Question 2 Code:

def print\_abacus(num):

'''

Print\_Abacus takes in a string of a number

Then it goes through and adjust the default abacus

Finally it prints the new abacus

'''

print(num + " is represent by: ")

sizeOf = len(num)

bottomRowEmpty = [5,6,7,8]

abacus = [['o','o','o','o','o','o','o','o','o','o','o','o','o'],

['o','o','o','o','o','o','o','o','o','o','o','o','o'],

[' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],

[' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],

['-','-','-','-','-','-','-','-','-','-','-','-','-'],

[' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],

[' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],

[' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],

[' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],

['o','o','o','o','o','o','o','o','o','o','o','o','o'],

['o','o','o','o','o','o','o','o','o','o','o','o','o'],

['o','o','o','o','o','o','o','o','o','o','o','o','o'],

['o','o','o','o','o','o','o','o','o','o','o','o','o']]

for i in range(sizeOf):

colIndex = (13-sizeOf)+i

numSpacesToMove = int(num[i])

if (numSpacesToMove >= 5):

# If we have more than 5, adjust the top row first

abacus[3][colIndex] = 'o'

abacus[1][colIndex] = ' '

numSpacesToMove=numSpacesToMove-5

# Then we change the bottom row

# To keep track of how many bottom rows we've changes

bottomPlaced = 0

while (numSpacesToMove > 0):

# Parse the next empty index from our list of bottom row

rowIndex = bottomRowEmpty[bottomPlaced]

abacus[rowIndex][colIndex] = 'o'

bottomPlaced+=1

numSpacesToMove-=1

''' After we place the beads in the right spot, we put empty spots

in the used beads spots '''

for x in range(9,9+bottomPlaced):

abacus[x][colIndex] = ' '

else:

# If we have less than 5, we only adjust the bottom row

# To keep track of how many bottom rows we've changes

bottomPlaced = 0

while (numSpacesToMove > 0):

# Parse the next empty index from our list of bottom row

rowIndex = bottomRowEmpty[bottomPlaced]

abacus[rowIndex][colIndex] = 'o'

bottomPlaced+=1

numSpacesToMove-=1

''' After we place the beads in the right spot, we put empty spots

in the used beads spots '''

for x in range(9,9+bottomPlaced):

abacus[x][colIndex] = ' '

# Once we finish editing the abacus, we print out the abacus

for a in range(len(abacus)):

row = str(abacus[a])

print(row)

def add\_abacus(num1, num2):

print\_abacus(str(num1))

print('')

print\_abacus(str(num2))

print('')

summation = num1 + num2

print\_abacus(str(summation))

# 2a

print('QUESTION 2A: ')

add\_abacus(54321, 90678)

# 2b

print('QUESTION 2B: ')

add\_abacus(559876543210, 27623428724)

# 2c

print('QUESTION 2C: ')

add\_abacus(127002343627, 23412876241)

Question 2 Output:

A picture containing graphical user interface

Description automatically generatedA picture containing graphical user interface

Description automatically generatedA picture containing background pattern

Description automatically generated

Question 3 Code:

def uncompact\_subtractives(num):

'''

This function is used for uncompacting subtractives by converting the

roman numeral word into an integer and then converting that integer into

roman numerals without subtractives

Returning a roman numeral word without subtractives

'''

rom\_val = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}

int\_val = 0

# Convert the roman numerals to a number

for i in range(len(num)):

if i > 0 and rom\_val[num[i]] > rom\_val[num[i-1]]:

# Subtractives

int\_val += rom\_val[num[i]] - 2 \* rom\_val[num[i-1]]

else:

int\_val += rom\_val[num[i]]

# Break down what symbols need without subtractives

symbolsNeeded = {'I': 0, 'V': 0, 'X': 0, 'L': 0, 'C': 0, 'D': 0, 'M': 0}

while int\_val > 1000:

symbolsNeeded['M']+=1

int\_val-=1000

while int\_val > 500:

symbolsNeeded['D']+=1

int\_val-=500

while int\_val > 100:

symbolsNeeded['C']+=1

int\_val-=100

while int\_val > 50:

symbolsNeeded['L']+=1

int\_val-=50

while int\_val > 10:

symbolsNeeded['X']+=1

int\_val-=10

while int\_val > 5:

symbolsNeeded['V']+=1

int\_val-=5

while int\_val > 0:

symbolsNeeded['I']+=1

int\_val-=1

# Write the new roman numeral word sorted from largest to smallest

newRomanWord = ""

while symbolsNeeded['M'] > 0:

newRomanWord+='M'

symbolsNeeded['M']-=1

while symbolsNeeded['D'] > 0:

newRomanWord+='D'

symbolsNeeded['D']-=1

while symbolsNeeded['C'] > 0:

newRomanWord+='C'

symbolsNeeded['C']-=1

while symbolsNeeded['L'] > 0:

newRomanWord+='L'

symbolsNeeded['L']-=1

while symbolsNeeded['X'] > 0:

newRomanWord+='X'

symbolsNeeded['X']-=1

while symbolsNeeded['V'] > 0:

newRomanWord+='V'

symbolsNeeded['V']-=1

while symbolsNeeded['I'] > 0:

newRomanWord+='I'

symbolsNeeded['I']-=1

return newRomanWord

def add\_romanNums(num1, num2):

# Take out the subtractives

num1 = uncompact\_subtractives(num1)

num2 = uncompact\_subtractives(num2)

# Concatenate

combined = num1 + num2

combined = uncompact\_subtractives(combined)

return combined

def subtract\_romanNums(num1, num2):

# Take out the subtractives

num1 = uncompact\_subtractives(num1)

num2 = uncompact\_subtractives(num2)

num1\_dict = {'I': 0, 'V': 0, 'X': 0, 'L': 0, 'C': 0, 'D': 0, 'M': 0}

num2\_dict = {'I': 0, 'V': 0, 'X': 0, 'L': 0, 'C': 0, 'D': 0, 'M': 0}

for i in num1:

num1\_dict[i]+=1

for j in num2:

num2\_dict[j]+=1

newNum = ''

for x in ['M','D','C','L','X','V','I']:

diff = num1\_dict[x] - num2\_dict[x]

if (diff > 0):

newNum+=(diff\*x)

return newNum

# 3a

print('QUESTION 3A: ')

print(add\_romanNums(add\_romanNums('CXCV','XXXI'), 'LXXXVIII'))

# 3b

print('QUESTION 3B: ')

print(subtract\_romanNums(subtract\_romanNums('CCCXCV','CXV'),'LXX'))

# 3c

print('QUESTION 3C: ')

print(add\_romanNums('CCMCCCII', 'MLCLCIII'))

Question 3 Output:

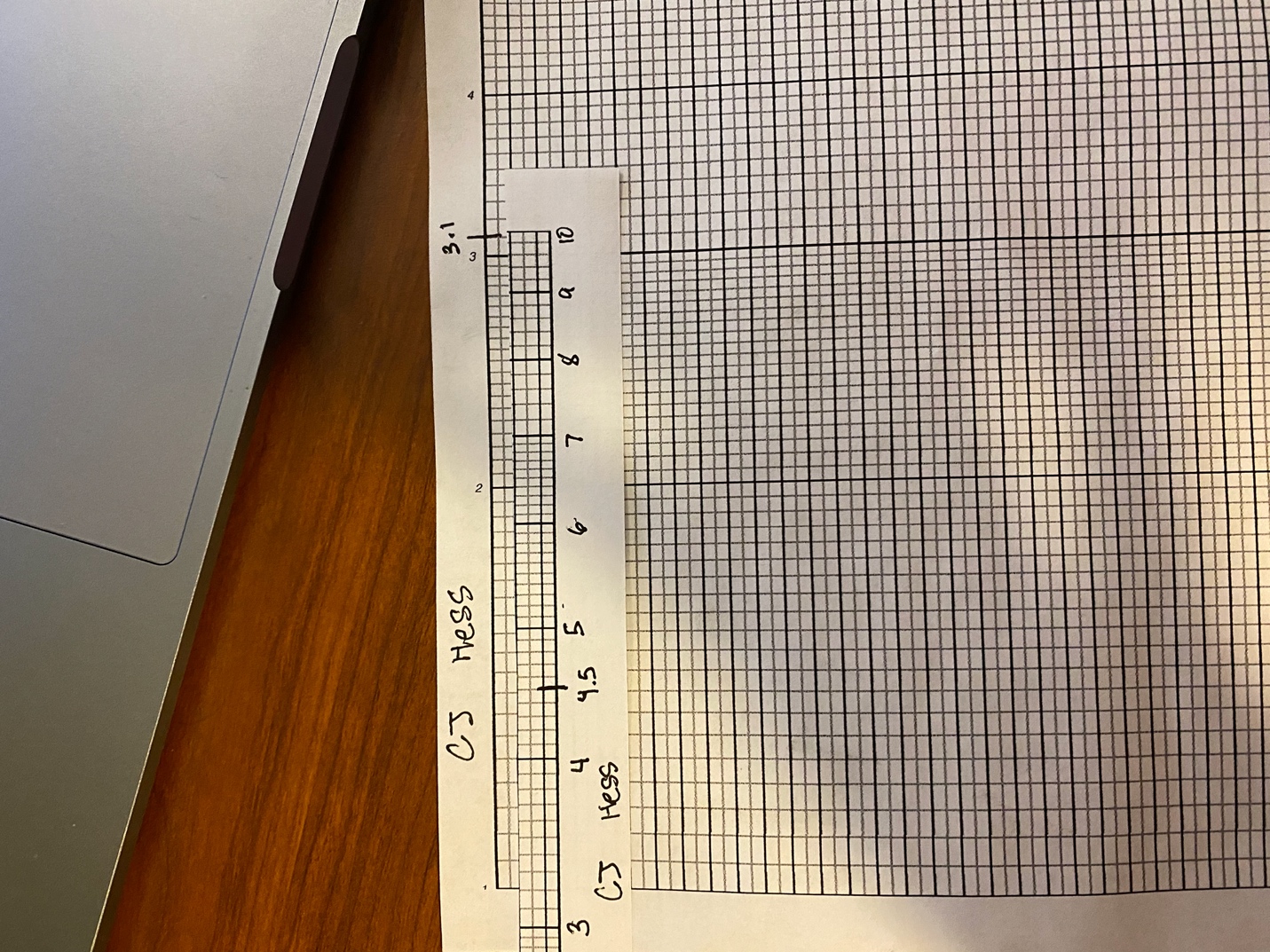
Text

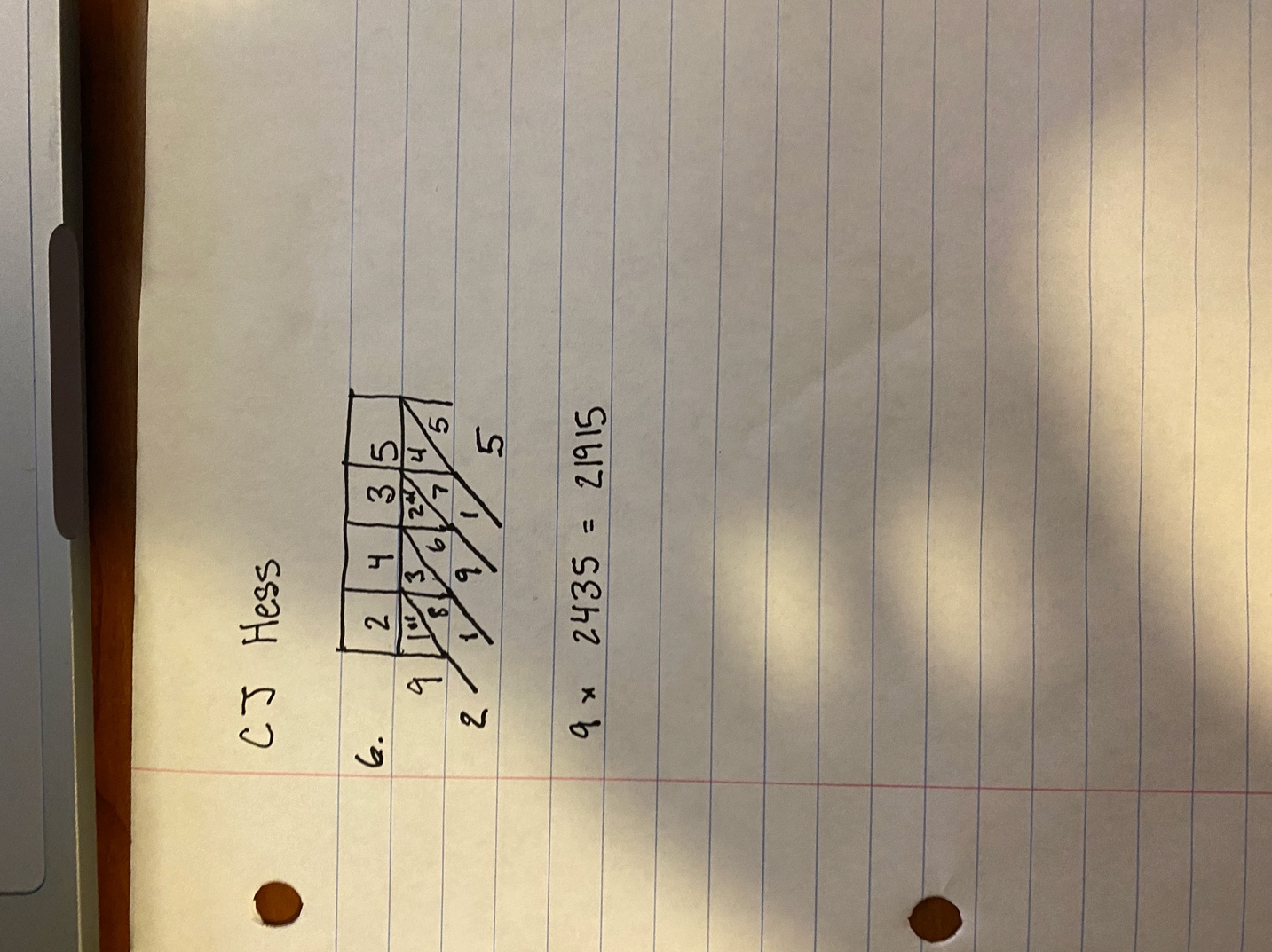
Description automatically generated

\*Question 4a-c ^\*

Question 4d: My initial thought for approaching multiplying two big roman numeral numbers was of confusion. The multiplication method that we use unfortunately could not work for roman numerals and therefore it could not be solved efficiently. The processing time of adding that many roman numerals x number of times would exponentially increase as the amount of digits increased. I do not believe that the Romans could have had a great scientific revolution with the use of only Roman Numerals to work with.

5. I changed the scales by some magnitude of 10 to make them fit on the sheet. So instead of 44.98, I used 4.5 by rounding and dividing by 10. For 3127, I used 3.1 by dividing by 1000 and rounding. With how the slide aligned, I did 4.5 \* 3.1\*10 = 1.395. Then accounting for the scaling, I multiplied by 1000 to get 139500.





7a. The ENIAC was very big and was able to be walked through. It was great for watching processes run and lights execute.

7b. 20 words

7c. Subroutines were not common at all in ENIAC programming due to the extremely limited storage

7d. Ballistic Research Laboratories

7e. The cards would soak up moisture, so they needed to be able to dry out.