Lab Assignment 2: Code Decoder

Objective

This assignment is designed to provide you with practice using selection statements to solve an encoding problem.

Description

Your friend Maverick recently landed a lucrative engineering co-op work term in an oil-rich country. Meanwhile, you chose to make a modest salary as a co-op intern with the Canadian Security Intelligence Service (CSIS). A week after arriving in the foreign country, Maverick is kidnapped by rebels opposed to oil exploration. He is held captive in a small village within rebel territory. Luckily, a group of disenchanted rebels are willing to smuggle Maverick to one of seven rendezvous points:

- the village bridge
- the village library
- river crossing
- nearby airport
- bus terminal
- hospital
- railway station

where an allied helicopter will pick them up and fly them to safety. For the plan to work, the friendly rebels need to know two pieces of information:

- the day of the rendezvous
- the rendezvous point

CSIS has devised a nine-digit code to carry this information to the friendly rebels. When the time is ripe, CSIS will insert a coded message into the online newspaper of the oil-rich country, as a number on the front page. The rebels have Internet access via satellite links and follow the news. While it is helpful (as a decoy strategy) that the newspaper has other numbers on the front page, e.g. in real adverts, it is vital that the rebel insiders can detect and decode the secret message. CSIS has found a way to smuggle a decoder to the friendly rebels. You are asked to program the decoder in Python. The safety of your friend Maverick depends on your ability to program the decoder correctly.

Version 0: Getting Started

Download and unzip the files in the **V0GetStarted.zip** file into your Current Folder. Open the lab2V0.py file in the Editor Window and inspect it before running it. The program will prompt the user to enter a code to break. Try the following input

Test Case #1 – 123456789

The output for the test case in the command console should match the image in the CommandOutputV0.pdf file that was unzipped. For further testing, go ahead and try other combinations.

Version 1: Partial Decoder

Download and unzip the files in the **V1PartialDecoder.zip** file into your Current Folder. The decoder will implement 4 different rules to determine if the code is correct or not. In version 1, you will implement Rules 1 and 2 and the partial implementation of Rule 3 and 4. Please use the provide lab2V1.py file to write your program.

NOTE. Only Rules 1 and 2 may display an invalid message. When an invalid message is encountered, do not proceed with the remaining Rule checks. For example, if Rule 1 is invalid, do not check Rules 2, 3 or 4.

Rule 1. A valid code must be a 9-digit number.

e.g. 123456789, message MAY be valid
e.g. 12345678, message invalid
e.g. 1234567890, message invalid

If the code entered is invalid, display the message 'Decoy Message: Not a nine-digit number.' to the command window and terminate the program

Rule 2. The code must pass the odd-even "truth" test. (Do not use any loops to do this)

- If the sum of the digits is even, the message is **invalid**.
  e.g. 222222222, sum = 2+2+2+2+2+2+2+2+2 = 18, message invalid

- If the sum of the digits is odd, the message MAY be **valid**.
  e.g. 222222223, sum = 2+2+2+2+2+2+2+2+3 = 19, message may be valid

If the code entered is invalid, display the message 'Decoy Message: Sum is even.' to the command window and terminate the program

Rule 3. Multiply the 3rd digit by the 2nd digit, then subtract the 1st digit. Display this number to the command window (Displaying the text 'day =' is optional)

Rule 4. Calculate the value of the 3rd digit to the power of the 2nd digit.

- If the value is divisible by 3 , then subtract the
  6th digit - 5th digit

- If the value is not divisible by 3, then subtract the
  5th digit - 6th digit

Display this number to the command console. (Displaying the text 'place =' is optional)

Test Case 1 – 12345678
Test Case 2 – 123456788
Test Case 3 – 123456789
Test Case 4 – 732456789

The output for the test cases in the command window should match the images in the CommandOutputV1.pdf file that was unzipped. For further testing, go ahead and try other combinations.

Version 2: Full Decoder

Download and unzip the files in the **V2FullDecoder.zip** file into your Current Folder. Modify/Add to your Version 1 code for full implementation of Rule 3 and 4.

Rule 3. To determine the rescue day, multiply the 3rd digit by the 2nd digit, then subtract the 1st digit. The answer indicates the rescue day, as follows:

1 = Monday	2 = Tuesday	3 = Wednesday	
4 = Thursday	5 = Friday	6 = Saturday	7 = Sunday

Any other number indicates that the message is invalid.

Examples:	233407789	`3x3-2 = 7`	Rescue is on Sunday
	732456789	`2x3-7 = -1`	Message is invalid.

Rule 4. To determine the rendezvous point, calculate the value of the 3rd digit to the power of the 2nd digit

- If the number is divisible by 3, then the rendezvous point is
  6th digit - 5th digit
- If the number is not is divisible by 3, then the rendezvous point is
  5th digit - 6th digit

Rendezvous Number	Rendezvous Point
1	bridge
2	library
3	river crossing
4	airport
5	bus terminal
6	hospital
7	railway station

If the result is one of the following rendezvous numbers, the message is valid:
Any other rendezvous number means that the message is invalid.

The output for invalid messages should indicate only the first reason encountered for rejecting the message. The reasons are one of the following, in this order:

- Decoy Message: Not a nine-digit number.
- Decoy Message: Sum is even.
- Decoy Message: Invalid rescue day.
- Decoy Message: Invalid rendezvous point.

Note - Multiple invalid messages displayed to the command window is incorrect.

If the code successfully passes all 4 rules, please display a message to the command console indicating both the rescue date and rendezvous point on a **single line.** Please use the print() instruction to accomplish this.

Test Case 1 – 233407789
Test Case 2 – 732456789
Test Case 3 – 12345678
Test Case 4 – 123456788
Test Case 5 – 724120779
Test Case 6 – 115984236
Test Case 7 – 643158994
Test Case 8 – 132456789

The output in the command console should match the image in the CommandOutputV2.pdf file that was unzipped. For further testing, go ahead and try other combinations. Include a section comment header

called `Rule1`, `Rule2`, `Rule3`, and `Rule4` and summarize what the sections does.

Code Requirements/Submission details

1. Any program using a terminating instruction is not allowed (ie, sys.exit(),os_exit()etc). Using it will result in a possible maximum mark of **'Min. Pass'**.

2. The 'match' instruction can be used in implementing rules 3 and 4. This is **optional.**

3. Do not implement your solution with any loops (for or while). Using loops will result in a possible maximum mark of **'Min. Pass'**.

4. File naming convention.

lab2<Version>_<ccid>.py

For example **lab2V1_whoy.py** or **lab2V2_whoy.py**

5. Please consult the schedule posted on Eclass for due dates. Late submission for Version 1 will not be accepted. Late submission for Version 2 will result in a **reduction of 10%** per business day (1 day max). Late submission penalties can be waived at the Lab instructor's discretion with valid proof of documentation (ie, doctor's note). Non valid reasons will not be accepted such as but not limited to - forgetting due date, computer/laptop crashing, submitting wrong assignment, etc). It is your responsibility to verify that the submitted file is correct and uploaded properly to the Eclass submission link.