 2, then lists of size 4 before finally being merged into one final list of size 8. Note that it is the act of merging that does the sorting; values are **not** merged out of order and then sorted. The candidate puts boxes around their numbers to clearly indicate the different lists. This is an example of best practice.

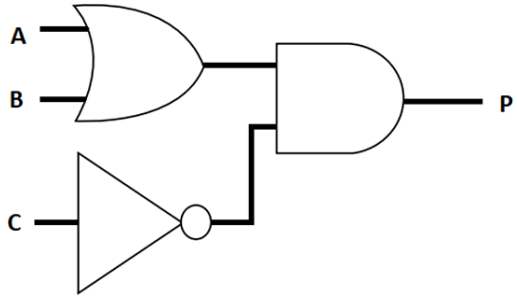
Question			Answer/Indicative content	Marks	Guidance
	b		• Uses an array of numbers with a sorted part and unsorted part **(top box)** • Swaps them if they are in the incorrect order **(bottom box)** • Is declared with a fixed length **(top box)**	3	No mark if more than one box ticked per section. Examiner's Comments The majority of candidates were able to select the correct answers for the first two tick boxes, showing good understanding of sorting algorithms. Fewer candidates were able to define an array as being declared with a fixed length. This may be due to Python being the most common programming language used, and Python support **lists** by default instead of arrays. Lists are dynamic data structures and so can be appended to, whereas arrays are static data structures with a fixed length.  Assessment for learning If Python is used as the main programming language in a centre, care must be taken to make sure that elements of the specification not easily covered in Python, (e.g. arrays, `do...until` loops, `switch/case`) are covered suitably in a different way – for example, perhaps by introducing a second programming language. Note : `Switch/case` is supported in Python version 3.10 and later using the `match` and `case` keywords.

Question			Answer/Indicative content	Marks	Guidance
	c	i	• Linear search	1	Accept “linear” by itself Do not allow “linear **sort**” Examiner’s Comments Most candidates were able to identify the algorithm given as a linear search. The other stopping condition for part (ii) had to be precisely and correctly defined by candidates. Many responded “when the value is not found”, but how would this be written as a condition for a high-level programming language to check? The condition is technically when the end of the list is reached **and** the value has not been found, but responses relating to simply reaching the end of the list or having checked each item were accepted.
		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. Examiner’s Comments Most candidates were able to identify the algorithm given as a linear search. The other stopping condition for part (ii) had to be precisely and correctly defined by candidates. Many responded “when the value is not found”, but how would this be written as a condition for a high-level programming language to check? The condition is technically when the end of the list is reached **and** the value has not been found, but responses relating to simply reaching the end of the list or having checked each item were accepted.
			Total	9	

Question			Answer/Indicative content	Marks	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)`
					
					Examiner's Comments
					
					Casting was well understood by most candidates. Casting converts one data type to another data type. This question was an AO2 question and so requires an answer in context.
					
					Candidates were required to identify how casting is used **in the given algorithm.** In this case, this meant identifying that a value was converted to an integer (on the 4th line). Candidates could have identified any value or variable to be converted to an integer, as they were not double penalised if they had incorrectly answered the previous question.

	b		• Convert data to an **integer / int**	1	Do not allow generic "change to different data type" Examiner's Comments Casting was well understood by most candidates. Casting converts one data type to another data type. This question was an AO2 question and so requires an answer in context. Candidates were required to identify how casting is used **in the given algorithm.** In this case, this meant identifying that a value was converted to an integer (on the 4th line). Candidates could have identified any value or variable to be converted to an integer, as they were not double penalised if they had incorrectly answered the previous question.
			Total	6	

Question			Answer/Indicative content	Marks	Guidance
5	a		• Any use of an **OR** gate with **2 inputs** • NOT **C** • Correctly joined by **AND** gate  	3	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. <u>Examiner's Comments</u> This question was extremely well answered by candidates, showing excellent understanding of constructing logic gate diagrams from a real-world scenario. Misconceptions or mistakes were rare, but where they did occur they tended to be around shapes of gates. Candidates should be aware that the circle on the NOT gate is crucial (as otherwise a triangle with no circle is simply a buffer) and that AND and OR gates do not need circles, or they would instead of NAND or NOR gates. Examiners are instructed to mark the shapes of the gates and ignore any annotation, so it is crucial that candidates can accurately draw shapes that are clearly AND and OR gates.

<table>
<tr><th colspan="3">Question</th><th>Answer/Indicative content</th><th>Marks</th><th>Guidance</th></tr>
<tr><td></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

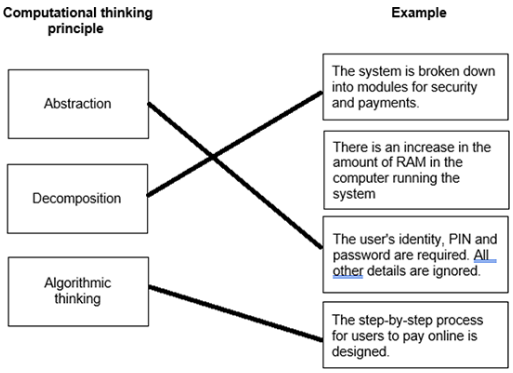
<table><tr><th>A</th><th>B</th><th>P</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>
Accept T/F, Yes/No, ticks/crosses, etc.

<u>Examiner's Comments</u>

This question was also extremely well answered. Most candidates filled in columns A and B with every possible combination of inputs. 1s and 0s were the expected options, but Yes/No and True/False were also accepted if given by candidates.

Inputs do not have to be in any particular order, but best practice to start with 00 and then essentially count up in binary, through 01, 10 then 11. This means that where larger systems are presented (perhaps with 3 or 4 inputs), no input combination is missed.

The final column (P) should have contained the output for the system, and with P = A AND B, this means that only the row with A and B as both 1 should have had a 1 output, with every other output being 0.</td></tr>
</table>

Question	Answer/Indicative content	Marks	Guidance
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. 

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.
			
			<u>Examiner's Comments</u>
			
			As a question in Section A, candidates are free to use a high-level programming

Question			Answer/Indicative content	Marks	Guidance
					language, flowcharts or structured English. Pleasingly, the vast majority attempted to use a high-level programming language and this proved to be very successful for many. Candidates proved able to take input as a string, to use in-built length functions to check for length and to output messages based on these checks. The major misconception that affected many students related to checking two invalid PINs, 1234 or 4321. Where two checks are done within a single IF statement it is important that each condition **must** be a valid comparison and therefore include the variable to check against for both comparisons. **This is the same misconception highlighted on Question 1 (b) (i).**  **Misconception** Selection statements using two conditions joined together with a Boolean condition such as OR must make sure that both conditions are logically valid comparisons. In many cases, this will mean referencing a variable twice, once in each condition. For example, **`if PIN == "1234" OR "4321"`** is incorrect because the second condition is not a valid comparison – what is being compared to "4321" in order for the condition to be evaluated as True? The correct statement would be **`if PIN == "1234" OR PIN == "4321"`**.
			Total	14	

Question			Answer/Indicative content	Marks	Guidance
6	a	i	• 3rd box ticked	1	Examiner's Comments Pleasingly, the vast majority of candidates were able to identify the correct syntax for code to store data in two distinct variables, despite the distractors being designed around mistakes and misconceptions that have been seen commonly in previous examination series. Centres should be applauded for preparing candidates so well to avoid these common pitfalls.

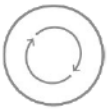
Question			Answer/Indicative content	Marks	Guidance

		ii	• Checks if `passA` and `passB` 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.
				3	Example code
					`if passA == passB:`
					`  print("passwords match")`
					`else`
					`  print("passwords do not match")`
					`end if`
					
					Allow alternative messages.
					
					Ignore superfluous code (including inputs or overwriting variables) as long as it does not affect outcome.
					
					Could be done as separate `if` statements. Do not allow `elseif/elif` etc by itself for BP1 unless complete logical comparison (e.g. `elif passA != passB`). FT to BP2 and 3.
					
					BP2 and 3 require a sensible attempt at **comparing** variables in BP1 to be given.
					
					`passA` and `passB` must be correct with no spaces. Ignore case. Penalise spaces only if clear and obvious.
					
					Examiner's Comments
					
					This question was answered exceptionally well by most candidates - including candidates who struggled on other questions. This shows that basic fundamental programming skills are being taught consistently across centres and candidates are able to complete questions such as this to a high standard.
					
					This shows a clear progression from previous years and is extremely pleasing to see. Although the constructs assessed here should be relatively simple, many candidates were not able to design algorithms of this nature historically.

	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.

Question	Answer/Indicative content	Marks	Guidance

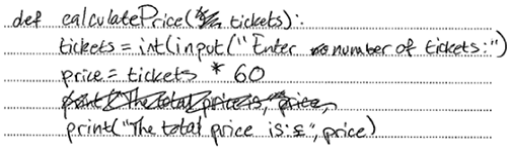
			For same error repeated (e.g. : included), penalise once then FT.
			
			Examiner's Comments
			
			This question was surprisingly challenging for many candidates, with very few candidates achieving full marks even when they scored very highly across the rest of the paper.
			
			Most candidates were able to identify the incorrect table name and that **IF** should have been **WHERE** to be valid SQL. However, far fewer candidates replaced the AND between the field names with a comma or noticed the missing string delimiters (either single or double quotation marks) around Saturday.
			
			Where candidates introduced other errors, this was recognised by the mark scheme. A very common error was introducing spaces into field or table names.
			
			  Misconception
			
			In SQL, field names are separated by commas and string data has to be delimited using quotation marks. The correct, full mark answer for this question was
			
			`SELECT ActId, Stage`
			
			`FROM TblMusic`
			
			`WHERE Day = "Saturday"`
			
			SQL is case-insensitive, so capitals do not have to be used for keywords in candidate answers but are in examination paper questions by convention.
			
			However, field and table names do not contain spaces, so **Tbl Music** is incorrect.

Question			Answer/Indicative content	Marks	Guidance
		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. <u>Examiner's Comments</u> This was the lowest scoring question on the entire examination paper. Only a small percentage of candidates were able to give an example of a record as an entire row in the database table, such as `[1, Platinum, Saturday, 0.5]`. Examiners were instructed to be generous and give this mark wherever four field values were identified, whether they were already in the table or even the correct data type or not. However, even with this generosity, most candidates were not successful.  **Assessment for learning** "The use of records to store data" is bullet point 3 in section 2.2.3 of the specification, with the further clarification that "use of 2D arrays to emulate database tables of a collection of fields, and records" is required content to be covered.

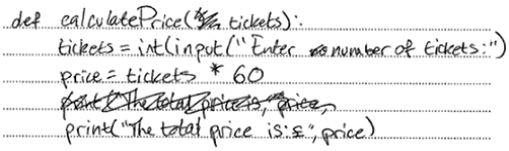
Question			Answer/Indicative content	Marks	Guidance
		iii	• Stage: **String** • SetLength: **Real** // **Float**	2	Accept equivalent data types, including those that may be common in databases such as : • String = text / varchar / char etc • Real / float = decimal / single / double / numeric etc <u>Examiner's Comments</u> This question required candidates to give the most appropriate data type for each field listed. Where candidates understood the concept of data types, the vast majority were able to correctly identify string and real/float (either was acceptable) as the correct answers. Iincorrect answers tended to show that candidates did not know what was meant by "data type" and so gave responses such as "variable" or "input". Data types that exist with common DBMS software such as MySQL or Microsoft Access were also accepted. This meant that (for example), varchar and single/double were accepted as synonyms for string and real/float.

Question			Answer/Indicative content	Marks	Guidance

	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.
					
					Examiner's Comments
					
					These questions proved to be excellent discriminators, with only candidates who achieved very highly overall able to do well. Although teaching of programming by centres has improved students' general overall responses, it is clear that defining functions and then utilising them is an area that still requires focus.
					
					For part (c) (ii), candidates were asked to define a function. Many candidates were able to begin to define a function. However, a common misconception was either the lack of a parameter in the definition, or overwriting/ignoring the parameter and asking for separate input within the function.
					
					Another very common misconception was to print or otherwise output the calculated price instead of returning it as required.
					
					For part c (iii), candidates were asked to

Question	Answer/Indicative content	Marks	Guidance
			then use this function to calculate the price of 3 tickets and store the answer in specified variable. Very few candidates did this successfully. Instead many candidates attempted to redefine the function again or to work out the price of the tickets in some other way. **Misconception** When a function is defined, all data passed to the function should be done via parameters. It is not appropriate to ask for input inside the function, especially when this overwrites or replaces an existing parameter. Functions also return a value instead of printing anything, as clearly requested in the question. Exemplar 2  In this example, the candidate has successfully defined a function with a single parameter, tickets. This first line would have achieved mark points 1 and 2. However, the tickets' parameter is then overwritten with an unnecessary input statement. The price is then correctly calculated but is printed instead of returned. As a result, this example receives 2 out of 4 marks.

Question	Answer/Indicative content	Marks	Guidance
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. Examiner's Comments These questions proved to be excellent discriminators, with only candidates who achieved very highly overall able to do well. Although teaching of programming by centres has improved students' general overall responses, it is clear that defining functions and then utilising them is an area that still requires focus. For part (c) (ii), candidates were asked to define a function. Many candidates were able to begin to define a function. However, a common misconception was either the lack of a parameter in the definition, or overwriting/ignoring the parameter and asking for separate input within the function. Another very common misconception was to print or otherwise output the calculated price instead of returning it as required. For part c (iii), candidates were asked to then use this function to calculate the price of 3 tickets and store the answer in specified variable. Very few candidates did this successfully. Instead many candidates attempted to redefine the function again or to work out the price of the tickets in some other way. Misconception

Question			Answer/Indicative content	Marks	Guidance
					When a function is defined, all data passed to the function should be done via parameters. It is not appropriate to ask for input inside the function, especially when this overwrites or replaces an existing parameter. Functions also return a value instead of printing anything, as clearly requested in the question. Exemplar 2  In this example, the candidate has successfully defined a function with a single parameter, tickets. This first line would have achieved mark points 1 and 2. However, the tickets' parameter is then overwritten with an unnecessary input statement. The price is then correctly calculated but is printed instead of returned. As a result, this example receives 2 out of 4 marks.
	d	i	• `Count` **initially** set to 0 • `count` increases to 10, 24, 30 and no other changes • `x` goes from 0, 1, 2 (, 3) and no other changes. • 30 output and no other output.	4	Example | Line number	count	x	Output
---	---	---	---
1	0		
2		0	
3	10		
4			
2		1	
3	24		
4			
2		2	
3	30		
4			
5			30	 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

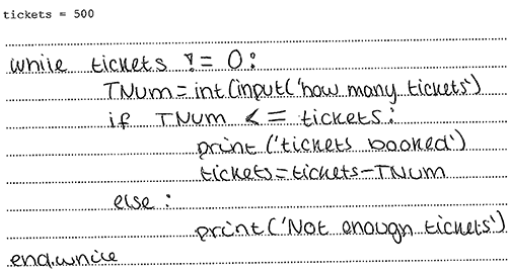
Question			Answer/Indicative content	Marks	Guidance
					`print(30)` or equivalent for output. Examiner's Comments Trace tables are commonplace in Paper 2 and have examined before. Candidates showed better understanding of tracing a program through line by line, and this confidence was clear in candidate responses, with many able to correctly do this. This question differed slightly from historical examination questions in two ways. Firstly, this is the first time that a for loop has been asked about in a trace table. The challenge of the counter variable being updated was handled with ease by most candidates. Secondly, although line numbers were asked for, these were not used in the marking of candidates' responses and instead one mark was given per set of changes to variables. This process worked extremely well and will be used in the marking of any future trace table questions.
		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` Examiner's Comments Candidates were able to confidently identify line 02 as the line to be changed and also identified the upper bound of the for loop as the value to be changed. Where candidates provided other solutions that logically worked, these were of course accepted.

Question			Answer/Indicative content	Marks	Guidance
		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")` Examiner's Comments As with many other smaller programming questions on this paper, candidates performed exceptionally well. The majority of candidates were able to use the **`count`** variable to output the correct value. Where this was not successfully done, candidates tended to include the variable identifier inside the string delimiters, thereby outputting the word count and not the value held in the variable. Examiners were instructed to credit responses that used string interpolation, such as f strings in Python or the use of $ before strings in PHP.
	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	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.

Question	Answer/Indicative content	Marks	Guidance

	tickets != 0 // until tickets <=0 // until tickets = 0		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`
			
			Examiner’s Comments
			
			This year, candidates appeared much more willing to attempt the question and therefore many received marks for their efforts.
			
			Very few candidates left their response completely blank. This is pleasing to see, as usually we see an increase in the number of candidates who do not attempt this question at all.
			
			Marks were allocated on the mark scheme for the correct input from the user and any attempt at iteration, with most candidates achieving at least the first of these.
			
			Many candidates made a very strong attempt at checking for tickets available. Where this was done correctly, it included the scenario where exactly the available number of tickets were requested, (e.g. 7 tickets remaining and 7 tickets were requested). If, however, a candidate checked **`if tickets > num`** (where num is the number requested), this was not correct as it would not allow this exact scenario.

Question	Answer/Indicative content	Marks	Guidance

			Other common mistakes included:
			
			• checking if tickets was less than 500 instead of checking against the number requested
			• subtracting the number of tickets required even when the booking was not successful.
			
			Altogether however, this was an well answered question with most candidates achieving some marks and many achieving all or almost all available marks.
			
			Exemplar 3
			
			 
			tickets = 500
			while tickets != 0:
			TNum = int(input('how many tickets'))
			if TNum <= tickets:
			print('tickets booked')
			tickets = tickets-TNum
			else:
			print('Not enough tickets')
			endwhile
			
			In this example, the candidate successfully achieves all mark points for a 6 mark response.
			
			• The number of tickets requested is entered and stored in the variable **`TNum`**.
			• This value is then checked against the **`tickets`** variable, using <= to make sure that there are enough tickets remaining.
			• If this check was successful, a message is printed and the value in tickets reduced correctly.
			• If the check was not successful then a "not enough tickets" message is printed instead.
			• All of this is repeated while the number of **`tickets`** does not equal 0.
			
			This was a typical response that achieved full marks.

	Total	30	