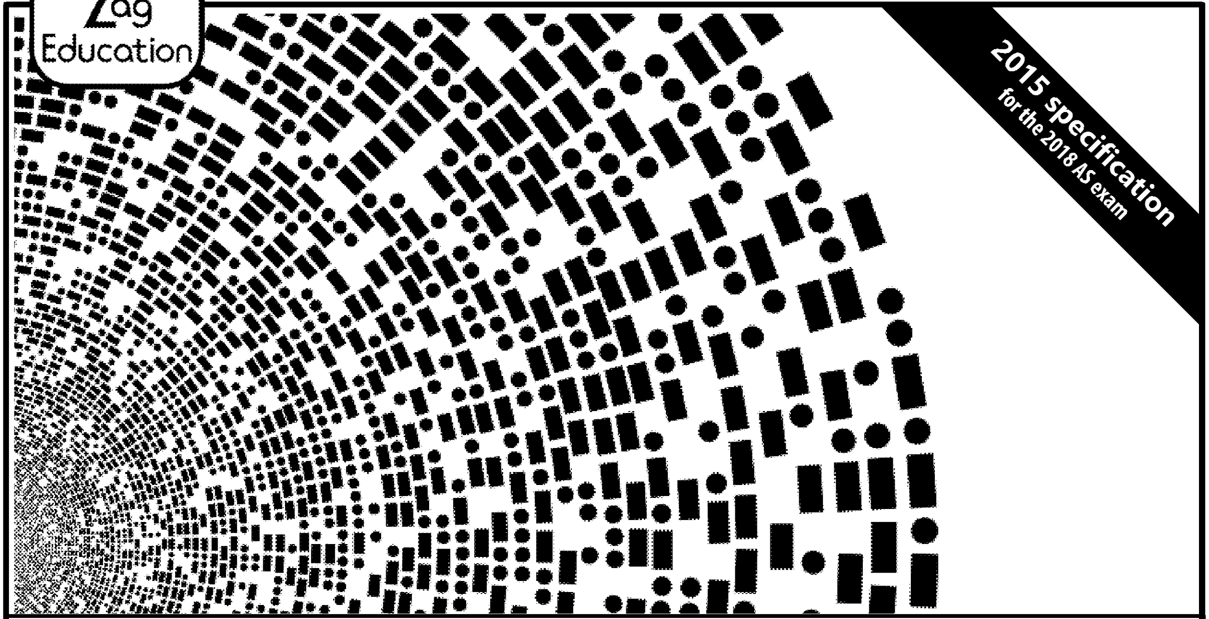
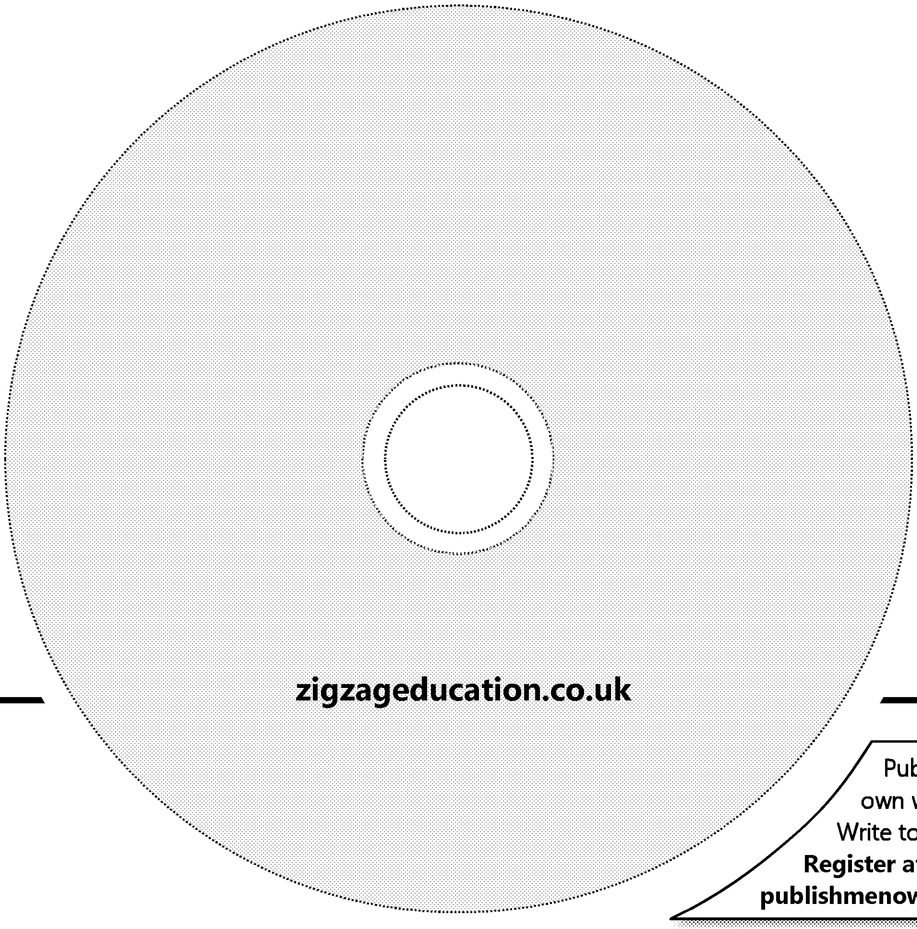


PAPER 1 EXAM RESOURCE PACK 2018

MORSE CODE

PYTHON3

for AS AQA Computer Science



CH9/
8528

POD
8528

zigzageducation.co.uk

Contents

Teacher's Introduction

This resource pack is designed to help you support your students taking the **AS Computer Science Paper 1** examination. It is based on the ***Morse Code*** preliminary material (Python3) – for examination June 2018.

On the CD, you will find the following files.

🗀	`MorseCode`	for student use – this folder contains all of the content, accessible via a HTML interface
🗀	`editable`	for teacher use – this folder contains ALL of the documents in editable (docx) formats
🗎	`Passwords.txt`	for teacher use – this file contains all of the passwords for the protected PDFs (also listed below)

* PRINTED COPIES OF ALL THE MATERIALS IN THIS DIGITAL RESOURCE PACK ARE INCLUDED FOR REFERENCE.

Installation: Copy the entire `MorseCode` folder onto a network location that is accessible for students, and provide them with a shortcut to the index.html file. All content can be accessed from this page.

Passwords: All of the PDFs in the 'Answers & Solutions' HTML page (`answers.html`) are password-protected, so that students can only access them with your permission. Each password is a four-digit code, as follows:

🗎	`Commentary.pdf`	1005
🗎	`DiagramComplete.pdf`	6113
🗎	`QuestionsMarkScheme.pdf`	2887
🗎	`TaskMarkScheme.pdf`	4392

Should you wish to give students access to ALL protected-PDFs, the master password for all files is:

`zz2ghc4`

The resource pack consists of the following:

① **Pre-release Commentary**, consisting of two parts:

– A general walkthrough of the skeleton program, including a written description and flowcharts giving a visual demonstration of the game.

– A detailed, technical overview of the skeleton program, describing how all subroutines and the various code elements work.

Note: although this section is intended to give extra support to teachers and students, it should in no way be seen as a substitute to a student exploring the code for themselves. For this reason, this content has been placed on the 'Answers & Solutions' HTML page as a password-protected file, to allow you to control if/when students access it.

② **Structure Diagram Activity**

Partially completed structure diagram activity for students to complete while getting to grips with the skeleton program. Any missing subroutine names, return values, parameters and directional arrows must be added to the diagram. An A4 printed copy is provided in this pack for reference, however it is recommended that you print this in A3 size from the PDF. Solutions are provided on the *Answers & Solutions* page as a protected PDF.

③ **Written Questions**

Theory questions testing students' understanding of the *Morse Code* program. These questions require access to the skeleton code, but no modifications need to be made to the program. Write-on (with answer lines) and non-write-on version are available format. Solutions are provided on the *Answers & Solutions* page as a protected PDF.

④ **Programming Tasks**

Fifteen modification exercises put students' programming skills to the test. Solutions are provided on the *Answers & Solutions* page as a protected PDF. Note that these are example solutions and you must use your discretion to award marks accordingly where there are valid alternative solutions.

Free Updates

Register your email address to receive any future free minor updates made to this resource or other Computing resources your school has purchased, and details of any promotions for your subject.

** resulting from minor specification changes, suggestions from teachers and peer reviews, or occasional errors reported by customers*

zzed.uk/freeupdates

An **Electronic Answer Document (EAD)** is provided should you wish students to use it for ③ and/or ④ above.

This resource is intended to supplement your teaching only. **Please read full disclaimer (p. iv) before using it.**

-- --- .-. ... .

MORSE CODE

-.-. --- -.. .

Description of the Program

The program is a system that converts between plaintext and Morse code.

Plain text is language printed alphabetically (A, B, C, etc.) whereas Morse code u
to represent each letter in the alphabet:

Plaintext	Morse code	Plaintext	Morse code
A	.-	J	.---
B	-...	K	-.-
C	-.-.	L	.-..
D	-..	M	--
E	.	N	-.
F	..-.	O	---
G	--.	P	.--.
H	....	Q	--.-
I	..	R	.-.

Each character is separated by a space, so the word HELLO is represented as foll

.... . .-.. .-.. ---

H	E	L	L
....		.-..	.-

Included with the pre-release material is a text file called 'message.txt'. The co

===⌂⌂⌂=⌂⌂⌂=⌂===⌂⌂⌂⌂⌂⌂⌂===

Note: *The ⌂ symbols are not included in the text file, they have been included in the*
to make them more visible for this explanation. The message.txt file consists of spac

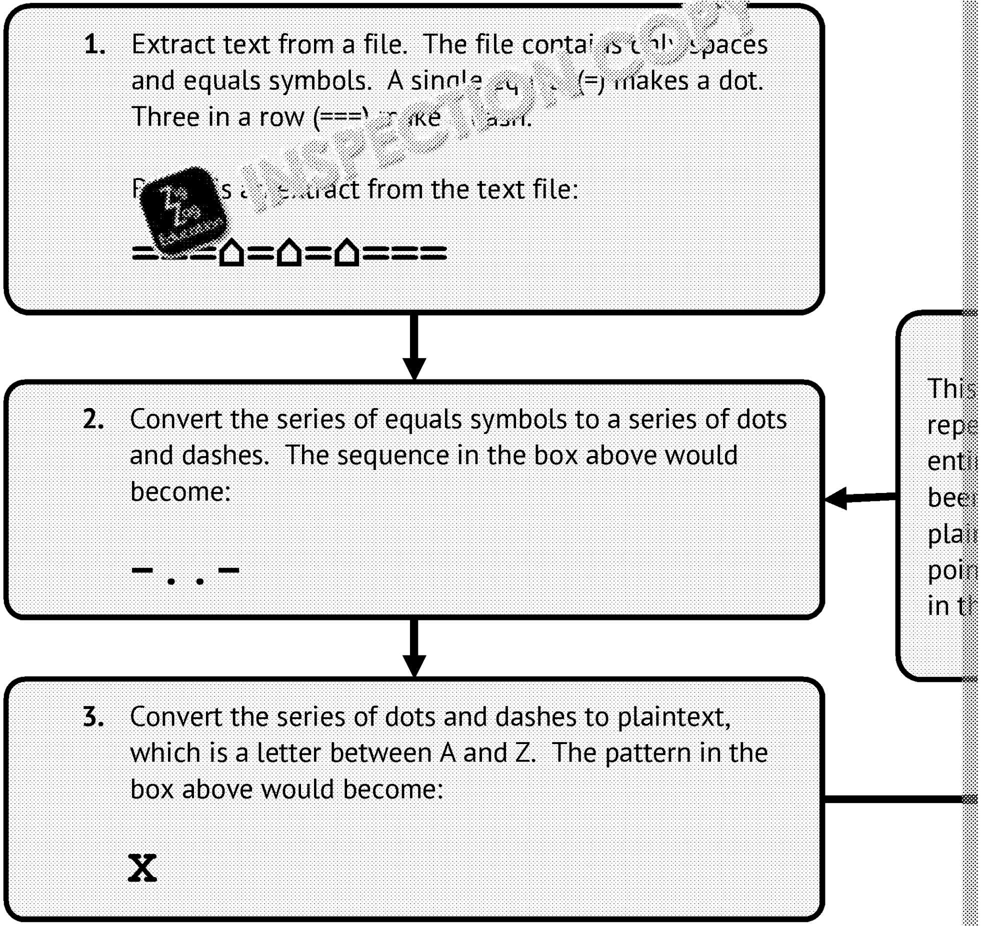
Overview

The program has two subroutines that handle conversion between plaintext and

ReceiveMorseCode

The subroutine `ReceiveMorseCode` reads Morse code from a text file and conv key subroutines used to perform this conversion is `Decode`. The subroutine `Se` from the user at the keyboard and converts it to Morse code.

`ReceiveMorseCode` consists of three main stages:



1. Extract text from a file. The file contains only spaces and equals symbols. A single equals (=) makes a dot. Three in a row (===) make a dash.

   Here is an extract from the text file:

   ===⌂=⌂=⌂===

2. Convert the series of equals symbols to a series of dots and dashes. The sequence in the box above would become:

   -..-

3. Convert the series of dots and dashes to plaintext, which is a letter between A and Z. The pattern in the box above would become:

   X

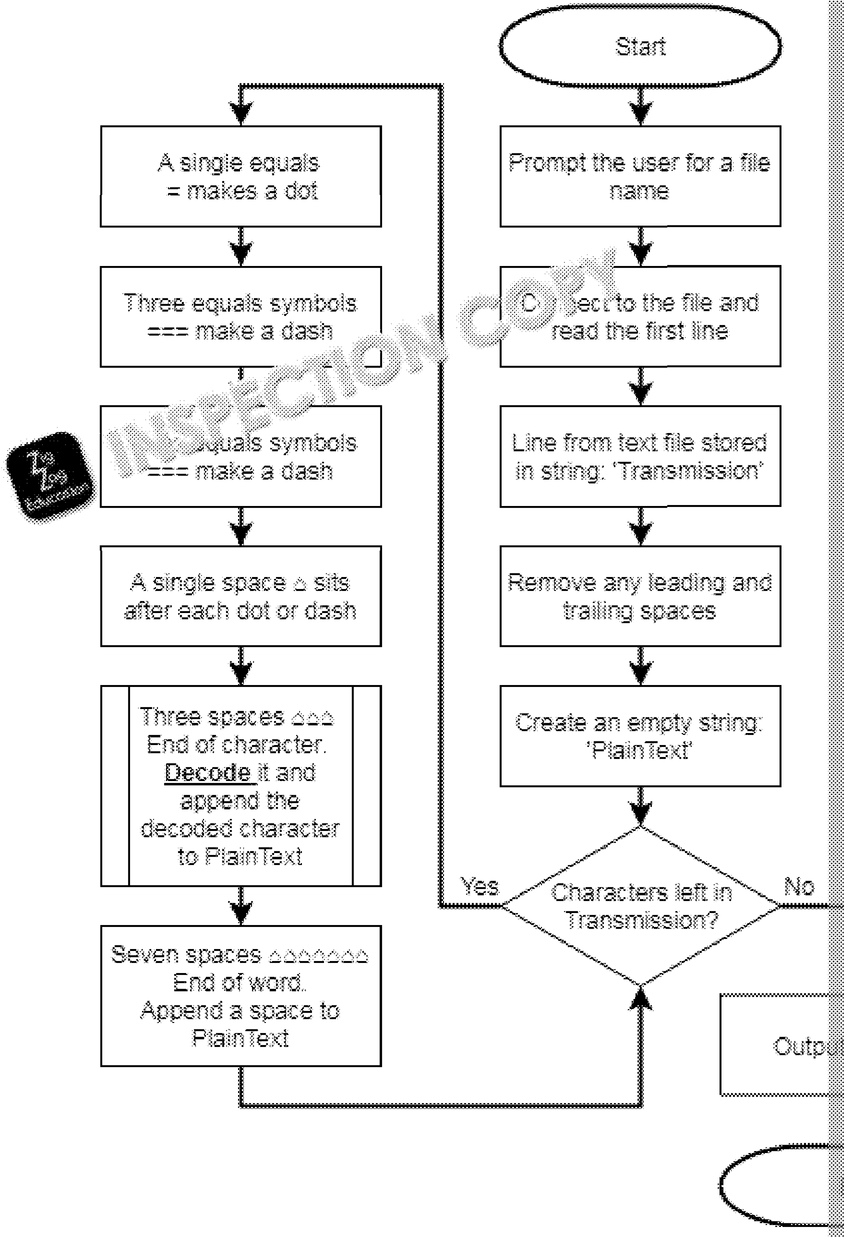
This
repe
enti
bee
plai
poi
in t

SendMorseCode

`SendMorseCode` is less involved. The user types uppercase plaintext at the con Morse code. The Morse code is then displayed on the console. Any spaces in the three spaces in Morse code.

Input	Output
COMPUTING	-.-. --- -- .--. ..-
AQA AS	.- --.- .- .- ...

ReceiveMorseCode Subroutine


Start
Prompt the user for a file name
Connect to the file and read the first line
Line from text file stored in string: 'Transmission'
Remove any leading and trailing spaces
Create an empty string: 'PlainText'
Characters left in Transmission?
Yes
No
Outpu
A single equals = makes a dot
Three equals symbols === make a dash
Three equals symbols === make a dash
A single space ⌂ sits after each dot or dash
Three spaces ⌂⌂⌂ End of character. Decode it and append the decoded character to PlainText
Seven spaces ⌂⌂⌂⌂⌂⌂⌂ End of word. Append a space to PlainText

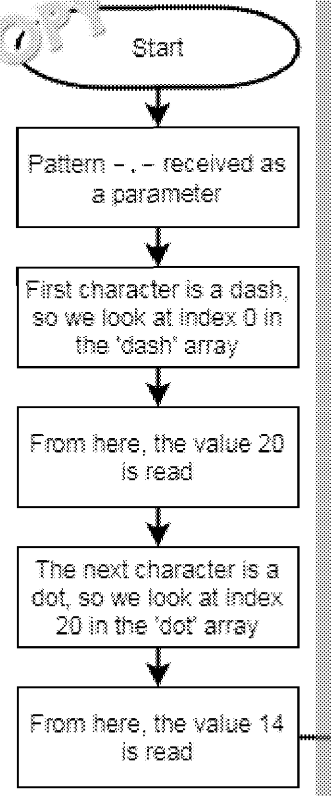
`ReceiveMorseCode` calls seven other subroutines, either directly or indirectly in the flowchart, as the flowchart exists only to provide a top-level understandin

Decode Subroutine

Element index in list:	Dot	Dash	Letter
0	5	20	⌂
1	18	23	A
2	0	0	B
3	0	0	C
4	2	24	D
5	9	1	E
6	0	0	F
7	26	17	G
8	0	0	H
9	19	21	I
10	0	0	J
11	3	25	K
12	0	0	L
13	7	15	M
14	4	11	N
15	0	0	O
16	0	0	P
17	0	0	Q
18	12	0	R
19	8	22	S
20	14	13	T
21	6	0	U
22	0	0	V
23	16	10	W
24	0	0	X
25	0	0	Y
26	0	0	Z

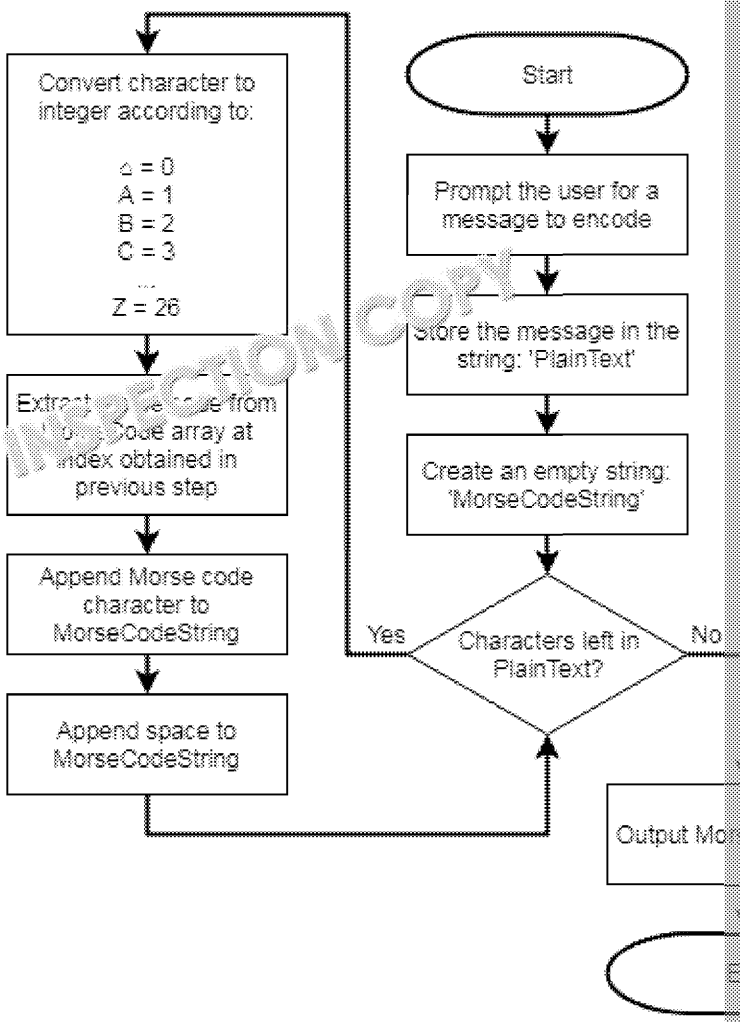
The subroutine `Decode` use `Dot`, `Dash` and `Letter`, wh throughout execution.

The flowchart below shows the pattern – . – into the pla


Start
Pattern – . – received as a parameter
First character is a dash, so we look at index 0 in the 'dash' array
From here, the value 20 is read
The next character is a dot, so we look at index 20 in the 'dot' array
From here, the value 14 is read

If the first character is a das looking at index 0 in the Da a dot (.), the starting point i

SendMorseCode Subroutine


Start
Prompt the user for a message to encode
Store the message in the string: 'PlainText'
Create an empty string: 'MorseCodeString'
Characters left in PlainText?
Yes
No
Convert character to integer according to:
⌂ = 0
A = 1
B = 2
C = 3
...
Z = 26
Extract ... from MorseCode array at index obtained in previous step
Append Morse code character to MorseCodeString
Append space to MorseCodeString
Output Mo

Unlike ReceiveMorseCode, which calls several other subroutines, SendMors calls no other subroutines. The user enters a message, which is validated to ens characters and spaces. The message is then translated, one character at-a-time, taken from an list called MorseCode.

The Text File (message.txt)

The contents of the text file are explained below:

===⌂⌂⌂=⌂⌂⌂=⌂===⌂⌂⌂⌂⌂⌂⌂===⌂

===⌂⌂⌂	This is a dash (===), followed by three spa Three spaces signals the end of a characte The character that is made up of a single
=⌂⌂⌂	This is the second character, which is a si
=⌂===	This character is a dot followed by a dash. A single space is used between them (inst the character is not finished yet. The Morse code comprising a dot followe
⌂⌂⌂⌂⌂⌂⌂	This is then followed by seven spaces, wh between two words.
===⌂=⌂=⌂===⌂	This is a character that is made up of a da by a dot, followed by a dash, which make

The whole message, therefore, is **TEA X**

Subroutine Calls, Parameters and Return Values

The numbers to the left do **not** indicate the order in which subroutines are called, as there are multiple possible orders.
Instead, these numbers relate to the numbers in the structure diagram.

Call	Parameters	Return
1 Main **calls** SendReceiveMessages	–	–
2 SendReceiveMessages **calls** DisplayMenu	–	–
3 SendReceiveMessages **calls** GetMenuOption	–	MenuOption
4 SendReceiveMessages **calls** ReceiveMorseCode	Dash Letter Dot	–
5 SendReceiveMessages **calls** SendMorseCode	MorseCode	–
6 ReceiveMorseCode **calls** GetTransmission	–	Transmission
7 ReceiveMorseCode **calls** GetNextLetter	i Transmission	i SymbolString
8 ReceiveMorseCode **calls** Decode	CodedLetter Dash Letter Dot	Letter[Pointer] This returns a string, but the string is always one character long, and is a character within the string Letter, at location Pointer. If Letter contains the string "Hello" , then Letter[0] = H, Letter[1] = E, etc.

Description of Subroutines

Each subroutine is described below.

Subroutine Name	Description	
Decode *Receives a coded letter (i.e. a letter in Morse code, such as -..-), and returns the corresponding plain text letter (X in this case)*	Parameters: CodedLetter, Dash, Letter, Dot Returns: Letter[Pointer] Called from: ReceiveMorseCode Calls: –	1. Initialise an integer variable CodedLetterLength to be equal to the length of the parameter CodedLetter 2. Initialise an integer variable Pointer to zero 3. Set up a loop to iterate through each character in CodedLetter, using the variable i 4. If i points to a space, this subroutine returns a space to ReceiveMorseCode 5. If i points to a dash, Pointer is changed to navigate the Morse code binary tree (see Preliminary Material, page 4), one step to the left 6. If i points to a dot, Pointer is changed to navigate the Morse code binary tree, one step to the right 7. By the time i has looped through each dot/dash in the encoded character, the value of Pointer should point (in the Letter list) to the letter that corresponds to the Morse code letter 8. If a space is not returned to ReceiveMorseCode in step 4 (above), the letter identified in step 7 is returned as a string
DisplayMenu *Displays three options to the user – send Morse code, receive Morse code or end the program*	Parameters: – Returns: – Called from: SendReceiveMessages Calls: –	1. Output three menu options (R, S, X), one on each line

Subroutine Name	Description	
GetNextLetter *A Morse code transmission usually consists of multiple letters. This subroutine extracts the next letter from a transmission.*	Parameters: i Transmission Returns: SymbolString Called from: ReceiveMorseCode Calls: GetNextSymbol	1. Declare string variable SymbolString and initialise it to an empty string 2. Set up a loop to repeat until any **one** of these conditions is met: • A space is returned from a call to GetNextSymbol (meaning the Morse character being parsed has ended) • The EOL character (#) is reached (meaning the end of the entire message has been reached) • The two characters after the current character are both spaces (meaning the letter has ended) 3. Within the loop, a call is made to GetNextSymbol, which will return a space, a dash or a dot. A space (see first bullet point) terminates the loop 4. If the call to GetNextSymbol returns a dash or a dot, that dash or dot is appended to the string variable SymbolString 5. At the end of the word (see bullet points), SymbolString is returned to ReceiveMorseCode
GetNextSymbol *A Morse code letter can consist of multiple symbols (combinations of dots and dashes). There are also spaces, which are used to separate them. This subroutine determines whether the next symbol is a dot, a dash or a space.*	Parameters: i Transmission Returns: Symbol Called from: GetNextLetter Calls: ReportError	1. When the parameter i is initially passed to this subroutine, its value is zero 2. Integer variable SymbolLength initialised to zero 3. i is used to point to characters within the string variable Transmission 4. If i points to the # character, '*End of transmission*' is written to the console, and an empty string is returned to GetNextLetter 5. Otherwise, i is incremented until it reaches either a space or the EOF character (#) within Transmission 6. As i is incremented, SymbolLength is also incremented

Subroutine Name	Description	
GetTransmission *This subroutine prompts the user for a filename, then reads the first line of the corresponding file, passing it to* *ReceiveMorseCode*	Parameters: – Returns: Transmission Called from: ReceiveMorseCode Calls: StripLeadingSpaces StripTrailingSpaces ReportError	1. Prompt the user for a file name 2. Create a FileHandle connected to the specified file 3. Read the first line of the file into the variable Transmission 4. Pass the variable Transmission to the subroutine StripLeadingSpaces, from which it should be returned 5. If the length of Transmission at this point is greater than zero, pass it to StripTrailingSpaces, from which it should be returned 6. Append the EOL symbol (currently #) to the variable Transmission 7. If any errors occur between steps 2 and 6, call ReportError (passing '*No transmission found*' as a parameter) and set the variable Transmission to an empty string 8. Return the variable Transmission to the subroutine ReceiveMorseCode
Main *This subroutine only exists to start the program (by calling* *SendReceiveMessages)*	Parameters: – Returns: – Called from: – Calls: SendReceiveMessages	1. Call SendReceiveMessages
ReceiveMorseCode *Calls other subroutines to manage the process of retrieving an encoded message (in Morse code), extracting each letter in turn and decoding each letter as it is extracted*	Parameters: Dash Letter Dot Returns: – Called from: SendReceiveMessages Calls: GetTransmission GetNextLetter	1. Set string variables PlainText and MorseCodeString to contain empty strings 2. Set the string variable Transmission to contain the return value from a call to the subroutine GetTransmission 3. Set the integer variable LastChar to point to the index of the last character in Transmission

Subroutine Name	Description	

`ReportError`
Writes an error to the console between two asterisks
	Parameters: `s`
	Returns: –
	Called from: `GetTransmission`
	`StripLeadingSpaces`
	`GetNextSymbol`
	Calls: –
		1. The error message arrives as a string parameter called `s`
		2. Parameter `s` is displayed between two asterisks

`SendMorseCode` *Accepts a plain text input from the user, translates it into Morse code and outputs the translation to the console*	Parameters: `MorseCode` Returns: – Called from: `SendReceiveMessages` Calls: –	1. Prompt the user for a message to be encoded 2. Store the message in the variable `PlainText` 3. Store the length of the message in the variable `PlainTextLength` 4. Initialise variable `MorseCodeString` as an empty string 5. Set up a loop to iterate through each character in `PlainText` 6. If the character is a space, the integer variable `Index` is set to 0 7. Otherwise, `Index` is set to a number that represents that letter's position in the alphabet, e.g. if the letter is A, `Index` will be set to 1; if the letter is B, `Index` will be set to 2; etc. 8. The value of `Index` is used as an index in the `MorseCode` list that was passed in as a parameter. For example, if the letter being examined was A, the value of `Index` would be 1. Element 1 would then be retrieved from the `MorseCode` list. 9. The Morse code value retrieved from the list is appended to the variable `MorseCodeString`, followed by a space 10. Once steps 6–9 have been performed on each character in the variable `PlainText`, the value of the variable `MorseCodeString` is printed

Subroutine Name	Description	
SendReceiveMessages *This contains the main program loop, which repeatedly displays the menu, prompts the user for an input and calls the appropriate subroutine in response. This loop ends when the user indicates a desire to end the program.*	Parameters: – Returns: – Called from: Main Calls: DisplayMenu GetMenuOption ReceiveMorseCode SendMorseCode	1. Initialise Dash list (to contain integer pointers that relate to the Morse code binary tree) 2. Initialise the Letter list (SPACE, 'A', B', 'C' … 'Z') 3. Initialise the Dot list (to contain integer pointers that relate to the Morse code binary tree) 4. Initialise the MorseCode list (to contain the Morse equivalents of letters, in the same order as the Letter list) 5. Begin a loop that continues until the user indicates that they want to end the program 5. Call DisplayMenu to display the menu 6. Call GetMenuOption to get user input from menu 7. Either call ReceiveMorseCode, call SendMorseCode, or terminate the loop, depending on user input
StripLeadingSpaces *Removes any spaces from the left of a string*	Parameters: Transmission Returns: Transmission Called from: GetTransmission Calls: ReportError	1. Store the length of the transmission in the integer variable TransmissionLength 2. Set up a loop that repeats while the first character of Transmission is a space, and while the length of Transmission is greater than zero 3. Within that loop, decrement the variable TransmissionLength and remove the first character of Transmission 4. If, after the loop, the length of Transmission is zero, call the subroutine ReportError, passing to it the string '*No signal received*' as a parameter
StripTrailingSpaces	Parameters: Transmission	1. Set the integer variable LastChar to point to the index of the last

Description of Variables, Constants and Parameters

The following table contains variables ⓥ, constants ⓒ and parameters ⓟ

Name	Type	Description	Created in / Passed to
`CodedLetter` ⓟ	`String`	Contains a single Morse code letter that is about to be decoded (passed by value)	`Decode`
`CodedLetter` ⓥ	`String`	Contains a single Morse code letter that is about to be decoded or has just been encoded	`ReceiveMorseCode` `SendMorseCode`
`CodedLetterLength` ⓥ	`Integer`	The number of Morse symbols in an encoded letter	`Decode`
`Dash` ⓟ	`Integer list`	Contains pointers to left branches of the binary tree seen on the Preliminary Material document, page 4 (passed by value)	`Decode` `ReceiveMorseCode`
`Dash` ⓥ	`Integer list`	Contains pointers to left branches of the binary tree seen on the Preliminary Material document, page 4	`SendReceiveMessages`
`Dot` ⓟ	`Integer list`	Contains pointers to right branches of the binary tree seen on the Preliminary Material document, page 4 (passed by value)	`Decode` `ReceiveMorseCode`
`Dot` ⓥ	`Integer list`	Contains pointers to right branches of the binary tree seen on the Preliminary Material document, page 4	`SendReceiveMessages`
`EMPTYSTRING` ⓒ	`String`	Constant to store an empty string: “”	`(global)`
`EOL` ⓒ	`Char`	Constant to store # symbol, which marks the end of a line	`(global)`
`FileHandle` ⓥ	`File Handle`	Used to store a reference to the text file containing the transmission	`GetTransmission`

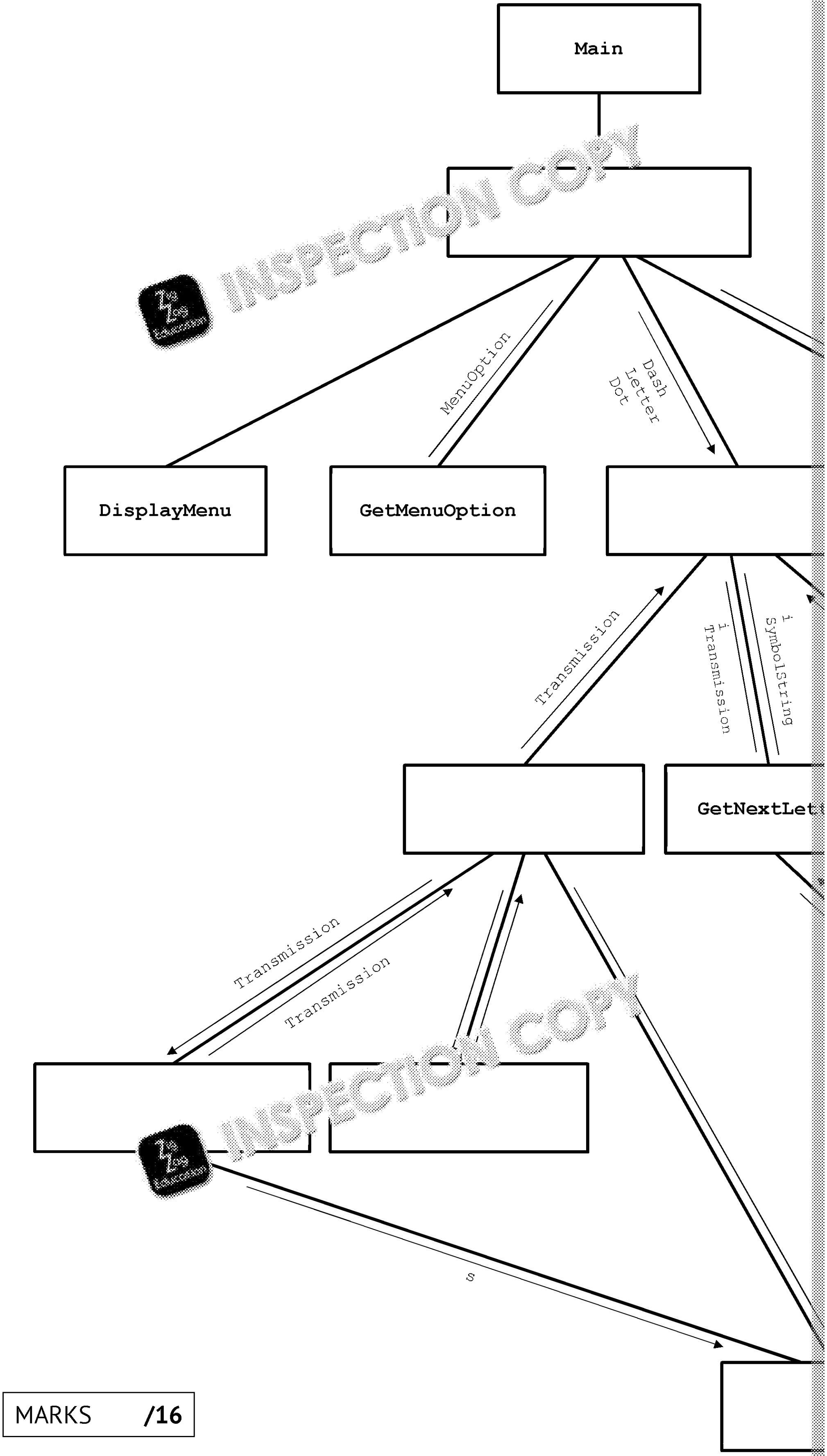
Name	Type	Description	Created in / Passed to
`Index` ⓥ	`Integer`	Stores a pointer used to access the correct Morse code character within a list	`SendMorseCode`
`LastChar` ⓥ	`Integer`	Points to the index of the last character in `Transmission`	`StripTrailingSpaces` `ReceiveMorseCode`
`Letter` ⓟ	`String list`	Contains a space in the first element, followed by the lower-case alphabet, with each letter in its own element (passed by value)	`Decode` `ReceiveMorseCode`
`Letter` ⓥ	`String list`	Contains a space in the first element, followed by the upper-case alphabet, with each letter in its own element	`SendReceiveMessages`
`LetterEnd` ⓥ	`Boolean`	Set to true if the end of a Morse code letter has been reached while it is being parsed character by character	`GetNextLetter`
`MenuOption` ⓥ	`String`	Contains the user's response when presented with the program's main menu	`GetMenuOption` `SendReceiveMessages`
`MorseCode` ⓟ	`String list`	Contains a space in the first element, followed by Morse code equivalents for each letter, with one such letter per element (passed by value)	`SendMorseCode`
`MorseCode` ⓥ	`String list`	Contains a space in the first element, followed by Morse code equivalents for each letter, with one such letter per element	`SendReceiveMessages`
`MorseCodeString` ⓥ	`String`	An entire Morse code message, which can contain any number of Morse code characters	`ReceiveMorseCode`
`MorseCodeString` ⓥ	`String`	Contains a Morse code message, constructed character by character	`SendMorseCode`
`PlainText` ⓥ	`String`	Contains a message that has been (or is about to be) decoded from its Morse code equivalent	`ReceiveMorseCode` `SendMorseCode`
`PlainTextLength` ⓥ	`Integer`	The number of characters to be converted to Morse code	`SendMorseCode`

Name	Type	Description	Created in / Passed to
Signal ⓥ	String	Variable to examine each character of Transmission in turn	GetNextSymbol
SPACE ⓒ	Char	Constant to store a single space character	(global)
Symbol ⓥ	Char	Contains a dot, dash or space within a Morse code letter	GetNextSymbol Decode
Symbol ⓥ	String	Contains the value returned from GetNextSymbol (i.e. single dot, dash or space) that forms part of a Morse code letter	GetNextLetter
SymbolLength ⓥ	Integer	Stores the number of characters in a single Morse code letter	GetNextSymbol
SymbolString ⓥ	String	Built up, one dot or dash at a time, into a Morse code letter	GetNextLetter
Transmission ⓟ	String	Stores a sequence of Morse code letters (passed by value)	StripLeadingSpaces StripTrailingSpaces GetNextSymbol GetNextLetter
Transmission ⓥ	String	Stores a sequence of equals signs and spaces, used to represent Morse code as described in the Preliminary Material Document	GetTransmission ReceiveMorseCode
TransmissionLength ⓥ	Integer	Stores the length of the Transmission variable	StripLeadingSpaces

Structure Diagram (Activity)

The following structure diagram is incomplete, and you will need to make the following changes, as requir

- Adding a subroutine's name, e.g. **`ReceiveMorseCode`**
- Adding or completing one or more parameters, e.g. **`Dash`**
- Adding a return value, e.g. **`Symbol`**
- Completing the arrow by drawing its head – parameters in this diagram are passed downwards; retu



MARKS /16

Programming Questions

These questions refer to the preliminary material and require you to load the skel program, but do not require any additional programming.

1. State the name of an identifier for:

   a) A string constant (or variable used as a constant) **[1]**

   ....

   b) A subroutine with two parameters **[1]**

   ....

   c) A subroutine that returns a tuple (more than one value) **[1]**

   ....

   d) A Boolean variable **[1]**

   ....

   e) A parameter that is a list **[1]**

   ....

   f) An integer list **[1]**

   ....

   g) A built-in function called from within the `GetMenuOption` subroutine [

   ....

   h) The identifier for a user-defined function called from the `GetNextLett`

   ....

2. State the purpose of each of the following lines in the `GetTransmission` s

```
FileName = input("Enter file name: ")
...
    FileHandle = open(FileName, 'r')
    Transmission = FileHandle.readline(0029
    FileHandle.close()
    ...
```

....

....

....

....

3. Describe the purpose of the `While` loop within the `SendReceiveMessag`

....

....

....

4. Describe the nature and purpose of the `Dash` data structure in `SendRecei`

....

....

....

....

....

5. Look at the subroutine `StripLeadingSpaces`. Describe the purpose and u
`FirstSignal`. **[2]**

....

....

....

6. Describe each of the following lines of code, taken from the `StripTraili`

```
LastChar = len(Transmission) - 1
while Transmission[LastChar] == SPACE:
    LastChar -= 1
    Transmission = Transmission[:-1]
return Transmission
```

....

....

....

....

....

....

7. Describe the function of the following line from the `SendMorseCode` subro

```
Index = ord(PlainTextLetter) - ord('A') + 1
```

....

....

....

....

....

8. Describe the purpose of the `except:` block in the `GetTransmission` sub

   State one situation in which the code in the `except:` block would be execut

   ....

   ....

   ....

9. The skeleton program begins with a number of constants (or variables used a

   State two benefits of the program being written in this way. **[2]**

   ....

   ....

   ....

10. The `StripLeadingSpaces` subroutine uses the `[1:]` operation.

    Describe the purpose of the `[1:]` operation and explain how it is used in `St`

    ....

    ....

    ....

    ....

    ....

11. Describe each of the circumstances that would lead to the subroutine `Repor`

    ....

    ....

    ....

    ....

    ....

12. Describe fully the operation of the `Decode` subroutine if the value of `Coded`

    ....

    ....

    ....

    ....

    ....

    ....

    ....

MORSE CODE: Programming

The following tasks require you to open the skeleton program and m

Task 1

This task refers to `GetMenuOption`.

Currently, the program allows any single character to be entered as a choice from `GetMenuOption` subroutine so that all values entered are converted to upper ca be made. If an invalid choice is entered, the user should be prompted with the m

```
    Invalid choice, please choose a letter from the menu
```

This this should repeat until they have entered a valid choice. For example:

- Entering ‘a’ should result in the above prompt
- Then pressing Enter should make the same prompt appear again
- Finally entering ‘S’ should take you to the ‘Send Morse code’ option

Note that the prompt to enter a choice from the menu should remain the sa

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `GetMenuOption`
- One screen capture showing the *menu choice*, the *prompt* and *result* for th
  - Begin the program and enter ‘y’ at the prompt
  - Press Enter at the prompt
  - Enter ‘SS’ at the prompt
  - Enter ‘R’ at the prompt
- One screen capture showing the *menu choice*, the *prompts* and *result* for th
  - Begin the program and enter ‘x’ at the prompt

Task 2

This task refers to `SendReceiveMessages` and `SendMorseCode`.

The program currently only accepts upper case letters. Modify the code so that fu the sequence ‘. – . – . –‘ (dot, dash, dot, dash, dot dash):

- Alter the main code to add an additional constant called FULLSTOP (tech but we use the convention of uppercase to indicate a constant).
- Modify the `Letter` and `MorseCode` lists in `SendReceiveMessages` s added onto the end of each list for the full stop. Modify the `Dot` and `Da` stop can be correctly received in a transmission.
- Modify `SendMorseCode` so that a full stop is correctly identified (using the 28th element of the `MorseCode` list).

Note that you will need to put the *message2.txt* file into the same folder as yo

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM snippet showing the additiona
- Your amended SOURCE CODE PROGRAM for `SendReceiveMessages`
- Your amended SOURCE CODE PROGRAM for `SendMorseCode`
- One screen capture showing the *values entered* and the *result* for the fol
  - Run the program and enter ‘S’ at the prompt
  - Enter ‘S.O.S.’ at the prompt
- One screen capture showing the *result* for the following sequence:
  - Run the program and enter ‘R’ at the prompt
  - Enter ‘message2.txt’ at the prompt

Task 3

This task refers to DisplayMenu and SendReceiveMessages. It also invol subroutine PrintMorseCodeSymbols which will have two parameters, the lis from SendReceiveMessages.

Modify DisplayMenu and SendReceiveMessages to add the following as th

```
    P – Print Morse code symbols
```

This new menu option will need to call a new subroutine PrintMorseCodeSy and pass two arguments, the lists Letter and MorseCode. The subroutine sh print out a table of all the Morse code letters and symbols in the following forma

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for DisplayMenu
- Your amended SOURCE CODE PROGRAM for SendReceiveMessages
- Your new SOURCE CODE PROGRAM for PrintMorseCodeSymbols
- One screen capture showing the main menu, selection of option P and t

Task 4

This task refers to DisplayMenu, SendMorseCode and SendReceiveMes creation of a new subroutine TransmitMorseCode which will have one param SendReceiveMessages.

Modify DisplayMenu and SendReceiveMessages to add the following as th

```
    T – Transmit Morse code
```

This new menu option will need to call a new subroutine, TransmitMorseCod list MorseCode. The new subroutine should call the existing subroutine SendM be modified to return the message instead of printing it out. (Note that you will SendReceiveMessages to print out the return value instead of just calling it.) then ask you for a file name and convert the Morse code message to transmissio

For example:

- The user selects option 'T' from the menu and is asked to enter their mes
- They enter 'TEA TIME'
- The program prompts them for a file name and they enter 'message4.txt'
- The program generates the transmission (=== = = === = = == file message4.txt

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for DisplayMenu
- Your amended SOURCE CODE PROGRAM for SendReceiveMessages
- Your new SOURCE CODE PROGRAM for TransmitMorseCode
- One screen capture showing the following sequence:
    - Run the program and enter 'T' from the main menu
    - Enter the message: 'ZIG ZAG'
    - Enter 'message4.txt' as the file name
    - Select option R from the main menu
    - Enter the file name 'message4.txt'

Task 5

This task refers to SendReceiveMessages, ReceiveMorseCode and De

Currently, if an invalid sequence of dots and dashes is received, the program will instead of presenting a suitable error message.

Modify SendReceiveMessages to pass the list of valid symbols as the (new) f ReceiveMorseCode and modify ReceiveMorseCode to pass the list of vali argument to Decode.

You should decode an invalid character(s) as the asterisk (*) symbol and print out invalid sequence of dots and dashes that was received. You will use the messag

For example:

```
        Enter your choice: R
        Enter file name: message5.txt
        *   Invalid Symbol (-.---) received.     *
         -.---
        *T
```

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for SendReceiveMessages
- Your amended SOURCE CODE PROGRAM for ReceiveMorseCode
- Your amended SOURCE CODE PROGRAM for Decode
- One screen capture showing choosing option R from the main menu an 'message5.txt'. (Note you will need to put message5.txt in the same fold

Task 6

This task refers to GetTransmission.

The program currently expects the full file name to be typed in (including the .tx better if this was flexible.

Modify the GetTransmission subroutine so that it functions properly, with or

No changes should be made to any of the prompts.

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for GetTransmission
- One screen capture showing choosing option R from the main menu an 'message6'
- One screen capture showing choosing option R from the main menu an 'message6.txt'

Task 7

This task refers to `DisplayMenu` and `SendReceiveMessages` and involves c `ConvertMorseCode`.

Currently, there is no option for the message to be entered in Morse code.

Modify `DisplayMenu` and `SendReceiveMessages` to add the following as th

```
    C - Convert Morse code
```

This new menu option will need to call a new subroutine `ConvertMorseCode` lists `MorseCode` and `Letter`. The new subroutine should ask the user to ente print out the decoded message. It should accept only valid Morse code and prin symbol is invalid.

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `DisplayMenu`
- Your amended SOURCE CODE PROGRAM for `SendReceiveMessages`
- Your new SOURCE CODE PROGRAM for `ConvertMorseCode`
- One screen capture showing all of the input and output for the followi
    - Run the program and enter 'C' from the main menu
    - Enter the Morse code: .... ..   - .... . .-. .
- One screen capture showing all of the input and output for the followi
    - Run the program and enter 'C' from the main menu
    - Enter the Morse code: .... .-.--- .-.. .-.. ---

Task 8

This task refers to `SendMorseCode`.

Modify this subroutine to also generate the quaternary for the message to be se the encoded message in Morse code (on a separate line).

Quaternary Symbols:

- Letter separator (0)
- Word separator (1)
- Dot (2)
- Dash (3)

Encoding Examples:

- Three dots: 22
- Three dashes:
- The word 'son'
- The phrase 'is

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `SendMorseCode`
- One screen capture showing all of the input and output for the followi
    - Run the program and enter 'S' from the main menu
    - Enter the message: 'TEST MSG'

Task 9

This task refers to `DisplayMenu` and `SendReceiveMessages`. It also invol subroutine `SendEncryptedMorseCode` which will have one parameter, `Mors`

Modify `DisplayMenu` and `SendReceiveMessages` to add the following as th

```
E – Send encrypted message
```

This new menu option will need to call a new subroutine, `SendEncryptedMor` the list `MorseCode`. The new subroutine should ask the user to enter a messag Caesar Cipher Shift for the message is. It should then apply the shift (but not shi Morse code for based on the cipher text for the message.

For example:

- User enters the message 'I AM'
- User chooses a Caesar Cipher Shift of 3
- Message is shifted by 3 to L DP (not displayed)
- Morse code version of the message is displayed: `.-..     -.. .--`

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `DisplayMenu`
- Your amended SOURCE CODE PROGRAM for `SendReceiveMessages`
- Your new SOURCE CODE PROGRAM for `SendEncryptedMorseCode`
- One screen capture showing all of the input and output for the followin
  - Run the program and enter 'S' from the main menu
  - Enter the message: 'TEST MSG'
  - Enter a Caesar Cipher Shift of: '12'
- One screen capture showing all of the input and output for the followin
  - Run the program and enter 'S' from the main menu
  - Enter the message: 'TEST MSG'
  - Enter a Caesar Cipher Shift of: '-5'
- One screen capture showing all of the input and output for the followin
  - Run the program and enter 'S' from the main menu
  - Enter the message: 'TEST MSG'
  - Enter a Caesar Cipher Shift of: '50'

Task 10

This task refers to `SendMorseCode` and involves the creation of a new subrouti `CalculateTransmissionTime` that will take one parameter (the message in integer which represents the number of time units required to send the message.

Modify `SendMorseCode` so that it makes a call to `CalculateTransmissio` containing the message in Morse code as the argument. It should retrieve the va suitable message of the following format:

Your message will take 80 time units to send.

Note: *When calculating the length of time in time units, a dot is 1 time unit and a da between dots, dashes or spaces is 1 time unit. The gap between letters is 3 time unit 2 additional time units to indicate the end of a letter, the gap between words (a spa space at the end of a letter and 4 additional time units to indicate the end of a word).*

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `SendMorseCode`
- Your new SOURCE CODE PROGRAM for `CalculateTransmissionTi`
- One screen capture showing all of the input and output for the followin
  - Run the program and enter 'S' from the main menu
  - Enter the message: 'TEST MSG'

Task 11

This task refers to `SendMorseCode`.

Modify the subroutine so that the user can put in the message in any case (upper,

If the input includes at least one lower case letter then the subroutine should pri

```
    Only uppercase letters can be used, your message has
```

... followed by the message in uppercase.

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `SendMorseCode`
- One screen capture showing all of the input and output for the followin
  - Run the program and enter 'S' from the main menu
  - Enter the message: 'TEST MSG'
- One screen capture showing all of the input and output for the followin
  - Run the program and enter 'S' from the main menu
  - Enter the message: 'Test Message'
- One screen capture showing all of the input and output for the followin
  - Run the program and enter 'S' from the main menu
  - Enter the message: 'test msg'

Task 12

This task refers to `ReceiveMorseCode`.

Modify the subroutine so that is prints out a message showing how many symbo received. Only dots and dashes count as symbols and only letters count as chara

For example:

- User selects 'R' from the menu and enters a file name containing a trans
- 8 symbols received: `-  .  .-     -..-`
- Which represent 4 characters: `TEA X`

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `ReceiveMorseCode`
- One screen capture showing all of the input and output for the followin
  - Run the program and enter 'R' from the main menu
  - Enter the file name: message12.txt

Task 13

This task refers to `SendReceiveMessages` and `DisplayMenu`.

The program currently uses the *International Morse Code* but needs to be updated that and the *American Morse Code* system.

Modify the subroutine `DisplayMenu` so that the menu informs the user what st used. You will need to pass in a Boolean argument (`InternationalMorseCo` specify either International (True) or American (False).

Create the following new menu option:

```
V - Change to American Morse code
```

Once this menu option has been chosen and American Morse code is being used,

```
V - Change to International Morse code
```

This new menu option should appear as the third menu option before X.   For ex

```
Main Menu
=========
R - Receive Morse code
S - Send Morse code
V - Change to American Morse code
X - Exit program

System is currently using the International version o
Enter your choice:
```

Note there is no need to actually change all of the symbols and mappings for this tas followed through by actually changing the lists Dash, Dot and MorseCode were the e

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `SendReceiveMessages`
- Your amended SOURCE CODE PROGRAM for `DisplayMenu`
- One screen capture showing all of the input and output for the followin
  - Run the program and enter 'V' from the main menu
  - Enter 'V' again from the main menu
  - Enter 'V' a third time from the main menu

Task 14

This task is an extension of Task 4 which you will need to have solved first in ord

This task refers to `TransmitMorseCode`.

Modify your solution so that before it writes the transmission signals to the file, i
and asks the user whether they would like to overwrite the file or choose anothe

For example:

```
      Enter file name: message4.txt
      File already exists, would you like to overwrite it
```

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `TransmitMorseCode`
- One screen capture showing all of the input and output for the followin
  - Run the program and enter 'T' from the main menu
  - Enter the message: 'TEST MSG'
  - Enter the file name: 'message4.txt'
  - Choose 'N'
  - Enter the file name 'message14.txt'
- One screen capture showing all of the input and output for the followin
  - Run the program and enter 'T' from the main menu
  - Enter the message: 'TEST MESSAGE'
  - Enter the file name: 'message14.txt'
  - Choose 'Y'
  - Choose 'R' from the main menu
  - Enter the file name: 'message14.txt'

Task 15

This task refers to `GetTransmission.`

After the transmission has been received, display a message saying how many sy
this down into the number of units of a signal (=) and the number of units of no s
trailing spaces should not be counted.

For example:

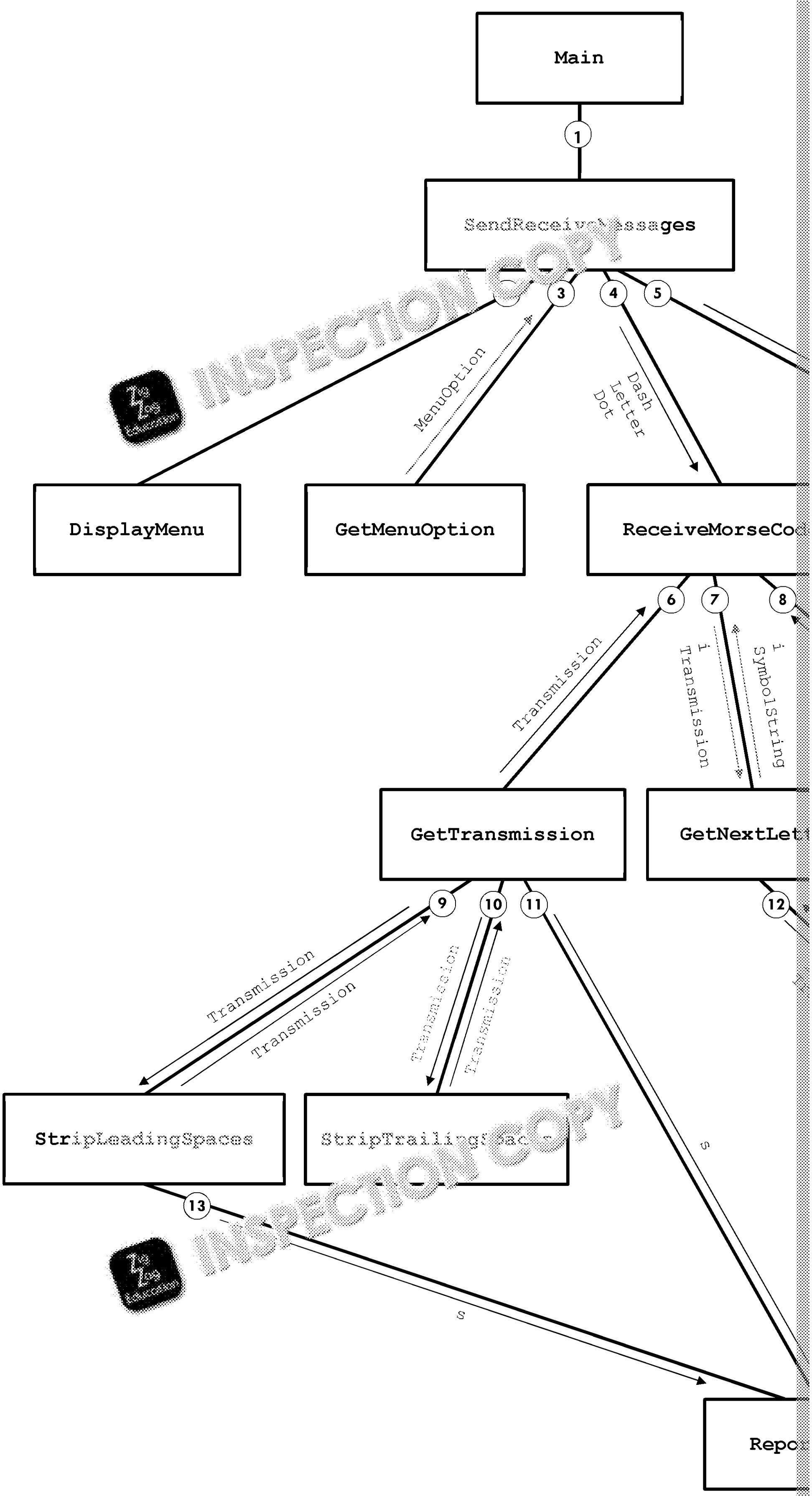
```
      45 symbols received in transmission consisting of 30
```

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `GetTransmission`
- One screen capture showing all of the input and output for the followin
  - Run the program and enter 'R' from the main menu
  - Enter the file name: 'message.txt'

Structure Diagram (Complete)

Subroutines are called downwards, i.e. `Main` calls `SendReceiveMessages`, not the other way aro
Arrows pointing downwards indicate parameters; arrows pointing upwards indicate return values.



Programming Questions (Solutions)

Q	Answer/Guidance
1a	`EMPTYSTRING`
1b	`GetNextSymbol // GetNextLetter`
1c	`GetNextSymbol // GetNextLetter`
1d	`LetterEnd // ProgramEnd`
1e	`Dash // Letter // Dot // MorseCode`
1f	`Dash // Dot`
1g	`len // input`
1h	`GetNextSymbol`
2	1 mark for each of the following: • (String) variable (`FileName`) initialised to user input • (Variable) `FileHandle` assigned to specified file opened for rea `open` function • `Transmission` variable set to first line of the file • File is closed
3	1 mark for each of the following: • (Repeatedly) prompt the user / accept user input • until X is entered / loop terminates at X
4	1 mark for each of the following **(max 3)**: • Integer array • Contains pointers • Indicates which element to move to next… • … if the next Morse signal is a dash
5	1 mark for each of the following: • Initially set to the first character in `Transmission` • As spaces are removed, it points to the new first character
6	1 mark for each of the following: • (Integer) variable `LastChar` set to the index of the last characte • Using the built-in function `len()` to get the length of the `Trans` • Loop repeats while `LastChar` / last character is a space • If the last character is a space, remove it from `Transmission`) • Decrement `LastChar` / index variable • Return `Transmission` with all spaces removed from end/right
7	1 mark for each of the following **(max 3)**: • Gets ASCII value of `PlainTextCharacter` • Gets ASCII value of A / Gets value 65 • Subtracts ASCII value of A / 65 from ASCII value of `PlainTextC` • If `PlainTextLetter` is A, `Index` is 1 (for example)
8	1 mark for each of the following: • `except:` block executed if `try:` block fails to execute correctly • File name mistyped // file not found // error reading file // error/e `StripLeadingSpaces` // error/exception in `StripTrailing` `Transmission`/`EOL` not being a valid string

Q	Answer/Guidance
9	1 mark for each of the following **(max 2)**: • Constants won't be accidentally changed • By being at the start of the code, the code is easier to read/under *this is for the benefit of the human, not the computer*) • No need to remember (precise) values // constant names more me code is more readable
10	1 mark for each of the following **(max 3)**: • It is used to remove the first item in a list • It Is used here to trim the first character/space from the `Transmi` • By treating the string as a list • Is called repeatedly if multiple spaces exist
11	1 mark for `StripLeadingSpaces` instance: • If the length of `Transmission` is zero 1 mark for `GetTransmission` instance: • If there is a file error (accept any error relating to code in the `try` if the `except:` block executes // if the `try:` block fails / genera 3 marks for `GetNextSymbol` instance: • If the symbol is not a dot… • … not a dash / minus sign… • … not a space
12	1 mark for each of the following: • `CodedLetterLength` variable set to the length of the sequenc • `for` loop to run four times • `Symbol` initially set to the first symbol in the sequence to be dec • `Pointer` set to 20 • `Symbol` then set to dot (on next iteration) • `Pointer` set to 14 • `Pointer` set to 4 (on next iteration) • `Symbol` then set to dash (on next iteration) • `Pointer` set to 24 • X retrieved from `Letter` array / X returned (*only credit this mark parse the arrays*)
TOTAL MARKS	

MORSE CODE: Programmin

Suggested Solutions and Mark Schem

The following are recommended solutions, and not an exhaustive list of all possible s guidance should be used as a guide only. Discretion should be used in awarding credit v

Task 1

1 mark The user is always prompted with “Enter your choice: “ when the prog

1 mark The input is converted to uppercase (or equivalent logic later such as case letters in the selection/iterative statement

1 mark There is an iterative statement that will continue to prompt the user (even if it doesn't work properly. Also accept an iterative statement w appropriate selection statement inside the iterative statement which

1 mark The condition for the iterative statement specifically prompts the use choose a letter from the menu: “ when anything other than “R”. “S” or equivalents too if the input wasn’t converted to uppercase).

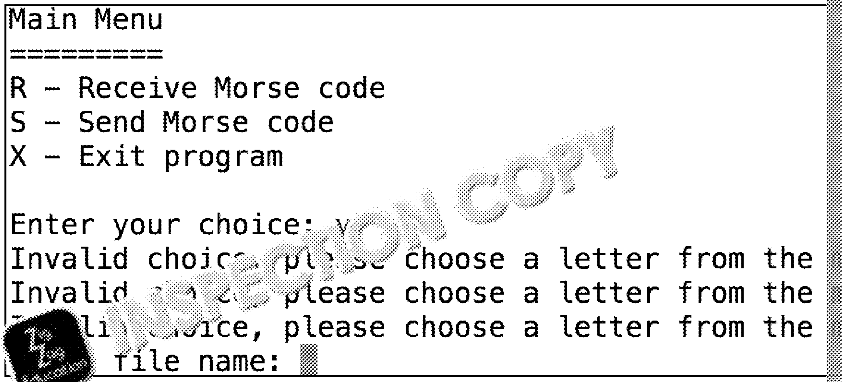
1 mark The input from the “Invalid choice” prompt is converted to uppercase checking both upper and lower case letters in the selection/iterative

```
def GetMenuOption():
  MenuOption = input("Enter your choice: ").upper()
  while MenuOption not in ['R', 'S', 'X']:
    MenuOption = input("Invalid choice, please choose a lett
  return MenuOption
```

1 mark Screenshot shows ‘y’ was entered, resulting in the Invalid choice pror

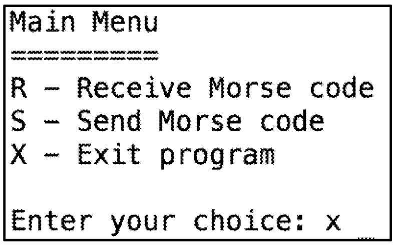
1 mark Screenshot shows nothing was entered (i.e. enter was pressed with n choice prompt, followed by ‘SS’ being entered at the prompt and ano

1 mark Screenshot shows ‘R’ was entered, resulting in the Enter file name: pr



```
Main Menu
=========
R - Receive Morse code
S - Send Morse code
X - Exit program

Enter your choice: y
Invalid choice, please choose a letter from the
Invalid choice, please choose a letter from the
Invalid choice, please choose a letter from the
Enter file name: 
```

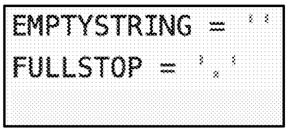
1 mark Screenshot shows ‘x’ was entered, followed by the program exiting:



```
Main Menu
=========
R - Receive Morse code
S - Send Morse code
X - Exit program

Enter your choice: x
```

Task 2

1 mark Addition of constant called FULLSTOP and set to the value '.'



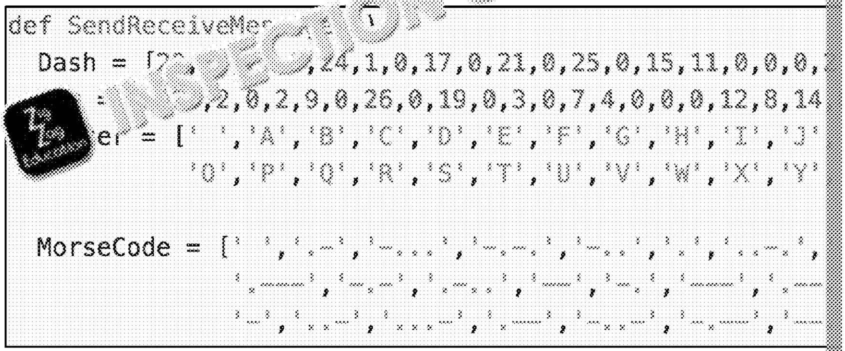
```
EMPTYSTRING = ''
FULLSTOP = '.'
```

1 mark Adding '.' as the 28th element of the list Letter

1 mark Adding '.-.-.-' as the 28th element of the list MorseCode

1 mark Modifying the lists `Dot` and `Dash` correctly sc[illegible] a sequence of dot

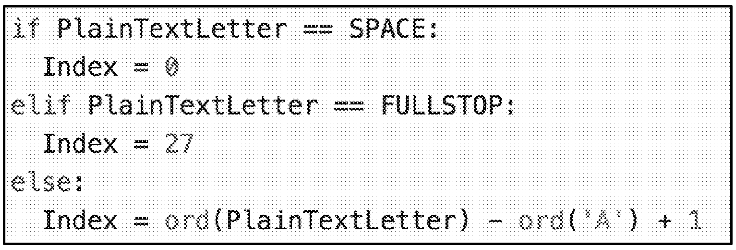
in the number 27



```
def SendReceiveMe[illegible]
  Dash = [2[illegible],24,1,0,17,0,21,0,25,0,15,11,0,0,0,
  [illegible],2,0,2,9,0,26,0,19,0,3,0,7,4,0,0,0,12,8,14
  Letter = [' ','A','B','C','D','E','F','G','H','I','J'
            'O','P','Q','R','S','T','U','V','W','X','Y'

  MorseCode = [' ','.-','-...','-.-.','-..','.','..-.',
               '.---','-.-','.-..','--','-.','---','.--
               '-','..-','...-','.--','-..-','-.--','--
```

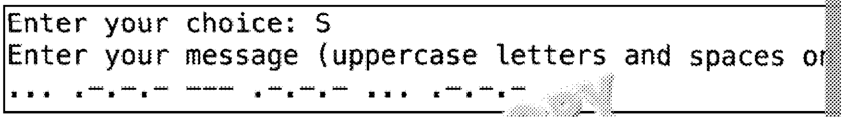
1 mark Modifying the selection statement in to correctly detect a full stop us

1 mark Selection statement correctly uses the sequence from the 28th elemen



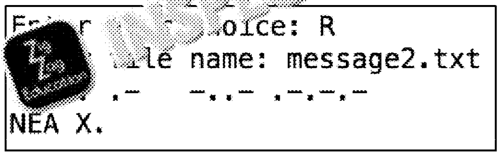
```
if PlainTextLetter == SPACE:
  Index = 0
elif PlainTextLetter == FULLSTOP:
  Index = 27
else:
  Index = ord(PlainTextLetter) - ord('A') + 1
```

1 mark Screenshot shows selecting S to send a message and entering S.O.S. Morse Code being output as below:



```
Enter your choice: S
Enter your message (uppercase letters and spaces or
... .-.-.- --- .-.-.- ... .-.-.-
```

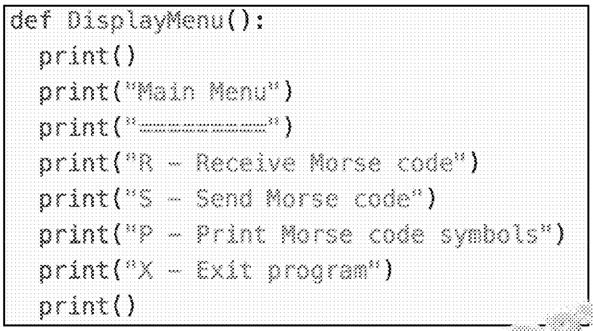
1 mark Screenshot shows selecting R t[illegible] message and entering mes then the message is d[illegible] show below including the full stop.



```
E[illegible]r [illegible]oice: R
[illegible]ile name: message2.txt
[illegible] .-   -..- .-.-.-
NEA X.
```

Task 3

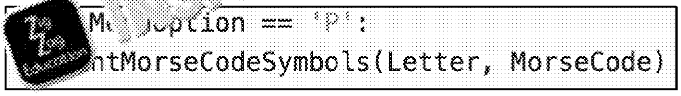
1 mark Addition of new option to the menu by modifying DisplayMenu, on



```
def DisplayMenu():
  print()
  print("Main Menu")
  print("=========")
  print("R - Receive Morse code")
  print("S - Send Morse code")
  print("P - Print Morse code symbols")
  print("X - Exit program")
  print()
```

1 mark Inclusion of menu option P in a new selection structure in SendRece

1 mark New option calls the new subroutine PrintMorseCodeSymbols a Letter and MorseCode



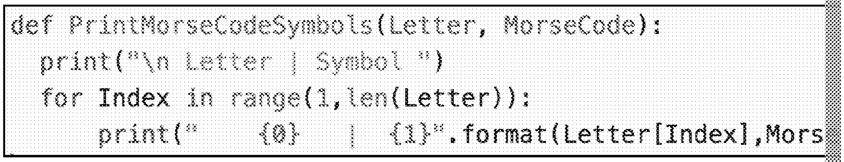
```
elif MenuOption == 'P':
  PrintMorseCodeSymbols(Letter, MorseCode)
```

1 mark Code for subroutine PrintMorseCodeSymbols has two parameter correctly named)

1 mark Print statement for the table heading outside of any iterative or selec

1 mark Iterative structure that will iterate through the Letter and MorseCode for the length)

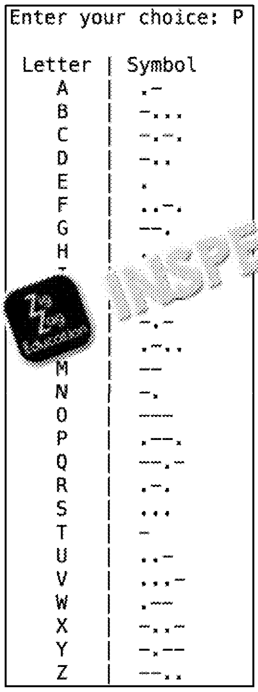
1 mark Suitable code inside the iterative structure to print out a letter with i



```
def PrintMorseCodeSymbols(Letter, MorseCode):
  print("\n Letter | Symbol ")
  for Index in range(1,len(Letter)):
      print("    {0}    |  {1}".format(Letter[Index],Mors
```

1 mark Screenshot shows two columns, one for Letter and another for Symb letters and symbols correctly mapped to each other

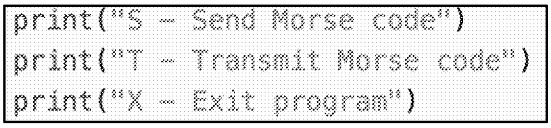
1 mark Screenshot shows the table in precisely the correct format as per the capitalisation and correct spacing (ignore failure to leave a blank line



```
Enter your choice: P

 Letter | Symbol
    A   |  .-
    B   |  -...
    C   |  -.-.
    D   |  -..
    E   |  .
    F   |  ..-.
    G   |  --.
    H   |  .
[illegible]
        |  -.-
        |  .-..
    M   |  --
    N   |  -.
    O   |  ---
    P   |  .--.
    Q   |  --.-
    R   |  .-.
    S   |  ...
    T   |  -
    U   |  ..-
    V   |  ...-
    W   |  .--
    X   |  -..-
    Y   |  -.--
    Z   |  --..
```

Task 4

1 mark Addition of new option to the menu by modifying DisplayMenu, on

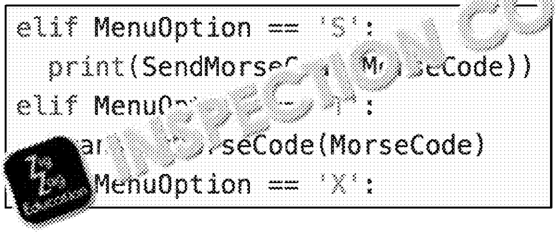


```
print("S - Send Morse code")
print("T - Transmit Morse code")
print("X - Exit program")
```

1 mark Inclusion of menu option T in a new selection structure in SendRec

1 mark New option calls the new subroutine TransmitMorseCodeSymbo MorseCode

1 mark Modification of menu option S to print out the result of the call to Se



```
elif MenuOption == 'S':
  print(SendMorseCode(MorseCode))
elif MenuOption == 'T':
  TransmitMorseCode(MorseCode)
elif MenuOption == 'X':
```

1 mark Code for subroutine TransmitMorseCode has one parameter (ever named)

1 mark Call to SendMorseCode passing the argument of MorseCode (acce

1 mark Result of call to SendMorseCode stored in a variable

1 mark Suitable iterative structure to go through the Morse code version of t

1 mark Selection statement to store different transmission strings based on t

1 mark Inclusion of space, dot and dash in the selection statement

1 mark Selection statement correctly handles putting a single space betweer

1 mark Selection statement correctly handles putting a total of three spaces

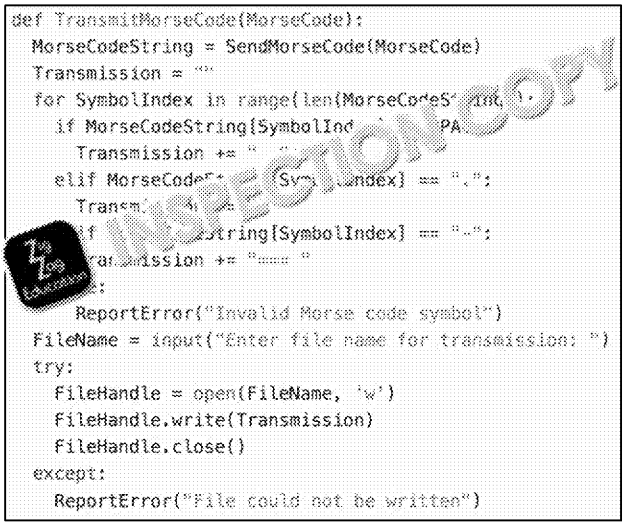
1 mark Selection statement correctly handles putting a total of seven spaces

1 mark Suitable prompt to enter a file name

1 mark Transmission string correctly written to the file

1 mark File is closed after being written to

1 mark Using a try...except... structure with an appropriate error message to



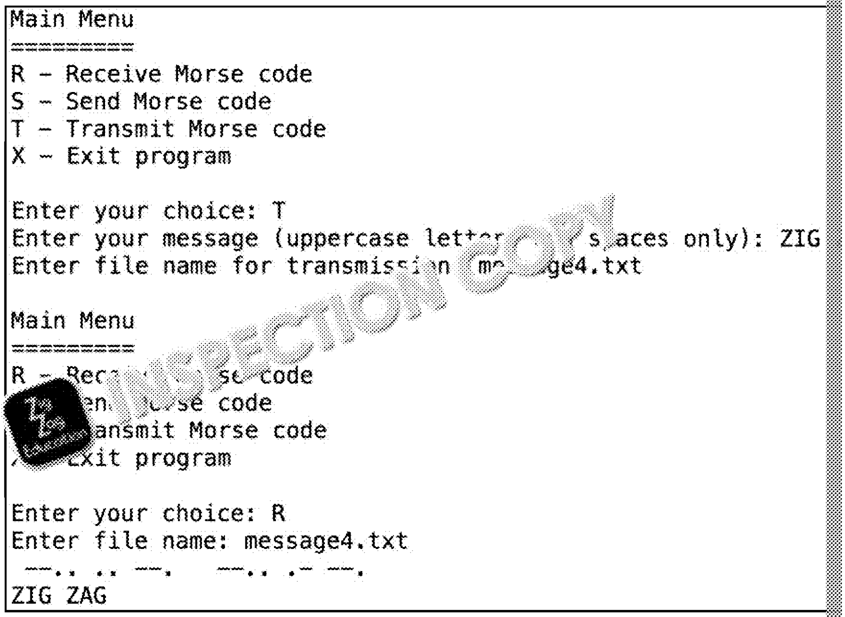
```
def TransmitMorseCode(MorseCode):
  MorseCodeString = SendMorseCode(MorseCode)
  Transmission = ""
  for SymbolIndex in range(len(MorseCodeString)):
    if MorseCodeString[SymbolIndex] == SPACE:
      Transmission += "  "
    elif MorseCodeString[SymbolIndex] == ".":
      Transmission += "= "
    elif MorseCodeString[SymbolIndex] == "-":
      Transmission += "=== "
    else:
      ReportError("Invalid Morse code symbol")
  FileName = input("Enter file name for transmission: ")
  try:
    FileHandle = open(FileName, 'w')
    FileHandle.write(Transmission)
    FileHandle.close()
  except:
    ReportError("File could not be written")
```

1 mark Screenshot shows choosing option T from the menu and entering the with a space between the two words)

1 mark Screenshot shows a prompt for the file name and the user entering m

1 mark Screenshot shows the user choosing option R from the menu and ent at the prompt

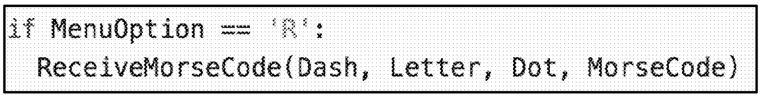
1 mark Screenshot shows the correct Morse code and decoded message as p



```
Main Menu
=========
R - Receive Morse code
S - Send Morse code
T - Transmit Morse code
X - Exit program

Enter your choice: T
Enter your message (uppercase letters and spaces only): ZIG
Enter file name for transmission: message4.txt

Main Menu
=========
R - Receive Morse code
S - Send Morse code
T - Transmit Morse code
X - Exit program

Enter your choice: R
Enter file name: message4.txt
 --.. .. --.   --.. .- --.
ZIG ZAG
```

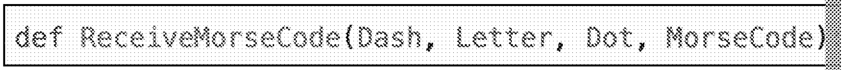
Task 5

1 mark Inclusion of new argument in the called to ReceiveMorseCode in



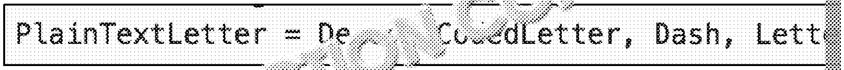
```
if MenuOption == 'R':
  ReceiveMorseCode(Dash, Letter, Dot, MorseCode)
```

1 mark Addition of new parameter to ReceiveMorseCode



```
def ReceiveMorseCode(Dash, Letter, Dot, MorseCode)
```

1 mark Modification of the call to Decode in ReceiveMorseCode to speci



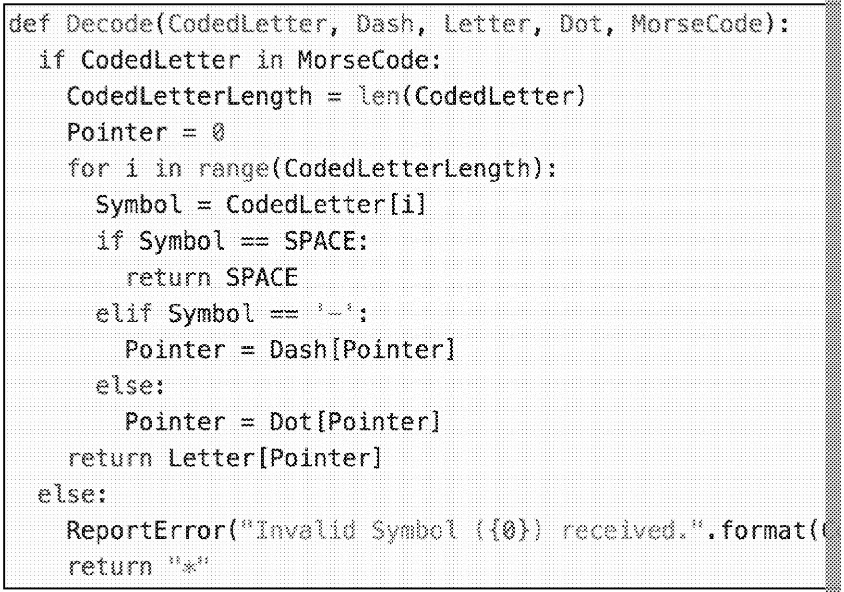
```
PlainTextLetter = Decode(CodedLetter, Dash, Lett
```

1 mark Addition of new parameter to Decode

1 mark Selection statement to include/exclude valid letters (depending on lo

1 mark Error is reported if the CodedLetter doesn't represent a valid sequ

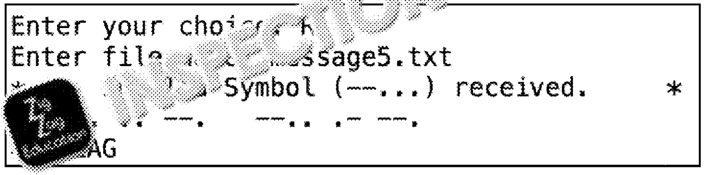
1 mark An asterisk is returned if the sequence of dots and dashes is invalid



```
def Decode(CodedLetter, Dash, Letter, Dot, MorseCode):
  if CodedLetter in MorseCode:
    CodedLetterLength = len(CodedLetter)
    Pointer = 0
    for i in range(CodedLetterLength):
      Symbol = CodedLetter[i]
      if Symbol == SPACE:
        return SPACE
      elif Symbol == '-':
        Pointer = Dash[Pointer]
      else:
        Pointer = Dot[Pointer]
    return Letter[Pointer]
  else:
    ReportError("Invalid Symbol ({0}) received.".format(C
    return "*"
```

1 mark Screenshot shows an error message containing the invalid symbol (--

1 mark Screenshot shows the decoded message as *IG ZAG

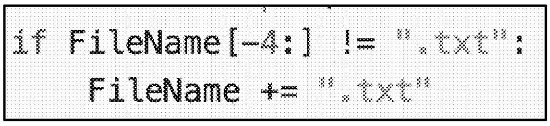


```
Enter your choice: R
Enter file name: message5.txt
* Invalid Symbol (--...) received.    *
--... .. --.   --.. .- --.
*IG ZAG
```

Task 6

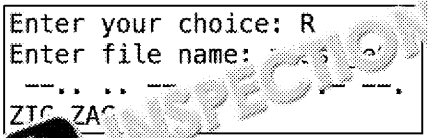
1 mark Selection statement to check if the last four characters of the FileN
reasonable method of isolating and checking the last four characters

1 mark .txt correctly appended to the FileName if it is not already the last



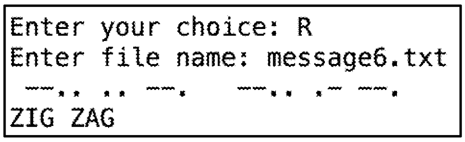
```
if FileName[-4:] != ".txt":
    FileName += ".txt"
```

1 mark Screenshot shows the filename entered as message6 without an exte
received exactly as shown



```
Enter your choice: R
Enter file name: message6
 --.. .. --.    --.. .- --.
ZIG ZAG
```

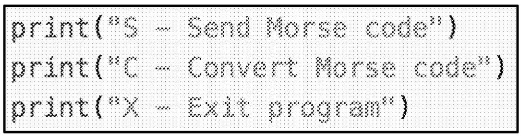
1 mark Screenshot shows the filename entered as message6.txt including th
being received exactly as shown



```
Enter your choice: R
Enter file name: message6.txt
 --.. .. --.    --.. .- --.
ZIG ZAG
```

Task 7

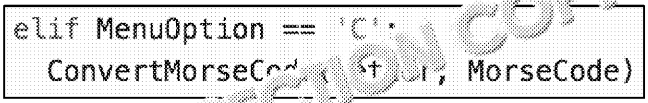
1 mark Addition of new option to the menu by modifying DisplayMenu, or



```
print("S - Send Morse code")
print("C - Convert Morse code")
print("X - Exit program")
```

1 mark Inclusion of menu option C in a new selection structure in SendRec

1 mark New option calls the new subroutine ConvertMorseCodeSymbol Letter and MorseCode (accept them in either order)



```
elif MenuOption == 'C':
  ConvertMorseCode(Letter, MorseCode)
```

1 mark Code for subroutine ConvertMorseCode has two parameters (even

1 mark User is asked to enter a message in Morse code

1 mark Input of Morse code from user stored in a variable with a meaningful

1 mark Suitable iterative structure to go through the Morse code version of t

1 mark Selection statement checks whether the symbol is a valid Morse code

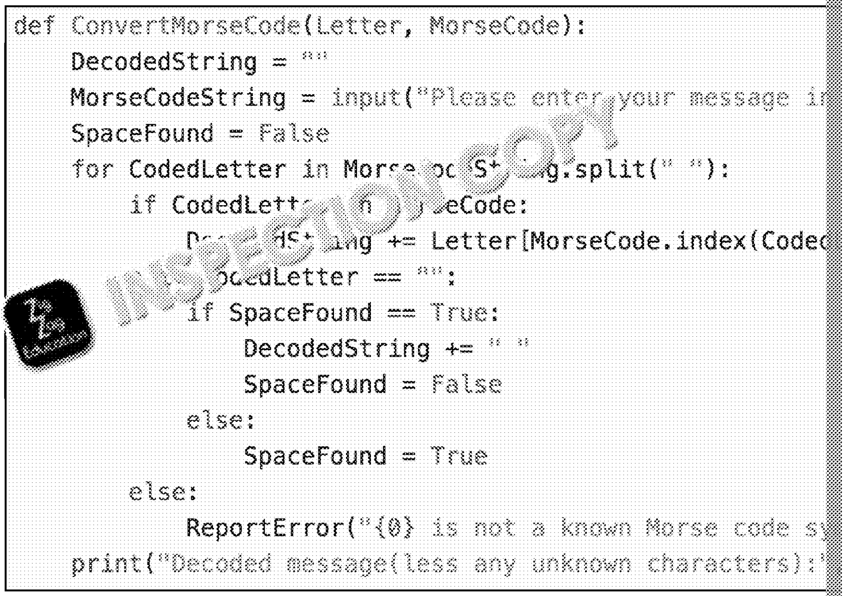
1 mark Inclusion of space in the selection statement

1 mark Selection statement correctly handles a single space between symbo decoded message

1 mark Selection statement correctly handles a total of three spaces betwee in the decoded message

1 mark Printing out any invalid symbols received

Accept alternative working solutions (at full marks) that call the subroutine Decode mark if it's not modified to correctly detect any invalid symbols

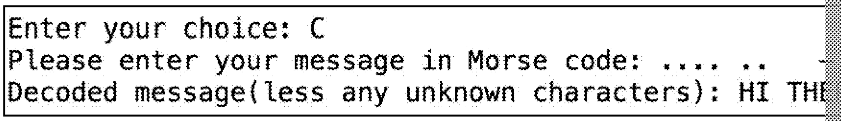


```
def ConvertMorseCode(Letter, MorseCode):
    DecodedString = ""
    MorseCodeString = input("Please enter your message i
    SpaceFound = False
    for CodedLetter in MorseCodeString.split(" "):
        if CodedLetter in MorseCode:
            DecodedString += Letter[MorseCode.index(Coded
        elif CodedLetter == "":
            if SpaceFound == True:
                DecodedString += " "
                SpaceFound = False
            else:
                SpaceFound = True
        else:
            ReportError("{0} is not a known Morse code sy
    print("Decoded message(less any unknown characters):
```

1 mark Screenshot shows choosing option C from the menu and entering the

```
.... ..   - .... . .-. .
```

1 mark Screenshot shows the decoded message as: HI THERE



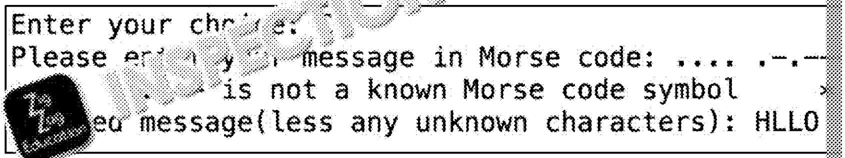
```
Enter your choice: C
Please enter your message in Morse code: .... ..   -
Decoded message(less any unknown characters): HI THE
```

1 mark Screenshot shows choosing option C from the menu and entering the

```
.... .-.--- .-.. .-.. ---
```

1 mark Screenshot shows the decoded message as: HLLO

1 mark Screenshot shows the symbol .-.--- as being invalid/not known



```
Enter your choice: C
Please enter your message in Morse code: .... .-.--
.-.--- is not a known Morse code symbol     >
Decoded message(less any unknown characters): HLLO
```

Task 8

1 mark Suitable variable with meaningful identifier initialised to store the qu

1 mark Selection statement to detect whether the letter is a space or a Mors

1 mark Selection statement placed inside appropriate iterative structure (wh

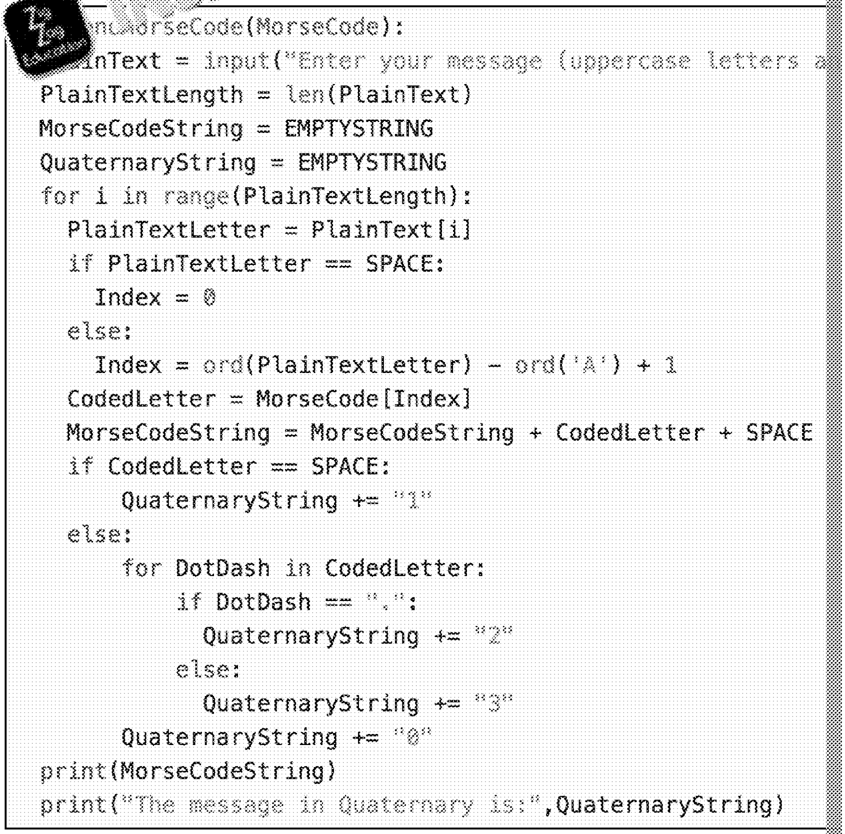
1 mark Selection statement correctly handles a space between words as 1 in

1 mark Selection statement contains an iterative statement to go through al Morse code symbol

1 mark Selection statement correctly handles a dot in a symbol as 2 in quate

1 mark Selection statement correctly handles a dash in a symbol as 3 in quat

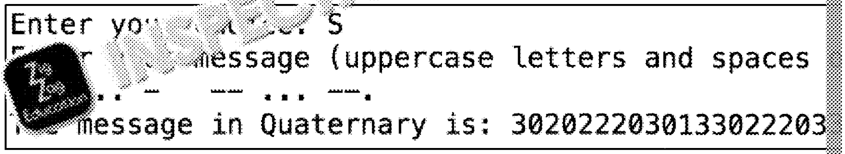
1 mark Selection statement correctly adds a 0 in quaternary after each comp



```
def SendMorseCode(MorseCode):
  PlainText = input("Enter your message (uppercase letters a
  PlainTextLength = len(PlainText)
  MorseCodeString = EMPTYSTRING
  QuaternaryString = 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
    if CodedLetter == SPACE:
        QuaternaryString += "1"
    else:
        for DotDash in CodedLetter:
            if DotDash == ".":
              QuaternaryString += "2"
            else:
              QuaternaryString += "3"
        QuaternaryString += "0"
  print(MorseCodeString)
  print("The message in Quaternary is:",QuaternaryString)
```

1 mark Screenshot shows S chosen from the main menu and the message en

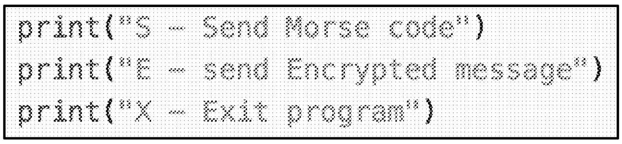
1 mark Screenshot shows the message correctly in quaternary AFTER the Mo



```
Enter your choice: S
Enter your message (uppercase letters and spaces
... -   -- ... --.
The message in Quaternary is: 3020222030133022203
```

Task 9

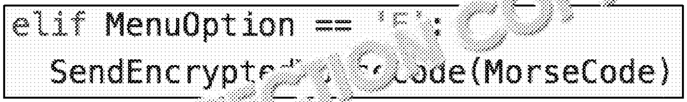
1 mark Addition of new option to the menu by modifying `DisplayMenu`, ac



```
print("S - Send Morse code")
print("E - send Encrypted message")
print("X - Exit program")
```

1 mark Inclusion of menu option E in a new selection structure in `SendRece`

1 mark New option calls the new subroutine `SendEncryptedMorseCode` `MorseCode`



```
elif MenuOption == 'E':
  SendEncryptedMorseCode(MorseCode)
```

1 mark Code for subroutine `SendEncryptedMorseCode` has one paramet

1 mark User is asked to enter a message in plain text which is stored in a var identifier

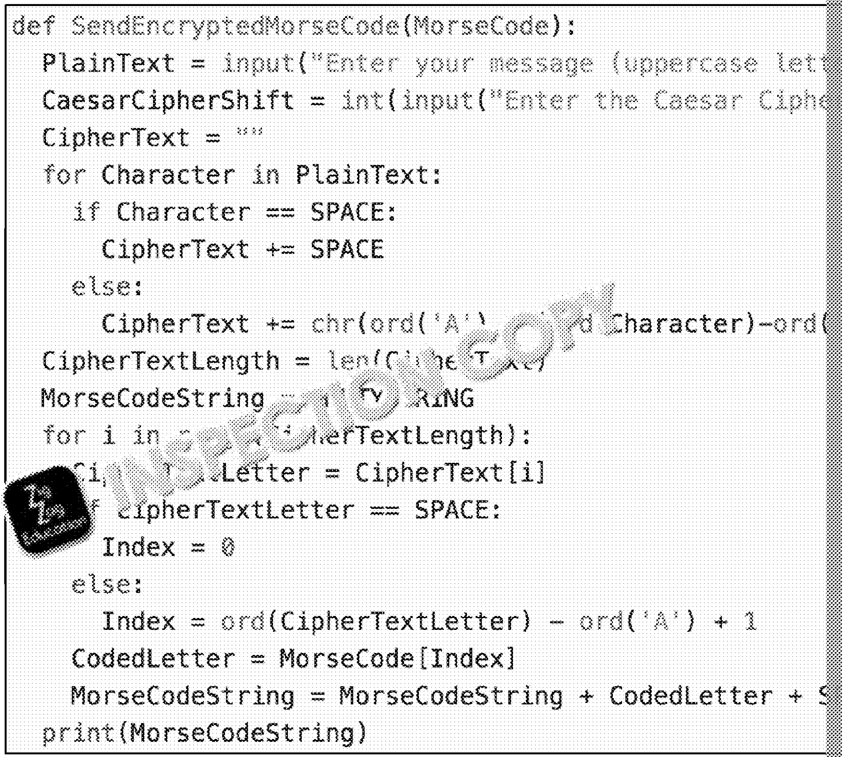
1 mark User is asked to enter a Caesar Cipher Shift which is converted to an with a meaningful identifier

1 mark Iterative structure to go through the message entered, character by c

1 mark Selection statement inside the iterative structure that differentiates b

1 mark Character is correctly Caesar cipher shifted inside the selection stater functions that do this for you). Do not award the mark if they fail to w to A)

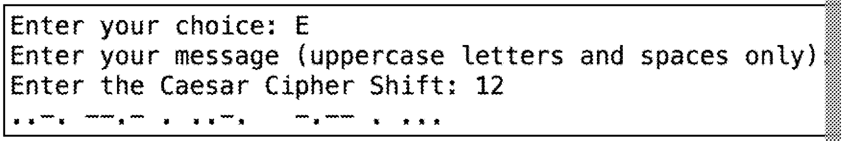
1 mark Cipher text is then correctly converted to Morse code and printed out



```
def SendEncryptedMorseCode(MorseCode):
  PlainText = input("Enter your message (uppercase lett
  CaesarCipherShift = int(input("Enter the Caesar Ciphe
  CipherText = ""
  for Character in PlainText:
    if Character == SPACE:
      CipherText += SPACE
    else:
      CipherText += chr(ord('A') + (ord(Character)-ord(
  CipherTextLength = len(CipherText)
  MorseCodeString = EMPTYSTRING
  for i in range(CipherTextLength):
    CipherTextLetter = CipherText[i]
    if CipherTextLetter == SPACE:
      Index = 0
    else:
      Index = ord(CipherTextLetter) - ord('A') + 1
    CodedLetter = MorseCode[Index]
    MorseCodeString = MorseCodeString + CodedLetter + S
  print(MorseCodeString)
```

1 mark Screenshot shows choosing option E from the menu and entering the Caesar Cipher Shift of 12

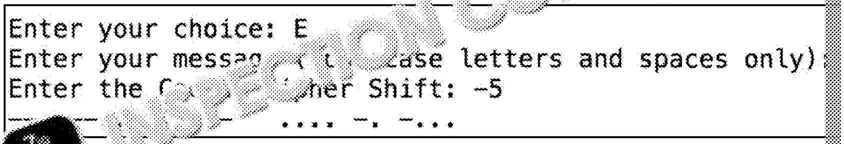
1 mark Screenshot shows the encoded message correctly



```
Enter your choice: E
Enter your message (uppercase letters and spaces only)
Enter the Caesar Cipher Shift: 12
..-. --.- . ..-.   -.-- . ...
```

1 mark Screenshot shows choosing option E from the menu and entering the Caesar Cipher Shift of -5

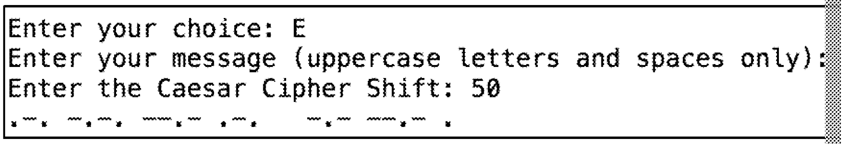
1 mark Screenshot shows the encoded message correctly



```
Enter your choice: E
Enter your message (uppercase letters and spaces only):
Enter the Caesar Cipher Shift: -5
--- --.. -. ---   .... -. -...
```

1 mark Screenshot shows choosing option E from the menu and entering the Caesar Cipher Shift of 50

1 mark Screenshot shows the encoded message correctly



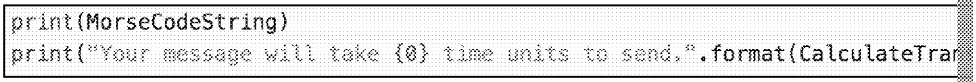
```
Enter your choice: E
Enter your message (uppercase letters and spaces only):
Enter the Caesar Cipher Shift: 50
.-. -.-. --.- .-.   -.- --.- .
```

Task 10

1 mark Print statement appears after the one to print out the message in Mo

1 mark Message prints out the value from the call to `CalculateTransmis`

1 mark Variable `MorseCodeString` correctly passed as the argument



```
print(MorseCodeString)
print("Your message will take {0} time units to send.".format(CalculateTran
```

1 mark Subroutine takes one parameter which has a meaningful identifier

1 mark There is a variable to hold the total transmission time which is initial

1 mark There is an iterative statement to loop through the entire message

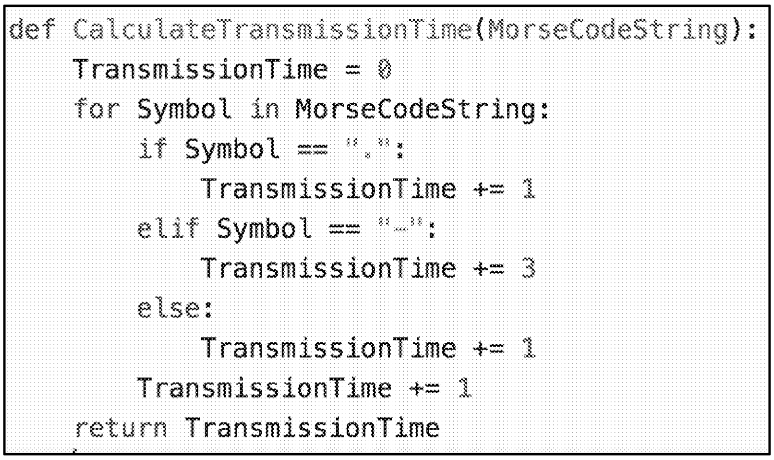
1 mark There is a selection statement inside the iterative statement

1 mark The selection statement adds 1 for a dot and 3 for a dash

1 mark There is an additional +1 time unit after every dot or dash

1 mark The total additional time for an end of letter is +3

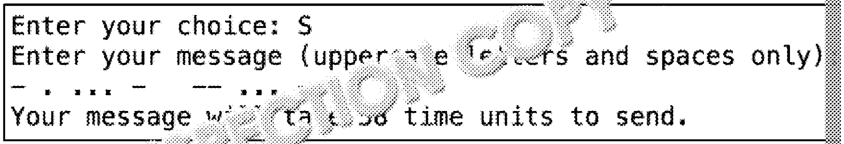
1 mark The total additional time for an end of word is +7



```
def CalculateTransmissionTime(MorseCodeString):
    TransmissionTime = 0
    for Symbol in MorseCodeString:
        if Symbol == ".":
            TransmissionTime += 1
        elif Symbol == "-":
            TransmissionTime += 3
        else:
            TransmissionTime += 1
        TransmissionTime += 1
    return TransmissionTime
```

1 mark Screenshot show S being chosen from the menu and the message TE

1 mark Screenshot shows 58 time units (after the Morse code)



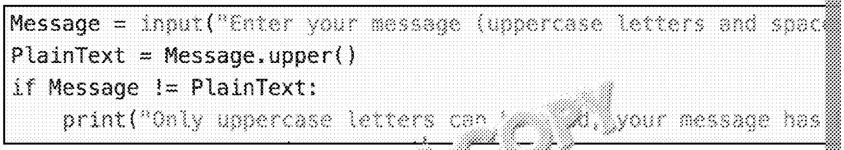
```
Enter your choice: S
Enter your message (uppercase letters and spaces only)
- . ... -   -- ...
Your message will take 58 time units to send.
```

Task 11

1 mark Message is not converted to uppercase as it is input

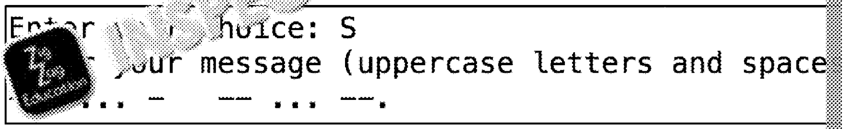
1 mark Selection statement comparing the message to an uppercase version
the message contains at least one lowercase letter)

1 mark Selection statement contains a print statement which explains that t
and shows the uppercase message



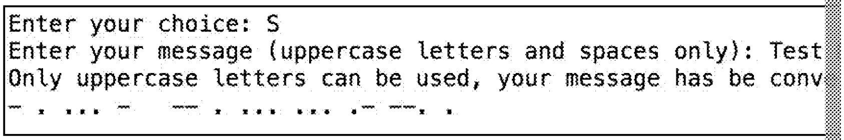
```
Message = input("Enter your message (uppercase letters and spac
PlainText = Message.upper()
if Message != PlainText:
    print("Only uppercase letters can be used, your message has
```

1 mark Screenshot shows as below (with no message about converting it)



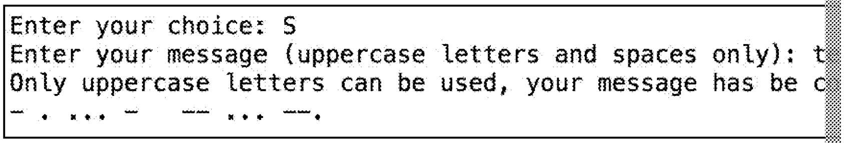
```
Enter your choice: S
Enter your message (uppercase letters and space
- . ... -   -- ... --.
```

1 mark Screenshot the message converted to uppercase including an explan



```
Enter your choice: S
Enter your message (uppercase letters and spaces only): Test
Only uppercase letters can be used, your message has be conv
- . ... -   -- . ... ... .- --. .
```

1 mark Screenshot the message converted to uppercase including an explan



```
Enter your choice: S
Enter your message (uppercase letters and spaces only): t
Only uppercase letters can be used, your message has be c
- . ... -   -- ... --.
```

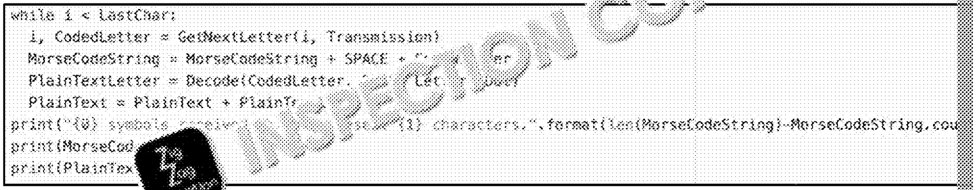
Task 12

1 mark Print statement appears after the iterative structure that parses the m

1 mark Number of symbols computed either by counting the number of dots length of the `MorseCodeString` and deducting the number of spac means)

1 mark Number of characters computed either by counting the number of let length of `PlainText` and deducting the number of spaces (or by so

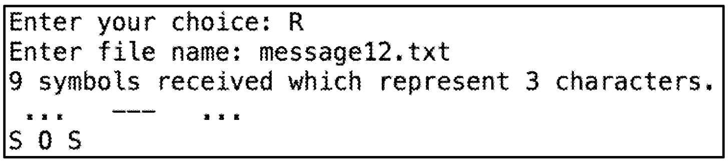
1 mark Print statement is of exactly the same format as the question with th capitalisation



```
while i < LastChar:
  i, CodedLetter = GetNextLetter(i, Transmission)
  MorseCodeString = MorseCodeString + SPACE + [illegible]er
  PlainTextLetter = Decode(CodedLetter, [illegible] Let[illegible]
  PlainText = PlainText + PlainT[illegible]
print("{0} symbols [illegible] {1} characters.".format(len(MorseCodeString)-MorseCodeString.cou
print(MorseCod[illegible]
print(PlainTex[illegible]
```

1 mark Screenshot shows five lines of messages of similar content and forma SAME ORDER as those shown below

1 mark Screenshot shows 9 symbols received and 3 characters received



```
Enter your choice: R
Enter file name: message12.txt
9 symbols received which represent 3 characters.
 ...   ---   ...
S O S
```

Task 13

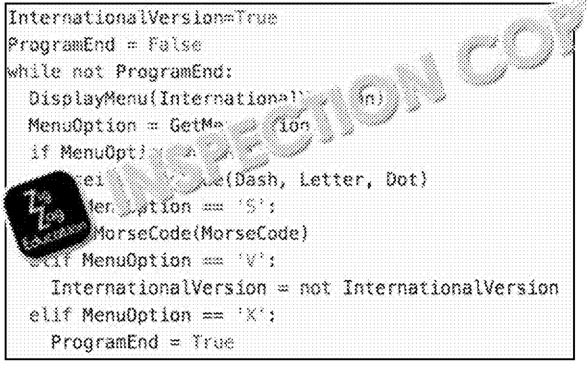
1 mark New variable created with a sensible identifier for International

1 mark Variable is defined and initialised to True within SendReceiveMes

1 mark Call to DisplayMenu now passes the argument InternationalV

1 mark Menu option V is added to the selection statement

1 mark Selection statement for option V changes the value of Internatio or vice-versa



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

1 mark New parameter for DisplayMenu added with meaningful identifier

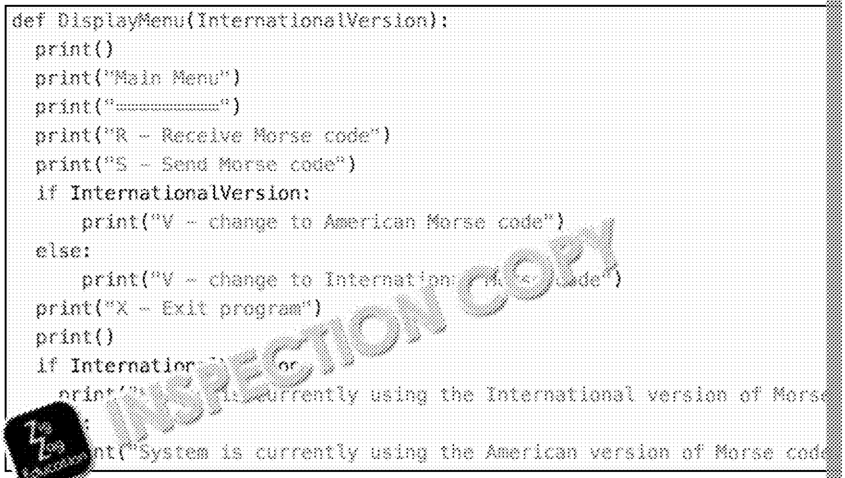
1 mark Selection statement added for InternationalVersion

1 mark Selection statement affects what is displayed on the menu

1 mark Menu options refer to either American version or International versio

1 mark After the menu has printed, there is another selection statement for

1 mark Selection statement will print out a suitable message according to th InternationalVersion correctly stating which version of Morse



```
def DisplayMenu(InternationalVersion):
  print()
  print("Main Menu")
  print("=========")
  print("R - Receive Morse code")
  print("S - Send Morse code")
  if InternationalVersion:
      print("V - change to American Morse code")
  else:
      print("V - change to International Morse code")
  print("X - Exit program")
  print()
  if InternationalVersion:
    print("System is currently using the International version of Morse
  else:
    print("System is currently using the American version of Morse code
```

1 mark Screenshot shows menu option V has been added

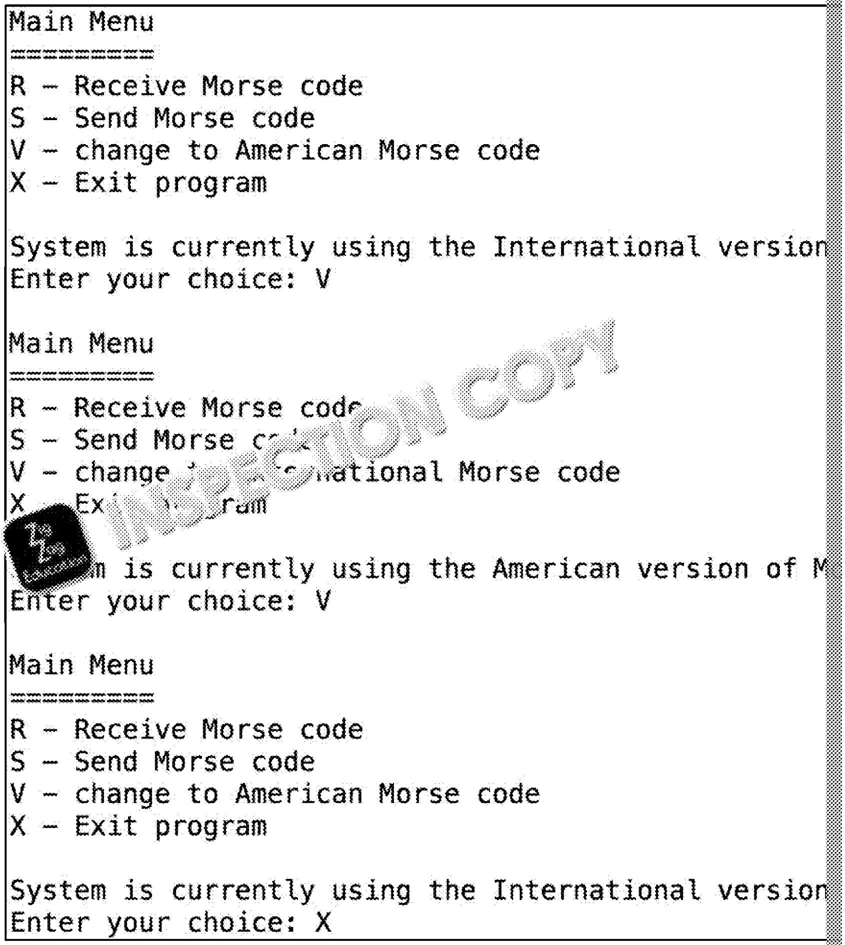
1 mark First menu refers to change to American Morse code

1 mark Screenshot shows that the initial version of Morse code is the Interna

1 mark Screenshot shows that V was selected from the first menu

1 mark Screenshot shows that the menu option correctly toggles to Internati

1 mark Screenshot shows that the message correctly toggles from Internatio
American version after the second and then back again after the thirc



```
Main Menu
=========
R - Receive Morse code
S - Send Morse code
V - change to American Morse code
X - Exit program

System is currently using the International version
Enter your choice: V

Main Menu
=========
R - Receive Morse code
S - Send Morse code
V - change to International Morse code
X - Exit program

System is currently using the American version of M
Enter your choice: V

Main Menu
=========
R - Receive Morse code
S - Send Morse code
V - change to American Morse code
X - Exit program

System is currently using the International version
Enter your choice: X
```

Task 14

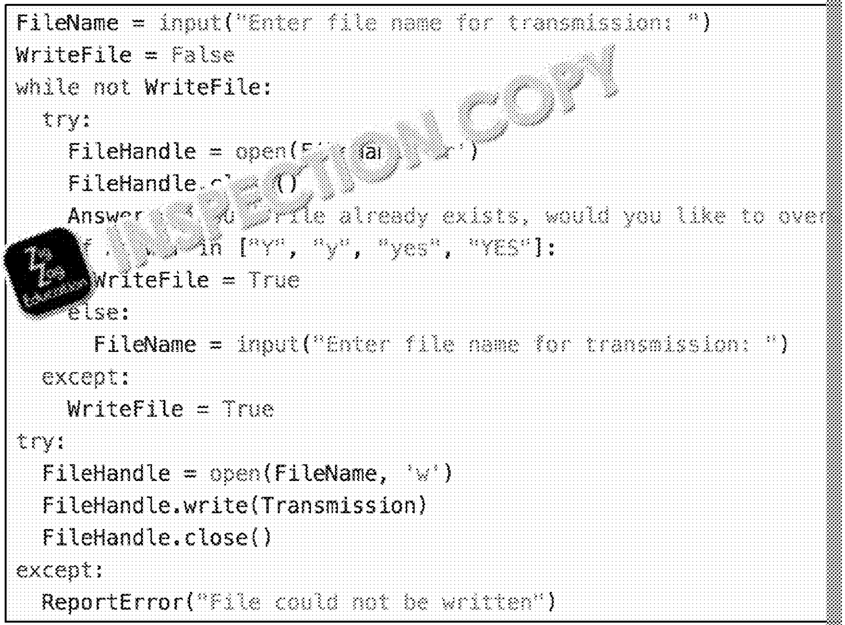
1 mark Iterative statement with a sensible condition to keeping checking un or the user chooses to overwrite the file

1 mark Structure such as try… except… with an open statement which tests i

1 mark Prompt asking the user if they would like to overwrite the file or not and

1 mark Selection statement exits the loop if they want to overwrite the file

1 mark Selection statement asks for a new file name if they don't want to ov

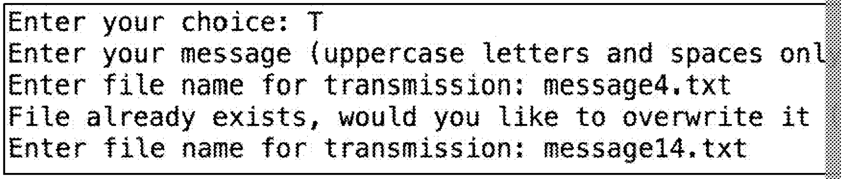


```
FileName = input("Enter file name for transmission: ")
WriteFile = False
while not WriteFile:
  try:
    FileHandle = open(FileName, 'r')
    FileHandle.close()
    Answer = input("File already exists, would you like to over
    if Answer in ["Y", "y", "yes", "YES"]:
      WriteFile = True
    else:
      FileName = input("Enter file name for transmission: ")
  except:
    WriteFile = True
try:
  FileHandle = open(FileName, 'w')
  FileHandle.write(Transmission)
  FileHandle.close()
except:
  ReportError("File could not be written")
```

1 mark Screenshot shows user entering T and then TEST MSG correctly

1 mark User enters message4.txt and the program responds with file already would like to overwrite it

1 mark User selects N and enters message14.txt which results in the output to the main menu)

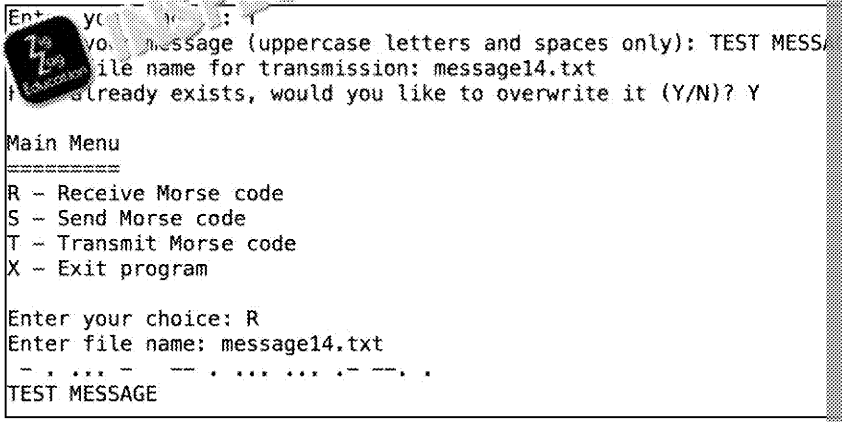


```
Enter your choice: T
Enter your message (uppercase letters and spaces onl
Enter file name for transmission: message4.txt
File already exists, would you like to overwrite it
Enter file name for transmission: message14.txt
```

1 mark Screenshot shows user entering T and then TEST MESSAGE correctly

1 mark User enters message14.txt and then Y, program responds with the r

1 mark User selects R from the main menu and message14.txt which results



```
Enter your choice: T
Enter your message (uppercase letters and spaces only): TEST MESS
Enter file name for transmission: message14.txt
File already exists, would you like to overwrite it (Y/N)? Y

Main Menu
=========
R - Receive Morse code
S - Send Morse code
T - Transmit Morse code
X - Exit program

Enter your choice: R
Enter file name: message14.txt
 - . ... -   -- . ... ... .- --. .
TEST MESSAGE
```

Task 15

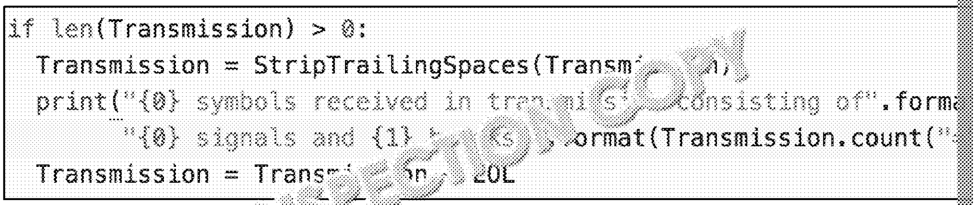
1 mark Print statement is inside the selection statement shown below

1 mark Message prints out the length of `Transmission` as the total numb

1 mark Message correctly counts the number of “=” in the Transmission

1 mark Message correctly counts the number of “ ” in the Transmission

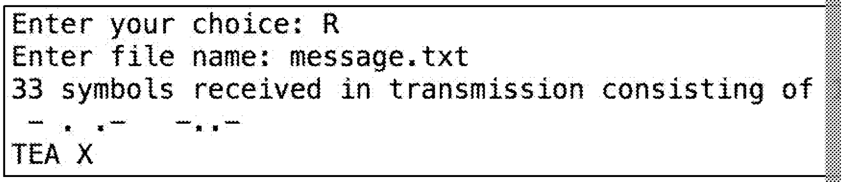
1 mark Message printed is of the correct format and matches the example in



```
if len(Transmission) > 0:
  Transmission = StripTrailingSpaces(Transmission)
  print("{0} symbols received in transmission consisting of".forma
      "{0} signals and {1} breaks".format(Transmission.count("=
  Transmission = Transmission + EOL
```

1 mark Screenshot shows 33 symbols received in total

1 mark Screenshot shows that there were 16 signals and 17 breaks



```
Enter your choice: R
Enter file name: message.txt
33 symbols received in transmission consisting of
 - . .-   -..-
TEA X
```

Name	

ZigZag Education supporting

AS AQA Computer Science Paper 1

Summer 2018

MORSE CODE

Electronic Answer Document (EAD)

Instructions

- Enter your name in the box at the top of this page
- Answer **all** questions by entering your answers into this document
- Remember to **save** this document regularly
- Save and print this document and any additional pages

- Answer **all** questions
- The marks available for each question are shown in brackets

- You will need:
  - □ access to a computer
  - □ access to a printer
  - □ access to appropriate software
  - □ electronic copies of the required skeleton code
  - □ EAD (Electronic Answer Document)

Total marks:

Written Questions

Answer all questions.
Remember to save this document regularly.

Q	Answer	
1	(a)	
	(b)	
	(c)	
	(d)	
	(e)	
	(f)	
	(g)	
	(h)	
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		

Programming Tasks

Answer all questions.
Remember to save this document regularly.

Q	Answer
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	