**WBSU VI SEMESTER COMPUTER SCIENCE EXAM :**

**PAPER :- CMSADSE06P (DSE4)**

**MEMBERS :**

**STUDENT 1 :**

**NAME :** Arnab Dey

**REGISTRATION NO :** 1251811400201 of 2018 - 2021

**ROLL :** 6211125

**NO :** 16585

**STUDENT 2 :**

**NAME :** Diya Brahmachari

**REGISTRATION NO :** 1251821400205 of 2018 - 2021

**ROLL :** 6212125

**NO :** 16588

**STUDENT 3 :**

**NAME :** Pritam Shil

**REGISTRATION NO :** 1251811300204 of 2018 - 2021

**ROLL :** 6211125

**NO :** 16584

**Converting a pdf data to excel file using Natural Language Processing**

Guide : Dr. Tamal Chakraborty

Members : Arnab Dey Reg. No : 1251811400201

Diya Brahmachari Reg.No : 1251821400205

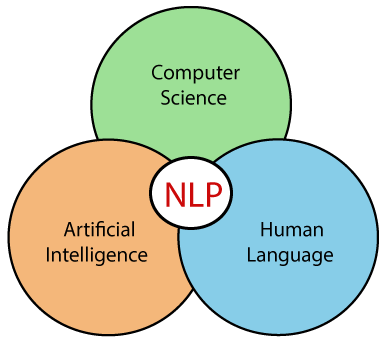
Pritam Shil Reg.No : 1251811300204

**INTRODUCTION**

Our main objective is to collect data from a pdf and translate the data in an excel sheet. We have achieved it using natural language processing via regular expressions.

**NATURAL LANGUGAGE PROCESSING:**

NLP stands for **Natural Language Processing**, which is a part of **Computer Science, Human language,** and **Artificial Intelligence**. It is the technology that is used by machines to understand, analyse, manipulate, and interpret human's languages. It helps developers to organize knowledge for performing tasks such as **translation, automatic summarization, Named Entity Recognition (NER), speech recognition, relationship extraction,**and**topic segmentation**.



## Ambiguity and Uncertainty in Language:

## Ambiguity, generally used in natural language processing, can be referred as the ability of being understood in more than one way. In simple terms, we can say that ambiguity is the capability of being understood in more than one way. Natural language is very ambiguous. NLP has the following types of ambiguities-

### 1.Lexical Ambiguity:

The ambiguity of a single word is called lexical ambiguity. For example, treating the word silver as a noun, an adjective, or a verb.

### 2.Syntactic Ambiguity:

This kind of ambiguity occurs when a sentence is parsed in different ways. For example, the sentence “The man saw the girl with the telescope”. It is ambiguous whether the man saw the girl carrying a telescope or he saw her through his telescope.

### 3.Semantic Ambiguity:

This kind of ambiguity occurs when the meaning of the words themselves can be misinterpreted. In other words, semantic ambiguity happens when a sentence contains an ambiguous word or phrase. For example, the sentence “The car hit the pole while it was moving” is having semantic ambiguity because the interpretations can be “The car, while moving, hit the pole” and “The car hit the pole while the pole was moving”.

### 4.Anaphoric Ambiguity:

This kind of ambiguity arises due to the use of anaphora entities in discourse. For example, the horse ran up the hill. It was very steep. It soon got tired. Here, the anaphoric reference of “it” in two situations cause ambiguity.

### 5.Pragmatic ambiguity:

Such kind of ambiguity refers to the situation where the context of a phrase gives it multiple interpretations. In simple words, we can say that pragmatic ambiguity arises when the statement is not specific. For example, the sentence “I like you too” can have multiple interpretations like I like you (just like you like me), I like you (just like someone else dose).

## NLP Phases:

## Following diagram shows the phases or logical steps in natural language processing:-

## 

### 1.Morphological Processing:

It is the first phase of NLP. The purpose of this phase is to break chunks of language input into sets of tokens corresponding to paragraphs, sentences and words. For example, a word like “uneasy” can be broken into two sub-word tokens as “un-easy”.

### 2.Syntax Analysis:

It is the second phase of NLP. The purpose of this phase is two folds: to check that a sentence is well formed or not and to break it up into a structure that shows the syntactic relationships between the different words. For example, the sentence like “The school goes to the boy” would be rejected by syntax analyzer or parser.

### 3.Semantic Analysis:

It is the third phase of NLP. The purpose of this phase is to draw exact meaning, or you can say dictionary meaning from the text. The text is checked for meaningfulness. For example, semantic analyzer would reject a sentence like “Hot ice-cream”.

### 4.Pragmatic Analysis:

It is the fourth phase of NLP. Pragmatic analysis simply fits the actual objects/events, which exist in a given context with object references obtained during the last phase (semantic analysis). For example, the sentence “Put the banana in the basket on the shelf” can have two semantic interpretations and pragmatic analyzer will choose between these two possibilities.

We have used regular expressions to achieve natural language processing. It is one of the key concepts of Natural Language Processing. A regular expression is ”instruction” given to a function on what and how to match or replace a set of strings. In short, if there’s a pattern in any string, we can easily extract, substitute and do variety of other string manipulation operations using regular expressions.

**MOTIVATION**

During the pandemic we students are giving examinations over the internet and submitting the answer sheet in electronic format(likely pdf) to the institution. Teachers are checking the answer sheet (which is in pdf format)and they might face a problem while evaluating the answer sheet. It is very inconvenient to write and evaluate on the pdf file. So we want to create a solution where the algorithm will translate the question paper into an excel sheet and thus making it convenient for the teachers to put the marks in and calculating the total marks. This solution will be very convenient for the teachers during these times.

In the covid situation almost every university is taking exams in the same method. All students are giving examination through an online base from their home. That’s why every teacher is facing the same problem while checking the answer sheets of students. So we thought of this project for the benefit of all teachers.

We make sure that this solution will be convenient for all teachers in this situation.

**Algorithm**

parseQuestion(question pdf name){

1. First open the file using the pdfreader and store inside a variable.
2. Create a loop according to the number of pages of the pdf.
3. Then using regular language split the pdf for better readability.
4. Ignore some lines keywords from the file
5. Line that starts with “CBCS”
6. From other lines which are not starting with those keywords find the marks for each of the questions.
7. Put the question and their marks inside a dictionary data structure where question numbers and the part marks are the key.
8. Pass the data through check\_parts function

}

check\_parts(pass the data for all the question){

1. Check if there is any ‘+’ symbol in the marks
2. If yes then the question has sub questions and calculate their number of part questions according to the number of + exist in the question.
3. If not, go to the next question until the number of questions ends.
4. After all questions end, exit the function.

}

Write\_xl(pass the data for all the question and the name of the Excel file){

1. Print all the data of the questions.
2. Create a new file tmpl\_paperchecking.xlsx
3. Copy the necessary data using the xlsxwriter module into some variables..
4. Then add the format for each column using the add\_format function.
5. Then we copy the head part of the old excel file to the new file using write\_head

function.

1. Copy the values of the column name “NAME” from the old excel file to the new file.
2. Copy the all column name from the old file using the write\_title function.
3. Then write the calculated and collected data from the pdf using write\_body function.

}

Write\_title(number of columns and the column names in a list){

1. Is the title of all columns has been written
2. If no

a.Write name of the column

1. Call write\_questions function

}

Write\_body(pass the names of the column title and the calculated data){

1. Create a loop based on the number of rows
2. write all the data in each column from the existing data.

}

Write\_head(pass the number of rows){

1. Create a loop based on the number of rows
2. Copy the data from old file to the new one

}

Write\_questions(pass the dictionary of questions and the columns details and the worksheet as a file){

1. Create a new list
2. Take data of all questions and put them in the list
3. Create all the worksheet formulas for the sub questions and different marks
   1. Create each column for sub questions.
   2. So the no one can give more marks then the full marks
   3. Only take the best marks from the sub questions if more than required questions are answered .
4. Return the list

}

getSubQuestion(pass all the questions and their sub questions){

1. Check if the summation of the parts of a sub question mark are equal to the full marks or not .
2. If not then change the part marks to the calculated marks using division.
3. If yes, go to the next question.
4. Repeat step 1 till there are no questions

}

**Code**

**Program 1:**

import PyPDF2

import sys, re, os, io

import excel\_writer

def check\_parts(questions):

for q in questions:

parts = questions[q]['parts']

partmarks = questions[q]['partmarks']

if len(parts) != len(partmarks):

partmarks = "".join(partmarks)

if partmarks.count('+') + 1 == len(parts):

partmarks = partmarks.split('+')

questions[q]['partmarks'] = partmarks

def parseQuestion(qfile):

with open(qfile, 'rb') as pdf:

pdfReader = PyPDF2.PdfFileReader(pdf)

q = 0

questions = {}

for pageNo in range(pdfReader.numPages):

txt = pdfReader.getPage(pageNo).extractText()

txt = re.split('\n\s\n\s\n', txt)

for field in txt:

field = field.replace('\n', '')

field.strip()

match = re.search('(\d+)\.\s\*Answer any.+(\d+×\d+\s\*=\s\*\d+)\s\*\(', field)

if match:

q = match.group(1)

pms = match.group(2)

else:

match = re.search('\s\*(\d+)\.', field)

if match:

q = match.group(1)

cbcs = re.search('^\s\*(CBCS.\*)\s\*\d+\.', field)

if cbcs:

field = re.sub(cbcs.group(1), '', field)

if (q != 0 and 'CBCS' not in field):

#print(field)

items = re.findall('\(([a-h])\)', field)

question = questions.get(q, {})

parts = question.get('parts', [])

for item in items:

parts.append(item)

question['parts'] = parts

questions[q] = question

partmarks = questions[q].get('partmarks', [])

if (len(pms) == 0):

for match in re.finditer('\s\*(\S+)\s\*(\(|$)', field):

if bool(re.search('\d(?!\.)', match.group(1))):

if not match.group(1).isnumeric() or int(match.group(1)) <= 10:

partmarks.append(match.group(1))

else:

partmarks.append(pms)

pms = []

questions[q]['partmarks'] = partmarks

check\_parts(questions)

return questions

questions = parseQuestion(sys.argv[1])

excel\_writer.write\_xl(sys.argv[2], questions)

**Program 2:**

import pandas as pd

import numpy as np

import os

def column(c):

q = c // 26

r = c % 26

if q > 0:

col = chr(65 + q - 1)+chr(65 + r)

else:

col = chr(65 + r)

return col

def getColRange(start, end):

col1 = column(start)

col2 = column(end)

return col1 + ':' + col2

def getCellRange(start\_row, start\_col, end\_row, end\_col):

return column(start\_col) + str(start\_row + 1) + ':' + column(end\_col) + str(end\_row + 1)

def getPermittedValues(marks):

return [x for x in np.arange(0, float(marks)+0.5, 0.5)]

def getSubQuestion(pm):

info = pm.split('×')

parts = int(info[1].split('=')[0])

total = int(info[1].split('=')[1])

if total / parts != info[0]:

marks = total / parts

else:

marks = info[0]

return (parts, marks)

def addDataValidation(ws, start\_row, start\_col, end\_row, end\_col, fm):

cell\_range = getCellRange(start\_row, start\_col, end\_row, end\_col)

validation = {'validate': 'list', 'source': getPermittedValues(fm)}

ws.data\_validation(cell\_range, validation)

def write\_questions(ws, r, c, questions, nr\_of\_students, col\_fmt):

qCols = []

r1, c1 = r, c

for q in questions.keys():

s = c

if len(questions[q]['parts']) > len(questions[q]['partmarks']):

(ans, marks) = getSubQuestion(questions[q]['partmarks'][0])

for p in questions[q]['parts']:

ws.write(r, c, q+'.'+p)

addDataValidation(ws, r + 1, c, r + nr\_of\_students, c, marks)

c += 1

ws.set\_column(getColRange(s, c - 1), 5, None, {'level' : 1, 'hidden': True})

for row in range(1, nr\_of\_students + 1):

cell\_range = getCellRange(r + row, s, r + row, c - 1)

formula = "=iferror(sum(large({},{})), sum({}))".format(

cell\_range, '{'+','.join([str(x) for x in range(1, ans+1)])+'}', cell\_range)

ws.write\_formula(r+row, c, formula, col\_fmt)

else:

for i in range(len(questions[q]['partmarks'])):

pm = questions[q]['partmarks'][i].split('+')

p = ''

if i < len(questions[q]['parts']):

p = questions[q]['parts'][i]

if len(pm) > 1:

for n in range(len(pm)):

ws.write(r, c, q+'.'+p+'.'+str(n+1))

addDataValidation(ws, r + 1, c, r + nr\_of\_students, c, pm[n])

c += 1

else:

if '×' in questions[q]['partmarks'][i]:

(subparts, submarks) = getSubQuestion(questions[q]['partmarks'][i])

for k in range(subparts):

ws.write(r, c, q+'.'+p+'.'+str(k+1))

addDataValidation(ws, r + 1, c, r + nr\_of\_students, c, submarks)

c += 1

else:

ws.write(r, c, q+'.'+p)

addDataValidation(ws, r + 1, c, r + nr\_of\_students, c, pm[0])

c += 1

ws.set\_column(getColRange(s, c - 1), 5, None, {'level' : 1, 'hidden': True})

for row in range(1, nr\_of\_students + 1):

cell\_range = getCellRange(r + row, s, r + row, c - 1)

ws.write\_formula(r+row, c, "=sum({})".format(cell\_range), col\_fmt)

ws.write(r, c, 'Q'+q, col\_fmt)

qCols.append(column(c))

c += 1

return qCols

def write\_head(ws, rows):

lines = 1

for r in range(len(rows)):

for c in range(len(rows[r])):

if pd.notnull(rows[r][c]):

ws.write(r, c, rows[r][c])

else:

ws.write(r, c, '')

lines += 1

return lines

def write\_body(ws, rows, nr\_of\_students, start, nameCol, nam\_format, col\_format, qCols):

for r in range(len(rows)):

for c in range(len(rows[r])):

(width, fmt) = (15, col\_format) if c != nameCol else (45, nam\_format)

if pd.notnull(rows[r][c]):

ws.write(start+r, c, rows[r][c], fmt)

else:

ws.write\_formula(start + r, c,

"=ceiling(sum({}), 1)".format(",".join([x + str(start+r+1) for x in qCols])),

fmt)

ws.set\_column(c, c, width)

def write\_title(ws, cols, nStudents, row, col\_fmt, questions):

for col in range(len(cols)):

ws.write(row, col, cols[col], col\_fmt)

return write\_questions(ws, row, len(cols), questions, nStudents, col\_fmt)

def write\_xl(template, questions):

print(questions)

filename = os.path.basename(template).split('.')[0] + "\_paperchecking.xlsx"

infile = pd.ExcelFile(template)

head = pd.read\_excel(infile, header=None, nrows=5, usecols = [0])

title = pd.read\_excel(infile, skiprows=6, nrows=1)

students = pd.read\_excel(infile, header=None, dtype=np.str, skiprows=7)

nr\_of\_students = len(students.index)

writer = pd.ExcelWriter(filename, engine = "xlsxwriter")

wb = writer.book

col\_fmt = wb.add\_format({'num\_format' : '0.0;;', 'align' : 'center', 'border' : 1})

nam\_fmt = wb.add\_format({'align' : 'left', 'border' : 1})

ws = wb.add\_worksheet()

lines = write\_head(ws, head.values.tolist())

nameCol = title.columns.get\_loc('NAME')

qCols = write\_title(ws, title.columns.values.tolist(), nr\_of\_students, lines, col\_fmt, questions)

write\_body(ws, students.values.tolist(), nr\_of\_students,

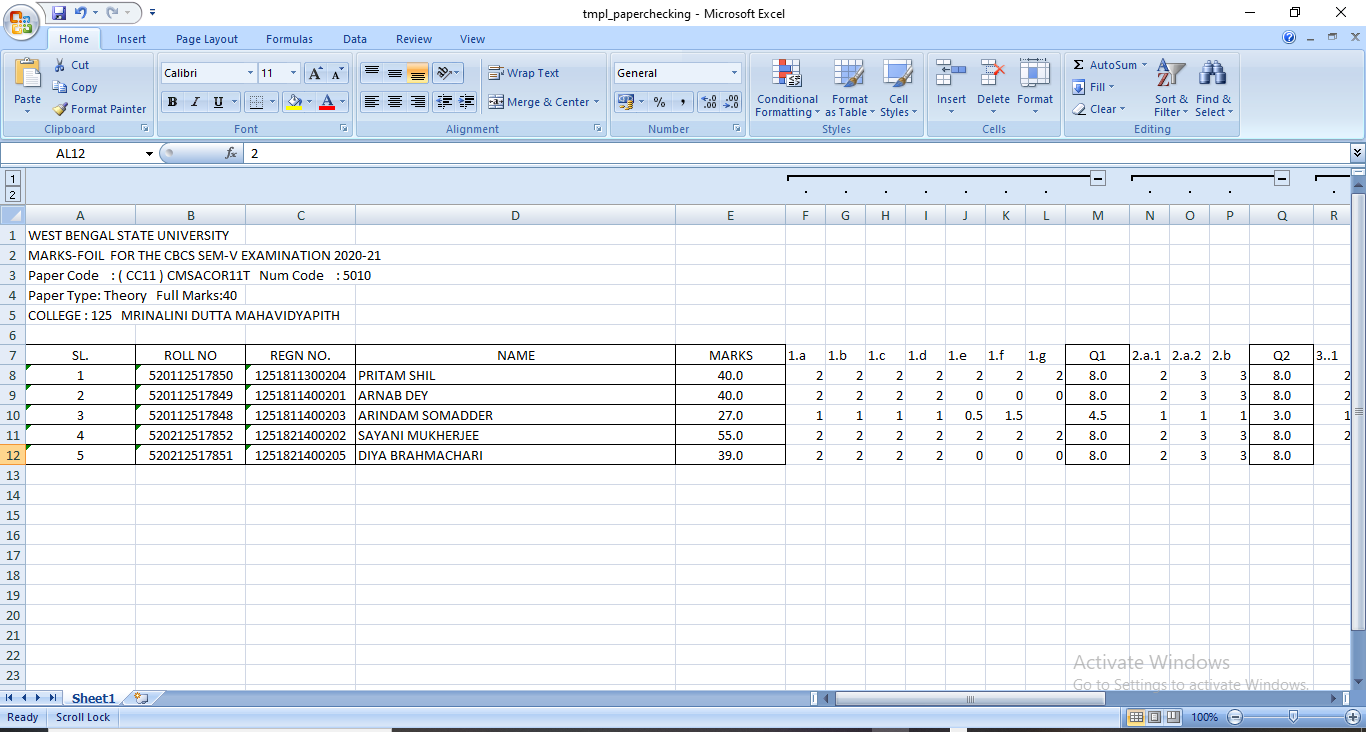
lines+1, nameCol, nam\_fmt, col\_fmt, qCols)

writer.save()

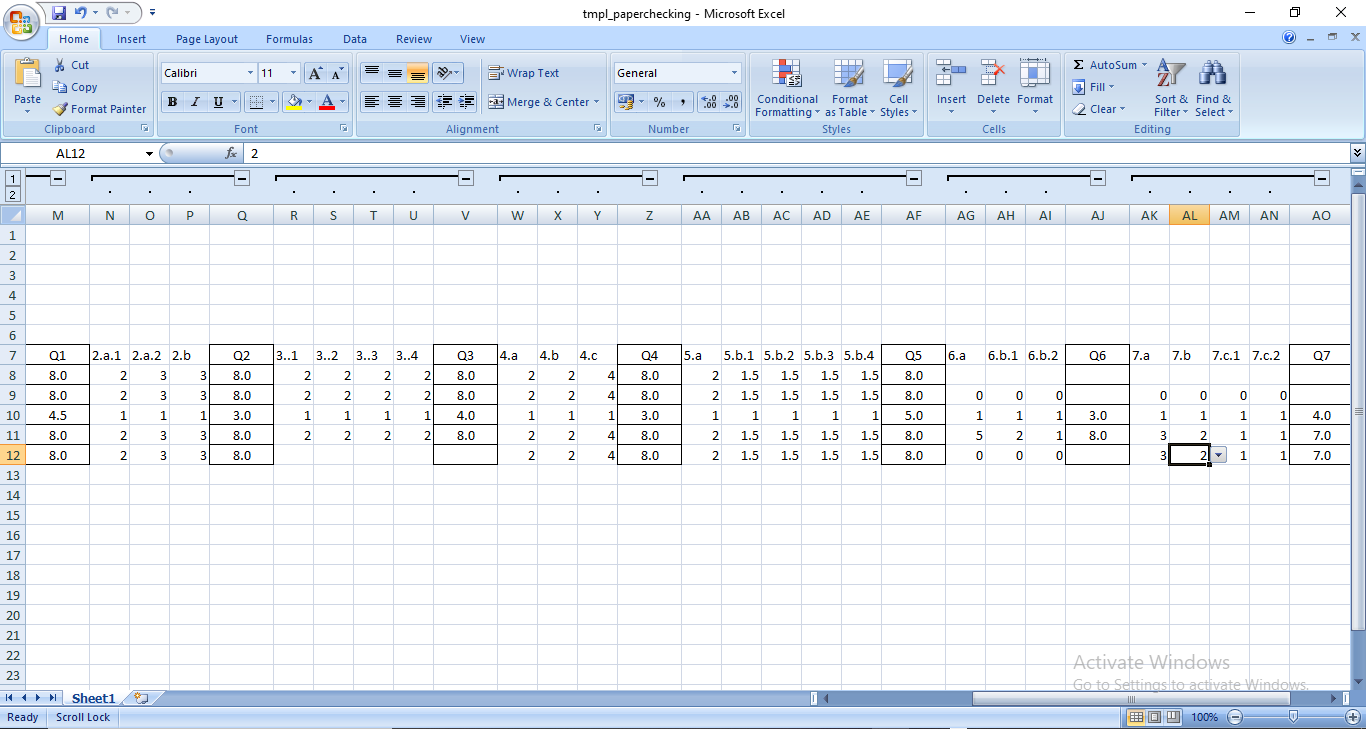
**Run:**

To run our program we need to pass the pdf file and the excel file with the command line. for example : 

**Output:**

****

Output1



Output2

**Discussion**

Here our main objective was to keep the necessary data from the pdf and translate it into a string. Our first obstacle was to read the pdf file and translate it into a string and we were able to achieve it with the help of Pypdf2 module in python.

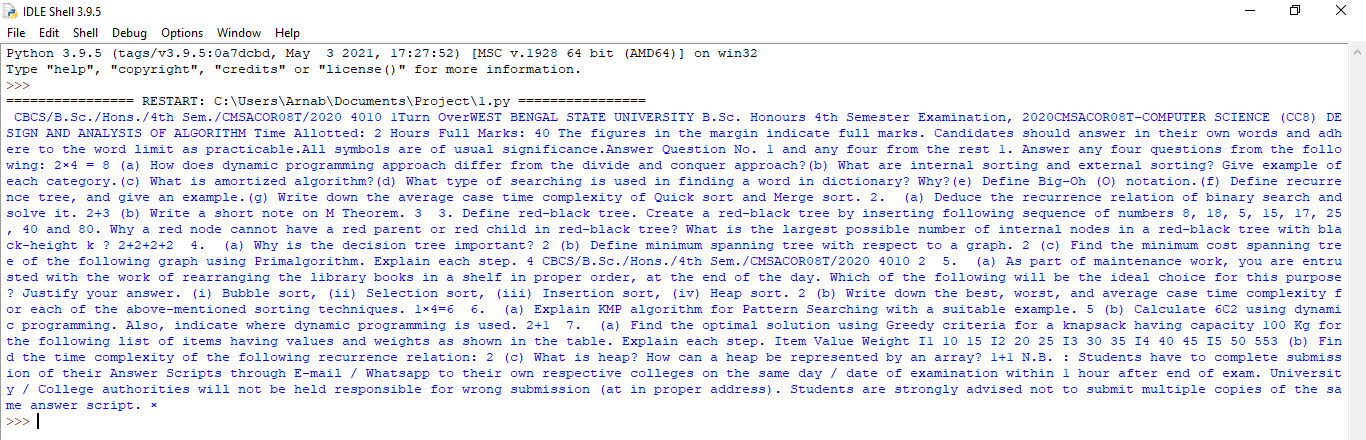


Fig.(i)

The output of our first solution is given in Fig.(i). The only problem in this solution is that needless data is also present in the string. We need to clean this data and create a list were important data is present. We have been able to achieve it via regular language.

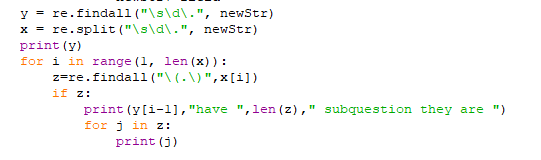


Fig.(ii)

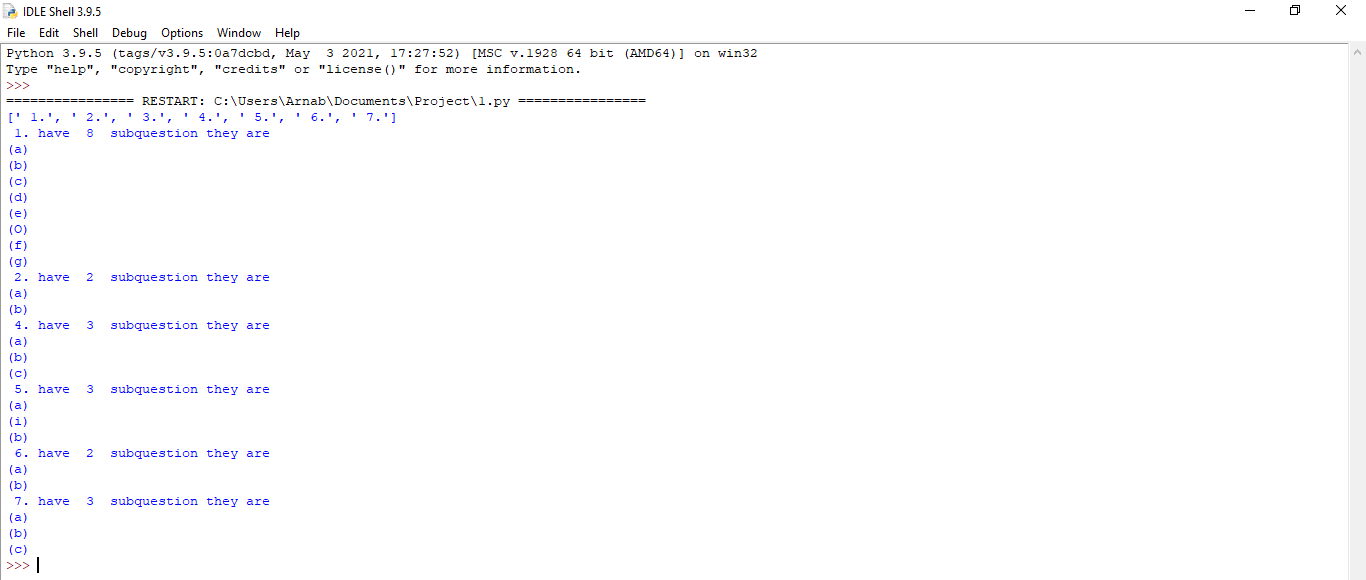


Fig.(iii)

The output of our second solution is given in Fig.(iii). The only problem in this solution is that everything which is in “(single character)”(for eg. (c),(a),(1),etc.) format is considered as a sub-question. Also keeping the data in a dictionary is more beneficial than keeping in a list.

It is more efficient to use a dictionary for lookup of elements because it takes less time to traverse in the dictionary than a list.

Data is fetched by giving a key element in a dictionary than giving the index value in a list.

Here the question number is the key element and its sub-questions and their respective part-marks are stored corresponding to the key element.

Now, the sub-questions and questions have been extracted from the string, marks are needed to be extracted and to be kept in a dictionary.

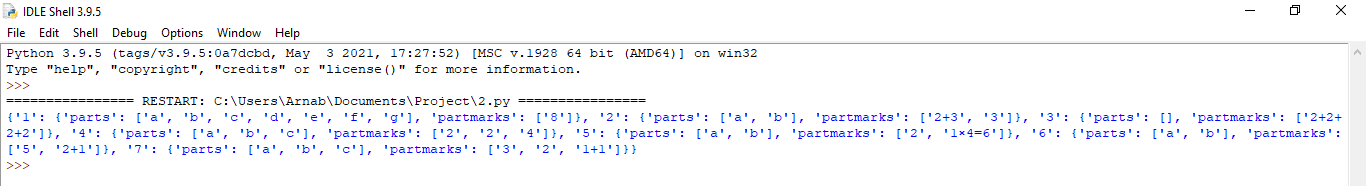


Fig.(iv)

The output of our third solution is given in Fig.(iv). Now we have the necessary data. We are now going to translate this data into an excel file.

An excel file has been created by the name tmpl in the same directory as the program files as shown in Fig.(v).

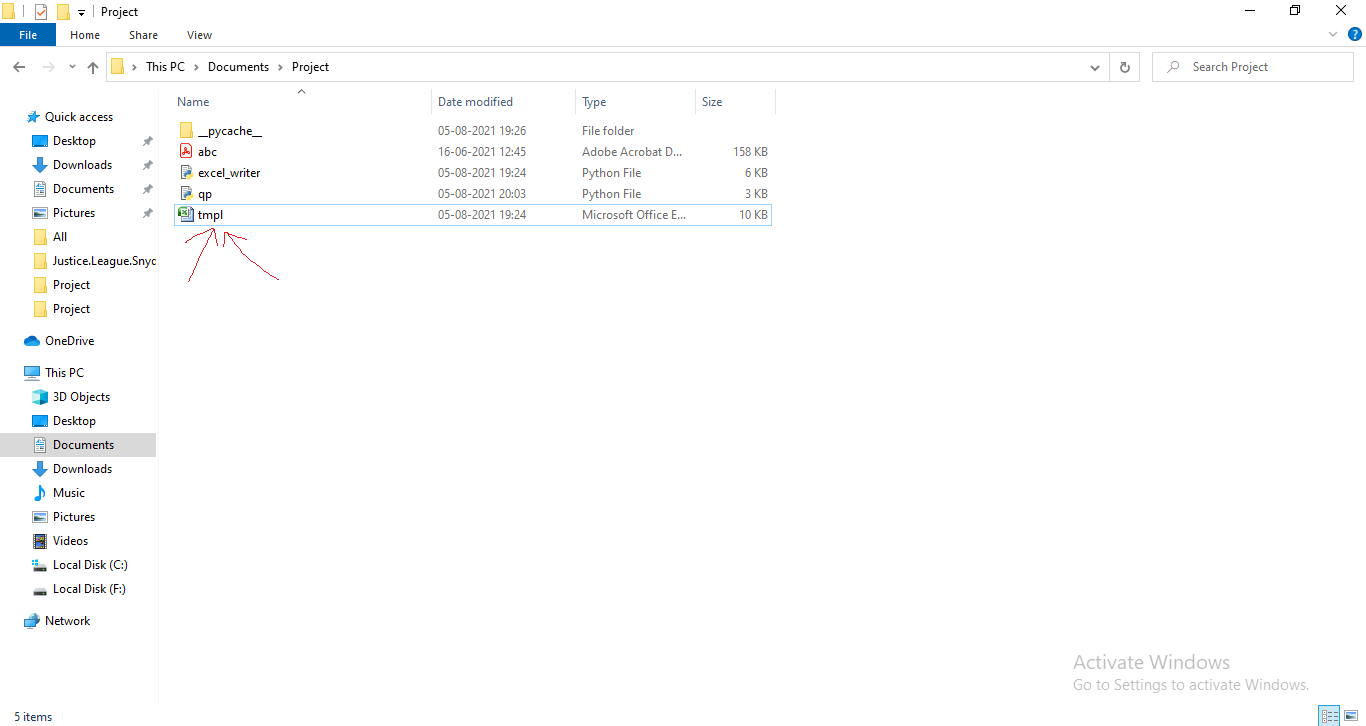


Fig.(v)

The tmpl file looks like this as shown in Fig.(vi).

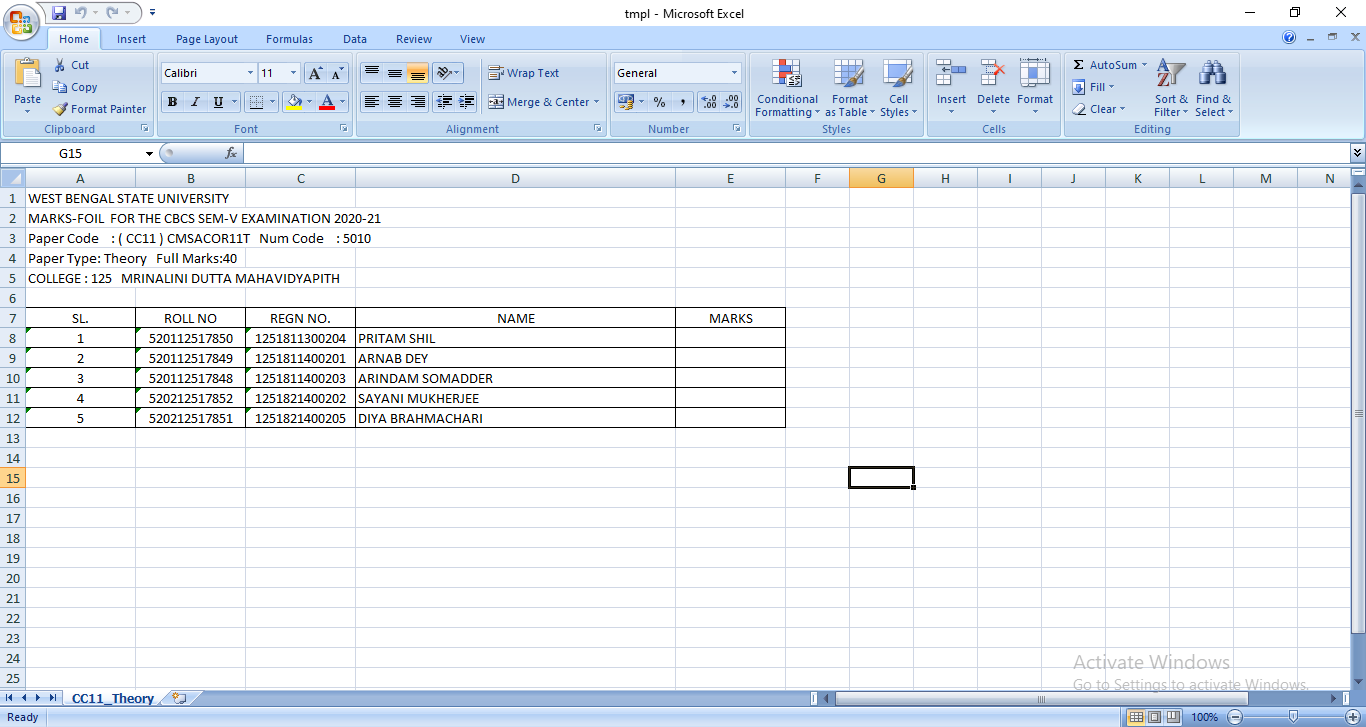


Fig.(vi)

Now, in the second program named excel\_writer.py, the contents of the dictionary as shown in Fig.(iv) is passed as an input and this program is to be kept in the same directory as the main program file. After executing the second program, a new excel file has been created with the name tmpl\_paperchecking.xlsx as shown in Fig.(vii).

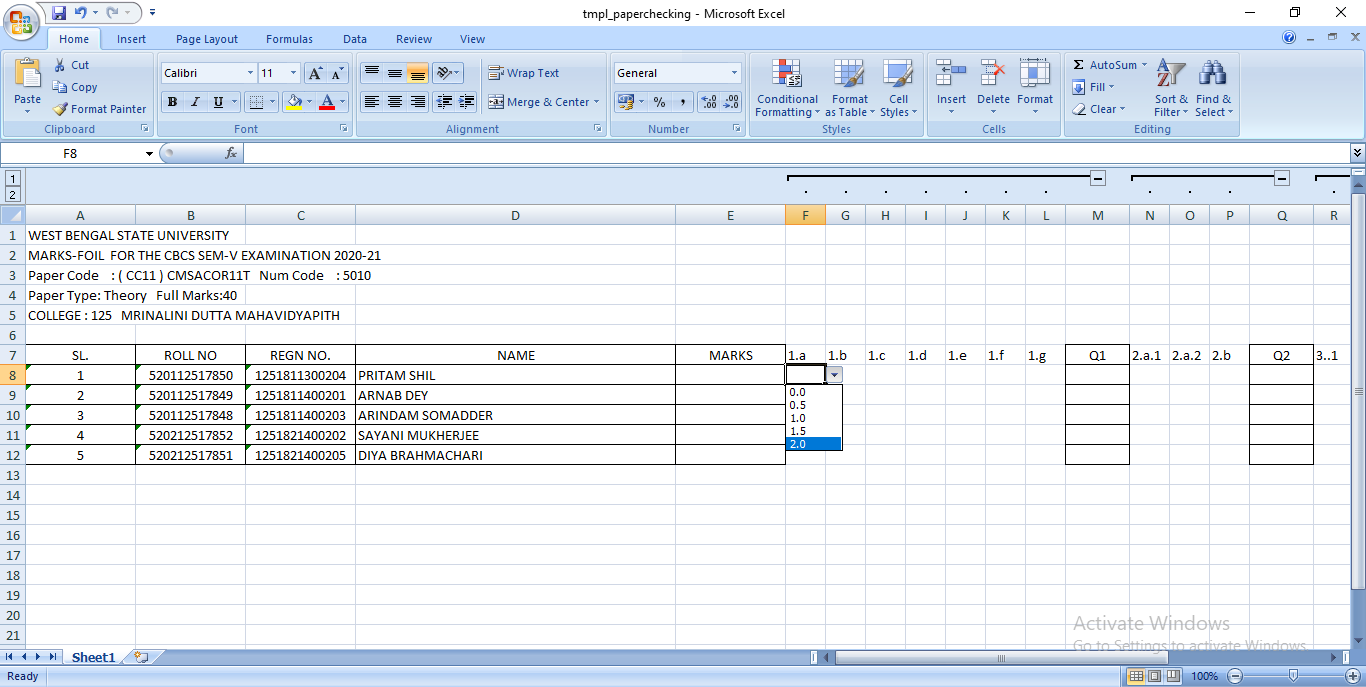


Fig.(vii)

Now the program has executed and the user has to input the marks in the tmpl\_paperchecking.xlsx and the excel file will now handle the rest of the operation.

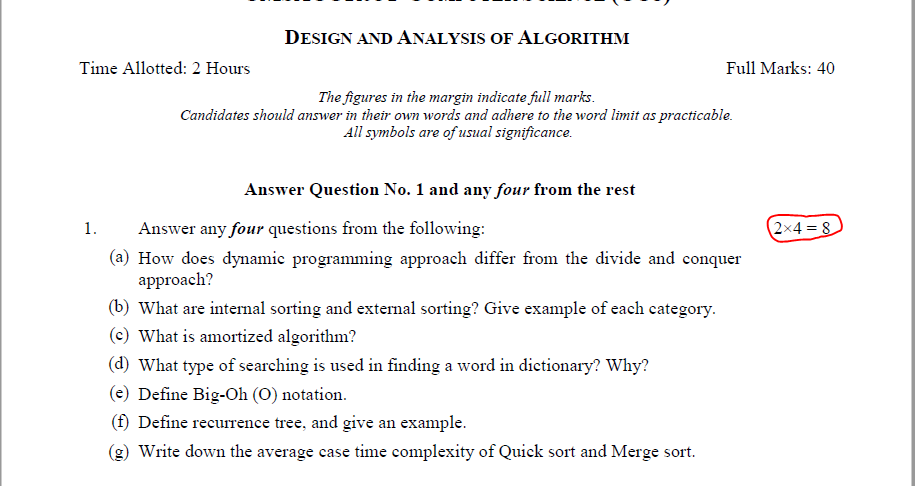


Fig.(viii)

As we can see in Fig.(vii), there are 5 possible inputs allowed, 0,0.5,1,1.5,2 because the total marks each sub-question can have is 2, so chances of mistakes are less. The user cannot give more marks than the prescribed value.

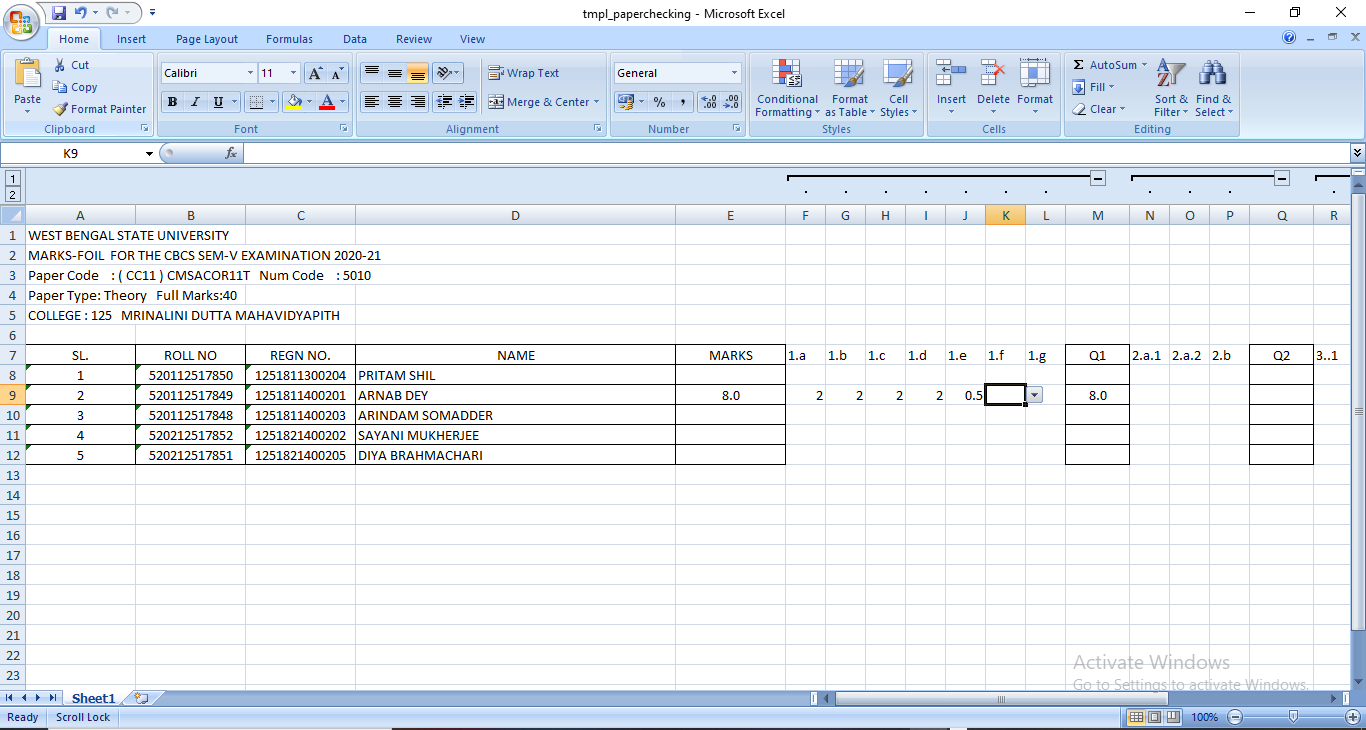


Fig.(ix)

Here, as we can see in Fig.(viii), only four sub-questions are to be answered as the marking format is given as “2x4=8” so, here the program deciphers “2” as the maximum marks for each sub-question, “4” as the maximum number of marks that can be given as an input and “8” as the total marks of that question. Now, the user has entered marks for five sub-questions as seen in Fig.(ix), the program has considered only four maximum marks and added them as the total marks for that question as well as the student’s total marks and miscalculation is avoided.

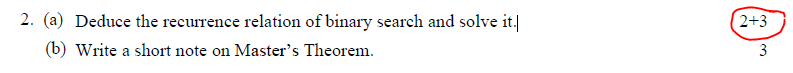


Fig.(x)

In Fig.(x) we can see that question number 2 has only two sub-questions but in Fig.(xi) there are three sub-questions, because in the sub-question the marking format is given as “2+3”, so the program considers it “2” as total number of sub-questions to be created and “3” as the maximum marks to be allowed as an input in those sub-questions.

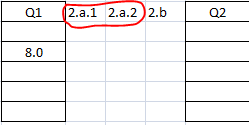
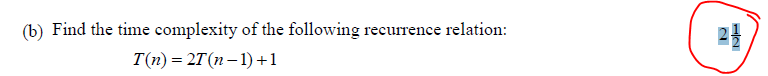


Fig.(xi)

**Conclusion**

Our main objective has been achieved but there are still some areas where improvement will be appreciated like, this program will only work for WBSU and similar types of question paper.



As we can see in the above picture, we are unable to convert this into a string in the first process in Fig.(i), only 2 is passing through the filter and not 21/2.