COMP 1039

Problem Solving and Programming

Programming Assignment 1

Eynesbury Modifications
Michael Ulpen

School of Computer and Information Science
The University of South Australia
July 2022

CONTENTS

INTRODUCTION

This document describes the first assignment for Problem Solving and Programming.

The assignment is intended to provide you with the opportunity to put into practice what you have learned in the course by applying your knowledge and skills to the implementation of a **simple encryption algorithm**.

This assignment is an **individual task** that will require an **individual submission**.

This document is a specification of the required programs and their output. Please ask your practical supervisor if you do not understand any part of this document or the assignment.

ENCRYPTION

A simple way to encrypt data is attributed to Julius Caesar, the Roman Emperor. (If you are interested, you may like to read the following… http://en.wikipedia.org/wiki/Caesar_cipher). This method takes each character in a message and replaces it with one which is a certain distance (offset) along the alphabet from it.

For example:

```
1  2  3  4  5  6  7  8  9  .  .  .  .  .
A  B  C  D  E  F  G  H  I  J  K  L  M  .  .  .
+3 offset →
      A  B  C  D  E  F  G  H  I  J  .  .  .
```


Encrypting “ACE” with the offset of +3 will result in “DFH”. To decrypt “DFH”, offset by the same amount in the other direction (i.e. with the negative offset -3). Decrypting “DFH” with the offset -3 will result in “ACE”.

You will need to use the following two functions:

- **`ord(c)`**
  If `c` is a character (a string of length 1), `ord(c)` returns an integer representing the ASCII position of that character. For example: `ord('a')` returns the integer `97`, `ord('b')` returns the integer `98`, etc.

- **`chr(i)`**
  If `i` is an integer, `chr(i)` returns a string containing only one character with an ASCII code equal to the integer `i`. For example: `chr(97)` returns the string `'a'`, `chr(98)` returns the string `'b'`

Instead of restricting the message to the alphabetic characters only, we will use all the printable ASCII characters i.e., all the characters from ASCII 32 (Space) to ASCII 126 (~).

0	NUL	16	DLE	32	SP	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

MENU

Your task is to write a **menu driven program** called *yourId*`_encryption.py` that will allow the user to enter commands and process these commands until the quit command is entered.

The following commands should be allowed:

1. **Enter Message:**
   Prompt for and read (from the keyboard) a string to be encrypted.

2. **Encrypt Message:**
   Encrypts the previously entered message or displays an error message if no string was entered. The message will be encrypted using a randomly generated number between 32 and 126 as the offset/encryption key. This encryption key will be converted into a character using `chr(i)` and appended to the encrypted string.

3. **Decrypt Message:**
   Decrypts the previously entered message or displays an error message if no string was entered. The message will be decrypted by converting the last letter in the string (the encryption key) to its ASCII value, and using this to offset all other characters in the negative direction. The encryption key should then be removed from the message.

4. **Quit:**
   Displays a goodbye message to the screen and quits the program.

REQUIREMENTS

It is recommended that you develop this part of the assignment in the suggested stages. Each stage is worth a portion of the marks.

It is expected that your solution will include the use of:

- Your solution in a file called *`yourEmailId`*`_encryption.py`.
- Appropriate and well-constructed while and for loops.
- Appropriate if, if-else, if-elif-else statements.
- The use of the `ord(c)` and `chr(i)` functions.
- The use of the `random.randint(a, b)` function in order to generate an encryption key.
- Output that **strictly** adheres to the assignment specifications. If you are not sure about these details, you should check with the provided file: 'Sample Output_Encryption'.
- Good programming practice:
  - Consistent commenting, layout, and indentation. You are to provide comments to describe: your details, program description, all variable definitions, and every significant section of code.
  - Meaningful variable names.

Your solution **MUST NOT** use:

- `break`, or `continue` statements in your solution. **Do not** use the `quit`() or `exit`() functions or the `break` or `return` statements (or any other techniques) as a way to break out of loops. Doing so will result in a significant mark deduction.

Please ensure that you use Python 3.8.0 *or later* to complete your assignments. Your programs **MUST** run using Python 3.8.0.

STAGES

It is recommended that you develop this part of the assignment in the suggested stages. Many problems in later stages are due to errors in early stages. **Make sure you have finished and thoroughly tested each stage before continuing.**

Stage 1

Write code to display your details to the screen. Your code should output your details to the screen in this format:

```
File       : wayby001_petals.py
Author     : Batman
Student ID : 0123456X
Description: Programming Assignment 1 - Encryption
This is my own work as defined by the Eynesbury Academic Misconduct Policy.
```

Ensure your output follows this format exactly. Ensure you change the file, author, and student ID to your own.

Stage 2

Write code to display the menu and to prompt for and read the user's choice.


```
-------------------
     MAIN MENU
-------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4):
```

Ensure this code works correctly before moving onto the next stage.

Stage 3

Set up a loop to prompt for and read menu commands until the user enters 4 (to quit). You do not need to perform any encryption/decryption at this point; you may simply display appropriate messages to the screen.

Sample output:

```
------------------
     MAIN MENU
------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 1
Option 1: Enter Message

------------------
     MAIN MENU
------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 2
Option 2: Encrypt Message

------------------
     MAIN MENU
------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 3
Option 3: Decrypt Message

------------------
     MAIN MENU
------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 4

Goodbye
```


Test your program to ensure this is working before you move onto the next stage. Ensure the loop ends after the user enters option 4.

Once you have that working, back up your program. *Note: When developing software, you should always have fixed points in your development where you know your software is bug free and runs correctly.*

Stage 4

Include code to detect if the user inputs a menu option less than 1 or greater than 4. If this happens print an error message and allow them to enter another number.

Sample output:

```
-------------------
     MAIN MENU
-------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 5
Invalid choice. Enter an option (1,2,3,4): 0
Invalid choice. Enter an option (1,2,3,4): 4

Goodbye
```

Make sure the program runs correctly.

Stage 5

Create a variable called message set to an empty string above your loop. It is important that all menu options use and modify this same variable.

Add code to implement option 1: **Enter Message**.

Prompt for and read a string from the keyboard. This string should be saved into the variable message. The same message variable will be used again when you select to encrypt or decrypt it. If the message is not an empty string, it should be displayed after executing each command.

Test your program to ensure it displays the output seen below:

```
-------------------
     MAIN MENU
-------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 1
Please enter a new message: The crow laughs loudest.
Your message is: 'The crow laughs loudest.'.

-------------------
     MAIN MENU
-------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 4
Your message is: 'The crow laughs loudest.'.

Goodbye
```

Stage 6

Add code to implement command 2: **Encrypt Message**.

This command will use the message entered previously in option 1: **Enter Message**. If this message is currently empty, display the error message "`Error: Cannot encrypt an empty message.`" Otherwise, the message should be encrypted.

You will need to use a loop and the `ord(c)` function to get the ASCII position of each character in the message. Test this with the message "abC" and ensure the ASCII positions are 97, 98, and 67.

You will need to add an offset to the ASCII position and then convert this number back to a character using the `chr(i)` function. The encrypted characters should be concatenated together to create your encrypted message. The encrypted message should then be put into the message variable.

To start with, choose 1 as your encryption offset. Test your program by first entering the message 'abC', then encrypt it. The result should be 'bcD'.

Stage 7

Modify your code so that every time the message is encrypted the encryption offset is a different random integer from 32 to 126. You should use the random.randint(a, b) function to do this.

If you test this, you may notice some strange output as some of the characters will be unprintable e.g., characters beyond ASCII position 126. Your program must only work with the printable ASCII character set: all the characters from ASCII 32 (Space) to ASCII 126 (~). When the ASCII position of the encrypted character points to a character beyond 126 it should *wrap around* to the beginning of the set, to position 32.

For any character beyond 126, you should subtract 95 (the total number of characters in the set) to *wrap back* to the beginning. You may have to use a while loop to subtract 95 multiple times until it is within the set, for example, '}' + 125 is 250, and minus 95 is 155. This is still out of bounds. Use a loop.

Test your code to ensure your message is encrypted using a random offset. Note that if the encrypted string contains characters that are not in the table below, then your ASCII values are not 'wrapping' correctly.

0	NUL	16	DLE	32	SP	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

Note: It is very important to test your code at multiple points throughout development. Not only does this ensure there are no errors; it also ensures that you understand how your code works. You may like to use print statements to display your variables as they change within your loops. Remove these print statements when you are sure your code works.

Stage 8

Your encryption key will need to be saved so you can decrypt the message in the future. Modify your code to convert the encryption offset (your encryption key) to a character. Convert the offset into a character using `chr(i)` and add it to the end of the encrypted message.

For example, if the offset was 33 (ASCII value '!'), 33 would be used to offset each character in the message, and '!' (ASCII 33) would be appended to the end of the string. If the message was "ABC", the encrypted string with offset 33 would be "bcd!"

Test this to ensure your message is becoming 1 character longer every time it is encrypted.

Sample output:

```
-------------------
     MAIN MENU
-------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 2
Error: Cannot encrypt an empty message.

-------------------
     MAIN MENU
-------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 1
Please enter a new message: FULLmoon
Your message is: 'FULLmoon'.

-------------------
     MAIN MENU
-------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 2
Your message was successfully encrypted.
Your message is: 'n}tt6887('.

-------------------
     MAIN MENU
-------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 4
Your message is: 'n}tt6887('.

Goodbye
```


Note: The random offset encryption key was 40 (ASCII '('). Every character in the message was offset by 40. The characters in "moon" were wrapped back by 95 because they were moved to positions outside the range 32 – 126. The ASCII character for 81 '(' was appended to the string, which made the string 1 character longer.

Stage 9

Add code to implement option 3: **Decrypt Message**.

This command will use the same message entered previously in option 1: **Enter Message**, and then encrypted using option 2: **Encrypt Message**. If this message is currently empty, display the error message "`Error: Cannot decrypt an empty message`". Otherwise, the message should be decrypted.

The encryption key will be found as the last character in the string. You will need to use the `ord(c)` function to get the ASCII value of this character. The number that is returned is the offset. You will need to subtract this offset from the ASCII position of each other character in the message variable. These ASCII values will then need to be converted back to characters using the `chr(i)` function. The decrypted characters should be concatenated together to create your decrypted message. The last character (the encryption key) should not be included in the decrypted message. The decrypted message should then be put into the message variable.

Your program must only work with the printable ASCII character set: all the characters from ASCII 32 (Space) to ASCII 126 (~). When the offset position points to a character that is less than 32, it should wrap around to the end of the set. You may have to add 95 multiple times until it is within 32 to 126. Use a loop.

Sample output:

```
--------------------
     MAIN MENU
--------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 3
Error: Cannot decrypt an empty message.

--------------------
     MAIN MENU
--------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 1
Please enter a new message: Long hallways
here.
Your message is: 'Long hallways here.'.

--------------------
     MAIN MENU
--------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit
```

```
Enter an option (1,2,3,4): 2
Your message was successfully encrypted.
Your message is: 'lQPIaJONNYC[UaJGTGoA'.

--------------------
     MAIN MENU
--------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit
Enter an option (1,2,3,4): 3
Your message was successfully decrypted.
Your message is: 'Long hallways here.'.

--------------------
     MAIN MENU
--------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 4
Your message is: 'Long hallways here.'.

Goodbye
```

Stage 10

If you decrypt your message before it was encrypted or decrypt it more times than it was encrypted, you may lose your original message. For example, if you decrypt the message “ABC” three times, the message becomes an empty string.

Modify your code to count how many times the message is encrypted and subtract 1 from the counter whenever it is decrypted. If you attempt to decrypt the message when the counter is less than 1, display the warning, “Your message may already be fully decrypted...” and ask the user, “Are you sure you want to decrypt (y/n)? “. Only decrypt the message if the user enters “y”. Ensure you reset the counter whenever you enter a new message.

Include an input validation loop to ensure the user enters only ‘y’ or ‘n’.

```
--------------------
     MAIN MENU
--------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 1
Please enter a new message: ABC
Your message is: 'ABC'.

--------------------
     MAIN MENU
--------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 3
Your message may be fully decrypted...
Are you sure you want to decrypt (y/n)? y

Your message was successfully decrypted.
Your message is: ']^'.

--------------------
     MAIN MENU
--------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 3
Your message may be fully decrypted...
Are you sure you want to decrypt (y/n)? no
Invalid choice. Enter y or n: yes
Invalid choice. Enter y or n: y

Your message was successfully decrypted.
Your message is: '^'.
```

```
--------------------
     MAIN MENU
--------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 3
Your message may be fully decrypted...
Are you sure you want to decrypt (y/n)? y
Your message was successfully decrypted.

--------------------
     MAIN MENU
--------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 3
Error: Cannot decrypt an empty message.

--------------------
     MAIN MENU
--------------------
1. Enter Message
2. Encrypt Message
3. Decrypt Message
4. Quit

Enter an option (1,2,3,4): 4

Goodbye
```

Stage 11

Check the sample output file, 'Sample Output_Part2_Encryption', and if necessary, modify your code so that:

- The output produced by your program **EXACTLY** adheres to the sample output provided.
- Your program behaves as described in these specs **and** the sample output provided.
- Thoroughly test your program to ensure it produces the correct output in all situations.

SUBMISSION DETAILS

Make sure your .py files are included in a zip file. The zip file should be called `yourIdAss1.zip`. For example: `503123Ass1.zip`

Ensure that the following files are included in your submission:

- yourId_encryption.py

For example:

- 503291_encryption.py

All files that you submit must include the following comments:

```
#
# File: fileName.py
# Author: your name
# Id: your id
# Description: Assignment 1 – place assignment description here…
# This is my own work as defined by Eynesbury
# Academic Misconduct policy.
#
```

Assignments that do not contain these details may not be marked.

It is expected that students will make copies of all assignments and be able to provide these if required.

Students may also be expected to explain parts of their assignment to the marking lecturer to show their full understanding of the work.

EXTENSIONS AND LATE SUBMISSIONS

There will be **no** extensions/late submissions for this course without one of the following exceptions:

1. A medical certificate is provided that has the timing and duration of the illness and an opinion on how much the student's ability to perform has been compromised by the illness. **Please note** if this information is not provided the medical certificate WILL NOT BE ACCEPTED. Late assessment items will not be accepted unless a medical certificate is presented to the Course Coordinator. The certificate must be produced as soon as possible and must cover the dates during which the assessment was to be attempted. In the case where you have a valid medical certificate, the due date will be extended by the number of days stated on the certificate up to five working days.
2. An Eynesbury counsellor contacts the Course Coordinator on your behalf requesting an extension. Normally you would use this if you have events outside your control adversely affecting your course work.
3. Unexpected work commitments. In this case, you will need to attach a letter from your work supervisor with your application stating the impact on your ability to complete your assessment.
4. Military obligations with proof.

Applications for extensions must be lodged with the Course Coordinator before the due date of the assignment

Note: Equipment failure, loss of data, 'Heavy work commitments' or late starting of the course are not sufficient grounds for an extension.

ACADEMIC MISCONDUCT

Deliberate academic misconduct, such as plagiarism, is subject to penalties.

- Do **NOT** share your code with others
- Do **NOT** copy code found on the internet or from any other source

Information about Academic integrity can be found in the "Policies and Procedures" section of the Eynesbury website.

MARKING CRITERIA

NAME:	Problem Solving and Programming (COMP 1039) Assignment 1 - Weighting: 10% - Due: Week 8, 202202		
OUTPUT	**SPECIFICATION**	**MARK**	**MAX MARK**
File : wayby001_petals.py Author : Batman Student ID : 0123456X Description: Programming Assignment 1 - Encryption This is my own work as defined by the Eynesbury Academic Misconduct Policy.	Your own details displayed. Correct indentation and alignment.		1 1
---------- MAIN MENU ---------- 1. Enter Message 2. Encrypt Message 3. Decrypt Message 4. Quit	Menu displayed correctly.		1
Enter an option (1,2,3,4):	Correct input option		1
Enter an option (1,2,3,4): 0 Invalid choice. Enter an option (1,2,3,4): 5	Correct error checking Input validation loop		1 1
Enter an option (1,2,3,4): 1 Please enter a new message: hello	Correct enter message option Counter reset		1 1
Your message is: 'hello'.	Output message Output message only when not empty string		1 1
Enter an option (1,2,3,4): 2 Your message was successfully encrypted. Your message is: '1.55(\'.	Encrypts all characters Random encrypt offset Wraps between 32:126 Appends encryption key to string Correct output Counter increased		2 1 2 2 1 1
Enter an option (1,2,3,4): 3 Your message was successfully decrypted. Your message is: 'hello'.	Decrypts all characters Removes encryption key Correct output Counter decreased		2 2 1 1
Enter an option (1,2,3,4): 3 Your message may be fully decrypted.!. Are you sure you want to decrypt (y/n)? yes Invalid choice. Enter y or n: No Invalid choice. Enter y or n: y Your message was successfully decrypted. Your message is: 'XU\\'.	Decryption warning when fully decrypted "Are you sure?" input Input validation loop on "Are you sure?"		2 1 1
Enter an option (1,2,3,4): 2 Error: Cannot encrypt an empty message.	Cannot encrypt an empty message		1
Enter an option (1,2,3,4): 3 Error: Cannot decrypt an empty message.	Cannot decrypt an empty message		1
Enter an option (1,2,3,4): 4 Goodbye	Correctly ends loop Correct output		1 1
Comment header containing academic integrity statement			2
Comments throughout code			2
Descriptive variable names			2
Correct indentation			2
No break or continue statements			2
Appropriate while loops (no inappropriate "flag" variables)			2
COMMENTS:	**TOTAL**		**45 MARKS**