



AS Computer Science

7516/1 - Paper 1

Mark scheme

June 2018

Version/Stage: 1.0 Final

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 Assessment Writer.

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.

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

The following annotation is used in the mark scheme:

- ;** - means a single mark
- //** - means alternative response
- /** - means an alternative word or sub-phrase
- A** - means 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.

Pages 5 to 14 contain the generic mark scheme.

Pages 15 to 48 contain the 'Program Source Codes' specific to the programming languages for questions 03, 11, 12, 13 and 14;

- pages 20 to 29 – VB.NET
- pages 30 to 36 – PYTHON 2
- pages 37 to 43 – PYTHON 3
- pages 44 to 50 – PASCAL/Delphi
- pages 51 to 61 – C#
- pages 62 to 69 – JAVA

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.

Examiners are required to assign each of the candidates' responses to the most appropriate level according to **its overall quality**, 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 11.1, the marks available for the AO3 elements are as follows:

AO3 (design) – 3 marks

AO3 (programming) – 9 marks

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

Qu	Marks										
01	1	<p>All marks for AO1 (knowledge)</p> <table border="1" data-bbox="317 344 1313 947"> <tbody> <tr> <td data-bbox="317 344 1163 495">Breaking a problem into a number of sub-problems</td> <td data-bbox="1163 344 1313 495">F</td> </tr> <tr> <td data-bbox="317 495 1163 645">Models are put into action to solve problems</td> <td data-bbox="1163 495 1313 645">H</td> </tr> <tr> <td data-bbox="317 645 1163 795">Combining procedures into compound procedures</td> <td data-bbox="1163 645 1313 795">G</td> </tr> <tr> <td data-bbox="317 795 1163 947">Details are removed until the problem is represented in a way that is possible to solve because the problem reduces to one that has already been solved</td> <td data-bbox="1163 795 1313 947">E</td> </tr> </tbody> </table> <p>1 mark per correct label</p> <p>Note: each label must only be used once (if given more than once, ignore all occurrences)</p> <p>A. handwritten answers</p> <p>A. lower case</p>	Breaking a problem into a number of sub-problems	F	Models are put into action to solve problems	H	Combining procedures into compound procedures	G	Details are removed until the problem is represented in a way that is possible to solve because the problem reduces to one that has already been solved	E	4
Breaking a problem into a number of sub-problems	F										
Models are put into action to solve problems	H										
Combining procedures into compound procedures	G										
Details are removed until the problem is represented in a way that is possible to solve because the problem reduces to one that has already been solved	E										

Qu	Marks						
02	1	All marks for A02 (apply)				3	
		Number	Root	d	FactorFound	r	Output
		5	1				
			2				
			3				
				2	FALSE	1	
				3		2	
				4			Prime
		<p>1 mark for correct columns Root and d</p> <p>1 mark for correct column r</p> <p>1 mark for correct columns FactorFound and Output</p> <p>Max 2 marks if any incorrect values written in table</p> <p style="text-align: center;">3</p> <p>I. Annotation indicating 'no value'</p> <p>I. Indication of repeating values</p> <p>I. Quotes</p> <p>I. Case & spelling for FactorFound & Output</p> <p>A. T, F instead of True False</p>					

Qu	Marks																																																																			
02	2	All marks for A02 (apply)																																																																		
		<table border="1"> <thead> <tr> <th>Number</th> <th>Root</th> <th>d</th> <th>FactorFound</th> <th>r</th> <th>Output</th> </tr> </thead> <tbody> <tr> <td>25</td> <td>1</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td>2</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td>3</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td>4</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td>5</td> <td>2</td> <td>False</td> <td>1</td> <td></td> </tr> <tr> <td></td> <td></td> <td>3</td> <td></td> <td>1</td> <td></td> </tr> <tr> <td></td> <td></td> <td>4</td> <td></td> <td>1</td> <td></td> </tr> <tr> <td></td> <td></td> <td>5</td> <td></td> <td>0</td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td>True</td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td>6</td> <td></td> <td></td> <td>Not prime</td> </tr> </tbody> </table> <p> 1 mark for correct columns <code>Root</code> and <code>d</code> 1 mark for correct column <code>r</code> 1 mark for correct columns <code>FactorFound</code> and <code>Output</code> Max 2 marks if any incorrect values written in table </p>	Number	Root	d	FactorFound	r	Output	25	1						2						3						4						5	2	False	1				3		1				4		1				5		0					True					6			Not prime
Number	Root	d	FactorFound	r	Output																																																															
25	1																																																																			
	2																																																																			
	3																																																																			
	4																																																																			
	5	2	False	1																																																																
		3		1																																																																
		4		1																																																																
		5		0																																																																
			True																																																																	
		6			Not prime																																																															
		3																																																																		

03	1	<p>All marks for AO3 (programming)</p> <p>Mark as follows:</p> <ol style="list-style-type: none"> 1) Correct variable declarations for <code>Number</code>, <code>c</code>, <code>k</code>; <p>Note to examiners If a language allows variables to be used without explicit declaration (eg Python) then this mark should be awarded if the correct variables exist in the program code and the first value they are assigned is of the correct data type.</p> <ol style="list-style-type: none"> 2) <code>WHILE</code> loop with syntax allowed by the programming language and one correct condition for termination of the loop; 3) Second correct condition for while loop; 4) Correct prompt <code>"Enter a positive whole number: "</code> and <code>Number</code> assigned value entered by user; 5) Correct syntax for the <code>IF</code> statements inside attempt at loop; A. <code>IF ... ELSEIF;</code> 6) correct contents in <code>IF</code> statements; 7) <code>FOR</code> loop with syntax allowed by the programming language over correct range; 8) Correct assignment to <code>c</code> inside <code>FOR</code> loop; 9) Output statement giving correct output; A. accept without spaces <p>I. Ignore minor differences in case and spelling R. real <code>Number</code></p> <p>Max 8 if code does not function correctly</p>	9
----	---	--	---

03	2	<p>Mark is for AO3 (evaluate)</p> <p>**** SCREEN CAPTURE ****</p> <p><i>Must match code from 03.1, including prompts on screen capture matching those in code.</i></p> <p><i>Code for 03.1 must be sensible.</i></p> <p>Screen capture showing:</p> <p>'-3' being entered and the message 'Not a positive number.' displayed</p> <p>'11' being entered and the message 'Number too large.' displayed</p> <p>'10' being entered and line of numbers displayed</p> <pre> Enter a positive whole number: -3 Not a positive number. Enter a positive whole number: 11 Number too large. Enter a positive whole number: 10 1 9 36 84 126 126 84 36 9 1 </pre> <p>></p> <p>A. Alternative layout :</p> <pre> Enter a positive whole number: -3 Not a positive number. Enter a positive whole number: 11 Number too large. Enter a positive whole number: 10 1 9 36 84 126 126 84 36 9 1 > </pre> <p>A. input on new line</p>	1
----	---	---	---

04	1	Mark is for AO1 (understanding) LetterEnd // ProgramEnd ; R. if any additional code R. if spelt incorrectly I. case & spacing	1
----	---	---	---

04	2	<p>Mark is for AO1 (understanding)</p> <p>GetMenuOption // Decode ; A. GetNextSymbol</p> <p>R. if any additional code R. if spelt incorrectly I. case & spacing</p>	1										
05		<p>All marks for AO1 (understanding)</p> <table border="1" data-bbox="240 629 1315 1081"> <thead> <tr> <th data-bbox="240 629 568 719">Identifier</th> <th data-bbox="568 629 1315 719">Description</th> </tr> </thead> <tbody> <tr> <td data-bbox="240 719 568 808">PlainTextLetter</td> <td data-bbox="568 719 1315 808">uncoded letter, part of PlainText</td> </tr> <tr> <td data-bbox="240 808 568 898">Signal</td> <td data-bbox="568 808 1315 898">single unit of Transmission (= or SPACE or EOL)</td> </tr> <tr> <td data-bbox="240 898 568 987">FirstSignal</td> <td data-bbox="568 898 1315 987">first character in Transmission</td> </tr> <tr> <td data-bbox="240 987 568 1077">Symbol</td> <td data-bbox="568 987 1315 1077">used to build SymbolString (. or -)</td> </tr> </tbody> </table> <p>R. if any additional code R. if spelt incorrectly I. case & spacing</p> <p>1 mark per correct identifier</p> <p>Note: each identifier must only be used once (if given more than once, ignore all occurrences)</p>	Identifier	Description	PlainTextLetter	uncoded letter, part of PlainText	Signal	single unit of Transmission (= or SPACE or EOL)	FirstSignal	first character in Transmission	Symbol	used to build SymbolString (. or -)	4
Identifier	Description												
PlainTextLetter	uncoded letter, part of PlainText												
Signal	single unit of Transmission (= or SPACE or EOL)												
FirstSignal	first character in Transmission												
Symbol	used to build SymbolString (. or -)												

06		<p>Mark is for AO1 (understanding)</p> <p>1) makes a subroutine self-contained; 2) releases storage when subroutine terminates; 3) able to test/debug subroutine independently ; 4) easier to re-use subroutine in another program; 5) local variable values cannot be inadvertently/accidentally altered by a subroutine call from the subroutine;</p> <p>Max 1</p>	1
07	1	<p>All marks for AO2 (analyse)</p> <p>the variable is used as the index of /pointer to / iterator for / place value; ... the current character/symbol/signal in the transmission string;</p>	2
07	2	<p>All marks for AO2 (analyse)</p> <p>1) empty string returned from <code>StripLeadingSpaces</code> (and assigned to <code>Transmission</code>) // generate empty string in <code>StripLeadingSpaces</code>; 2) <code>StripLeadingSpaces</code> (calls <code>ReportError</code>) to display “No signal received” 3) empty string is returned to <code>ReceiveMorseCode</code> // empty string is returned from <code>GetTransmission</code>; 4) <code>LastChar</code> is set to -1; 5) so loop is not entered; 6) <code>MorseCodeString / PlainText</code> remain empty strings;</p> <p>Max 5</p>	5
08		<p>All marks for AO2 (analyse)</p> <p>1) Any example string with only two consecutive symbols, for example “==” // any example string with more than 3 consecutive symbols, such as “====” ; Note: “xxx” would not cause an error. Note: It does not matter which non-space symbol is used in transmission. 2) The while loop counts the number of consecutive non-spaces; 3) If this number is not 0, 1 or 3, (it calls the <code>ReportError</code> subroutine);</p> <p>I. quotes</p>	3

MARK SCHEME – AS COMPUTER SCIENCE – 7516/1 – JUNE 2018

09	1	<p>Mark is for AO2 (analyse)</p> <p><code>GetNextLetter;</code></p> <p>R. if any additional code R. if spelt incorrectly I. case & spacing</p>	1
09	2	<p>Mark is for AO2 (analyse)</p> <p><code>GetNextSymbol;</code></p> <p>R. if any additional code R. if spelt incorrectly I. case & spacing</p>	1
10		<p>All marks AO2 (analyse)</p> <p>mark as follows:</p> <ol style="list-style-type: none"> 1) include digits 0 to 9 in <code>Letter</code> array; 2) include Morse codes in <code>MorseCode</code> array for digit characters; 3) extend <code>Dash</code> and <code>Dot</code> array; 4) at the corresponding positions some of the zeros will need to change (to include new pointers) // binary tree to include routes to digit characters; 5) <code>Decode</code> subroutine needs no changes; 6) <code>SendMorseCode</code> needs to test for digits; 7) Explain a method to look up Morse code for digits (eg linear search of <code>Letter</code> array); <p>Unusual answers should be referred to the PE A. answers using dictionary for digits A. <code>Letter</code> and <code>MorseCode</code> arrays need changing; for 1 mark if (1) and (2) not awarded Max 6</p>	6

11	1	<p>1 mark for AO3 (design) and 3 marks for AO3 (programming)</p> <p>Mark as follows:</p> <p>AO3 (design) – 1 mark:</p> <p>1) Identifying that a selection statement (or equivalent method) is required to test that character is within range of uppercase letters or is a space // identifying that selection statement needs modifying (e.g. <code>if Char in Letter...</code>);</p> <p>AO3 (programming) – 3 marks:</p> <p>2) Selection structure is created with correct logic so that if error detected it ensures error message is displayed only once & subroutine exits;</p> <p>3) calls <code>ReportError</code> subroutine with suitable message if error in input string;</p> <p>4) final value of <code>MorseCodeString</code> set to <code>EMPTYSTRING</code> (accept ' ' or <code>SPACE</code>) if error in input string;</p> <p>A. accept if <code>MorseCodeString</code> set to <code>EMPTYSTRING</code> initially and not changed.</p>	4
11	2	<p>Mark is for AO3 (evaluate)</p> <p>**** SCREEN CAPTURE ****</p> <p><i>Must match code from 11.1, including prompts on screen capture matching those in code.</i></p> <p><i>Code for 11.1 must be sensible.</i></p> <p>Screen capture showing: ‘S’ being entered followed by ‘Help’ and suitable message displayed</p> <pre>Main Menu ===== R - Receive Morse code S - Send Morse code X - Exit program Enter your choice: S Enter your message (uppercase letters and spaces only): Help * Invalid character entered *</pre> <p>A. any suitable message, but must be within *s</p>	1

12	1	<p>1 mark for AO3 (design) and 6 marks for AO3 (programming)</p> <p>Mark as follows:</p> <p>AO3 (design) – 1 mark:</p> <p>1) Identifying that within an iterative statement a selection statement (or equivalent method) is required to test whether the Morse code is a dot, a dash or a space;</p> <p>AO3 (programming) – 5 marks:</p> <p>2) Correct subroutine heading (<code>SendSignals</code>) and ending and correct parameter (<code>MorseCodeString</code>);</p> <p>3) loop for each character in <code>MorseCodeString</code>;</p> <p>4) start with empty string and keep adding a symbol string;</p> <p>5) at least one conversion of dot, dash or space to the correct symbol string;</p> <p>6) dot, dash and space converted to the correct symbol string;</p> <p>7) output the signals correctly;</p>	7
12	2	<p>Mark is for AO3 (evaluate)</p> <p>**** SCREEN CAPTURE ****</p> <p><i>Must match code from 12.1, including prompts on screen capture matching those in code.</i></p> <p><i>Code for 12.1 must be sensible.</i></p> <p>Screen capture showing: S being entered followed by MORSE X and the string <code>=== === === === === = === = === = === = === =</code> ' being displayed after the Morse code.</p> <pre> Main Menu ===== R - Receive Morse code S - Send Morse code X - Exit program Enter your choice: S Enter your message (uppercase letters and spaces only): MORSE X -- --- .-. -.- === === === === === = === = = = = = === = = === </pre>	1

13	1	2 marks for AO3 (design) and 4 marks for AO3 (programming)	6												
<p>Note that AO3 (design) marks 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 regardless of whether the solution works.</p>															
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Level</th> <th style="text-align: center;">Description</th> <th style="text-align: center;">Mark Range</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">3</td> <td>A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. Code is written to ensure that all letters are output with their corresponding Morse code. The formatting of each line has been considered. A formal interface is used to pass the data structures' data into the subroutine. All of the appropriate design decisions have been taken.</td> <td style="text-align: center;">5-6</td> </tr> <tr> <td style="text-align: center;">2</td> <td>There is evidence that a line of reasoning has been partially followed. The formatting of each line does not fully comply with requirements. There is evidence of some appropriate design work. There is Morse code output for each letter.</td> <td style="text-align: center;">3-4</td> </tr> <tr> <td style="text-align: center;">1</td> <td>An attempt has been made to create <code>OutputAlphabetWithCode</code> and some appropriate programming statements have been written. There is insufficient evidence to suggest that a line of reasoning has been followed or that the solution has been designed. The statements written may or may not be syntactically correct and the subroutine will have very little or none of the required functionality. It is unlikely that any of the key design elements of the task have been recognised.</td> <td style="text-align: center;">1-2</td> </tr> </tbody> </table>				Level	Description	Mark Range	3	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. Code is written to ensure that all letters are output with their corresponding Morse code. The formatting of each line has been considered. A formal interface is used to pass the data structures' data into the subroutine. All of the appropriate design decisions have been taken.	5-6	2	There is evidence that a line of reasoning has been partially followed. The formatting of each line does not fully comply with requirements. There is evidence of some appropriate design work. There is Morse code output for each letter.	3-4	1	An attempt has been made to create <code>OutputAlphabetWithCode</code> and some appropriate programming statements have been written. There is insufficient evidence to suggest that a line of reasoning has been followed or that the solution has been designed. The statements written may or may not be syntactically correct and the subroutine will have very little or none of the required functionality. It is unlikely that any of the key design elements of the task have been recognised.	1-2
Level	Description	Mark Range													
3	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. Code is written to ensure that all letters are output with their corresponding Morse code. The formatting of each line has been considered. A formal interface is used to pass the data structures' data into the subroutine. All of the appropriate design decisions have been taken.	5-6													
2	There is evidence that a line of reasoning has been partially followed. The formatting of each line does not fully comply with requirements. There is evidence of some appropriate design work. There is Morse code output for each letter.	3-4													
1	An attempt has been made to create <code>OutputAlphabetWithCode</code> and some appropriate programming statements have been written. There is insufficient evidence to suggest that a line of reasoning has been followed or that the solution has been designed. The statements written may or may not be syntactically correct and the subroutine will have very little or none of the required functionality. It is unlikely that any of the key design elements of the task have been recognised.	1-2													
<p>Marking guidance:</p> <p>Evidence of AO3 design – 2 points:</p> <p>Evidence of design to look for in response:</p> <ol style="list-style-type: none"> 1) identify the need for an iterative statement to act on each letter in turn 2) identify a method to output four letters per line <p>Evidence of AO3 programming – 7 points:</p> <p>Evidence of programming to look for in response:</p> <ol style="list-style-type: none"> 3) add option A to <code>DisplayMenu</code> subroutine 4) add test for new option and call <code>OutputAlphabetWithCode</code> with correct parameters 5) create new subroutine <code>OutputAlphabetWithCode</code> with correct parameters 6) loop from A to Z to output each letter and corresponding code separated from 															

		letter by one space (A. two spaces)	
--	--	---	--

13	2	<p>Mark is for AO3 (evaluate)</p> <p>**** SCREEN CAPTURE ****</p> <p><i>Must match code from 13.1, including prompts on screen capture matching those in code.</i></p> <p><i>Code for 13.1 must be sensible.</i></p> <p>Screen capture showing: main menu with new option A 'A' being entered and alphabet with Morse codes displayed</p> <pre>Main Menu ===== R - Receive Morse code S - Send Morse code A - Output alphabet with Morse code X - Exit program Enter your choice: A A .- B -... C -.-. D -.. E . F ..-. G --. H I .. J .--- K -.- L .-.. M -- N -. O --- P .--. Q ---. R .-. S ... T - U ..- V ...- W .-- X -.-. Y -.- Z ---..</pre> <p>If not in columns as shown, do not award screen capture mark</p>	1
----	---	---	---

14	1	<p>3 marks for AO3 (design) and 6 marks for AO3 (programming)</p> <p>Note that AO3 (design) marks 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 regardless of whether the solution works.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Level</th> <th style="text-align: center;">Description</th> <th style="text-align: center;">Mark Range</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">3</td> <td>A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. Code is written to ensure that each letter of the message is encrypted using the user-supplied keys. All of the appropriate design decisions have been taken.</td> <td style="text-align: center;">7-9</td> </tr> <tr> <td style="text-align: center;">2</td> <td>There is evidence that a line of reasoning has been partially followed. The encryption of each character does not fully comply with requirements. There is evidence of some appropriate design work.</td> <td style="text-align: center;">4-6</td> </tr> <tr> <td style="text-align: center;">1</td> <td>An attempt has been made to amend the subroutines. Some appropriate programming statements have been written. There is little evidence to suggest that a line of reasoning has been followed or that the solution has been designed. The statements written may or may not be syntactically correct and the subroutines will have very little or none of the extra required functionality. It is unlikely that any of the key design elements of the task have been recognised.</td> <td style="text-align: center;">1-3</td> </tr> </tbody> </table> <p>Marking guidance:</p> <p>Evidence of AO3 design – 3 points:</p> <p>Evidence of design to look for in response:</p> <ol style="list-style-type: none"> 1) identifying the need to validate a key is an integer 2) identifying a method to encrypt each character with a key 3) identifying suitable method to alternate keys depending on character position in message <p>Evidence of AO3 programming – 6 points:</p> <p>Evidence of programming to look for in response:</p> <ol style="list-style-type: none"> 4) in <code>SendReceiveMessages</code> correctly store 3 integer keys entered by the user (in a list or separate variables) 5) amend call and subroutine header of <code>SendMorseCode</code> to include keys as parameter(s) 6) correctly encrypt first three characters of message 7) correctly encrypt all characters in message 8) ensure index is within range of array subscripts 9) code to encrypt character inserted in suitable place in <code>SendMorseCode</code> 	Level	Description	Mark Range	3	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. Code is written to ensure that each letter of the message is encrypted using the user-supplied keys. All of the appropriate design decisions have been taken.	7-9	2	There is evidence that a line of reasoning has been partially followed. The encryption of each character does not fully comply with requirements. There is evidence of some appropriate design work.	4-6	1	An attempt has been made to amend the subroutines. Some appropriate programming statements have been written. There is little evidence to suggest that a line of reasoning has been followed or that the solution has been designed. The statements written may or may not be syntactically correct and the subroutines will have very little or none of the extra required functionality. It is unlikely that any of the key design elements of the task have been recognised.	1-3	9
Level	Description	Mark Range													
3	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. Code is written to ensure that each letter of the message is encrypted using the user-supplied keys. All of the appropriate design decisions have been taken.	7-9													
2	There is evidence that a line of reasoning has been partially followed. The encryption of each character does not fully comply with requirements. There is evidence of some appropriate design work.	4-6													
1	An attempt has been made to amend the subroutines. Some appropriate programming statements have been written. There is little evidence to suggest that a line of reasoning has been followed or that the solution has been designed. The statements written may or may not be syntactically correct and the subroutines will have very little or none of the extra required functionality. It is unlikely that any of the key design elements of the task have been recognised.	1-3													

14	2	<p>Mark is for AO3 (evaluate)</p> <p>**** SCREEN CAPTURE ****</p> <p><i>Must match code from 14.1, including prompts on screen capture matching those in code.</i></p> <p><i>Code for 14.1 must be sensible.</i></p> <p>Screen capture showing: 17, 5 and -3 being entered followed by option S and then TEA X followed by the output .---- .---- -.- -.- -.- -....</p> <pre>Enter encryption key (integer): 17 Enter encryption key (integer): 5 Enter encryption key (integer): -3 Main Menu ===== R - Receive Morse code S - Send Morse code X - Exit program Enter your choice: S Enter your message (uppercase letters and spaces only): TEA X .-.-.- .-.-.- -.- -.- -.- -....</pre>	1
		Total	75

VB.NET

03	1	<pre>Sub Main() Dim Number As Integer Dim c As Integer Number = 0 While Number < 1 Or Number > 10 Console.WriteLine("Enter a positive whole number: ") Number = Console.ReadLine If Number > 10 Then Console.WriteLine("Number too large") Else If Number < 1 Then Console.WriteLine("Not a positive number.") End If End If End While c = 1 For k = 0 To Number - 1 Console.WriteLine(c) c = (c * (Number - 1 - k)) \ (k + 1) Next Console.ReadLine() End Sub</pre>	9
----	---	--	---

11	1	<pre> Sub SendMorseCode(ByVal MorseCode() As String) Dim PlainText As String Dim PlainTextLength As Integer Dim MorseCodeString As String Dim PlainTextLetter As Char Dim CodedLetter As String Dim Index As Integer Console.Write("Enter your message (uppercase letters and spaces only): ") PlainText = Console.ReadLine() PlainTextLength = PlainText.Length() MorseCodeString = EMPTYSTRING For i = 0 To PlainTextLength - 1 PlainTextLetter = PlainText(i) If PlainTextLetter = SPACE Then Index = 0 ElseIf PlainTextLetter >= "A" And PlainTextLetter <= "Z" Then Index = Asc(PlainTextLetter) - Asc("A") + 1 Else ReportError("Invalid character entered") Index = 0 MorseCodeString = EMPTYSTRING Exit For End If CodedLetter = MorseCode(Index) MorseCodeString = MorseCodeString + CodedLetter + SPACE Next Console.WriteLine(MorseCodeString) End Sub Alternative answer: Sub SendMorseCode(ByVal MorseCode() As String) Dim PlainText As String Dim PlainTextLength As Integer Dim MorseCodeString As String Dim PlainTextLetter As Char Dim CodedLetter As String Dim Index As Integer Console.Write("Enter your message (uppercase letters and spaces only): ") PlainText = Console.ReadLine() Dim Valid As Boolean = True For Each ch In PlainText If ch <> SPACE Then If ch < "A" Or ch > "Z" Then Valid = False MorseCodeString = EMPTYSTYRING ReportError("Invalid character entered") End If </pre>	4
----	---	---	---

```
    End If
Next
If Valid Then
    PlainTextLength = PlainText.Length()
    MorseCodeString = EMPTYSTRING
    For i = 0 To PlainTextLength - 1
        PlainTextLetter = PlainText(i)
        If PlainTextLetter = SPACE Then
            Index = 0
        Else
            Index = Asc(PlainTextLetter) - Asc("A") + 1
        End If
        CodedLetter = MorseCode(Index)
        MorseCodeString = MorseCodeString + CodedLetter +
SPACE
    Next
End If
Console.WriteLine(MorseCodeString)
End Sub
```

MARK SCHEME – AS COMPUTER SCIENCE – 7516/1 – JUNE 2018

12	1	<pre>Sub SendSignals(ByVal MorseCodeString As String) Dim Transmission As String = EMPTYSTRING Dim CodeStringLength = MorseCodeString.Length() Dim Symbol As String Dim SymbolString As String For i = 0 To CodeStringLength - 1 Symbol = MorseCodeString(i) If Symbol = "." Then SymbolString = "= " ElseIf Symbol = "-" Then SymbolString = "=== " ElseIf Symbol = Space Then SymbolString = SPACE + SPACE End If Transmission += SymbolString Next Console.WriteLine(Transmission) End Sub</pre>	7
----	---	---	---

13	1	<pre> Sub OutputAlphabetWithCode(ByVal Letter() As String, ByVal MorseCode() As String) For Ptr = 1 To 26 Console.Write(Letter(Ptr) + " ") Console.Write(MorseCode(Ptr).PadRight(6)) If Ptr Mod 4 = 0 Then Console.WriteLine() End If Next Console.WriteLine() End Sub Sub DisplayMenu() Console.WriteLine() Console.WriteLine("Main Menu") Console.WriteLine("=====") Console.WriteLine("R - Receive Morse code") Console.WriteLine("S - Send Morse code") Console.WriteLine("A - Output alphabet with Morse code") Console.WriteLine("X - Exit Program") Console.WriteLine() End Sub Sub SendReceiveMessages() Dim Dash = {20, 23, 0, 0, 24, 1, 0, 17, 0, 21, 0, 25, 0, 15, 11, 0, 0, 0, 0, 22, 13, 0, 0, 10, 0, 0, 0} Dim Letter = {"SPACE", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"} Dim Dot = {5, 18, 0, 0, 2, 9, 0, 26, 0, 19, 0, 3, 0, 7, 4, 0, 0, 0, 12, 8, 14, 6, 0, 16, 0, 0, 0} Dim MorseCode = {"SPACE", ".-", "-...", "-.-.", "-..", ".", "..-", "--.", "...", ".:", ":-:", "-.-", ".-.-", "-.-", "-.", "...", "-", "-.-", "...-", ".--", "-.-.-", "-.-.-", "-.-.-"} Dim MenuOption As String Dim ProgramEnd As Boolean = False While Not ProgramEnd DisplayMenu() MenuOption = GetMenuOption() If MenuOption = "R" Then ReceiveMorseCode(Dash, Letter, Dot) ElseIf MenuOption = "S" Then SendMorseCode(MorseCode) ElseIf MenuOption = "A" Then OutputAlphabetWithCode(Letter, MorseCode) ElseIf MenuOption = "X" Then ProgramEnd = True End If End While End Sub </pre>	6
----	---	--	---

14	1	<pre> Sub SendMorseCode(ByVal MorseCode() As String, ByVal Keys() As Integer) Dim PlainText As String Dim PlainTextLength As Integer Dim MorseCodeString As String Dim PlainTextLetter As Char Dim CodedLetter As String Dim Index As Integer Console.Write("Enter your message (uppercase letters and spaces only): ") PlainText = Console.ReadLine() Dim Valid As Boolean = True For Each ch In PlainText If ch <> SPACE Then If ch < "A" Or ch > "Z" Then Valid = False MorseCodeString = EMPTYSTRING ReportError("Invalid character entered") End If End If Next If Valid Then PlainTextLength = PlainText.Length() MorseCodeString = EMPTYSTRING For i = 0 To PlainTextLength - 1 PlainTextLetter = PlainText(i) If PlainTextLetter = SPACE Then Index = 0 Else Index = Asc(PlainTextLetter) - Asc("A") + 1 End If Index += Keys(i Mod 3) While Index < 0 Index += 27 End While While Index >= 27 Index -= 27 End While CodedLetter = MorseCode(Index) MorseCodeString = MorseCodeString + CodedLetter + SPACE Next End If Console.WriteLine(MorseCodeString) SendSignals(MorseCodeString) End Sub Sub SendReceiveMessages() Dim Dash = {20, 23, 0, 0, 24, 1, 0, 17, 0, 21, 0, 25, 0, 15, 11, 0, 0, 0, 0, 22, 13, 0, 0, 10, 0, 0, 0} Dim Letter = {"SPACE", "A", "B", "C", "D", "E", "F", </pre>	9
----	---	---	---

```

"G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q",
"R", "S", "T", "U", "V", "W", "X", "Y", "Z"}
    Dim Dot = {5, 18, 0, 0, 2, 9, 0, 26, 0, 19, 0, 3, 0, 7,
4, 0, 0, 0, 12, 8, 14, 6, 0, 16, 0, 0, 0}
    Dim MorseCode = {"SPACE", ".-", "-...", "-.-.", "-..",
".", "..-", "--.", "...", ".-", ".---", "-.-", ".-.-",
"--", "-.", "---", ".--.", "---", "-.", "...", "-", "-.-",
",", "...-", ".--", "-.-", "-.-", "-.-"}
    Dim MenuOption As String
    Dim ProgramEnd As Boolean = False
Dim Keys() As Integer = {0, 0, 0}
Dim ValidDisplacement As Boolean
Dim Displacement As Integer
For i = 0 To 2
    ValidDisplacement = False
    While Not ValidDisplacement
        Try
            Console.Write("Enter encryption key (integer): ")
            Displacement = Console.ReadLine
            ValidDisplacement = True
        Catch ex As Exception
        End Try
        Keys(i) = Displacement
    End While
Next
While Not ProgramEnd
    DisplayMenu()
    MenuOption = GetMenuOption()
    If MenuOption = "R" Then
        ReceiveMorseCode(Dash, Letter, Dot)
    ElseIf MenuOption = "S" Then
        SendMorseCode(MorseCode, Keys)
    ElseIf MenuOption = "A" Then
        OutputAlphabetWithCode(Letter, MorseCode)
    ElseIf MenuOption = "X" Then
        ProgramEnd = True
    End If
End While
End Sub

```

Alternative answer:

```

Sub SendMorseCode(ByVal MorseCode() As String, ByVal Key1
As Integer, ByVal Key2 As Integer, ByVal Key3 As Integer)
    Dim PlainText As String
    Dim PlainTextLength As Integer
    Dim MorseCodeString As String
    Dim PlainTextLetter As Char
    Dim CodedLetter As String
    Dim Index As Integer
    Dim Displacement As Integer
    Console.WriteLine("Enter your message (uppercase letters and
spaces only): ")
    PlainText = Console.ReadLine()
    Dim Valid As Boolean = True
    For Each ch In PlainText
        If ch <> SPACE Then
            If ch < "A" Or ch > "Z" Then
                Valid = False
                MorseCodeString = EMPTYSTRING
                ReportError("Invalid character entered")
            End If
        End If
    Next
    If Valid Then
        PlainTextLength = PlainText.Length()
        MorseCodeString = EMPTYSTRING
        For i = 0 To PlainTextLength - 1
            PlainTextLetter = PlainText(i)
            If PlainTextLetter = SPACE Then
                Index = 0
            Else
                Index = Asc(PlainTextLetter) - Asc("A") + 1
            End If
            If i Mod 3 = 0 Then
                Displacement = Key1
            ElseIf i Mod 3 = 1 Then
                Displacement = Key2
            Else
                Displacement = Key3
            End If
            Index += Displacement
            While Index < 0
                Index += 27
            End While
            While Index >= 27
                Index -= 27
            End While
            CodedLetter = MorseCode(Index)
            MorseCodeString = MorseCodeString + CodedLetter +
SPACE
        Next
    End If
End Sub

```

```

End If
Console.WriteLine(MorseCodeString)
SendSignals(MorseCodeString)
End Sub

Sub SendReceiveMessages()
    Dim Dash = {20, 23, 0, 0, 24, 1, 0, 17, 0, 21, 0, 25, 0,
15, 11, 0, 0, 0, 0, 22, 13, 0, 0, 10, 0, 0, 0}
    Dim Letter = {"SPACE", "A", "B", "C", "D", "E", "F",
"G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q",
"R", "S", "T", "U", "V", "W", "X", "Y", "Z"}
    Dim Dot = {5, 18, 0, 0, 2, 9, 0, 26, 0, 19, 0, 3, 0, 7,
4, 0, 0, 0, 12, 8, 14, 6, 0, 16, 0, 0, 0}
    Dim MorseCode = {"SPACE", ".-", "-...", "-.-.", "-..",
".", "-.-.", "--.", "...", "..", ".---", "-.-", "-...",
"--", "-.", "---", ".---", "--.-", ".-.", "...", "-", "-.-",
",", "...-", ".--", "-..-", "-.-", "--.."}
    Dim MenuOption As String
    Dim ProgramEnd As Boolean = False
    Dim ValidDisplacement As Boolean
    Dim Key1 As Integer
    Dim Key2 As Integer
    Dim Key3 As Integer
    ValidDisplacement = False
    While Not ValidDisplacement
        Try
            Console.Write("Enter encryption key (integer): ")
            Key1 = Console.ReadLine
            ValidDisplacement = True
        Catch ex As Exception
        End Try
    End While
    ValidDisplacement = False
    While Not ValidDisplacement
        Try
            Console.Write("Enter encryption key (integer): ")
            Key2 = Console.ReadLine
            ValidDisplacement = True
        Catch ex As Exception
        End Try
    End While
    ValidDisplacement = False
    While Not ValidDisplacement
        Try
            Console.Write("Enter encryption key (integer): ")
            Key3 = Console.ReadLine
            ValidDisplacement = True
        Catch ex As Exception
        End Try
    End While
    While Not ProgramEnd

```

	<pre>DisplayMenu() MenuOption = GetMenuOption() If MenuOption = "R" Then ReceiveMorseCode(Dash, Letter, Dot) ElseIf MenuOption = "S" Then SendMorseCode(MorseCode, Key1, Key2, Key3) ElseIf MenuOption = "A" Then OutputAlphabetWithCode(Letter, MorseCode) ElseIf MenuOption = "X" Then ProgramEnd = True End If End While End Sub</pre>	
--	--	--

Python 2

03	1	<pre>Number = 0 while Number < 1 or Number > 10: Number = int(raw_input("Enter a positive whole number: ")) if Number > 10: print "Number too large" elif Number < 1: print "Not a positive number" c = 1 for k in range(Number): print c c = (c * (Number - 1 - k)) // (k + 1)</pre>	9
----	---	---	---

11	1	<pre>def SendMorseCode(MorseCode): PlainText = raw_input("Enter your message (uppercase letters and spaces only): ") PlainTextLength = len(PlainText) MorseCodeString = EMPTYSTRING for i in range(PlainTextLength): PlainTextLetter = PlainText[i] if PlainTextLetter == SPACE: Index = 0 elif PlainTextLetter >= 'A' and PlainTextLetter <= 'Z': Index = ord(PlainTextLetter) - ord('A') + 1 else: ReportError("Invalid character entered") Index = 0 MorseCodeString = EMPTYSTRING break CodedLetter = MorseCode[Index] MorseCodeString = MorseCodeString + CodedLetter + SPACE print MorseCodeString</pre> <p>Alternative answer:</p> <pre>def SendMorseCode(MorseCode): PlainText = raw_input("Enter your message (uppercase letters and spaces only): ") Valid = True for Character in PlainText: if Character != SPACE: if Character < "A" or Character > "Z": Valid = False MorseCodeString = EMPTYSTRING ReportError("Invalid character entered") break if Valid: PlainTextLength = len(PlainText) MorseCodeString = EMPTYSTRING for i in range(PlainTextLength): PlainTextLetter = PlainText[i] if PlainTextLetter == SPACE: Index = 0 else: Index = ord(PlainTextLetter) - ord('A') + 1 CodedLetter = MorseCode[Index] MorseCodeString = MorseCodeString + CodedLetter + SPACE print MorseCodeString</pre>	4
----	---	--	---

12	1	<pre>def SendSignals(MorseCodeString): Transmission = EMPTYSTRING CodeStringLength = len(MorseCodeString) for i in range(CodeStringLength): Symbol = MorseCodeString[i] if Symbol == '.': SymbolString = "= " elif Symbol == '-': SymbolString = "=== " elif Symbol == SPACE: SymbolString = SPACE + SPACE Transmission = Transmission + SymbolString print Transmission</pre>	7
----	---	--	---

1 3	1	<pre> def OutputAlphabetWithCode(Letter, MorseCode): for Ptr in range(1, 27): print Letter[Ptr], end=SPACE print '{0:<5}'.format(MorseCode[Ptr]), end=SPACE if Ptr % 4 == 0: print print def DisplayMenu(): print print "Main Menu" print "======" print "R - Receive Morse code" print "S - Send Morse code" print "A - Output alphabet with Morse code" print "X - Exit program" print def SendReceiveMessages(): Dash = [20,23,0,0,24,1,0,17,0,21,0,25,0,15,11,0,0,0,0,22,13,0,0,10,0,0,0] Dot = [5,18,0,0,2,9,0,26,0,19,0,3,0,7,4,0,0,0,12,8,14,6,0,16,0,0,0] Letter = ["SPACE",'A','B','C','D','E','F','G','H','I','J','K','L','M','N',' O','P','Q','R','S','T','U','V','W','X','Y','Z'] MorseCode = ["SPACE",'.-','-...','-.-.','-..','.',','.-..','-- .',',...','..','-.--','-.-','-...','--','-.','-.-','-.-.',',-.- ',',.-.','...','-',',..-','...-',',.-','-..-',',-.-','-.-..'] ProgramEnd = False while not ProgramEnd: DisplayMenu() MenuOption = GetMenuOption() if MenuOption == 'R': ReceiveMorseCode(Dash, Letter, Dot) elif MenuOption == 'S': SendMorseCode(MorseCode) elif MenuOption == 'A': OutputAlphabetWithCode(Letter, MorseCode) elif MenuOption == 'X': ProgramEnd = True </pre>	6
--------	---	---	---

1 4	1	<pre> def SendMorseCode(MorseCode, Keys): PlainText = raw_input("Enter your message (uppercase letters and spaces only): ") PlainTextLength = len(PlainText) MorseCodeString = EMPTYSTRING for i in range(PlainTextLength): PlainTextLetter = PlainText[i] if PlainTextLetter == SPACE: Index = 0 else: Index = ord(PlainTextLetter) - ord('A') + 1 Index = Index + Keys[i % 3] while Index < 0: Index = Index + 27 while Index >= 27: Index = Index - 27 CodedLetter = MorseCode[Index] MorseCodeString = MorseCodeString + CodedLetter + SPACE print(MorseCodeString) def SendReceiveMessages(): Dash = [20,23,0,0,24,1,0,17,0,21,0,25,0,15,11,0,0,0,0,22,13,0,0,10,0,0,0] Dot = [5,18,0,0,2,9,0,26,0,19,0,3,0,7,4,0,0,0,12,8,14,6,0,16,0,0,0] Letter = ["SPACE",'A','B','C','D','E','F','G','H','I','J','K','L','M','N',' O','P','Q','R','S','T','U','V','W','X','Y','Z'] MorseCode = ["SPACE",'.-','-...','-.-.','-...','.','.-.-.','-- .','.-...','..','-...','-.-.','-...','--','-.-.','-...','-.-.- ','-...','-','-...','-...-','-...','-.-.','-.-.-','-...'] ProgramEnd = False Keys = [0,0,0] for i in range(3): ValidDisplacement = False while not ValidDisplacement: try: Displacement = int(raw_input("Enter encryption key (integer): ")) ValidDisplacement = True except: pass Keys[i] = Displacement while not ProgramEnd: DisplayMenu() MenuOption = GetMenuOption() if MenuOption == 'R': ReceiveMorseCode(Dash, Letter, Dot) </pre>	9
--------	---	--	---

```

elif MenuOption == 'S':
    SendMorseCode(MorseCode, Keys)
elif MenuOption == 'X':
    ProgramEnd = True

```

Alternative answer:

```

def SendMorseCode(MorseCode, Key1, Key2, Key3):
    PlainText = input("Enter your message (uppercase letters and
spaces only): ")
    PlainTextLength = len(PlainText)
    MorseCodeString = EMPTYSTRING
    for i in range(PlainTextLength):
        PlainTextLetter = PlainText[i]
        if PlainTextLetter == SPACE:
            Index = 0
        else:
            Index = ord(PlainTextLetter) - ord('A') + 1
        if i % 3 == 0:
            Displacement = Key1
        elif i % 3 == 1:
            Displacement = Key2
        else:
            Displacement = Key3
        Index = Index + Displacement
        while Index < 0:
            Index = Index + 27
        while Index >= 27:
            Index = Index - 27
        CodedLetter = MorseCode[Index]
        MorseCodeString = MorseCodeString + CodedLetter + SPACE
    print(MorseCodeString)

def SendReceiveMessages():
    Dash =
[20,23,0,0,24,1,0,17,0,21,0,25,0,15,11,0,0,0,0,22,13,0,0,10,0,0,0]
    Dot =
[5,18,0,0,2,9,0,26,0,19,0,3,0,7,4,0,0,0,12,8,14,6,0,16,0,0,0]
    Letter =
["SPACE", 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', '
O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']

    MorseCode = ["SPACE", '.-', '-...', '-.-.', '-..', '.', '..-', '--
.', '....', '....', '---', '-.-', '-..', '--', '.-', '---', '-.-',
', '-.', '...', '-.', '...-', '...-', '---', '-.-', '-.-', '---..']
    ProgramEnd = False

ValidDisplacement = False
while not ValidDisplacement:
    try:
        Key1 = int(input("Enter encryption key (integer): "))
        ValidDisplacement = True

```

```
    except:
        pass
ValidDisplacement = False
while not ValidDisplacement:
    try:
        Key2 = int(input("Enter encryption key (integer): "))
        ValidDisplacement = True
    except:
        pass
ValidDisplacement = False
while not ValidDisplacement:
    try:
        Key3 = int(input("Enter encryption key (integer): "))
        ValidDisplacement = True
    except:
        pass

while not ProgramEnd:
    DisplayMenu()
    MenuOption = GetMenuOption()
    if MenuOption == 'R':
        ReceiveMorseCode(Dash, Letter, Dot)
    elif MenuOption == 'S':
        SendMorseCode(MorseCode, Key1, Key2, Key3)
    elif MenuOption == 'X':
        ProgramEnd = True
```

Python 3

03	1	<pre>Number = 0 while Number < 1 or Number > 10: Number = int(input("Enter a positive whole number: ")) if Number > 10: print("Number too large") elif Number < 1: print("Not a positive number") c = 1 for k in range(Number): print(c) c = (c * (Number - 1 - k)) // (k + 1)</pre>	9
-----------	----------	--	----------

11	1	<pre>def SendMorseCode(MorseCode): PlainText = input("Enter your message (uppercase letters and spaces only): ") PlainTextLength = len(PlainText) MorseCodeString = EMPTYSTRING for i in range(PlainTextLength): PlainTextLetter = PlainText[i] if PlainTextLetter == SPACE: Index = 0 elif PlainTextLetter >= 'A' and PlainTextLetter <= 'Z': Index = ord(PlainTextLetter) - ord('A') + 1 else: ReportError("Invalid character entered") Index = 0 MorseCodeString = EMPTYSTRING break CodedLetter = MorseCode[Index] MorseCodeString = MorseCodeString + CodedLetter + SPACE print(MorseCodeString)</pre>	4
-----------	----------	---	----------

Alternative answer:

```
def SendMorseCode(MorseCode):
    PlainText = input("Enter your message (uppercase letters and
spaces only): ")
    Valid = True
    for Character in PlainText:
        if Character != SPACE:
            if Character < "A" or Character > "Z":
                Valid = False
                MorseCodeString = EMPTYSTRING
                ReportError("Invalid character entered")
                break
    if Valid:
        PlainTextLength = len(PlainText)
        MorseCodeString = EMPTYSTRING
        for i in range(PlainTextLength):
            PlainTextLetter = PlainText[i]
            if PlainTextLetter == SPACE:
                Index = 0
            else:
                Index = ord(PlainTextLetter) - ord('A') + 1
            CodedLetter = MorseCode[Index]
            MorseCodeString = MorseCodeString + CodedLetter + SPACE
    print(MorseCodeString)
```

12	1	<pre>def SendSignals(MorseCodeString): Transmission = EMPTYSTRING CodeStringLength = len(MorseCodeString) for i in range(CodeStringLength): Symbol = MorseCodeString[i] if Symbol == '.': SymbolString = "= " elif Symbol == '-': SymbolString = "=== " elif Symbol == SPACE: SymbolString = SPACE + SPACE Transmission = Transmission + SymbolString print(Transmission)</pre>	7
-----------	----------	---	----------

```
13 1 def OutputAlphabetWithCode(Letter, MorseCode):
    for Ptr in range(1, 27):
        print(Letter[Ptr], end=SPACE)
        print('{0:<5}'.format(MorseCode[Ptr]), end=SPACE)
        if Ptr % 4 == 0:
            print()
    print()

def DisplayMenu():
    print()
    print("Main Menu")
    print("=====")
    print("R - Receive Morse code")
    print("S - Send Morse code")
    print("A - Output alphabet with Morse code")
    print("X - Exit program")
    print()

def SendReceiveMessages():
Dash = [20,23,0,0,24,1,0,17,0,21,0,25,0,15,11,0,0,0,0,22,13,0,0,10,0,0,0]
Dot = [5,18,0,0,2,9,0,26,0,19,0,3,0,7,4,0,0,0,12,8,14,6,0,16,0,0,0]
Letter =
["SPACE", 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']

MorseCode = ["SPACE", '.-', '-...', '-.-.', '-...', '.!', '...-', '---.', '.....', '...',
'.---.', '---.', '.-.', '...!', '-.', '...-', '...-', '---.', '-.-.', '---!']
ProgramEnd = False
while not ProgramEnd:
    DisplayMenu()
    MenuOption = GetMenuOption()
    if MenuOption == 'R':
        ReceiveMorseCode(Dash, Letter, Dot)
    elif MenuOption == 'S':
        SendMorseCode(MorseCode)
    elif MenuOption == 'A':
        OutputAlphabetWithCode(Letter, MorseCode)
    elif MenuOption == 'X':
        ProgramEnd = True
```



```

14 1 def SendMorseCode(MorseCode, Keys):
    PlainText = input("Enter your message (uppercase letters and spaces only)
    PlainTextLength = len(PlainText)
    MorseCodeString = EMPTYSTRING
    for i in range(PlainTextLength):
        PlainTextLetter = PlainText[i]
        if PlainTextLetter == SPACE:
            Index = 0
        else:
            Index = ord(PlainTextLetter) - ord('A') + 1
            Index = Index + Keys[i % 3]
            while Index < 0:
                Index = Index + 27
            while Index >= 27:
                Index = Index - 27
            CodedLetter = MorseCode[Index]
            MorseCodeString = MorseCodeString + CodedLetter + SPACE
    print(MorseCodeString)

def SendReceiveMessages():
Dash = [20,23,0,0,24,1,0,17,0,21,0,25,0,15,11,0,0,0,0,22,13,0,0,10,0,0,0]
Dot = [5,18,0,0,2,9,0,26,0,19,0,3,0,7,4,0,0,0,12,8,14,6,0,16,0,0,0]
Letter =
["SPACE", 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q',
'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '.']

MorseCode = ["SPACE", '.-', '-...', '-.-.', '-. .', '. . .', '.-.-.', '--.', '... .', '...',
'-. .', '--.-', '.-.', '...', '-.', '-.-', '...-', '---', '-.-.', '-.-.']
ProgramEnd = False

Keys = [0,0,0]
for i in range(3):
    ValidDisplacement = False
    while not ValidDisplacement:
        try:
            Displacement = int(input("Enter encryption key (integer): "))
            ValidDisplacement = True
        except:
            pass
    Keys[i] = Displacement

while not ProgramEnd:
    DisplayMenu()
    MenuOption = GetMenuOption()
    if MenuOption == 'R':
        ReceiveMorseCode(Dash, Letter, Dot)
    elif MenuOption == 'S':
        SendMorseCode(MorseCode, Keys)
    elif MenuOption == 'X':
        ProgramEnd = True

```

Alternative answer:

```

def SendMorseCode(MorseCode, Key1, Key2, Key3):
    PlainText = input("Enter your message (uppercase letters and spaces only)")
    PlainTextLength = len(PlainText)
    MorseCodeString = EMPTYSTRING
    for i in range(PlainTextLength):
        PlainTextLetter = PlainText[i]
        if PlainTextLetter == SPACE:
            Index = 0
        else:
            Index = ord(PlainTextLetter) - ord('A') + 1
        if i % 3 == 0:
            Displacement = Key1
        elif i % 3 == 1:
            Displacement = Key2
        else:
            Displacement = Key3
        Index = Index + Displacement
        while Index < 0:
            Index = Index + 27
        while Index >= 27:
            Index = Index - 27
        CodedLetter = MorseCode[Index]
        MorseCodeString = MorseCodeString + CodedLetter + SPACE
    print(MorseCodeString)

def SendReceiveMessages():
    Dash = [20,23,0,0,24,1,0,17,0,21,0,25,0,15,11,0,0,0,0,22,13,0,0,10,0,0,0]
    Dot = [5,18,0,0,2,9,0,26,0,19,0,3,0,7,4,0,0,0,12,8,14,6,0,16,0,0,0]
    Letter =
["SPACE", 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']

    MorseCode = ["SPACE", '.-', '-...', '-.-.', '-..', '.!', '...-', '--.', '....', '...',
    '.---', '---.', '.-.', '...', '-.', '...-', '---', '-...', '-.-.', '---']
    ProgramEnd = False

    ValidDisplacement = False
    while not ValidDisplacement:
        try:
            Key1 = int(input("Enter encryption key (integer): "))
            ValidDisplacement = True
        except:
            pass
    ValidDisplacement = False
    while not ValidDisplacement:
        try:
            Key2 = int(input("Enter encryption key (integer): "))
            ValidDisplacement = True
        except:
            pass
    ValidDisplacement = False

```

```
while not ValidDisplacement:
    try:
        Key3 = int(input("Enter encryption key (integer): "))
        ValidDisplacement = True
    except:
        pass

while not ProgramEnd:
    DisplayMenu()
    MenuOption = GetMenuOption()
    if MenuOption == 'R':
        ReceiveMorseCode(Dash, Letter, Dot)
    elif MenuOption == 'S':
        SendMorseCode(MorseCode, Key1, Key2, Key3)
    elif MenuOption == 'X':
        ProgramEnd = True
```

Pascal

03	1	<pre>var Number, c, k : Integer; begin Number := 0; while (Number < 1) or (Number > 10) do begin write('Enter a positive whole number: '); readln(Number); if Number > 10 then writeln('Number too large') else if Number < 1 then writeln('Not a positive number.') end; c := 1; for k := 0 to (Number - 1) do begin writeln(c); c := (c * (Number - 1 - k)) div (k + 1); end; readln; end.</pre>	9
----	---	--	---

11	1	<pre> Procedure SendMessage(MorseCode : Array of String); var PlainText, MorseCodeString, CodedLetter : String; PlainTextLength, i, Index : Integer; PlainTextLetter : Char; begin write('Enter your message (uppercase letters and spaces only): '); readln(PlainText); PlainTextLength := length(PlainText); MorseCodeString := EMPTYSTRING; for i := 1 to PlainTextLength do begin PlainTextLetter := PlainText[i]; if PlainTextLetter = SPACE then Index := 0 else if (PlainTextLetter >= 'A') and (PlainTextLetter <= 'Z') then Index := ord(PlainTextLetter) - ord('A') + 1 else begin ReportError('Invalid character entered'); Index := 0; MorseCodeString := EMPTYSTRING; break; end; CodedLetter := MorseCode[Index]; MorseCodeString := MorseCodeString + CodedLetter + SPACE; end; writeln(MorseCodeString); end; end; </pre>	4
----	---	---	---

12	1	<pre>Procedure SendSignals(MorseCodeString : String); var Transmission, SymbolString : String; CodeStringLength, i : Integer; Symbol : Char; begin Transmission := EMPTYSTRING; CodeStringLength := length(MorseCodeString); for i := 1 to CodeStringLength do begin Symbol := MorseCodeString[i]; if Symbol = '.' then SymbolString := '=' else if Symbol = '-' then SymbolString := '=== ' else if Symbol = SPACE then SymbolString := SPACE + SPACE; Transmission := Transmission + SymbolString; end; writeln(Transmission); end; end; end;</pre>	7
----	---	---	---

13	1	<pre>Procedure OutputAlphabetWithCode(Letter : Array of Char; MorseCode : Array of String); var Index : Integer; begin for Index := 1 to 26 do begin write(Letter[Index], SPACE); write(Format('%0:-6s', [MorseCode[Index]])); if index mod 4 = 0 then writeln; end; writeln; end; Procedure DisplayMenu(); begin writeln; writeln('Main Menu'); writeln('====='); writeln('R - Receive Morse code'); writeln('S = Send Morse code'); writeln('A - Output alphabet with Morse code'); writeln('X - Exit program'); writeln; end;</pre>	6
----	---	--	---

1 4	1	<pre>Procedure SendMorseCode (MorseCode : TStringArray; Keys : Array of Integer); var PlainText, MorseCodeString, CodedLetter : String; PlainTextLength, i, Index, Displacement : Integer; PlainTextLetter : Char; begin write('Enter your message (uppercase letters and spaces only): '); readln(PlainText); PlainTextLength := length(PlainText); MorseCodeString := EMPTYSTRING; for i := 1 to PlainTextLength do begin PlainTextLetter := PlainText[i]; if PlainTextLetter = SPACE then Index := 0 else Index := ord(PlainTextLetter) - ord('A') + 1 Displacement := Keys[(i-1) mod 3]; Index := Index + Displacement;</pre>
--------	---	--


```

    while Index < 0 do
        Index := Index + 27;
    while Index >= 27 do
        Index := Index - 27;
    CodedLetter := MorseCode[Index];
    MorseCodeString := MorseCodeString + CodedLetter + SPACE;
    end;
    writeln(MorseCodeString);
end;

Procedure SendReceiveMessages();
var
    Dash: array[0..26] of Integer =
(20,23,0,0,24,1,0,17,0,21,0,25,0,15,11,0,0,0,0,22,13,0,0,10,0,0,
0);
    Dot : array[0..26] of Integer =
(5,18,0,0,2,9,0,26,0,19,0,3,0,7,4,0,0,0,12,8,14,6,0,16,0,0,0);
    Letter : array[0..26] of Char = ('
','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P
','Q','R','S','T','U','V','W','X','Y','Z');
    MorseCode : array[0..26] of String = (' ','.-','-...','-.-
','-..','..','...-','---','....','...','---','-.-','-.-
','-.','---','--..','--.-','-.','...','-','..-','...-','--
'..'-'','-.-','--..');
    ProgramEnd : Boolean;
    MenuOption, Input : String;
    Keys : array [0..2] of Integer = (0,0,0);
    i, Error : Integer;
begin
    ProgramEnd := False;
    for i := 0 to 2 do
        begin
            write('Enter encryption key (integer): ');
            readln(Input);
            val(Input,Keys[i],Error);
            while Error <> 0 do
                begin
                    write('Error - invalid input - please re-enter: ');
                    readln(Input);
                    val(Input,Keys[i],Error);
                end;
            end;
        end;

    while not(ProgramEnd) do
        begin
            DisplayMenu();
            MenuOption := GetMenuOption();
            if MenuOption = 'R' then
                ReceiveMorseCode(Dash, Letter, Dot)
            else if MenuOption = 'S' then
                SendMorseCode(MorseCode, Keys)

```

	<pre> else if MenuOption = 'X' then ProgramEnd := True; end; end;</pre>	
--	--	--

C#

03	1	<pre>static void Main(string[] args) { int Number = 0; while (Number < 1 Number > 10) { Console.WriteLine("Enter a positive whole number: "); Number = Convert.ToInt32(Console.ReadLine()); if (Number > 10) { Console.WriteLine("Number too large."); } else { if (Number < 1) { Console.WriteLine("Not a positive number."); } } } int c = 1; for (int k = 0; k < Number ; k++) { Console.WriteLine(c); c = (c * (Number - 1 - k)) / (k + 1); } Console.ReadLine(); }</pre>	9
----	---	---	---

11	1	<pre> private static void SendMorseCode(string[] MorseCode) { Console.Write("Enter your message (uppercase letters and spaces only): "); string PlainText = Console.ReadLine(); int PlainTextLength = PlainText.Length; string MorseCodeString = EMPTYSTRING; char PlainTextLetter = SPACE; int Index = 0; for (int i = 0; i < PlainTextLength; i++) { PlainTextLetter = PlainText[i]; if (PlainTextLetter == SPACE) { Index = 0; } else if (PlainTextLetter >= 'A' && PlainTextLetter <= 'Z') { Index = (int)PlainTextLetter - (int)'A' + 1; } else { ReportError("Invalid character entered"); Index = 0; MorseCodeString = EMPTYSTRING; break; } string CodedLetter = MorseCode[Index]; MorseCodeString = MorseCodeString + CodedLetter + SPACE; } Console.WriteLine(MorseCodeString); } </pre> <p>Alternative answer</p> <pre> private static void SendMorseCode(string[] MorseCode) { bool Valid = true; Console.Write("Enter your message (uppercase letters and spaces only): "); string PlainText = Console.ReadLine(); int PlainTextLength = PlainText.Length; string MorseCodeString = EMPTYSTRING; char PlainTextLetter = ' '; int Index = 0; foreach (char Character in PlainText) { if (Character != SPACE) { if (Character < 'A' Character > 'Z') { Valid = false; MorseCodeString = EMPTYSTRING; </pre>	4
----	---	--	---

```
        ReportError("Invalid character entered");
        break;
    }
}
if (Valid)
{
    for (int i = 0; i < PlainTextLength; i++)
    {
        PlainTextLetter = PlainText[i];
        if (PlainTextLetter == SPACE)
        {
            Index = 0;
        }
        else
        {
            Index = (int)PlainTextLetter - (int)'A' + 1;
        }
        string CodedLetter = MorseCode[Index];
        MorseCodeString = MorseCodeString + CodedLetter +
SPACE;
    }
}
Console.WriteLine(MorseCodeString);
}
```

12	1	<pre>private static void SendSignals(string MorseCodeString) { string Transmission = EMPTYSTRING; char Symbol; string SymbolString = ""; int CodeStringLength = MorseCodeString.Length; for (int i = 0; i < CodeStringLength; i++) { Symbol = MorseCodeString[i]; if (Symbol == '.') { SymbolString = "= "; } else if (Symbol == '-') { SymbolString = "=== "; } if (Symbol == SPACE) { SymbolString = SPACE.ToString() + SPACE.ToString(); } Transmission = Transmission + SymbolString; } Console.WriteLine(Transmission); }</pre>	7
----	---	---	---

13	1	<pre> private static void OutputAlphabetWithCode(char[] Letter, string[] MorseCode) { for (int Ptr = 1; Ptr < 27; Ptr++) { Console.Write(Letter[Ptr]); Console.Write(SPACE); Console.Write("{0,-6}",MorseCode[Ptr]); if (Ptr % 4 == 0) { Console.WriteLine(); } } } private static void SendReceiveMessages() { int[] Dash = new int[] { 20, 23, 0, 0, 24, 1, 0, 17, 0, 21, 0, 25, 0, 15, 11, 0, 0, 0, 0, 22, 13, 0, 0, 10, 0, 0, 0 }; int[] Dot = new int[] { 5, 18, 0, 0, 2, 9, 0, 26, 0, 19, 0, 3, 0, 7, 4, 0, 0, 0, 12, 8, 14, 6, 0, 16, 0, 0, 0 }; char[] Letter = new char[] { "SPACE", 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z' }; string[] MorseCode = new string[] { " ", ".-", "-...", "-.-.", "-..", ".", "-.-.", "--.", "....", "..", ".---", "-.-", ".-..", "--", "-.", "----", ".--.", "---.", ".-.", "...", "-", ".-.", "...-", ".--", "-...-", "-.---", "--.." }; bool ProgramEnd = false; string MenuOption = EMPTYSTRING; while (!ProgramEnd) { DisplayMenu(); GetMenuOption(ref MenuOption); if (MenuOption == "R") { ReceiveMorseCode(Dash, Letter, Dot); } else if (MenuOption == "S") { SendMorseCode(MorseCode); } else if (MenuOption == "A") { OutputAlphabetWithCode(Letter, MorseCode); } else if (MenuOption == "X") { ProgramEnd = true; } } } </pre>	6
----	---	--	---

```
private static void DisplayMenu()  
{  
    Console.WriteLine();  
    Console.WriteLine("Main Menu");  
    Console.WriteLine("=====");  
    Console.WriteLine("R - Receive Morse code");  
    Console.WriteLine("S - Send Morse code");  
    Console.WriteLine("A - Output alphabet with Morse code");  
    Console.WriteLine("X - Exit program");  
    Console.WriteLine();  
}
```


14	1	<pre> private static void SendMorseCode(string[] MorseCode, int[] Keys) { Console.WriteLine("Enter your message (uppercase letters and spaces only): "); string PlainText = Console.ReadLine(); int PlainTextLength = PlainText.Length; string MorseCodeString = EMPTYSTRING; char PlainTextLetter = ' '; int Index = 0; for (int i = 0; i < PlainTextLength; i++) { PlainTextLetter = PlainText[i]; if (PlainTextLetter == SPACE) { Index = 0; } else { Index = (int)PlainTextLetter - (int)'A' + 1; } Index = Index + Keys[i % 3]; while (Index < 0) { Index = Index + 27; } while (Index >=27) { Index = Index - 27; } string CodedLetter = MorseCode[Index]; MorseCodeString = MorseCodeString + CodedLetter + SPACE; } Console.WriteLine(MorseCodeString); } private static void SendReceiveMessages() { int[] Dash = new int[] { 20, 23, 0, 0, 24, 1, 0, 17, 0, 21, 0, 25, 0, 15, 11, 0, 0, 0, 0, 22, 13, 0, 0, 10, 0, 0, 0 }; int[] Dot = new int[] { 5, 18, 0, 0, 2, 9, 0, 26, 0, 19, 0, 3, 0, 7, 4, 0, 0, 0, 12, 8, 14, 6, 0, 16, 0, 0, 0 }; char[] Letter = new char[] { "SPACE", 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z' }; string[] MorseCode = new string[] { " ", ".-", "-...", "-.-.", "-..", ".", "-.-.", "--.", "...", "..", ".---", "-.-.", "-.-.", "--", "-.", "---", ".--", "--.", ".-", "...", "-", "-.-", "...-", ".--", "-...-", "-.-.", "---." }; bool ProgramEnd = false; string MenuOption = EMPTYSTRING; int[] Keys = new int[3]; </pre>	9
----	---	---	---

```

int Displacement = 0;
for (int i = 0; i < 3; i++)
{
    bool ValidDisplacement = false;
    while (!ValidDisplacement)
    {
        Console.WriteLine("Enter encryption key (integer): ");
        try
        {
            Displacement = Convert.ToInt32(Console.ReadLine());
            ValidDisplacement = true;
        }
        catch (Exception)
        {
        }
    }
    Keys[i] = Displacement;
}
while (!ProgramEnd)
{
    DisplayMenu();
    GetMenuOption(ref MenuOption);
    if (MenuOption == "R")
    {
        ReceiveMorseCode(Dash, Letter, Dot);
    }
    else if (MenuOption == "S")
    {
        SendMorseCode(MorseCode, Keys);
    }
    else if (MenuOption == "X")
    {
        ProgramEnd = true;
    }
}
}

```

Alternative Answer:

```

private static void SendMorseCode(string[] MorseCode, int Key1,
int Key2, int Key3)
{
    Console.WriteLine("Enter your message (uppercase letters and
spaces only): ");
    string PlainText = Console.ReadLine();
    int PlainTextLength = PlainText.Length;
    string MorseCodeString = EMPTYSTRING;
    char PlainTextLetter = ' ';
    int Index = 0;
    int Displacement = 0;
    for (int i = 0; i < PlainTextLength; i++)

```

```

{
    PlainTextLetter = PlainText[i];
    if (PlainTextLetter == SPACE)
    {
        Index = 0;
    }
    else
    {
        Index = (int)PlainTextLetter - (int)'A' + 1;
    }
    if (i % 3 == 0)
    {
        Displacement = Key1;
    }
    else if (i % 3 == 1)
    {
        Displacement = Key2;
    }
    else
    {
        Displacement = Key3;
    }
    Index = Index + Displacement;
    while (Index < 0)
    {
        Index = Index + 27;
    }
    while (Index >= 27)
    {
        Index = Index - 27;
    }
    string CodedLetter = MorseCode[Index];
    MorseCodeString = MorseCodeString + CodedLetter + SPACE;
}
Console.WriteLine(MorseCodeString);
}

private static void SendReceiveMessages()
{
    int[] Dash = new int[] { 20, 23, 0, 0, 24, 1, 0, 17, 0, 21, 0,
25, 0, 15, 11, 0, 0, 0, 0, 22, 13, 0, 0, 10, 0, 0, 0 };
    int[] Dot = new int[] { 5, 18, 0, 0, 2, 9, 0, 26, 0, 19, 0, 3,
0, 7, 4, 0, 0, 0, 12, 8, 14, 6, 0, 16, 0, 0, 0 };
    char[] Letter = new char[] { "SPACE", 'A', 'B', 'C', 'D', 'E',
'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R',
'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z' };
    string[] MorseCode = new string[] { " ", ".-", "-...", "-.-.",
"-..", ".", "-.-.", "--.", "....", "..", ".---", "-.-", ".-..",
"--", "-.", "---", ".--.", "--.-", ".-.", "...", "-", ".-.",
"...-", ".--", "-.-", "-.---", "--.." };
    bool ProgramEnd = false;
}

```

```
string MenuOption = EMPTYSTRING;
int Key1 = 0, Key2 = 0, Key3 = 0;
bool ValidDisplacement = false;
while (!ValidDisplacement)
{
    Console.WriteLine("Enter encryption key (integer): ");
    try
    {
        Key1 = Convert.ToInt32(Console.ReadLine());
        ValidDisplacement = true;
    }
    catch (Exception)
    {
    }
}
ValidDisplacement = false;
while (!ValidDisplacement)
{
    Console.WriteLine("Enter encryption key (integer): ");
    try
    {
        Key2 = Convert.ToInt32(Console.ReadLine());
        ValidDisplacement = true;
    }
    catch (Exception)
    {
    }
}
ValidDisplacement = false;
while (!ValidDisplacement)
{
    Console.WriteLine("Enter encryption key (integer): ");
    try
    {
        Key3 = Convert.ToInt32(Console.ReadLine());
        ValidDisplacement = true;
    }
    catch (Exception)
    {
    }
}
while (!ProgramEnd)
{
    DisplayMenu();
    GetMenuOption(ref MenuOption);
    if (MenuOption == "R")
    {
        ReceiveMorseCode(Dash, Letter, Dot);
    }
    else if (MenuOption == "S")
    {
```

	<pre> SendMorseCode(MorseCode, Key1, Key2, Key3); } else if (MenuOption == "X") { ProgramEnd = true; } } }</pre>	
--	--	--

Java

03	1	<pre>int number = 0; while (number < 1 number > 10) { Console.WriteLine("Enter a positive whole number: "); number = Integer.parseInt(Console.ReadLine()); if (number > 10) { Console.WriteLine("Number too large"); } else if (number < 1) { Console.WriteLine("Not a positive number"); } } int c = 1; for (int k = 0; k < number; k++) { Console.write(c + " "); c = (c * (number - 1 - k) / (k + 1)); }</pre>	9
----	---	---	---

11	1	<pre>void sendMorseCode(String[] morseCode) { Console.write("Enter your message (uppercase letters and spaces only): "); String plainText = Console.readLine(); int plainTextLength = plainText.length(); String morseCodeString = EMPTYSTRING; int index; for (int i = 0; i < plainTextLength; i++) { char plainTextLetter = plainText.charAt(i); if (plainTextLetter == SPACE) { index = 0; } else { index = (int)plainTextLetter - (int)'A' + 1; } if (index >= 0 && index <= 26) { String codedLetter = morseCode[index]; morseCodeString = morseCodeString + codedLetter + SPACE; } else { morseCodeString = EMPTYSTRING; break; } } if(morseCodeString != EMPTYSTRING) { Console.WriteLine(morseCodeString); } else { reportError("Invalid character entered"); } }</pre>	4
----	---	---	---

12	1	<pre>void sendSignals(String morseCodeString) { int morseCodeLength = morseCodeString.length(); String signalString = EMPTYSTRING; for(int i = 0; i < morseCodeLength; i++) { if(morseCodeString.charAt(i) == '.') { signalString = signalString + "=" + SPACE; } else if(morseCodeString.charAt(i) == '-') { signalString = signalString + "===" + SPACE; } else if(morseCodeString.charAt(i) == SPACE) { signalString = signalString + SPACE + SPACE; } } Console.WriteLine(signalString); }</pre>	7
----	---	---	---

13	1	<pre> void outputAlphabetWithCode(char[] letter, String[] morseCode) { int counter = 0; for(int i = 1; i < 27; i++) { if(counter%4==0) { Console.WriteLine(); } Console.write(letter[i] + " "); Console.write(String.format("%-6s", morseCode[i])); counter++; } } void displayMenu() { Console.WriteLine(); Console.WriteLine("Main Menu"); Console.WriteLine("====="); Console.WriteLine("A - Output the alphabet with code"); Console.WriteLine("R - Receive Morse code"); Console.WriteLine("S - Send Morse code"); Console.WriteLine("X - Exit program"); Console.WriteLine(); } void sendReceiveMessages() { int[] dash = { 20, 23, 0, 0, 24, 1, 0, 17, 0, 21, 0, 25, 0, 15, 11, 0, 0, 0, 0, 22, 13, 0, 0, 10, 0, 0, 0 }; int[] dot = { 5, 18, 0, 0, 2, 9, 0, 26, 0, 19, 0, 3, 0, 7, 4, 0, 0, 0, 12, 8, 14, 6, 0, 16, 0, 0, 0 }; char[] letter = { "SPACE", 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z' }; String[] morseCode = { "SPACE + """, ".-", "-...", "-.-.", "-..", ". .", ". . .", "--.", "....", ". . .", ". ---", "-.-", "-.-.", "--", "-.", "----", ".---", "---.", ".-.", "...", "-", ".-.", "...-", ".--", "-.-.", "-.-.", "--.." }; boolean programEnd = false; while (!programEnd) { displayMenu(); char menuOption = getMenuOption(); if (menuOption == 'R') { receiveMorseCode(dash, letter, dot); } else if (menuOption == 'S') { sendMorseCode(morseCode); } } } </pre>	6
----	---	--	---

	<pre>else if (menuOption == 'A') { outputAlphabetWithCode(letter, morseCode); } else if (menuOption == 'X') { programEnd = true; } }</pre>	
--	--	--

14	1	<pre> void sendReceiveMessages() { int[] keys = new int[3]; for (int i = 1; i < 4; i++) { boolean validNumber = false; while(!validNumber) { try { Console.write("Enter key number " + i + ": "); keys[i-1] = Integer.parseInt(Console.readLine()); validNumber = true; } catch (Exception e) { reportError("Only enter integers"); } } } int[] dash = { 20, 23, 0, 0, 24, 1, 0, 17, 0, 21, 0, 25, 0, 15, 11, 0, 0, 0, 0, 22, 13, 0, 0, 10, 0, 0, 0 }; int[] dot = { 5, 18, 0, 0, 2, 9, 0, 26, 0, 19, 0, 3, 0, 7, 4, 0, 0, 0, 12, 8, 14, 6, 0, 16, 0, 0, 0 }; char[] letter = { "SPACE", 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z' }; String[] morseCode = { "SPACE + """, ".-", "-...", "-.-.", "- ..", ".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..", "- -", "-.", "----", ".--.", "---.", ".-.", "...", "-", ".-.", "...- ", ".--", "-..-", "-.-.", "--.." }; boolean programEnd = false; while (!programEnd) { displayMenu(); char menuOption = getMenuOption(); if (menuOption == 'R') { receiveMorseCode(dash, letter, dot); } else if (menuOption == 'S') { sendMorseCode(morseCode, keys); } else if (menuOption == 'A') { outputAlphabetWithCode(letter, morseCode); } else if (menuOption == 'X') { </pre>	9
----	---	--	---

```

        programEnd = true;
    }
}
void sendMorseCode(String[] morseCode, int [] keys)
{
    Console.WriteLine("Enter your message (uppercase letters and
spaces only): ");
    String plainText = Console.ReadLine();
    int plainTextLength = plainText.length();
    String morseCodeString = EMPTYSTRING;
    int index;
int keyNumber = 0;
    for (int i = 0; i < plainTextLength; i++)
    {
        char plainTextLetter = plainText.charAt(i);
        if (plainTextLetter == SPACE)
        {
            index = 0;
        }
        else
        {
            index = (int)plainTextLetter - (int)'A' + 1;
        }
index += keys[keyNumber];
keyNumber++;
if(keyNumber == 3)
        {
            keyNumber = 0;
        }
if(index > 26)
        {
            index = index - 27;
        }
else if(index < 0)
        {
            index = index + 27;
        }
        if (index >= 0 && index <= 26)
        {
            String codedLetter = morseCode[index];
            morseCodeString = morseCodeString + codedLetter + SPACE;
        }
        else
        {
            morseCodeString = EMPTYSTRING;
            break;
        }
    }

    if(morseCodeString != EMPTYSTRING)

```

	<pre>{ Console.WriteLine(morseCodeString); } else { reportError("Enter only space or uppercase letters"); } sendSignals(morseCodeString); }</pre>	
--	---	--