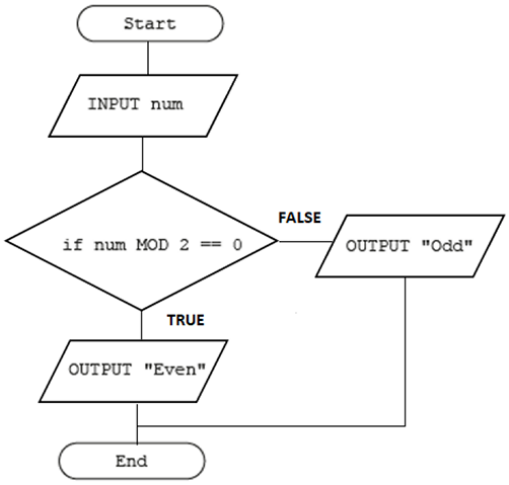
<table>
<tr><th colspan="3">Question</th><th>Answer/Indicative content</th><th>Marks</th><th>Guidance</th></tr>
<tr><td>1</td><td></td><td></td><td>
<table>
<tr><th rowspan="2">Keyword</th><th colspan="2">Programming construct</th></tr>
<tr><th>Selection</th><th>Iteration</th></tr>
<tr><td>if</td><td>✓</td><td></td></tr>
<tr><td>for</td><td></td><td>✓</td></tr>
<tr><td>while</td><td></td><td>✓</td></tr>
</table>
</td><td>3
(AO1)</td><td><u>Examiner's Comments</u>

This question was answered very well by the majority of candidates, showing a good understanding of the difference between the key programming constructs of selection and iteration.

Practical programming skills in lessons should be used to make this explicit link between technical definitions and use within a high-level language.</td></tr>
<tr><td></td><td></td><td></td><td>Total</td><td>3</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>
• Correct shape for all three inputs AND outputs (parallelogram)
• Correct shape for decision (diamond)
• True and False // Yes and No labelled correctly (true/Yes linking to "Even")
• All lines joined up correctly and link to End.


</td><td>4
(AO2)</td><td>No need for arrows – lines are acceptable.

BOD for correct answers that include a loop back to the start

<u>Examiner's Comments</u>

The majority of candidates recognised the correct flowchart shapes to be used for a decision and many also were able to use the parallelogram shape for inputs and outputs. However, fewer correctly connected up the boxes to make sure that all paths started and ended at appropriate points and even fewer candidates labelled up the decision box appropriately with True/False or Yes/No. These labels are crucial to be able to follow the path of the algorithm when a decision is made.</td></tr>
<tr><td></td><td></td><td></td><td>Total</td><td>4</td><td></td></tr>
<tr><td>3</td><td>a</td><td></td><td>
Max 1 mark for definition that is clearly different from a logic error.

• (an error that) breaks the rules/grammar of the programming language
• Stops the program from running // does not allow program to run // crashes the program // does not allow program to translate

Suitable example for 1 mark, e.g.

• misspelling key word (e.g. <code>printt</code>
</td><td>2
(AO1)</td><td>BOD code/program etc for BP1

Do not allow answers linked to data types.

"incorrect grammar" by itself is NE

Do not allow "stop working", "does not work", etc – TV.

Do not accept <u>missing</u> quotation marks e.g. <code>print(hello)</code> (could be a variable name)

BOD given code that could cause a syntax</td></tr>
</table>

Question			Answer/Indicative content	Marks	Guidance
			instead of **`print`**) • Missing / extra symbol (e.g. missing bracket, missing semicolon) • Mismatched quotes • Invalid variable or function names (e.g. variable starting with a number or including a space) • Incorrect use of operators • Use of reserved keywords for variables (e.g. **`print`** `= 3`) • Incorrect capitalisation of keywords (e.g. **`P`**`rint` instead of `print`) • Incorrect indentation (of code blocks) • Missing concatenation (e.g. `print(score x))`		error in a high-level language. Examiner's Comments This question firstly asked for a definition of the term 'syntax error'. Although many candidates were correctly able to do this in terms of rules of the programming language being broken, many instead gave examples of issues that would cause syntax errors, such as 'where a bracket is missing'. This would indeed cause a syntax error in many high-level languages but is not a definition in general and so was not credited by examiners for this part of the question. The second part did ask for an example and generous interpretation was asked of examiners so that any issue that could feasibly cause a syntax error was awarded, such as missing brackets or misspelling of key words. One common misconception here involved missing quotation marks/string delimiters around a string, which could instead be a reference to a variable if this was a single word. Misconception A syntax error is a mistake with the rules/grammar of the programming language that means the program will not run/execute or compile. Code such as `print(temp)` would not necessarily be a syntax error because `temp` could plausibly be a variable.
	b		1 mark each • line 03 • `total = num1 +` **`num2`** • Line 04 • `if total >= 10` **`and total <=20`** `then` Allow other logical equivalent code e.g. `total = int(num1) + int(num2)` `if 10 <= total <= 20`	4 (AO3)	Allow other logical corrections that will fix the problem identified and does not introduce any further errors. Allow descriptions of changes as long as clear exactly what will change. Do not allow ambiguous descriptions of changes to code. Ignore missing `then` from line 04. Ignore capitalisation.

Question	Answer/Indicative content	Marks	Guidance

			Examiner's Comments
			
			Line 03 was commonly identified as containing a logic error and many candidates were able to correct this. Where a candidate attempted to explain what changes should be made, this was only credited where the explanation was unambiguous and precise.
			
			The correction to line 04 was commonly done incorrectly due to the need to check multiple values.
			
			Misconception
			
			Where multiple values are required to be checked in a selection statement, a line such as :
			
			`if total >= 10 and <= 20 then`
			
			is incorrect as the second part of the statement has nothing to compare 20 against. The first part will clearly evaluate to true or false, but the second part is ambiguous. Instead, candidates should be encouraged to use :
			
			`if total >= 10 and` **`total <= 20`** `then`
			
			Alternatives that work in high-level languages would also obviously be accepted.
			
			Exemplar 1
			
			Line number 03
			Correction total = num1 + num2
			
			Line number 04
			Correction if total >=10 AND <=20 then
			
			Although this candidate has correctly identified lines 03 and 04, the correction for line 04 is incorrect due to the missing reference to total when comparing the upper boundary. This achieves three marks out of four.

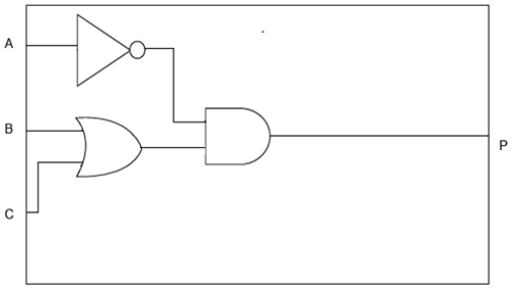
Question			Answer/Indicative content	Marks	Guidance
	c	i	1 mark each • Compare to / pick out middle value (which is 6) • **discard only left side // retain only right side** (because 6 < 10)… • … Compare to / pick out (middle value which is) 10	3 (AO2)	BP1 can be given for generic answer. BP2 and 3 must be linked to data set given For BP2, must remove 1, 2,5 and 6 from list if discussing individual numbers. Allow FT for BP3 if this done incorrectly.
		ii	• Data must be sorted / in order	1 (AO1)	
		iii	• Merge sort	1 (AO1)	
			Total	**11**	

Question			Answer/Indicative content	Marks	Guidance
4	a		Input e.g. • **Name / keyword** for video (to be searched for) // search **text** • **Controls** for watching video (e.g. play / pause) • **Rating** given to video Output e.g. • Video to be watched // audio • **Results** of search • (total / overall / average) rating of video • Number of views (of video) • Confirmation of data entry / data validity • Messages to user // example messages (e.g “enter a rating”, “your rating has been saved”) in **quotation marks**	2 (AO1)	1 mark for a suitable input, 1 mark for a suitable output Allow input / print pseudocode statements if meets mark point(s). Does not have to be valid pseudocode. Do not allow examples of inputs (e.g. “music videos”) <u>Examiner’s Comments</u> Identification of inputs and outputs for a problem is covered in specification point 2.1.2 and candidates are expected to be precise with their responses. Answers such as ‘video’ as input are problematic because the candidate would not upload or use a video for their input. Candidates who were more precise and used terms such as ‘name of video’ of ‘keywords searched for’ were positively recognised for this.
	b		Only 1 method asked for. Could be name and description/example or description and example • Authentication • …checking users allowed to access the site / know identity of users • …by example (e.g. username and password) • Anticipating misuse // preventing misuse • …stopping the user breaking / hacking into the system • …by example (e.g. restricting entry to integers) • Validation • …check / only allow sensible data to be entered / check data is sensible • …by example (e.g. restrict ratings to 1 to 10 / presence check / format check) • Input sanitisation • …removing invalid/special characters • …by example (e.g. remove quotation marks / semicolons) • Maintainability • …ensuring program is able to be understood by others • …by example (e.g. modularisation / comments)	2 (AO1)	Allow validation / input sanitisation / passwords as expansion of anticipating misuse. Allow mark for description with no / incorrect name Allow any 2 points from mark scheme as long as clearly linked to a single defensive design method. <u>Examiner’s Comments</u> The specification (Section 2.3.1) lists multiple ways that defensive design could be used in a program and any of these, plus other sensible options, were allowed as an acceptable response. The describe command word then required candidates to add further detail to obtain a second mark, in this case by describing how it could be used, either generically or as a specific example. Many points on the mark scheme crossed over with each other, such as a validation example being a sensible expansion for anticipating misuse, and hence two marks were able to be obtained in multiple ways.
			Total	**4**	

<table>
<tr><th colspan="3">Question</th><th>Answer/Indicative content</th><th>Marks</th><th>Guidance</th></tr>
<tr><td>5</td><td>a</td><td></td><td>1 mark per group of 2 rows

<table>
<tr><th>A</th><th>B</th><th>C</th><th>P</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table>
1 mark
1 mark
1 mark
1 mark</td><td>4
(AO2)</td><td>Accept True / False etc.</td></tr>
<tr><td></td><td>b</td><td></td><td>1 mark each

• NOT A
• B OR C
• AND gate with two inputs

 
</td><td>3
(AO3)</td><td>Max 2 if not logically correct or any additional / missing gates.

Shapes of gates must be correct with correct number of inputs. Ignore annotation of gate names.

NOT gate must include circle. Other gates must not include circle.

<u>Examiner's Comments</u>

Drawing logic circuits is now a commonly asked question and candidates are generally competent at doing this. Where issues did arise, they tended to be missing the circle from the NOT gate (and therefore turning it into a buffer, not a logic gate) or including the incorrect number of inputs into a gate.

Key point – logic gates

Candidates are expected to be able to draw the correct shapes for each gate. A number of candidates labelled their gates up, but this is not necessary; examiners are instructed to mark the shape of each gate and ignore any labelling.

Candidates are <u>not</u> allowed to take stencils or other tools that allow them to more easily draw these gates into an examination unless as part of a specific access arrangement agreed by OCR.</td></tr>
<tr><td></td><td></td><td></td><td>Total</td><td>7</td><td></td></tr>
</table>

Question			Answer/Indicative content	Marks	Guidance

6	a		1 mark for each output
			
			`print(message.upper)` — **`ABCD1234`** `(upper case)`
			`print(message.left(4))` — **`abcd`** `(lower case)`
			`print(int(message.right(4))*2)` — **`2468`**
				3 (AO2)	Case must be correct but BOD if ambiguous.
					
					Allow quotation marks in answer.
					
					Examiner's Comments
					
					This question assessed string manipulation and used OCR Exam Reference Language(ERL) to present each statement. This highlights the importance of candidates being taught how to interpret ERL as questions in the exam will be written in this format.
					
					Candidates were generally confident with the use of .upper and .left(). Candidates were less confident when this was combined with casting in the last part.

Question			Answer/Indicative content	Marks	Guidance

	b		1 mark per bullet point :
			
			• storing both strings correctly in `word1` and `word2`
			• correct concatenation (`word1` **`then`** `word2`)…
			• …storing in variable `message`
			
			Example
			`word1 = "Hello"`
			`word2 = "Everyone"`
			`message = word1 + word2`
				3
				(AO3)
					Accept & / + / . etc as valid methods of concatenation. Allow use of sensible concatenation functions e.g. concat() . **Do not allow commas.**
					
					Do not allow == for assigning value to string. Do not allow spaces in variable names. Penalise once then FT.
					
					Ignore additional code given. Ignore case.
					
					Reasonable attempt at BP2 needed to access BP3.
					
					Examiner's Comments
					
					This question was intended to be straightforward and aimed at all candidates, including those expecting to achieve lower grades. However, a significant number of candidates dropped marks, either through perhaps not reading the question requirements or through not understanding the term concatenation.
					
					There is no requirement in the question to print out the message obtained, simply to store it in the variable `message`. Where this was done in addition, this was not penalised but often it was done instead of storing the concatenated message.
					
					A common mistake was to use the comma for concatenation when in most high-level languages (and especially Python), this is simply used to print out two items or pass two arguments to a subroutine and not actually concatenate values. Another common mistake was with the names of the variables given – if these included spaces or differed from those given in the question (such as `word 1, wordone` or `world1` instead of `word1`) examiners were instructed to not allow these alternatives.

			Total	6	

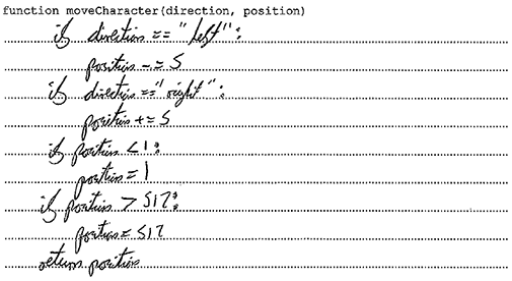
Question			Answer/Indicative content	Marks	Guidance
7	a		1 mark each to max 2 • (machine code) does not need to be translated / compiled / interpreted • Direct control of **hardware / memory** • Faster execution time • Code can be optimised / shorter code / use less memory • Can program for specific hardware • **Assembly language** is fast to translate.	2 (AO1)	"More efficient" by itself is TV. Mark first answer on each line. BP6 relates to Assembly language being a one-to-one direct mapping to machine code. <u>Examiner's Comments</u> These questions assessed understanding of low-level languages and the use of compilers to translate from high-level code. Many candidates were able to do this well and clearly understood the need for high-level languages to be translated into machine code before the processor can execute this. The questions, however, did not ask just for this but specifically asked for reasons and benefits. Where candidates left their response as simply a discussion of the difference between high-level and low-level code or between translators and compilers, it was difficult for examiners to award marks. Better responses were able to give clear reasons for the use of low-level code (such as direct control of hardware and faster execution of code) and the benefits of using a compiler instead of an interpreter (such as being able to distribute an executable file with no access to source code and not needing to translate code again once this has been done).

Question			Answer/Indicative content	Marks	Guidance
	b		1 mark each to max 3 • Can produce an **executable file** • program/code **runs/executes** faster (than interpreted version) • end users do not need translator • Can be run **again/multiple times** without re-translating // only needs to translate **once** • **End users** have no access to **source code** // distributed **with no source code**… • …cannot steal/copy/modify code/program • Shows **all/multiple** errors // shows errors **at the end** (of compilation) // creates **error file** • Compiler can optimise the code	3 (AO1)	Allow in reverse (e.g. “interpreter translates every time”) Do not allow "no access to source code" unless clearly talking about end user. Allow if in context of distribution. Do not allow descriptions of how a compiler translates (e.g. "translates whole code in one go") “Faster / quicker” by itself is TV <u>Examiner’s Comments</u> These questions assessed understanding of low-level languages and the use of compilers to translate from high-level code. Many candidates were able to do this well and clearly understood the need for high-level languages to be translated into machine code before the processor can execute this. The questions, however, did not ask just for this but specifically asked for reasons and benefits. Where candidates left their response as simply a discussion of the difference between high-level and low-level code or between translators and compilers, it was difficult for examiners to award marks. Better responses were able to give clear reasons for the use of low-level code (such as direct control of hardware and faster execution of code) and the benefits of using a compiler instead of an interpreter (such as being able to distribute an executable file with no access to source code and not needing to translate code again once this has been done).
			Total	5	

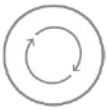
Question			Answer/Indicative content	Marks	Guidance
8	a		2 marks max per group • Meaningful **identifiers** // meaningful variable names • …to describe/show what they **store // purpose** of variable • An example of a meaningful variable identifier <u>for this algorithm</u> • Comments • …to make it easier for other programmers to **follow / understand** (part of) the code // explains what the code does // easier to **debug** • An example of a suitable comment <u>for this algorithm</u> • Use of subroutines • …to **reuse** blocks of code // make code easier to follow • An example of a subroutine <u>for this algorithm</u> • Use of constants • …to store data that will not change (during program execution) // so data can be changed in one place only • An example of a constant <u>for this algorithm</u> (e.g. store 512 as a constant)	4 (AO2)	Do not accept "what variables do" – incorrect verb, variables store/hold data. BOD notes (and alternatives) for comments. Do not allow instructions. Do not allow indentation (already done in program given) Allow whitespace / **blank** lines (same expansions as comments) Do not award expansion without being clear which method is being discussed. “Makes it easier to understand” by itself is TV. <u>Examiner’s Comments</u> It was pleasing that the majority of candidates understood the term maintainability and were able to suggest suitable improvements on a generic level, such as improving the naming of variables and adding comments. Better responses that achieved full marks were able to apply this to the code given, such as suggesting more suitable variable names. Candidates who suggested adding indentation were not credited with marks as this has clearly already been done in the code given and thus would not be an improvement. **Key point – generic versus. context driven responses** Where a question gives a context or scenario (such as data or a program being given), it is important that candidates refer to this context in their response if they are hoping to achieve full marks. For this question, the question text was: ‘Describe two ways to improve the maintainability of <u>this</u> algorithm’ This is a very different question from ‘Describe two ways to improve the maintainability of <u>an</u> algorithm’ where a generic response would suffice.

Question	Answer/Indicative content	Marks	Guidance

b	1 mark each to max 6
	
	• Appropriate use of **both parameters** and **no additional inputs / incorrect overwrites** that affect outcome of algorithm
	• Attempt at selection…
	• …correctly checking if `direction` is "left" **and** subtracting 5 from `position` (or equivalent)
	• …correctly checking if `direction` is "right" **and** adding 5 to `position` (or equivalent)
	• Ensuring position (or equivalent) is between 1 and 512 inclusive
	• **Returning** the updated position
	
	Example
	`if direction == "left" then`
	`position = position - 5`
	`elseif direction == "right" then`
	`position = position + 5`
	`endif`
	
	`if position < 1 then`
	`position = 1`
	`elseif position > 512 then`
	`position = 512`
	`endif`
	
	`return position`
		6 (AO3)	Allow `else` for BP3/4 (validated in question 8a)
			
			Allow <=, >= and equivalents (e.g. <= 0) for BP5.
			
			Do not award BP5 if before BP3 and 4 (otherwise will alter position value)
			
			BP6 only to be given if attempt made at calculating new position. Calculation can be partial/incorrect.
			
			Ignore repeat of function header / end.
			
			Accept flowchart / structured English but must not just repeat the question.
			
			If response uses loop to incorrectly change position multiple times, do not award BP1 (incorrect overwrite)
			
			For minor syntax errors (e.g. missing quotation marks or == for assignment, spaces in variable names) penalise once then FT.
			
			Examiner's Comments
			
			This question was an excellent discriminator in terms of candidate achievement. In particular, correct use of the given parameters (`direction` and `position`) was only seen from the most successful candidates. Many candidates instead started their response by asking for input from the user and therefore overwriting the parameters, losing crucial marks in the process.
			
			As on previous papers, this question provided one mark for any attempt made at a section of the requirement, in this case selection. Candidates who attempted to use an `if` or `select case` statement, even incorrectly or in the wrong context were able to achieve at least this mark; centres should therefore continue to encourage all candidates to attempt each and every question and not leave responses blank.
			
			It was challenging to achieve full marks, but many candidates did because of their excellent levels of practical programming

Question			Answer/Indicative content	Marks	Guidance
					experience in school. **Exemplar 2**  This candidate achieved full marks. The given parameters (direction and position) are both used and not overwritten by inputs before the position is modified depending on the direction given. The position is then validated to make sure that it is between 1 and 512 before being returned. Note the use of `position +=5` to add 5 to the position. This is an entirely acceptable alternative to `position = position + 5`
			Total	10	

Question			Answer/Indicative content	Marks	Guidance
9	a	i	• String • Integer • Real / Float	3 (AO3)	Accept alternative equivalent correct data types (e.g. single/double/decimal for BP3) Do not accept char for BP1 <u>Examiner's Comments</u> This question was completed very well by the vast majority of candidates, showing that the use of data types are now well understood by centres.
		ii	• `theTeam.length() - 1 // 5` • `count` • `studentName` • `True`	4 (AO3)	Accept 6 // `theTeam.length()` for BP1 (Python). Accept alternative length functions e.g. len() Accept **`count = 5`** (and equivalents) for BP1. Accept "True" for BP4. Do not allow obvious spaces in variable names. Ignore capitalisation. <u>Examiner's Comments</u> This was a challenging question revolving around the use of an array that is iterated through to implement a linear search. Many candidates achieved two marks for the first and last points, understanding that the loop would repeat from indexes 0 to 5 and that the value `True` would be returned if the item was found. As usual, allowance was given for off-by-one errors with this loop because of the prevalence of Python in centres and how loops are count controlled loops are handled in this language. It was far less common for candidates to achieve the middle two marks, perhaps because the level of technical knowledge needed was greater. A number of candidates attempted here to fashion a 2D array and refer to multiple indexes, but this was not appropriate given the data structure given. The most challenging mark was certainly the use of `count` as the index of the `theTeam` array, with very few candidates correctly identifying this as the missing element.

Question	Answer/Indicative content	Marks	Guidance
			 Assessment for learning Centres are encouraged to link the topics of arrays (both single and two-dimensional) to count controlled loops to be confident in answering questions like this one. A very common programming exercise is to iterate through every item in an array, either to search for an item, count or add items or as the pre-cursor for a search. Candidates are expected to have significant practical programming experience over the duration of their studies.

<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>

- `javelinThrow` set to 14.3 on line 01 and `yearGroup` set to 10 on line 02
- `score` set to 2 on line 06
- `score` set to 4 on line 11
- "The score is 4" output on line 13 with no additional outputs (allow input statements)

Example

<table>
<tr><th>Line number</th><th>javelinThrow</th><th>yearGroup</th><th>score</th><th>Output</th></tr>
<tr><td>01</td><td>14.3</td><td></td><td></td><td></td></tr>
<tr><td>02</td><td></td><td>10</td><td></td><td></td></tr>
<tr><td>06</td><td></td><td></td><td>2</td><td></td></tr>
<tr><td>11</td><td></td><td></td><td>4</td><td></td></tr>
<tr><td>13</td><td></td><td></td><td></td><td>The score is 4</td></tr>
</table>

Answer may include lines where no changes or output happens (i.e. lines 3, 4, 5, 7, 8, 9, 10, 12).

Where variable doesn't change, current value may be repeated on subsequent lines.
</td><td>4
(AO3)</td><td>

Max 3 if in wrong order or additional (incorrect) changes. Penalise line numbers once then FT.

Allow FT for BP4 for current value of `score`.

BP4 must not include comma. Ignore superfluous spaces. Ignore quotation marks.

Treat any entry in output column as an output, even if "x", "-" or "0".

Examiner's Comments

Trace tables have appeared multiple times in J277 examination papers now and candidates are hopefully familiar with the expectations, which are consistent across series.

Most candidates correctly traced through the given algorithm, which was more accessible than usual due to not containing any loops. Where mistakes were made, this was typically to do with either incorrect line numbers being given for each change (this was penalised once only and then subsequent mistakes with line numbers followed through) or additional incorrect output being given.

Candidates should be encouraged to simply leave boxes blank if no output is given on a particular line. If (for example) 'x' is written, examiners are unsure whether the candidate meant no output or the letter 'x' should be output. This ambiguity would mean that no mark could be given.
</td></tr>
<tr><td></td><td>c</td><td>i</td><td>

- inputs a value from the user and stores/uses
- checks min value (>= 40.0 // < 40)
- checks max value (<=180.0 // > 180)
- …outputs **both** valid / not valid correctly based on checks

Example 1 (checking for valid input)

```
h = input("Enter height
jumped")
if h >= 40 and h <= 180 then
  print("valid")
else
```
</td><td>4
(AO3)</td><td>

Answers using `AND/OR` for BP2 and BP3 must be logically correct e.g. `if height >=40 and height <=180`. Do not accept `if height >=40 and <=180`

Answers using OR will reverse output for BP4 (see examples).

BP4 needs reasonable attempt at **either** BP2 or BP3. Need to be sure what is being checked to be able to decide which way around valid/invalid should be.
</td></tr>
</table>

Question			Answer/Indicative content	Marks	Guidance

			`  print("not valid")`
			`endif`
			
			Example 2 (checking for invalid input)
			`h = input("Enter height jumped")`
			`if h < 40 or h > 180 then`
			`  print("not valid")`
			`else`
			`  print("valid")`
			`endif`
					Allow FT for BP4 if reasonable attempt at validating (must include at least one boundary)
					
					Ignore conversion to int on input. `input` cannot be used as a variable name.
					
					Greater than / less than symbols must be appropriate for a high-level language / ERL. Do not accept => (wrong way around) or ≥ (not available on keyboard). No obvious spaces in variable names. Penalise once and then FT.
					
					Examiner's Comments
					
					This question was done very well by the majority of candidates, with multiple ways of achieving the marks possible.
					
					One common problem was consistent with Question 3 (b), as again multiple conditions could be evaluated using a single `if` statement. Candidates needed to refer to their input value for each comparison if they were to achieve full marks. Another common problem was with the boundaries used; the tests must check 40 to 80 inclusive (for valid response) to be marked as correct. Obviously, if an input on these boundaries would produce the wrong output then full marks could not be given.
					
					One final common problem involved the use of greater than or equal to signs (and also less than or equal to). The common signs used in mathematics are not available on a typical keyboard and so would not be allowed in Section B of this paper. Instead, >= or <= should be used, and these should be the correct way around.

		ii	• Any normal value (between 40 and 180 inclusive) • 40.0 // 180.0 • Any value less then 40 // any value greater that 180 // any non-numeric value	3 (AO3)	No need to include decimals, e.g. accept 50. Ignore cm if given. Answer must be actual data (e.g. 50) and not description of data (e.g. "a value between 40 and 180"). If descriptions given, do not accept this as non-numeric for BP3

Question			Answer/Indicative content	Marks	Guidance
	d		• `TeamName` only in first space • `TblResult` in second space • `WHERE` • …`YearGroup = 11`	4 (AO3)	Max 3 if not in correct order / includes other logical errors. Ignore capitals. Do not accept * or additional fields for BP1 Spelling must be accurate (e.g. not TblResult<u>s</u>). No spaces in field names, penalise obvious spaces once and then FT. Allow quotation marks around field names, table name and 11 Accept == for BP4 (invalid SQL but works in some environments) <u>Examiner's Comments</u> A number of candidates struggled with this question. Problems included spaces in field names, misspelling of the table name (such as `TblResult`<u>s</u> plural when `TblResult` singular was given) or misunderstanding of the WHERE clause. Allowance was given where == was used for comparison and examiners were instructed to allow this (as this is used for comparison in high-level languages such as Python), although this is incorrect as defined in the most recent ANSI SQL standards.

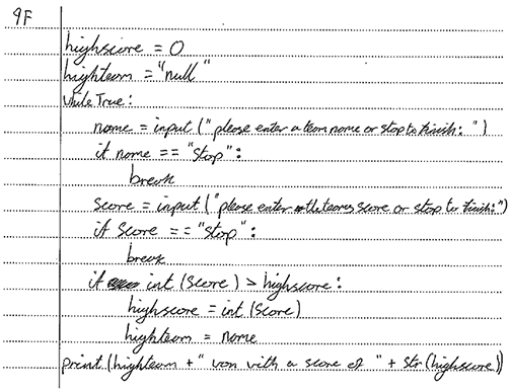
Question			Answer/Indicative content	Marks	Guidance
	e	i	• any **example** of simplification / removing data or focussing on data (in the design) Examples : - “focus on student names and events” - “ignore data such as students’ favourite subjects” - “store year groups instead of ages or DOB” - “shows student IDs instead of full student details”	1 (AO3)	Must be applicable to this program (in the context of students and a sports day), **not** a generic description of what abstraction is. Give BOD where this is unclear. Examiner’s Comments Both questions here asked about abstraction and decomposition of the sports day program. As explained previously in this report, where a scenario or context is given, candidates are expected to use this context. No marks were given by examiners for generic definitions of what the term abstraction or decomposition means. Abstraction in the sports day program could have been for focusing on anything sensible (such as event names) or removing/ignoring anything sensible (such as showing student IDs instead of names). Where the context of the sports day was used, candidates were generally successful in achieving this mark. Decomposition use was more tricky to correctly identify, as many candidates simply referred to how already separate data was stored. Where this extended to true decomposition (such as breaking down data into multiple tables, splitting up event data, etc.) this was credited but the average candidate fell short here. A much more successful approach was to discuss the decomposition of the program, such as having a separate algorithm for each event. Candidates attempting this angle of response did very well. Examiners were instructed for both questions to be generous in deciding whether candidates had indeed referred to the sports day context.
		ii	• any **example** of breaking down the **program** into sections/subroutines • any **example** of breaking down the **database** into tables Examples : - “splits the program up into different events” - “separates the validation routines into subroutines”	1 (AO3)	Must be applicable to this program, not a generic description of what decomposition is. Give BOD where this is unclear. Do not give answers discussing splitting into **fields** (e.g. split into StudentID, YearGroup, etc). BOD if answer discusses one table but suggests other tables could be used.

Question	Answer/Indicative content	Marks	Guidance
	- “breaks the database down into a table per event”		Do not give answers relating simply to data being split into smaller groups unless this clearly relates to how data is decomposed into **tables** in the DB. Allow reference to sports day to mean sports day program. <u>Examiner’s Comments</u> Both questions here asked about abstraction and decomposition of the sports day program. As explained previously in this report, where a scenario or context is given, candidates are expected to use this context. No marks were given by examiners for generic definitions of what the term abstraction or decomposition means. Abstraction in the sports day program could have been for focusing on anything sensible (such as event names) or removing/ignoring anything sensible (such as showing student IDs instead of names). Where the context of the sports day was used, candidates were generally successful in achieving this mark. Decomposition use was more tricky to correctly identify, as many candidates simply referred to how already separate data was stored. Where this extended to true decomposition (such as breaking down data into multiple tables, splitting up event data, etc.) this was credited but the average candidate fell short here. A much more successful approach was to discuss the decomposition of the program, such as having a separate algorithm for each event. Candidates attempting this angle of response did very well. Examiners were instructed for both questions to be generous in deciding whether candidates had indeed referred to the sports day context.

Question	Answer/Indicative content	Marks	Guidance

f	• Input team name AND score **and** store / use separately
	• Attempt at using iteration…
	• …to enter team/score until "stop" entered
	• Calculates highest score
	• Calculates winning team name…
	• …Outputs highest score **and** team name
	
	<u>Example 1</u>
	`highscore = 0`
	`while team != "stop"`
	`  team = input("enter team name")`
	`  score = input("enter score")`
	`  if score > highscore then`
	`  highscore = score`
	`  highteam = team`
	`  endif`
	`endwhile`
	`print(highscore)`
	`print(highteam)`
	
	<u>Example 2 (alternative)</u>
	`scores = []`
	`teams = []`
	`while team != "stop"`
	`  team = input("enter team name")`
	`  score = input("enter score")`
	`  scores.append(score)`
	`  teams.append(team)`
	`endwhile`
	`highscore = max[scores]`
	`highteam = teams[scores.index(highscore)]`
	`print(highscore)`
	`print(highteam)`
		6 (AO3)	For BP3, allow "stop" to be entered for either team name or score (or both). Allow third input (e.g. "do you wish to stop?")
			
			Allow use of `break` (or equivalent) to exit loop for BP3.
			
			Allow use of recursive function(s) for BP2/3.
			
			Initialisation of variables not needed - assume variables are 0 or empty string if not set.
			
			Ignore that multiple teams could get the same high score, assume only one team has the highest score.
			
			BP4/5 could be done in many ways – see examples. Allow any logically valid method. Allow use of max/sum functions and use of arrays/lists.
			
			FT for BP6 if attempt made at calculating highest score/name
			
			If answer simply asks for multiple entries (not using iteration), BP2 and 3 cannot be accessed but all others available.
			
			For minor syntax errors (e.g. missing quotation marks or == for assignment, spaces in variable names) penalise once then FT.
			
			`input` cannot be used as a variable name.
			
			<u>Examiner's Comments</u>
			
			The final question in Section B is expected to be challenging and this proved to be the case, although again was perhaps more accessible than previous papers' final questions.
			
			Marks were available for inputs (one mark) and correctly iterating over as required (two marks), with these three marks proving to be the easiest to achieve. The next three marks required significant processing in terms of calculating the highest score and team name from multiple values entered by the user. The vast majority of candidates simply kept a

Question	Answer/Indicative content	Marks	Guidance

			running 'highest score' and updated this on each iteration. Where this was attempted, it was mostly successful. Other candidates attempted more complex solutions, including adding data to arrays and then calculating highest values; where this was done successfully, this of course achieved full marks but frequently small logic mistakes meant that not all marks were given. Centres should encourage candidates to keep their responses simple and not produce over-elaborate solutions if a simpler alternative is available.
			
			A common mistake was where candidates attempted to use loops in the style of `for x in list :`. In this case, the variable `x` is a reference to an item in the array and not an index. It would therefore not be appropriate to try to access `list[x]` later in the code.
			
			A significant number of candidates were able to create a solution that fully met the requirements of the question, doing so in an elegant and efficient manner. This is extremely pleasing and show excellent understanding, produced from excellent teaching and significant amounts of practical programming experience.
			
			Exemplar 3
			
			 
			9F
			highscore = 0
			highteam = "null"
			while True:
			name = input("please enter a team name or stop to finish: ")
			if name == "stop":
			break
			score = input("please enter with teams score or stop to finish:")
			if score == "stop":
			break
			if ~~over~~ int(score) > highscore:
			highscore = int(score)
			highteam = name
			print(highteam + " won with a score of " + str(highscore))
			
			The candidate response shown here achieved six out of six marks. Both the `name` and `score` are input as required, with this being inside a while loop. Perhaps unconventionally (but acceptably), the candidate has used a `break` command to end the loop (which otherwise is infinite) upon stop being entered. This could have been more

Question			Answer/Indicative content	Marks	Guidance
					elegantly rewritten as `while name != 'stop'` but this would not have achieved any further marks. Within the loop, the candidate uses two variables to keep track of the current highest score and associated team, before printing these out in a message once the loop has ended.
			Total	30	