A-level
COMPUTER SCIENCE
7517/1

Paper 1

Mark scheme

June 2025

Version: 1.0 Final

256A7517/1/MS

Mark schemes are prepared by the Lead Assessment Writer and considered, together with the relevant questions, by a panel of subject teachers. This mark scheme includes any amendments made at the standardisation events which all associates participate in and is the scheme which was used by them in this examination. The standardisation process ensures that the mark scheme covers the students' responses to questions and that every associate understands and applies it in the same correct way. As preparation for standardisation each associate analyses a number of students' scripts. Alternative answers not already covered by the mark scheme are discussed and legislated for. If, after the standardisation process, associates encounter unusual answers which have not been raised they are required to refer these to the Lead Examiner.

It must be stressed that a mark scheme is a working document, in many cases further developed and expanded on the basis of students' reactions to a particular paper. Assumptions about future mark schemes on the basis of one year's document should be avoided; whilst the guiding principles of assessment remain constant, details will change, depending on the content of a particular examination paper.

No student should be disadvantaged on the basis of their gender identity and/or how they refer to the gender identity of others in their exam responses.

A consistent use of 'they/them' as a singular and pronouns beyond 'she/her' or 'he/him' will be credited in exam responses in line with existing mark scheme criteria.

Further copies of this mark scheme are available from aqa.org.uk

Level of response marking instructions

Level of response mark schemes are broken down into levels, each of which has a descriptor. The descriptor for the level shows the average performance for the level. There are marks in each level.

Before you apply the mark scheme to a student's answer read through the answer and annotate it (as instructed) to show the qualities that are being looked for. You can then apply the mark scheme.

Step 1 Determine a level

Start at the lowest level of the mark scheme and use it as a ladder to see whether the answer meets the descriptor for that level. The descriptor for the level indicates the different qualities that might be seen in the student's answer for that level. If it meets the lowest level then go to the next one and decide if it meets this level, and so on, until you have a match between the level descriptor and the answer. With practice and familiarity you will find that for better answers you will be able to quickly skip through the lower levels of the mark scheme.

When assigning a level you should look at the overall quality of the answer and not look to pick holes in small and specific parts of the answer where the student has not performed quite as well as the rest. If the answer covers different aspects of different levels of the mark scheme you should use a best fit approach for defining the level and then use the variability of the response to help decide the mark within the level, ie if the response is predominantly level 3 with a small amount of level 4 material it would be placed in level 3 but be awarded a mark near the top of the level because of the level 4 content.

Step 2 Determine a mark

Once you have assigned a level you need to decide on the mark. The descriptors on how to allocate marks can help with this. The exemplar materials used during standardisation will help. There will be an answer in the standardising materials which will correspond with each level of the mark scheme. This answer will have been awarded a mark by the Lead Examiner. You can compare the student's answer with the example to determine if it is the same standard, better or worse than the example. You can then use this to allocate a mark for the answer based on the Lead Examiner's mark on the example.

You may well need to read back through the answer as you apply the mark scheme to clarify points and assure yourself that the level and the mark are appropriate.

Indicative content in the mark scheme is provided as a guide for examiners. It is not intended to be exhaustive and you must credit other valid points. Students do not have to cover all of the points mentioned in the Indicative content to reach the highest level of the mark scheme.

An answer which contains nothing of relevance to the question must be awarded no marks.

A-level Computer Science

Paper 1 (7517/1) – applicable to all programming languages A, B, D and E

June 2025

The following annotation is used in the mark scheme:

; – means a single mark
// – means an alternative response
/ – means an alternative word or sub-phrase
A. – means an acceptable creditworthy answer
R. – means reject answer as not creditworthy
NE. – means not enough
I. – means ignore
DPT. – means 'Don't penalise twice'. In some questions a specific error made by a candidate, if repeated, could result in the loss of more than one mark. The **DPT** label indicates that this mistake should only result in a candidate losing one mark, on the first occasion that the error is made. Provided that the answer remains understandable, subsequent marks should be awarded as if the error was not being repeated.

Examiners are required to assign each of the candidate's responses to the most appropriate level according to **its overall quality**, and then allocate a single mark within the level. When deciding upon a mark in a level, examiners should bear in mind the relative weightings of the assessment objectives

eg

In question **04.1**, the marks available for the AO3 elements are as follows:

AO3 (design)	4 marks
AO3 (programming)	8 marks

Where a candidate's answer only reflects one element of the AO, the maximum mark they can receive will be restricted accordingly.

Question			Marks
01	1	**Mark is for AO1 (understanding)** Hashing algorithm/function will determine the memory location/index used; No need to search through records to find the required one; **Max 1**	1
01	2	**All marks AO1 (understanding)** 1. Apply hashing algorithm/function; 2. To (unique/primary) key; **NE.** data 3. Insert data at the index (**A.** address/memory location) obtained from this process; 4. If there is already data there, use a suitable method for collision handling; **A.** by example	4
01	3	**Mark is for AO1 (understanding)** More collisions occur meaning direct access will not happen as frequently // more collisions occur meaning more items will not be stored at the calculated index (**A.** address/memory location) (given by the hashing function/algorithm); **NE.** more collisions occur **A.** description of a collision resolution method being used, eg need for a linear search if multiple results have the same result from the hashing function	1
01	4	**Mark is for AO1 (understanding)** As list increases in size the maximum number of comparisons increases at the same rate; **A.** in the worst case, `n` comparisons will be needed **NE.** n comparisons are needed	1
01	5	**Mark is for AO1 (understanding)** As list increases in size the maximum number of comparisons increases at a decreasing rate; Each comparison halves the number of remaining items that need to be considered // each comparison halves the size of the list that has to still be searched through;	1
01	6	**Mark is for AO1 (understanding)** Can be used on unsorted lists;	1

01	**7**	**Mark is for AO1 (understanding)** Both have polynomial (or better) time complexity; Neither have exponential (or worse) time complexity; As the size of the list grows, the increase in the time taken is reasonable; **Max 1**	**1**
01	**8**	**Mark is for AO2 (apply)** 99;	**1**

Question			**Marks**
02		**All marks AO1 (knowledge)** Splitting a problem into smaller sub-problems; So that each sub-problem accomplishes an identifiable task, which might itself be further divided // Repeating this process until each sub-problem performs a single task;	**2**

Question			**Marks**

03	**1**	**Mark is for AO1 (knowledge)**
		
		Find the shortest path between two nodes (in a graph) //
		find the shortest path from a node to all other nodes (in a graph) //
		find the lowest cost (**A.** distance) path between two nodes (in a graph) //
		find the lowest cost (**A.** distance) path from a node to all other nodes (in a graph);
			1

03	**2**	**All marks for AO2 (analyse)** Not connected; It has cycles; **Max 1** if any additional incorrect answers eg directed **A.** by example	**2**

03	**3**	**All marks for AO2 (analyse)**	**2**

	1	2	3	4	5	6	7
1	0	1	1	0	0	0	1
2		0	0	0	0	1	0
3			0	0	0	1	1
4				0	1	0	0
5					0	0	0
6						0	0
7							0

Alternative answer

	1	2	3	4	5	6	7
1	0	1	1	0	0	0	1
2	1	0	0	0	0	1	0
3	1	0	0	0	0	1	1
4	0	0	0	0	1	0	0
5	0	0	0	1	0	0	0
6	0	1	1	0	0	0	0
7	1	0	1	0	0	0	0

Alternative answer

	1	2	3	4	5	6	7
1	0						
2	1	0					
3	1	0	0				
4	0	0	0	0			
5	0	0	0	1	0		
6	0	1	1	0	0	0	
7	1	0	1	0	0	0	0

Mark as follows:

1 mark: any three rows correct **I.** missing values for edges from node to itself **A.** use of third symbol for edges from a node to itself
2 marks: correct table

A. any suitable indicators instead of 1 and 0

Max 1 if more than two symbols used

<table>
<tr><td>03</td><td>4</td><td>Mark is for AO2 (analyse)

The start and end (nodes) are the same;
There are no unvisited nodes in the graph that can be reached from the start node;

NE. code without a description

Max 1</td><td>1</td></tr>
<tr><td>03</td><td>5</td><td>All marks AO2 (apply)

<table>
<tr><th></th><th></th><th></th><th colspan="7">V</th></tr>
<tr><th>j</th><th>Subroutine call to Traverse</th><th>i</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th></tr>
<tr><td>1</td><td></td><td></td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td><td>0</td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td><td>0</td><td></td><td></td><td></td></tr>
<tr><td>5</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td></td><td></td></tr>
<tr><td>6</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td></td></tr>
<tr><td>7</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr>
<tr><td>1</td><td>Traverse(3, 7)</td><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>Traverse(1, 7)</td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>Traverse(2, 7)</td><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>Traverse(6, 7)</td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td><td></td></tr>
<tr><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>Traverse(2, 7)</td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>Traverse(1, 7)</td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>3</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>Traverse(7, 7)</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>Traverse(1, 7)</td><td>3</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>Traverse(3, 7)</td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

Mark as follows:

1. <code>j</code> has values 1 to 7 then 1
2. All values of <code>V</code> set to 0
3. First call to <code>Traverse</code> is <code>Traverse(3,7)</code>, first value of <code>i</code> is 1 and <code>V[3]</code> is set to 1 I. <code>IsPath(3, 7)</code>
4. Final values of columns 1 to 7 for <code>V</code>
5. Second and third calls to <code>Traverse</code>
6. Remaining calls to <code>Traverse</code>

I. unnecessary repeated values in a column</td><td>6</td></tr>
</table>

Max 5 if any errors
I. missing parts in blue font

Question			Marks
04	1	**4 marks for AO3 (design) and 8 marks for AO3 (programming)** **Mark Scheme**	12

Level	Description	Mark Range
4	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution that meets most of the requirements. All of the appropriate design decisions have been taken. To award 12 marks, all of the requirements must be met.	10–12
3	There is evidence that a line of reasoning has been followed to produce a logically structured program. The program displays relevant prompts, inputs the required data, has at least one iterative structure and at least one selection structure and uses appropriate variables to store most of the needed data. An attempt has been made to test for alphabetic/non-alphabetic characters, although this may not work correctly under all circumstances. The solution demonstrates good design work as most of the correct design decisions have been made.	7–9
2	A program has been written and some appropriate, syntactically correct programming language statements have been written. There is evidence that a line of reasoning has been partially followed as, although the program may not have the required functionality, it can be seen that the response contains some of the statements that would be needed in a working solution. There is evidence of some appropriate design work as the response recognises at least one appropriate technique that could be used by a working solution, regardless of whether this has been implemented correctly.	4–6
1	A program has been written and a few appropriate programming language statements have been written, but there is no evidence that a line of reasoning has been followed to arrive at a working solution. The statements written may or may not be syntactically correct. It is unlikely that any of the key design elements of the task have been recognised.	1–3

Guidance

Evidence of AO3 design – 4 points:

Evidence of design to look for in responses.

1. Identifying that an iteration structure repeating based on the length of the plaintext entered by the user is needed.
2. Identifying that nested iteration is needed to encrypt the plaintext. (**A.** equivalent eg single iteration with use of reminder division to "wrap around")
3. Identifying that appropriate variables are needed to store the plaintext, ciphertext and number of columns. **A.** ciphertext not stored if clear attempt made to display the encrypted characters
4. Recognising the need to convert a character into a character code and comparing that with at least one of 65, 90, 97, 122 // recognising the need to check if a character is in a list of allowed values // recognising the need to compare a character with at least one of z, Z, a or A.

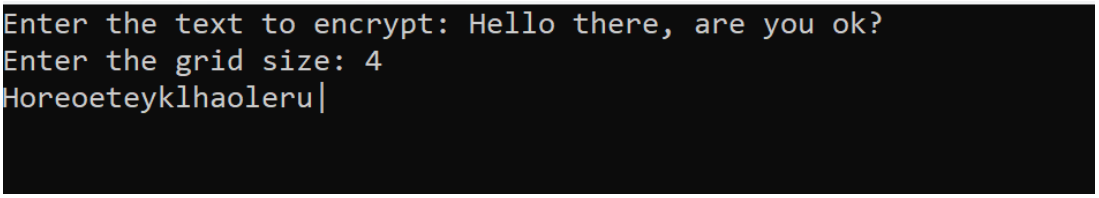
Note that AO3 (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not, and regardless of whether the solution works.

Evidence for AO3 programming – 8 points:

Evidence of programming to look for in response.

5. User input for plaintext being assigned to appropriate variable.
6. User input for number of columns being assigned to appropriate variable.
7. Removes a non-alphabetic character from the plaintext **R.** if it could remove an alphabetic character
8. Repeats their attempt at mark point 7 for every character in the plaintext.
9. Creates a string containing only alphabetic characters from the plaintext.
   A. only working for one of uppercase or lowercase.
10. Iteration structure that repeats once for each "column" // iteration structure that repeats once for each "row" // iteration structure that repeats until (alphabetic) plaintext is same length as ciphertext.
11. Creates/displays ciphertext that starts with all characters from the first "column" in the correct order
    A. non-alphabetic characters included.
12. After attempt at adding characters in first "column" it adds the first character in the second "column" to the plaintext.
    A. non-alphabetic characters included.

Max 11 if any errors

04	**2**	**Mark is for AO3 (evaluate)** ****** SCREEN CAPTURE ****** *Must match code from **04.1**, including messages on screen capture(s) matching those in code.* *Code for **04.1** must be sensible.* Screen captures showing the correct user inputs with the expected string being displayed;  	**1**

Question			Marks
05	1	**All marks AO1 (understanding)** Simpler for a machine/computer to evaluate // simpler to code algorithm; **A.** Easier **R.** understand Do not need brackets; Operators appear in the order needed for calculation; No need for order of precedence of operators; No need to backtrack when evaluating; **A.** RPN expressions cannot be ambiguous as **BOD** **Max 2**	**2**
05	2	**All marks AO1 (understanding)** Consists of key-value pairs; Keys are unique // value is looked up by providing the key;	**2**
05	3	**Mark is for AO2 (analyse)** Stack; **A.** LIFO // Last In, First Out	**1**

<table>
<tr><th colspan="2">Question</th><th></th><th>Marks</th></tr>
<tr><td>06</td><td>1</td><td>All marks AO2 (analyse)

<table>
<tr><th>Initial state</th><th>Input(s)</th><th>New state</th></tr>
<tr><td>S0</td><td>0–9</td><td>S1</td></tr>
<tr><td>S1</td><td>0–9</td><td>S1</td></tr>
<tr><td>S1</td><td>+ – * /</td><td>S2</td></tr>
<tr><td>S2</td><td>0–9</td><td>S3</td></tr>
<tr><td>S3</td><td>0–9</td><td>S3</td></tr>
<tr><td>S3</td><td>+ – * /</td><td>S2</td></tr>
</table>

Mark as follows:

1 mark: any two correct rows
2 marks: all rows correct

I. order of rows
A. any reasonable format for input</td><td>2</td></tr>
<tr><td>06</td><td>2</td><td>All marks AO2 (analyse)

Strings in which any number/operand, other than the first,; contain an even number of digits;

A. Other than the first number/operand; each number can only have an odd number of digits;</td><td>2</td></tr>
<tr><td>06</td><td>3</td><td>All marks AO2 (analyse)

The empty string;
Strings consisting of just one number/operand;
Strings that start with an operator followed by an operand;

Max 2</td><td>2</td></tr>
<tr><td>06</td><td>4</td><td>Mark is for AO2 (analyse)

For June 2025 series item 6.4 has discounted. Examiners must award all students 1 mark for item 6.4</td><td>1</td></tr>
<tr><td>06</td><td>5</td><td>Mark is for AO1 (understanding)

(Both mean the preceeding character/element is optional, but) ? allows at most one of the character/element whereas * allows an unlimited/any of the character/element;</td><td>1</td></tr>
</table>

06	**6**	**All marks for AO1 (understanding)** Unordered; No duplicate values;	**2**
06	**7**	**Mark is for AO2 (apply)** Can't ensure that there are equal number of open and close brackets // can't ensure that the count/number of two different items is the same (when there can be any number of those items); Can't do both left- and right-recursion (only one of the two); **A.** FSMs do not have a memory **Max 1**	**1**

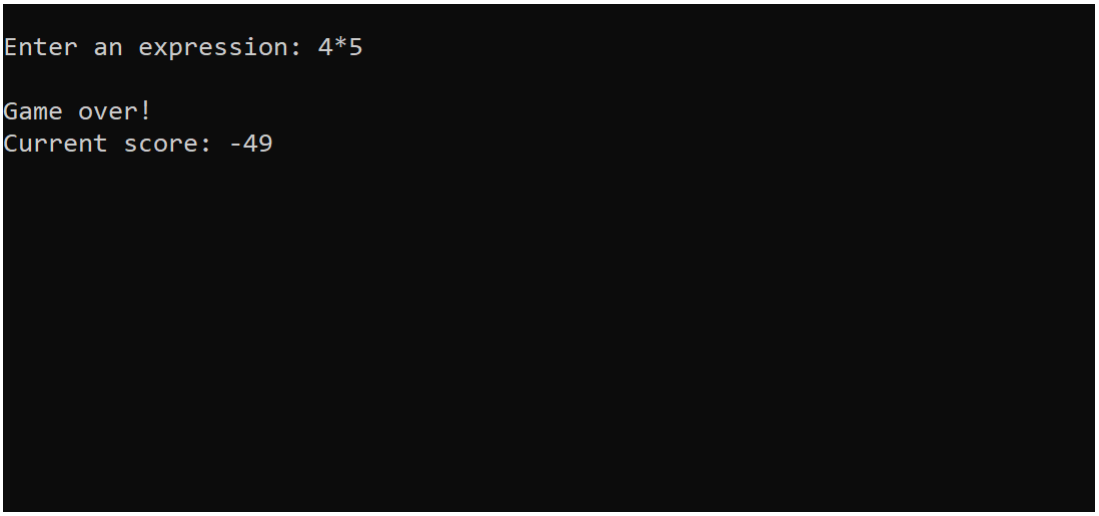
06	**8**	**All marks AO2 (apply)**
		
		`<expression> ::= <number><operator><number> | (<expression>)|<expression><operator><number>|<number><operator><expression>`
		
		Alternative answer
		
		`<expression> ::= <number><operator><number>`
		`<expression> ::= (<expression>)`
		`<expression> ::= <expression><operator><number>`
		`<expression> ::= <number><operator><expression>`
		
		Mark as follows:
		• 1 mark: one of the four definitions for expression is correct
		• 2 marks: two of the four definitions for expression is correct
		• 3 marks: fully correct answer
		
		DPT. Minor errors in notation used
			3

Question			Marks
07	1	**Mark is for AO2 (analyse)** Division by zero; When the result of a calculation is too large to be represented with the data type used; **Max 1**	1
07	2	**All marks AO2 (analyse)** Checks that the result of the expression is a whole number/integer; Returns either the evaluation of the expression or –1, to indicate that the evaluation was not an whole number/integer;	2

Question			Marks
08		**Mark is for AO2 (analyse)** `NumbersAllowed`; `Targets`; `Temp`; **R.** if additional code **I.** case **I.** spacing **Max 1**	1

Question			Marks
09	1	**All marks for AO3 (programming)** 1. New selection structure, in correct position in code, with score reduced by five inside the structure; 2. Correct condition for selection structure for mark point 1 used to check if there is a number shown in the second position; 3. Score reduced by ten and by the value of the number in the first position in `Targets`; 4. Attempt at code for mark point 3 is inside a selection structure that compares a position in `Targets` with a numeric value; **A.** new selection structure or using existing selection structure **Max 3** if code contains errors	4
09	2	**Mark is for AO3 (evaluate)** ****** SCREEN CAPTURE ****** *Must match code from **09.1**.* *Code for **09.1** must be sensible.* Screen capture showing the last two instances of 4*5 being entered followed by game over message being displayed. Final score should be –49:	1

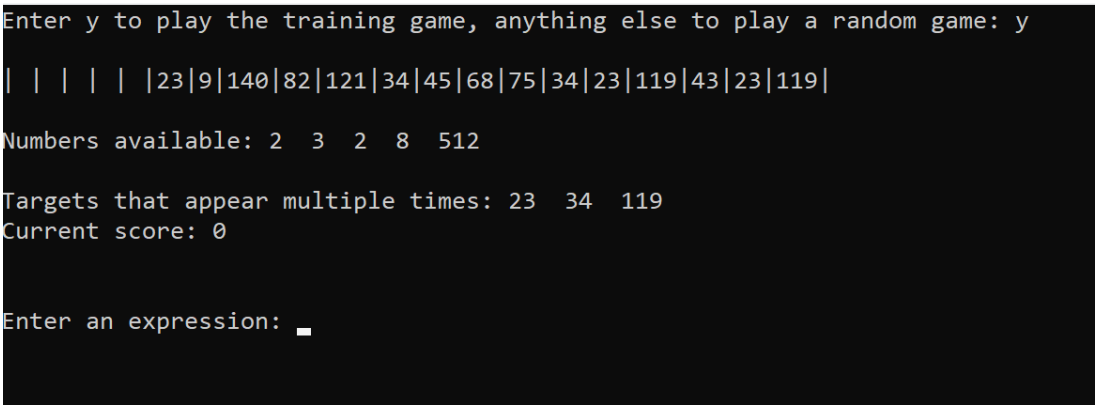
```
Enter y to play the training game, anything else to play a random game: y

| | | | | |23|9|140|82|121|34|45|68|75|34|23|119|43|23|119|

Numbers available: 2  3  2  8  512

Current score: 0


Enter an expression: 4*5

| | | | |23|9|140|82|121|34|45|68|75|34|23|119|43|23|119|119|

Numbers available: 2  3  2  8  512

Current score: -1


Enter an expression: 4*5

| | | |23|9|140|82|121|34|45|68|75|34|23|119|43|23|119|119|119|

Numbers available: 2  3  2  8  512

Current score: -2


Enter an expression: 4*5

| | |23|9|140|82|121|34|45|68|75|34|23|119|43|23|119|119|119|119|

Numbers available: 2  3  2  8  512

Current score: -3


Enter an expression: 4*5

| |23|9|140|82|121|34|45|68|75|34|23|119|43|23|119|119|119|119|119|

Numbers available: 2  3  2  8  512

Current score: -4


Enter an expression: 4*5

|23|9|140|82|121|34|45|68|75|34|23|119|43|23|119|119|119|119|119|119|

Numbers available: 2  3  2  8  512

Current score: -10
```


Enter an expression: 4*5
Game over!
Current score: -49

Question			Marks
10	1	**All marks for AO3 (programming)** Changes to `CheckIfUserInputValid` subroutine: 1. Correctly modified regular expression; Changes to `EvaluateRPN` subroutine: 2. Operator `^` added to list used in condition for second iterative structure; 3. Selection structure with correct condition for `^` operator; 4. Calculates `Num1` to the power of `Num2` and stores result in `Result`; Changes to `ConvertToRPN` subroutine: 5. New operator (`^`) added to dictionary and will have higher precedence than other operators; 6. Condition that checks if current operator is (or is not) `^`; 7. If current operator is `^` and last item in `Operators` is a `^` then `CurrentOperator` is added to end of `Operators`, the last item is not removed from `Operators` and the last item in `Operators` is not added to `UserInputInRPN` **Max 6 marks** if code contains errors	7

10	**2**	**Mark is for AO3 (evaluate)**
		
		****** SCREEN CAPTURE ******
		*Must match code from **10.1**.*
		*Code for **10.1** must be sensible.*
		
		`Enter y to play the training game, anything else to play a random game: y`
		
		`| | | | | |23|9|140|82|121|34|45|68|75|34|23|119|43|23|119|`
		
		`Numbers available: 2  3  2  8  512`
		
		`Current score: 0`
		
		`Enter an expression: 512/2^3^2+8`
		
		`| | | | |23| |140|82|121|34|45|68|75|34|23|119|43|23|119|119|`
		
		`Numbers available: 2  3  2  8  512`
		
		`Current score: 1`
		
		`Enter an expression: 3^2`
		
		`| | | |23| |140|82|121|34|45|68|75|34|23|119|43|23|119|119|119|`
		
		`Numbers available: 2  3  2  8  512`
		
		`Current score: 0`
		
		`Enter an expression:`
		
		Screen capture(s) showing training game was played followed by first expression being entered with 9 being removed from the targets and score increasing to one, then second expression being entered and score becoming zero;
			1

Question			Marks
11	1	**All marks for AO3 (programming)** Mark points 1 to 9 refer to the new subroutine `DisplayTargetMultiples`. Mark points 10 to 11 refer to `DisplayState`. 1. Create a new subroutine called `DisplayTargetMultiples`; **R.** other names for subroutine **I.** case and minor typos 2. Selection structure that compares two values in `Targets`; 3. Checks that one (or both) of the two compared values are not –1; 4. Checks if the value has not already been included in the list of duplicates; 5. Adds value that (their code) recognises as appearing multiple times to a list; **A.** equivalent 6. Iteration structure that selects each value in `Targets`; 7. Iteration structure that selects all other values in `Targets`; **A.** all values if there is a mechanism to check that the two values are not in the same position in `Targets` 8. Displays one duplicate value found; **R.** if a non-duplicate number is also displayed **I.** -1 being displayed 9. Displays all duplicate numbers found; **R.** if a duplicate number is displayed more than once **I.** -1 being displayed 10. Call the new subroutine; 11. Call to new subroutine in correct place in `DisplayState`; **Alternative solution** 4. Checks that a count of the number of occurences for a number is greater than 1; 6. Iteration structure that selects each number between 0 and the maximum number in `Targets` / the maximum target number that could be generated; 7. Counts how often that number appears in `Targets`; **Max 10 marks** if code contains errors	**11**

11	**2**	**Mark is for AO3 (evaluate)** ****** SCREEN CAPTURE ****** *Must match code from **11.1**.* *Code for **11.1** must be sensible.*  Screen capture(s) showing 23, 34, 119 are the targets that appear multiple times; **I.** order of 23, 34, 119	**1**

Question			Marks
12	1	**All marks for AO3 (programming)** Mark points 1 to 9 relate to the `CanGetATargetUsingThreeNumbers` subroutine. Mark points 10 to 11 relate to the `DisplayState` subroutine. 1. Creating a new subroutine called `CanGetATargetUsingThreeNumbers`; **R.** other subroutine identifiers **I.** case and minor typos 2. Iterative structure that repeats once for each target; **A.** equivalent 3. Selection structure that means targets of –1 will be ignored; 4. Nested iteration structures to get three positions in `NumbersAllowed` // nested iteration structures to get three numbers from `NumbersAllowed`; 5. Checks that the three positions from mark point 4 are all different // checks that the three numbers do not exceed the count of that number in `NumbersAllowed`; 6. Nested iteration to get two operators; **A.** equivalent 7. Forms an infix expression using three numbers from `NumbersAllowed` and two operators; **R.** if expressions cannot be formed for an operator 8. Converts expression to RPN and evaluates it // evaluates the infix expression; **R.** if expressions cannot be evaluated for an operator 9. Selection structure that returns true if their attempt at evaluation of expression is equal to a target and the subroutine returns false if no match is found; **R.** returns false after only checking one expression/target 10. Call to new subroutine and uses value returned; 11. Selection structure(s) with condition based on value returned by call to new subroutine and appropriate messages with exactly one of the two messages being displayed under all circumstances; **R.** if would always display the same message **Alternative solution** 2. Attempts call to `CheckIfUserInputEvaluationIsATarget` in appropriate place in the code; 3. Correct call to `CheckIfUserInputEvaluationIsATarget` including ensuring that the score and list of targets are not changed by the call; **Max 10** if code contains any errors	**11**

12	**2**	**Mark is for AO3 (evaluate)**
		
		****** SCREEN CAPTURE ******
		*Must match code from **12.1**, including prompts on screen capture matching those in code.*
		*Code for **12.1** must be sensible.*
		
		`Enter y to play the training game, anything else to play a random game: y`
		
		`| | | | | |23|9|140|82|121|34|45|68|75|34|23|119|43|23|119|`
		
		`Numbers available: 2  3  2  8  512`
		
		`Targets that appear multiple times: 23  34  119`
		`Can get a target when using exactly three of the numbers available`
		`Current score: 0`
		
		`Enter an expression: 3+8-2`
		
		`| | | | |23| |140|82|121|34|45|68|75|34|23|119|43|23|119|119|`
		
		`Numbers available: 2  3  2  8  512`
		
		`Targets that appear multiple times: 23  34  119`
		`Can't get a target using exactly three of the numbers available`
		`Current score: 1`
		
		`Enter an expression:`
		
		Screen capture(s) showing that the message changes from can get a target to cannot get a target after entering the expression; **A.** any appropriate messages
			1

12	**3**	**All marks AO2 (apply)** $5 \times 4 \times 4 \times 4 \times 3$ // 960;; If final answer is incorrect award a maximum of 1 mark for: • Multiplying by 4 twice • Multiplying by 5, 4 and 3 • Multiplying by 5 and 12	**2**

VB.Net

<table>
<tr><th colspan="2">Question</th><th></th><th>Marks</th></tr>
<tr><td>04</td><td>1</td><td>

```
Console.Write("Enter the text to encrypt: ")
Dim Plaintext As String = Console.ReadLine()
Console.Write("Enter the grid size: ")
Dim GridSize As Integer = Console.ReadLine
Dim PTLettersOnly As String = ""
Dim Ciphertext As String = ""
For Count = 0 To Plaintext.Length - 1
  If
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz".Contains(Plai
ntext(Count)) Then
    PTLettersOnly &= Plaintext(Count)
  End If
Next
For Column = 0 To GridSize - 1
  For Count = Column To PTLettersOnly.Length - 1 Step GridSize
    Ciphertext &= PTLettersOnly(Count)
  Next
Next
Console.Write(Ciphertext)
Console.ReadLine()
```

</td><td>12</td></tr>
<tr><td>09</td><td>1</td><td>

```
...
Score -= 1
If Targets(1) <> -1 Then
    Score -= 5
End If
If Targets(0) <> -1 Then
    Score -= Targets(0) + 10
    GameOver = True
Else
    UpdateTargets(Targets, TrainingGame, MaxTarget)
End If
...
```

</td><td>4</td></tr>
</table>

10	**1**	Function EvaluateRPN(ByVal UserInputInRPN as List(Of String)) As Integer
		    Dim S As New List(Of String)
		    While UserInputInRPN.Count > 0
		        While Not {"+", "-", "*", "/", "^"}.Contains(UserInputInRPN(0))
		            S.Add(UserInputInRPN(0))
		            UserInputInRPN.RemoveAt(0)
		        End While
		        ...
		        Select Case (UserInputInRPN(0))
		            Case "+"
		                Result = Num1 + Num2
		            Case "-"
		                Result = Num1 - Num2
		            Case "*"
		                Result = Num1 * Num2
		            Case "/"
		                Result = Num1 / Num2
		            Case "^"
		                Result = Num1 ^ Num2
		        End Select
		...
		End Function
		
		Function CheckIfUserInputValid(ByVal UserInput As String) As Boolean
		    Return Regex.IsMatch(UserInput, "^([0-9]+[\+\-*\/**\^**])+[0-9]+$")
		End Function
		
		Function ConvertToRPN(ByVal UserInput As String) As List(Of String)
		    Dim Position As Integer = 0
		    Dim Precedence As New Dictionary(Of String, Integer) From {{"+", 2}, {"-", 2}, {"*", 4}, {"/", 4}, **{"^", 5}**}
		    Dim Operators As New List(Of String)
		    ...
		    While Position < UserInput.Length
		        Operand = GetNumberFromUserInput(UserInput, Position)
		        UserInputInRPN.Add(Operand.ToString())
		        If Position < UserInput.Length Then
		            ...
		            If Operators.Count > 0 AndAlso Precedence(Operators(Operators.Count - 1)) = Precedence(CurrentOperator) **AndAlso CurrentOperator <>** "^" Then
		                UserInputInRPN.Add(Operators(Operators.Count - 1))
		                Operators.RemoveAt(Operators.Count - 1)
		            End If
		            Operators.Add(CurrentOperator)
		            ...
		End Function
			7

11	**1**	**Sub DisplayTargetMultiples(ByVal Targets As List(Of Integer))**
		Dim MultiplesFound As List(Of Integer) = New List(Of Integer)
		For i = 0 To Targets.Count - 1
		For j = i + 1 To Targets.Count - 1
		If Targets(i) = Targets(j) And Targets(i) <> -1 Then
		If Not MultiplesFound.Contains(Targets(i)) Then
		MultiplesFound.Add(Targets(i))
		End If
		End If
		Next
		Next
		Console.Write("Targets that appear multiple times: ")
		For Each Item In MultiplesFound
		Console.Write(Item)
		Console.Write(" ")
		Next
		Console.WriteLine()
		End Sub
		
		Sub DisplayState(ByVal Targets As List(Of Integer), ByVal NumbersAllowed As List(Of Integer), ByVal Score As Integer)
		DisplayTargets(Targets)
		DisplayNumbersAllowed(NumbersAllowed)
		DisplayTargetMultiples(Targets)
		DisplayScore(Score)
		End Sub
			11

12	**1**	**Function CanGetATargetUsingThreeNumbers(ByVal NumbersAllowed As List(Of Integer), ByVal Targets As List(Of Integer)) As Boolean**
		** Dim OperatorList As New List(Of String)({"+", "-", "*", "/"})**
		** For Each T In Targets**
		** If T <> -1 Then**
		** For i = 0 To NumbersAllowed.Count - 1**
		** For j = 0 To NumbersAllowed.Count - 1**
		** For k = 0 To NumbersAllowed.Count - 1**
		** If Not (i = j Or i = k Or j = k) Then**
		** For Each w In OperatorList**
		** For Each x In OperatorList**
		** Dim UserInput As String = NumbersAllowed(i).ToString & w & NumbersAllowed(j).ToString() & x & NumbersAllowed(k).ToString()**
		** Dim UserInputInRPN As List(Of String) = ConvertToRPN(UserInput)**
		** If T = EvaluateRPN(UserInputInRPN) Then**
		** Return True**
		** End If**
		** Next**
		** Next**
		** End If**
		** Next**
		** Next**
		** Next**
		** End If**
		** Next**
		** Return False**
		End Function
		
		Sub DisplayState(ByVal Targets As List(Of Integer), ByVal NumbersAllowed As List(Of Integer), ByVal Score As Integer)
		  DisplayTargets(Targets)
		  DisplayNumbersAllowed(NumbersAllowed)
		  If CanGetATargetUsingThreeNumbers(NumbersAllowed, Targets) Then
		  Console.WriteLine("Can get a target when using exactly three of the numbers available")
		  Else
		  Console.WriteLine("Can't get a target using exactly three of the numbers available")
		  End If
		  DisplayScore(Score)
		End Sub
			11

Python 3

Question			Marks

04	1	`Plaintext = input("Enter the text to encrypt: ")`
		`GridSize = int(input("Enter the grid size: "))`
		`PTLettersOnly = ""`
		`Ciphertext = ""`
		`for Count in range (0, len(Plaintext)):`
		`  if Plaintext[Count] in "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz":`
		`  PTLettersOnly += Plaintext[Count]`
		`for Column in range (0, GridSize):`
		`  for Count in range(Column, len(PTLettersOnly), GridSize):`
		`  Ciphertext += PTLettersOnly[Count]`
		`print(Ciphertext)`
			12

09	1	`  Score -= 1`
		`  if Targets[1] != -1:`
		`  Score -= 5`
		`  if Targets[0] != -1:`
		`  Score -= Targets[0] + 10`
		`  GameOver = True`
		`  else:`
		`...`
			4

10	**1**	`def EvaluateRPN(UserInputInRPN):`
		`  S = []`
		`  while len(UserInputInRPN) > 0:`
		`  while UserInputInRPN[0] not in ["+", "-", "*", "/", "^"]:`
		`  S.append(UserInputInRPN[0])`
		`  UserInputInRPN.pop(0)`
		`  Num2 = float(S[-1])`
		`  S.pop()`
		`  Num1 = float(S[-1])`
		`  S.pop()`
		`  Result = 0.0`
		`  if UserInputInRPN[0] == "+":`
		`  Result = Num1 + Num2`
		`  elif UserInputInRPN[0] == "-":`
		`  Result = Num1 - Num2`
		`  elif UserInputInRPN[0] == "*":`
		`  Result = Num1 * Num2`
		`  elif UserInputInRPN[0] == "/":`
		`  Result = Num1 / Num2`
		`  elif UserInputInRPN[0] == "^":`
		`  Result = Num1 ** Num2`
		`  UserInputInRPN.pop(0)`
		`  ...`
		
		`def CheckifUserInputValid(UserInput):`
		`  if re.search("^([0-9]+[\\+\\-\\*\\/`**`\\^`**`])+[0-9]+$", UserInput) is not None:`
		`  return True`
		`  else:`
		`  return False`
		
		`def ConvertToRPN(UserInput):`
		`  Position = 0`
		`  Precedence = {"+": 2, "-": 2, "*": 4, "/": 4, "^": `**`5`**`}`
		`  Operators = []`
		`  ...`
		`  while Position < len(UserInput):`
		`  Operand, Position = GetNumberFromUserInput(UserInput, Position)`
		`  UserInputInRPN.append(str(Operand))`
		`  if Position < len(UserInput):`
		`  CurrentOperator = UserInput[Position - 1]`
		`  while len(Operators) > 0 and Precedence[Operators[-1]] > Precedence[CurrentOperator]:`
		`  UserInputInRPN.append(Operators[-1])`
		`  Operators.pop()`
		`  if len(Operators) > 0 and Precedence[Operators[-1]] == Precedence[CurrentOperator]` **`and CurrentOperator != "^":`**
		`  UserInputInRPN.append(Operators[-1])`
		`  Operators.pop()`
		`  Operators.append(CurrentOperator)`
		`  ...`
			7

11	**1**	**def DisplayTargetMultiples(Targets):**
		** MultiplesFound = []**
		** for i in range(0, len(Targets)):**
		** for j in range(i + 1, len(Targets)):**
		** if Targets[i] == Targets[j] and Targets[i] != -1:**
		** if not(Targets[i] in MultiplesFound):**
		** MultiplesFound.append(Targets[i])**
		** print("Targets that appear multiple times: ")**
		** for Item in MultiplesFound:**
		** print(Item, end="")**
		** print(" ", end="")**
		** print()**
		
		def DisplayState(Targets, NumbersAllowed, Score):
		  DisplayTargets(Targets)
		  DisplayNumbersAllowed(NumbersAllowed)
		  DisplayTargetMultiples(Targets)
		  DisplayScore(Score)
			11

12	**1**	**def CanGetATargetUsingThreeNumbers(NumbersAllowed, Targets):**
		** OperatorList = ["+", "-", "*", "/"]**
		** for T in Targets:**
		** if T != -1:**
		** for i in range(0, len(NumbersAllowed)):**
		** for j in range(0, len(NumbersAllowed)):**
		** for k in range(0, len(NumbersAllowed)):**
		** if not(i == j or i == k or j == k):**
		** for w in OperatorList:**
		** for x in OperatorList:**
		** UserInput = str(NumbersAllowed[i]) + w + str(NumbersAllowed[j]) + x + str(NumbersAllowed[k])**
		** UserInputInRPN = ConvertToRPN(UserInput)**
		** if T == EvaluateRPN(UserInputInRPN):**
		** return True**
		** return False**
		
		def DisplayState(Targets, NumbersAllowed, Score):
		  DisplayTargets(Targets)
		  DisplayNumbersAllowed(NumbersAllowed)
		  if CanGetATargetUsingThreeNumbers(NumbersAllowed, Targets):
		** print("Can get a target when using exactly three of the numbers available")**
		** else:**
		** print("Can't get a target using exactly three of the numbers available")**
		  DisplayScore(Score)
			11

C#

<table>
<tr><th colspan="2">Question</th><th></th><th>Marks</th></tr>
<tr><td>04</td><td>1</td><td>

```
Console.Write("Enter the text to encrypt: ");
string Plaintext = Console.ReadLine();
Console.Write("Enter the grid size: ");
int GridSize = Convert.ToInt32(Console.ReadLine());
string PTLettersOnly = "";
string Ciphertext = "";
for (int Count = 0; Count < Plaintext.Length; Count++)
{
  if
("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz".Contains(Pla
intext[Count]))
  {
    PTLettersOnly += Plaintext[Count];
  }
}
for (int Column = 0; Column < GridSize; Column++)
{
  for (int Count = Column; Count < PTLettersOnly.Length; Count +=
GridSize)
  {
    Ciphertext += PTLettersOnly[Count];
  }
}
Console.Write(Ciphertext);
Console.ReadLine();
```

</td><td>12</td></tr>
<tr><td>09</td><td>1</td><td>

```
...
Score--;
if (Targets[1] != -1)
{
  Score -= 5;
}
if (Targets[0] != -1)
{
  Score -= Targets[0] + 10;
  GameOver = true;
}
else
{
  UpdateTargets(Targets, TrainingGame, MaxTarget);
}
...
```

</td><td>4</td></tr>
</table>

10	**1**	static int EvaluateRPN(List<string> UserInputInRPN)
		{
		  List<string> S = new List<string>();
		  while (UserInputInRPN.Count > 0)
		  {
		  while (!new string[] { "+", "-", "*", "/","^"
		}.Contains(UserInputInRPN[0]))
		  {
		  S.Add(UserInputInRPN[0]);
		  UserInputInRPN.RemoveAt(0);
		  }
		  double Num2 = Convert.ToDouble(S[S.Count - 1]);
		  S.RemoveAt(S.Count - 1);
		  double Num1 = Convert.ToDouble(S[S.Count - 1]);
		  S.RemoveAt(S.Count - 1);
		  double Result = 0;
		  switch (UserInputInRPN[0])
		  {
		  case "+":
		  Result = Num1 + Num2;
		  break;
		  case "-":
		  Result = Num1 - Num2;
		  break;
		  case "*":
		  Result = Num1 * Num2;
		  break;
		  case "/":
		  Result = Num1 / Num2;
		  break;
		  case "^":
		  Result = Math.Pow(Num1, Num2);
		  break;
		  }
		...
		
		static bool CheckIfUserInputValid(string UserInput)
		{
		  return Regex.IsMatch(UserInput, @"^([0-9]+[\+\-*\/**\^**])+[0-9]+$");
		}
		
		static List<string> ConvertToRPN(string UserInput)
		{
		  int Position = 0;
		  Dictionary<string, int> Precedence = new Dictionary<string, int>
		  {
		  { "+", 2 },
		  { "-", 2 },
		  { "*", 4 },
		  { "/", 4 },
		  { "^", 5 }
		  };
		  List<string> Operators = new List<string>();
		  int Operand = GetNumberFromUserInput(UserInput, ref Position);
		  List<string> UserInputInRPN = new List<string> {
		Operand.ToString() };
		  Operators.Add(UserInput[Position - 1].ToString());
		  ...
			7

		`if (Operators.Count > 0 && Precedence[Operators[Operators.Count - 1]] == Precedence[CurrentOperator] && CurrentOperator != "^")`
		`{`
		`  UserInputInRPN.Add(Operators[Operators.Count - 1]);`
		`  Operators.RemoveAt(Operators.Count - 1);`
		`}`
		`Operators.Add(CurrentOperator);`
		`...`	

11	**1**	**static void DisplayTargetMultiples(List<int> Targets)**
		{
		** List<int> MultiplesFound = new List<int>();**
		** for (int i = 0; i < Targets.Count; i++)**
		** {**
		** for (int j = i + 1; j < Targets.Count; j++)**
		** {**
		** if (Targets[i] == Targets[j] && Targets[i] != -1)**
		** {**
		** if (!MultiplesFound.Contains(Targets[i]))**
		** {**
		** MultiplesFound.Add(Targets[i]);**
		** }**
		** }**
		** }**
		** }**
		** Console.Write("Targets that appear multiple times: ");**
		** foreach (var Item in MultiplesFound)**
		** {**
		** Console.Write(Item + " ");**
		** }**
		** Console.WriteLine();**
		}
		
		static void DisplayState(List<int> Targets, List<int> NumbersAllowed, int Score)
		{
		  DisplayTargets(Targets);
		  DisplayNumbersAllowed(NumbersAllowed);
		  DisplayTargetMultiples(Targets);
		  DisplayScore(Score);
		}
			11

12	**1**	**public static bool CanGetATargetUsingThreeNumbers(List<int> NumbersAllowed, List<int> Targets)**
		{
		** List<string> OperatorList = new List<string> { "+", "-", "*", "/" };**
		** foreach (var T in Targets)**
		** {**
		** if (T != -1)**
		** {**
		** for (int i = 0; i < NumbersAllowed.Count; i++)**
		** {**
		** for (int j = 0; j < NumbersAllowed.Count; j++)**
		** {**
		** for (int k = 0; k < NumbersAllowed.Count; k++)**
		** {**
		** if (i != j && i != k && j != k)**
		** {**
		** foreach (var w in OperatorList)**
		** {**
		** foreach (var x in OperatorList)**
		** {**
		** string UserInput = $"{NumbersAllowed[i]}{w}{NumbersAllowed[j]}{x}{NumbersAllowed[k]}";**
		** List<string> UserInputInRPN = ConvertToRPN(UserInput);**
		** if (T == EvaluateRPN(UserInputInRPN))**
		** {**
		** return true;**
		** }**
		** }**
		** }**
		** }**
		** }**
		** }**
		** }**
		** }**
		** }**
		** return false;**
		}
		
		static void DisplayState(List<int> Targets, List<int> NumbersAllowed, int Score)
		{
		  DisplayTargets(Targets);
		  DisplayNumbersAllowed(NumbersAllowed);
		  DisplayTargetMultiples(Targets);
		** if (CanGetATargetUsingThreeNumbers(NumbersAllowed, Targets))**
		** {**
		** Console.WriteLine("Can get a target when using exactly three of the numbers available");**
		** }**
		** else**
		** {**
		** Console.WriteLine("Can't get a target using exactly three of the numbers available");**
		** }**
		  DisplayScore(Score);
		}
			11

Java

<table>
<tr><th colspan="2">Question</th><th></th><th>Marks</th></tr>
<tr><td>04</td><td>1</td><td>

```
System.out.print("Enter the plaintext: ");
String plainText = System.console().readLine();
String plainTextLetters = "";
for (int i = 0; i < plainText.length(); i++) {
    if (Character.isLetter(plainText.charAt(i))) {
        plainTextLetters += plainText.charAt(i);
    }
}
System.out.print("Enter the number of columns: ");
int columns = Integer.parseInt(System.console().readLine());
System.out.print("Ciphertext is: ");
for (int column = 0; column < columns; column++) {
    for (int i = column; i < plainTextLetters.length(); i +=
columns) {
        System.out.print(plainTextLetters.charAt(i));
    }
}
System.out.println();
System.console().readLine();
```

</td><td>12</td></tr>
<tr><td>09</td><td>1</td><td>

```
...
score.value -= 1;
if (targets.get(1) != -1) {
    score.value -= 5;
}
if (targets.get(0) != -1) {
    score.value -= targets.get(0) + 10;
}
if (targets.get(0) != -1) {
    gameOver = true;
} else {
    updateTargets(targets, trainingGame, maxTarget);
}
...
```

</td><td>4</td></tr>
</table>

10	**1**	`private static boolean checkIfUserInputValid(String userInput) {`
		`    return userInput.matches("^([0-9]+[\\+\\-\\*\\/**\\^**])+[0-9]+$");`
		`}`
		
		`private static int evaluateRPN(List<String> userInputInRPN) {`
		`    List<String> s = new ArrayList<String>();`
		`    while (userInputInRPN.size() > 0) { // List.of()`
		`        while (!"+-*/^".contains(userInputInRPN.get(0))) {`
		`            s.add(userInputInRPN.get(0));`
		`            userInputInRPN.remove(0);`
		`        }`
		`        ...`
		`        switch (userInputInRPN.get(0)) {`
		`            case "+":`
		`                result = num1 + num2;`
		`                break;`
		`            case "-":`
		`                result = num1 - num2;`
		`                break;`
		`            case "*":`
		`                result = num1 * num2;`
		`                break;`
		`            case "/":`
		`                result = num1 / num2;`
		`                break;`
		`            **case "^":**`
		`                **result = Math.pow(num1, num2);**`
		`                **break;**`
		`        }`
		`...`
		`}`
		
		`static List<String> convertToRPN(String userInput) {`
		`    IntWrapper position = new IntWrapper(0);`
		`    Map<String, Integer> precedence = new HashMap<String,`
		`Integer>();`
		`    precedence.put("+", 2);`
		`    precedence.put("-", 2);`
		`    precedence.put("*", 4);`
		`    precedence.put("/", 4);`
		`    **precedence.put("^", 5);**`
		`    List<String> operators = new ArrayList<String>();`
		`...`
		`    while (position.value < userInput.length()) {`
		`        operand = getNumberFromUserInput(userInput, position);`
		`        userInputInRPN.add(Integer.toString(operand));`
		`        if (position.value < userInput.length()) {`
		`            ...`
		`            if (operators.size() > 0`
		`                    && precedence.get(operators.get(operators.size()`
		`- 1)) == precedence.get(currentOperator)) {`
		`                **if (!operators.get(operators.size() -**`
		`**1).equals("^")) {**`
		
		`userInputInRPN.add(operators.get(operators.size() - 1));`
		`                    operators.remove(operators.size() - 1);`
		`                **}**`
		`            }`
		`            operators.add(currentOperator);`
		`            ...`
		`}`
			7

11	**1**	**static void displayTargetMultiples(List<Integer> targets) {**
		** Map<Integer, Integer> counts = new HashMap<Integer, Integer>();**
		** for (int t : targets) {**
		** if (t != -1 && counts.containsKey(t)) {**
		** counts.put(t, counts.get(t) + 1);**
		** } else {**
		** counts.put(t, 1);**
		** }**
		** }**
		** System.out.print("Targets appearing more than once: ");**
		** for (Map.Entry<Integer, Integer> count : counts.entrySet()) {**
		** if (count.getValue() > 1) {**
		** System.out.print(count.getKey() + " ");**
		** }**
		** }**
		** System.out.println();**
		}
		
		static void displayState(List<Integer> targets, List<Integer> numbersAllowed, int score) {
		    displayTargets(targets);
		    displayNumbersAllowed(numbersAllowed);
		    displayTargetMultiples(targets);
		    displayScore(score);
		}
			11

12	**1**	**static boolean canGetATargetUsingThreeNumbers(List<Integer> targets, List<Integer> allowed) {**
		for (Integer n1 : allowed) {
		for (Integer n2 : allowed) {
		for (Integer n3 : allowed) {
		String[] operators = {"+", "-", "*", "/"};
		for (String op1 : operators) {
		for (String op2 : operators) {
		String expression = n1.toString() + op1 + n2.toString() + op2 + n3.toString();
		List<String> RPN = convertToRPN(expression);
		IntWrapper score = new IntWrapper(0);
		if (checkNumbersUsedAreAllInNumbersAllowed(allowed, RPN, 1000) && checkIfUserInputEvaluationIsATarget(targets, RPN, score)) {
		return true;
		}
		}
		}
		}
		}
		}
		return false;
		}
		
		static void displayState(List<Integer> targets, List<Integer> numbersAllowed, int score) {
		if (canGetATargetUsingThreeNumbers(targets, numbersAllowed)) {
		System.out.println("You can make a target using three numbers");
		} else {
		System.out.println("You cannot make a target using three numbers");
		}
		displayTargets(targets);
		displayNumbersAllowed(numbersAllowed);
		displayTargetMultiples(targets);
		displayScore(score);
		}
			11