Chhetri Academy

chhetriacademy.com

GCSE COMPUTER SCIENCE

2.2

OCR J277 | Specification point 2.2.1

Programming fundamentals

Untiered

Answers and examiner comments

5	2018-2025	Untiered
question groups	available papers	tier

Using this booklet

1. Answer numbers match the companion question booklet.
2. Follow the original question number on marking tables.
3. Examiner comments are included where available.

Name _____ **Date** _____

Some entries use full source marking material in an appendix. Any unavailable mark schemes are identified beside the relevant answer.

2	**(a)**	**(i)**	• 2, 3, 4	1	All three numbers needed in the correct order (with no other numbers) for mark.
2	**(a)**	**(ii)**	• 15	1	Accept 3 x 5
2	**(b)**		1 mark per bullet, max 2 • Sequence • Iteration / loops / repetition	2	Ignore spelling. Do not allow examples (eg FOR loop / WHILE loop)
2	**(c)**	**(i)**	1 mark per bullet, max 2 • A (name/identifier for a) **memory location** • used to (temporarily) **holds/contains/stores** data / value // is assigned a value • that **can be** changed / **possible** to change (while the program is running)	2	Do not accept “will change” for bullet point 4. Do not allow “holds/stores something” or “holds/stores information” for bullet point 2. Do not accept name / identifier without reference to a memory location. Do not accept “a value given a name” or equivalent.
2	**(c)**	**(ii)**	1 mark per bullet, max 2 • k • p • m	2	Ignore capitalisation. Correct answer only. Do not allow other code in answer.

Question			Answer	Mark	Guidance
4	(a)		• Contents of variable **can** be changed; contents of constants **cannot** be changed (while the programming is running)	1	Both sides needed for mark.
	(b)	(i)	• 16	1	
		(ii)	• 2	1	
		(iii)	• 9	1	
	(c)	(i)	• second.substring(3,5)	1	Ignore print / lack of print. Allow other suitable methods of string manipulation as long as variables used. Allow any valid method that extracts rightmost 5 or 6 characters of `second` variable.
		(ii)	• first.substring(0,8)	1	Ignore print / lack of print. Allow other suitable methods of string manipulation as long as variables used. Allow any valid method that extracts leftmost 8 or 9 characters of `first` variable.
		(iii)	• first.substring(9,7) + " " + second • "Science " + second • first.substring(9,7) + " is great"	1	Ignore print / lack of print. Allow other suitable methods of string manipulation as long as variables(s) used. Allow alternative concatenation symbols (e.g. & or **.**). Allow concatenation functions Must have correct spacing in outcome.

Question			Answer	Mark	Guidance
4	(a)		Any two bullet points for one mark each: • Add comments • Name variables sensibly • **Put into** subroutine / procedure / function • Use loop / iteration	2 **(AO2 1b)**	Do not accept indentation (no code to sensibly indent in this example) "Use a subroutine" is not enough. Must be clear that existing code will be put into a new subroutine.
4	(b)	(i)	• Multiplication • Division	2 **(AO1 1a)**	Accept other correct answers that mean the same Accept floor / integer division // division with no remainder (Python v2.x)
4	(b)	(ii)	• high-level • stops // crashes • no • executable • without	5 **(AO1 1b, AO2 1b)**	Ignore spelling errors.

4	**(c)**		• input and stores/uses value with message
			• attempt at repeating…
			• …correctly repeats number of times given as input
			• …correctly take number as input within loop and calculates total of these numbers
			• …correctly calculate an average (total/num)
			• Output both total and average
				6
				(AO3 2b, AO3 2c)
					e.g.
					`num = input("Enter how many numbers")`
					`for x = 1 to num`
					`  temp = input("Enter a number")`
					`  total = total + temp`
					`next x`
					`print(total)`
					`print(total / num)`
					
					If flow chart used, correct shapes needed.
					
					Allow tolerance of 1 with number of loops for BP3 with `for` loops
					
					BP1 requires input with a message (can be two statements, e.g. print and then input or combined. Input must be stored or used.
					
					BP3, 4, 5 must be logically correct to be credited Ignore non-initialisation of total
					
					BP 5 can be given as FT as long as an attempt has bee made at working out a total within the loop.
					
					BP6 can be given as FT long as attempt made at total and average (not necessarily in a loop)

SECTION B

Note – algorithm questions in Section B (5bi, 5ci, 5cii, 5e) require candidates to answer **using OCR Reference Language or a high-level programming language**. Candidates **cannot** answer using structured English or flowcharts and responses of this nature should be marked as NAQ.

Candidates **do not** need to state which language they are using. Because of this, you must not assume use of a particular language. Mark for logical correctness and **not** for the syntax of any particular language. Ignore case sensitivity. Variable names must be correct/consistent.

<table>
<tr><th colspan="3">Question</th><th>Answer</th><th>Mark</th><th>Guidance</th></tr>
<tr><td>1</td><td>(a)</td><td></td><td>1 mark per row

<table>
<tr><th>Statement</th><th>Low-level</th><th>High-level</th></tr>
<tr><td>The same language can be used on computers that use different hardware</td><td></td><td>✓</td></tr>
<tr><td>It allows the user to directly manipulate memory</td><td>✓</td><td></td></tr>
<tr><td>It allows the user to write English-like words</td><td></td><td>✓</td></tr>
<tr><td>It always needs to be translated into object code or machine code</td><td></td><td>✓</td></tr>
</table></td><td>4
(AO1 1b)</td><td>No mark if more than 1 tick for that row.

Allow other indications of choice (e.g. cross) as long as clear.</td></tr>
<tr><td>1</td><td>(b)</td><td></td><td><u>total</u> = num1 + num2</td><td>1
(AO3 2b)</td><td>Allow other logically valid responses that result in total storing the correct value. Accept other suitable assignment operators (e.g. ←)

e.g.
total = sum(num1, num2)

total = num2 + num1

x = num1 + num2
total = x

Ignore any values given to the variable. Ignore capitalisation and minor misspelling. Ignore superfluous code that does not affect outcome.</td></tr>
</table>

Question			Answer	Mark	Guidance
1	(c)	(i)	`print(12 ^  2)`	1 (AO2 1a)	Accept ** or other sensible operator that indicates raising to a power. If pseudocode operator given, must be a single word/symbol (e.g. `pow`), not containing spaces.
1	(c)	(ii)	`if number MOD 2 == 0 then`	1 (AO2 1a)	Accept **%** or other sensible operator that indicates modulus If pseudocode operator given, must be a single word/symbol (e.g. `modulo`), not containing spaces.
1	(c)	(iii)	`difference = measurement1 - measurement2`	1 (AO2 1a)	Accept other sensible operator that indicates subtraction. If a pseudocode operator given, must be a single word/symbol (e.g. `minus`), not containing spaces.

<table>
<tr><th colspan="3">Question</th><th>Answer</th><th>Mark</th><th>Guidance</th></tr>
<tr><td>1</td><td>(d)</td><td></td><td>1 mark each:
• Start is set to 3 on line 01 and 3 is output on line 03.
• 2, 1 and 0 are output on next 3 iterations with start updated to 2, 1, 0, -1 on correct line numbers.
• Finished is output on line 06

<table>
<tr><th>Line</th><th>start</th><th>Output</th></tr>
<tr><td>01</td><td>3</td><td></td></tr>
<tr><td>03</td><td></td><td>3</td></tr>
<tr><td>04</td><td>2</td><td></td></tr>
<tr><td>03</td><td></td><td>2</td></tr>
<tr><td>04</td><td>1</td><td></td></tr>
<tr><td>03</td><td></td><td>1</td></tr>
<tr><td>04</td><td>0</td><td></td></tr>
<tr><td>03</td><td></td><td>0</td></tr>
<tr><td>04</td><td>-1</td><td></td></tr>
<tr><td>06</td><td></td><td>Finished</td></tr>
</table>
BP1
BP2
BP3</td><td>3
(AO3 2c)</td><td>Ignore lines 02 and 05 in answer unless these change or output any values.

Candidate may repeat start value when unchanged, this is acceptable.

Penalise incorrect or missing line numbers or additional output once only then FT. This includes where variable change and output appear on the same line.

-1 must not be output for BP2

Penalise missing or incorrect output once only for BP1 and FT for missing or incorrect output for BP2.

Finished may be with or without quotes. Ignore case or minor spelling error.

Max 2 marks if any incorrect output or changes to <code>start</code>.

Do not accept calculated values of start (e.g. <code>3-1</code>)</td></tr>
</table>

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 4 (a)

4 Jack is writing a program to add up some numbers. His first attempt at the program is shown.

```
a = input("Enter a number")
b = input("Enter a number")
c = input("Enter a number")
d = input("Enter a number")
e = input("Enter a number")
f = (a + b + c + d + e)
print(f)
```

(a) Give **two** ways that the maintainability of this program could be improved.

1 .....

.....

2 .....

.....

[2]

This question asked about maintainability. It was important that a candidate's response referred to the code given.

"Give two ways that maintainability of **a program** could be improved" is a different question than the one asked.

Because of this, responses that were accepted must genuinely improve the maintainability of the program shown. One very common wrong response was indentation; although this would be useful generically, there was no code presented that would benefit from being indented.

Many candidates were able to give responses such as use of comments and sensibly named variables that would genuinely improve the given program. Where candidates described putting the given code inside a newly defined subroutine. This response was credited. However some responses simply said "use a subroutine". This was not enough.

Question 4 (b) (i)

(b) Jack's program uses the addition (+) arithmetic operator. This adds together two numbers.

(i) State the purpose of each of the arithmetic operators in the table.

Arithmetic operator	Purpose
*	
/	

[2]

This question was answered well by the majority of candidates.

Question 4 (b) (ii)

(ii) Complete the description of programming languages and translators by writing the correct term from the box in each space.

continues	crashes	debugging	error	executable
high-level	interpreter	language	low-level	many
no	one	stops	with	without

Jack writes his program in a ..... language. This needs to be translated into assembly or machine code before it can be executed. This is done using a translator.

One type of translator is an interpreter. This converts one line of code and then executes it, before moving to the next line. It ..... when an error is found, and when corrected continues running from the same position. This translator is helpful when debugging code.

A second type of translator is a compiler. This converts all of the code and produces an error report. The code will not run until there are ..... errors.

The ..... file produced can be run ..... the compiler.

[5]

This question was answered well by the majority of candidates.

Question 4 (c)

(c) Jack decides to improve his program. He wants to be able to input how many numbers to add together each time the algorithm runs, and also wants it to calculate and display the average of these numbers.

Write an algorithm to:

- ask the user to input the quantity of numbers they want to enter and read this value as input
- repeatedly take a number as input, until the quantity of numbers the user input has been entered
- calculate and output the total of these numbers
- calculate and output the average of these numbers.

.....

.....

.....

.....

.....

..... **[6]**

As this question appeared in Section A, candidates are free to respond in any suitable way, including using flowcharts, structured English, pseudocode or a high-level language. The majority of candidates who scored highly tended to use a high-level language consistently.

The question is already decomposed for candidates and many were able to use these bullet points to build a solution that achieved the majority (or all) marks available.

Where mistakes were made, these tended to be with repeating code. For example, many candidates repeated the process of adding values up without repeatedly asking for a new number (and therefore continually adding the same number).

Other candidates used condition-controlled iteration to repeat the process but did not make sure that the condition being evaluated ever returned a `False` value, therefore repeating infinitely.

Where a mistake was made in one section (such as with iteration), examiners were instructed to use FT (follow through) marks where possible if later sections were logically constructed. This made sure that marks could be given where appropriate.

A few candidates were not able to access many of the marks available in this question, suggesting that they would benefit from more practical programming time in lessons.

Section B overview

Section B consists of multiple sub-questions all centred around one scenario (in this case, a hotel). Where algorithms are asked for, candidates must respond using either OCR Exam Reference Language or a high-level language that has been studied. Where responses were presented using flowcharts or structured English, no marks were given.

Candidates do not have to state which language they are using and so responses are marked for their logical correctness and consistency.

It is important for centres and candidates to understand that examiners are told to **not** penalise responses because they would not work in a particular language.

Question 1 (a)

1 (a) The table contains four statements about programming languages.

Tick (✓) **one** box in each row to identify whether each statement describes a low-level programming language or a high-level programming language.

Statement	Low-level	High-level
The same language can be used on computers that use different hardware		
It allows the user to directly manipulate memory		
It allows the user to write English-like words		
It always needs to be translated into object code or machine code		

[4]

This question was answered well by many candidates. The strongest responses showed a good understanding of the difference between low-level and high-level languages. Incorrect responses tended to be on the first row related to portability. A high-level language such as Python is portable, with translators available for many different types of processors. A low-level language is specific to one type of processor.

Question 1 (b)

(b) The variables `num1` and `num2` store integers.

Write pseudocode to add the integers stored in `num1` and `num2`. Store the result in a variable with the identifier `total`

.....

..... **[1]**

Candidates appear to be getting more confident at answering simple programming/pseudocode questions such as this. The majority of responses included code written to produce the required outcome. The use of multiple steps was allowed.

Question 1 (d)

(d) Read the following pseudocode algorithm:

```
01  start = 3
02  do
03    print(start)
04    start = start - 1
05  until start == -1
06  print("Finished")
```

Complete the following trace table for the given algorithm.

Line number	`start`	Output

[3]

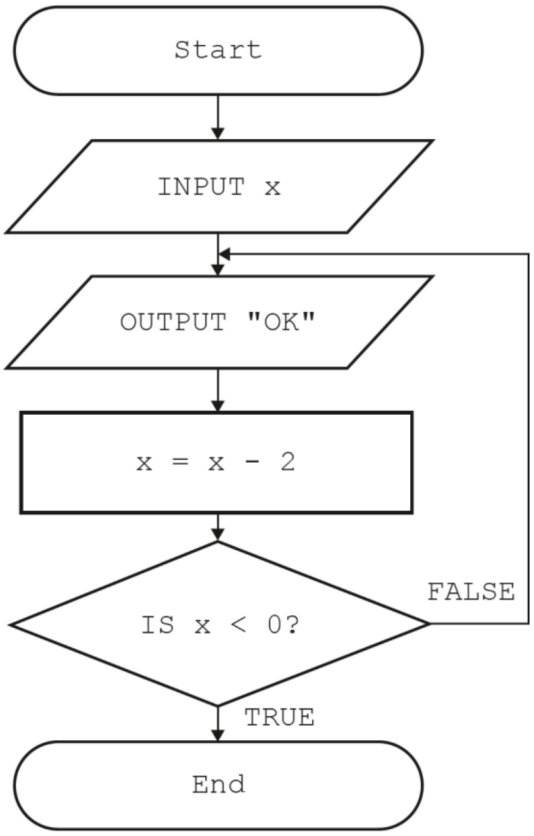
This question assessed candidates' ability to trace through and understand the steps taken by an algorithm. This also tested their understanding of condition-controlled loops. Many responses were very successful with this and achieved full marks.

Mistakes tended to be with identifying the line number where each change occurred or outputting values that were not actually output (e.g. -1).

Examiners were instructed to only penalise a misunderstanding once. Where (for example) line numbers were incorrect, this would still have allowed 2 out of 3 marks to be achieved.

Question 2 (a)

2 This flowchart uses iteration.



(a) Complete the table by identifying how many times "OK" will be output for when each value is input.

Input value	Number of times "OK" is output
3	
10	

[2]

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.

Question 2 (b) (i)

(b) Iteration is a programming construct.

(i) Identify **and** describe **two** different examples of iteration that can be used in a high-level language.

1 .....

.....

.....

2 .....

.....

.....

[4]

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.

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.

Question 2 (b) (ii)

(ii) Give the names of **two** other programming constructs that are used in a high-level language.

1 .....

2 .....

[2]

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).

Question 2 (c)

(c) A programmer implements the flowchart as a program in a high-level language using an Integrated Development Environment (IDE).

Identify and describe **two** tools that the programmer could use in the IDE to create a program from the flowchart.

Tool 1 .....

Description .....

.....

Tool 2 .....

Description .....

.....

[4]

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.