1. Use of pyPDF2
2. The sequence is:
   1. **text\_extraction\_pypdf2.py**
   2. **fresh\_extract\_all\_questions.py –** this module extracts all the questions and also all the answer options and creates the dictionary called {file name}\_answer\_options\_extracted\_dictionary
   3. **fresh\_extract\_answer\_options.py (**this extracts the question and answer parts separately) (should be renamed to parts extraction) creates the dictionary called {file name}\_answer\_all\_parts\_extracted\_dictionary
   4. **fresh\_extract\_question\_types.py** - it identifies only r\_and\_a\_type, creates the dictionary called {file name}\_answer\_all\_parts\_r\_and\_a\_extracted\_dictionary
   5. **identify\_list\_type.py -** it identifies the list type questions (both types) and creates the dictionaries called - disha\_26\_2020\_upsc\_pre\_gs1\_history\_ancient\_que\_answer\_all\_parts\_r\_and\_a\_extracted\_dictionary\_list\_type\_questions\_question\_part\_dictionary and disha\_26\_2020\_upsc\_pre\_gs1\_history\_ancient\_que\_answer\_all\_parts\_r\_and\_a\_extracted\_dictionary\_list\_type\_answers\_answer\_part\_dictionary
3. **text\_extraction\_pypdf2.py** which running successfully as on 10 Apr, 2024 is:

import PyPDF2  
import os  
from pathlib import Path  
  
  
def extract\_text\_from\_pdf(pdf\_path):  
 with open(pdf\_path, 'rb') as file:  
 reader = PyPDF2.PdfReader(file)  
 text = ''  
 for page\_number, page in enumerate(reader.pages, start=1):  
 page\_text = page.extract\_text()  
 if not page\_text:  
 print(f"Warning: No text found on page {page\_number}")  
 text += page\_text + '\n'  
 return text  
  
  
def create\_directories\_and\_extract\_text(pdf\_file\_name):  
 input\_path = Path(r'C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Input')  
 base\_path = Path(r'C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Area')  
  
 parts = pdf\_file\_name.split(' ')  
 print("Parts of the file name:")  
 for i, part in enumerate(parts):  
 print(f"Part {i + 1}: {part}")  
  
 # Let's use the fifth and sixth parts for the subfolder name  
 subfolder\_name = parts[4] + ' ' + parts[5]  
 subfolder\_path = base\_path / subfolder\_name.lower()  
 subfolder\_path.mkdir(parents=True, exist\_ok=True)  
  
 # Create the text file name without the "Pages X - Y" part  
 text\_file\_name = '\_'.join(parts[:7]).lower().replace(' ', '\_') + '.txt'  
 text\_file\_path = subfolder\_path / text\_file\_name  
  
 pdf\_path = input\_path / (pdf\_file\_name + '.pdf')  
 pdf\_text = extract\_text\_from\_pdf(pdf\_path)  
  
 with open(text\_file\_path, 'w', encoding='utf-8') as text\_file:  
 text\_file.write(pdf\_text)  
  
 print(f"Text extracted and saved to {text\_file\_path}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 pdf\_file\_name = input("Enter the name of the PDF file (without the .pdf extension): ")  
 create\_directories\_and\_extract\_text(pdf\_file\_name)

1. **fresh\_extract\_all\_questions.py** running successfully as on mid-day of 19 Apr, 2024 is:

from pathlib import Path  
import re  
import pandas as pd  
import importlib.util  
import os  
from openpyxl import load\_workbook  
from openpyxl.styles import Font  
from datetime import datetime  
from bs4 import BeautifulSoup  
import argparse  
from fresh\_text\_cleaner import clean\_and\_correct\_pre\_process\_text, replace\_undesirable\_characters  
import glob  
  
# refined\_questions = {}  
questions\_dict = {}  
def get\_refined\_files(directory):  
 *"""Retrieve a list of all files ending with '\_first\_refined.txt' in the specified directory."""* pattern = os.path.join(directory, '\*\_first\_refined.txt')  
 return glob.glob(pattern)  
  
  
def combine\_refined\_questions(output\_dir):  
 *"""Combine all refined question files into a single file."""* refined\_files = get\_refined\_files(output\_dir)  
 combined\_file\_path = os.path.join(output\_dir, "combined\_refined\_questions.txt")  
  
 with open(combined\_file\_path, 'w', encoding='utf-8') as combined\_file:  
 for file\_path in refined\_files:  
 with open(file\_path, 'r', encoding='utf-8') as file:  
 combined\_file.write(file.read())  
 # combined\_file.write("\n---\n") # Add a separator after each file's content  
  
 print(f"All refined questions have been combined and written to {combined\_file\_path}")  
  
  
"""def clean\_and\_preprocess\_text(text, file\_path):  
 # This is a placeholder function to handle text cleaning and preprocessing  
 # Implement or import your actual cleaning functions here  
 return text # Return cleaned text for now"""  
  
def clean\_file(input\_file\_path):  
 cleaned\_file\_path = input\_file\_path.parent / (input\_file\_path.stem + '\_cleaned.txt')  
 with open(input\_file\_path, 'r', encoding='utf-8') as file:  
 text = file.read()  
 cleaned\_text = clean\_and\_correct\_pre\_process\_text(text, input\_file\_path)  
 with open(cleaned\_file\_path, 'w', encoding='utf-8') as cleaned\_file:  
 cleaned\_file.write(cleaned\_text)  
 print(f"Cleaned text file '{cleaned\_file\_path.name}' created in '{cleaned\_file\_path.parent}'. Press Enter to continue with removing undesirable characters.")  
 input()  
 return cleaned\_file\_path  
  
  
def write\_dictionary(data\_dict, file\_path, dict\_name):  
 with open(file\_path, 'w', encoding='utf-8') as file:  
 # Write the dictionary with a specific name that matches the expected attribute  
 file.write(f'{dict\_name} = ' + repr(data\_dict))  
 print(f"Dictionary has been written to {file\_path}")  
  
  
def load\_replacement\_dictionary(file\_path):  
 with open(file\_path, 'r', encoding='utf-8') as file:  
 dictionary = eval(file.read())  
 return dictionary  
  
def extract\_question\_number(questions):  
 question\_numbers\_first\_round = []  
 for question in questions:  
 # Extract the number from the start of the question  
 match = re.match(r"^(\d+)\.", question)  
 if match:  
 # If a number is found, append it to the list of question numbers  
 question\_number = int(match.group(1))  
 question\_numbers\_first\_round.append(question\_number)  
 else:  
 # If no number is found, append None or some placeholder to indicate a missing number  
 question\_numbers\_first\_round.append(None)  
 return question\_numbers\_first\_round  
  
def assess\_extraction(questions\_dict, target):  
 # Extract question numbers directly from the dictionary keys, which represent the question numbers  
 question\_numbers = [int(q\_num) for q\_num in questions\_dict.keys() if q\_num is not None]  
  
 missing\_numbers = sorted(set(range(1, target + 1)) - set(question\_numbers))  
 valid\_numbers = sorted(set(question\_numbers))  
  
 percentage\_valid = len(valid\_numbers) / target \* 100 if target != 0 else 0  
  
 return {  
 "date\_time": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),  
 "target": target,  
 "valid\_question\_numbers": valid\_numbers,  
 "missing\_question\_numbers": missing\_numbers,  
 "percentage\_valid\_questions": percentage\_valid  
 }  
  
def write\_performance\_excel(data, file\_path):  
 # Convert the data to a DataFrame  
 new\_data\_df = pd.DataFrame([data])  
  
 # If the file already exists, read it, append the new data, and save it  
 if os.path.exists(file\_path):  
 try:  
 # Load the existing data  
 existing\_df = pd.read\_excel(file\_path)  
 # Append the new data  
 updated\_df = pd.concat([existing\_df, new\_data\_df], ignore\_index=True)  
 except Exception as e:  
 print(f"Failed to read the existing Excel file. Error: {e}")  
 print("New data will be written to a new file.")  
 updated\_df = new\_data\_df  
 else:  
 updated\_df = new\_data\_df  
  
 # Use XlsxWriter to write the DataFrame to the Excel file  
 with pd.ExcelWriter(file\_path, engine='xlsxwriter') as writer:  
 updated\_df.to\_excel(writer, index=False)  
  
 print(f"Performance data updated and saved in '{file\_path}'. The file is located in the folder: '{os.path.dirname(file\_path)}'.")  
  
def extract\_all\_questions(text, output\_dir, base\_filename):  
  
 # pattern\_for\_all\_questions = r'(\d+\.\s+(?:.\* ?(?:\n\s \*)?) \* ?(?=\(d\)[ ^ a - d] \* \n \*\d +\.\s |\Z))'  
 # pattern\_for\_all\_questions = r'(\d+\.\s+(?:.\*?(?:\n\s\*)?)\*?(?=\(d\)(?:(?!\([abcd]\)).)\*\n\*\d+\.\s|\Z))'  
 pattern\_for\_all\_questions = r'(\d+\.\s+[\s\S]+?(?=\s\*\(\w\)\s+[\s\S]\*?){3,4}\s\*\(\w\)\s+[\s\S]\*?(?=\d+\.\s+|\Z))'  
  
 questions = re.findall(pattern\_for\_all\_questions, text)  
  
 if not questions:  
 print("No questions extracted. Please check the file content and format.")  
 return {}, []  
  
 print(f"Number of questions found initially: {len(questions)}") # Initial count right after findall  
  
 # After extracting questions  
 question\_numbers\_first\_round = extract\_question\_number(questions)  
 # Uncommenting as initialised already at the beginning of the script  
 # questions\_dict = {}  
 output\_file\_path = output\_dir / (base\_filename + '\_extracted\_questions.txt')  
  
 with open(output\_file\_path, 'w', encoding='utf-8') as file:  
 for i, question in enumerate(questions, start=1):  
 question\_num = question\_numbers\_first\_round[i - 1]  
 if question\_num is not None:  
 questions\_dict[question\_num] = {  
 'question\_id': question\_num,  
 'question\_number': question\_num,  
 'question': question.strip(),  
 'source\_file\_hyperlink': str(output\_file\_path)  
 }  
 file.write(f"Question {question\_num}:\n{question.strip()}\n\n---\n\n")  
  
 print(f"Questions have been written to {output\_file\_path}")  
  
 # Write dictionary to a .py file  
 dict\_file\_path = output\_dir / (base\_filename + '\_first\_round\_dict.py')  
 dict\_name = base\_filename + '\_first\_round\_dict' # This should match the filename without '.py'  
 write\_dictionary(questions\_dict, dict\_file\_path, dict\_name)  
  
 # return questions\_dict, question\_numbers # Return both the dictionary and the list of question numbers  
 return questions\_dict, question\_numbers\_first\_round  
  
  
def write\_questions\_to\_file(questions, file\_path):  
 with open(file\_path, 'w', encoding='utf-8') as file:  
 for question in questions:  
 file.write(question.strip() + '\n\n---\n\n')  
 print(f"Questions have been written to {file\_path}")  
  
def first\_refine(questions\_dict, output\_dir):  
 pattern\_for\_breaking\_at\_a = r'(\d+\.\s+(?:[\s\S]\*?\([a-d]\)[\s\S]\*?)(?=\d+\.\s|\Z))'  
 refined\_questions\_dictionary = {}  
 refined\_question\_numbers = [] # Initialize to track which questions are refined  
  
 for question\_id, question\_info in questions\_dict.items():  
 matches = re.findall(pattern\_for\_breaking\_at\_a, question\_info['question'], re.DOTALL)  
 if len(matches) > 1:  
 for index, match in enumerate(matches, start=1):  
 match\_key = f"{question\_id}\_{index}" # Creating a unique key for each match  
 """refined\_questions\_dictionary[match\_key] = {  
 'question\_number': question\_info['question\_number'], # Keeping the original question number  
 'question': f"\n{match.strip()}\n" # Formatting the question with new lines  
 }  
 refined\_question\_numbers.append(match\_key)"""  
 file\_name = f"{match\_key}\_first\_refined.txt"  
 output\_file\_path = os.path.join(output\_dir, file\_name)  
 with open(output\_file\_path, 'w', encoding='utf-8') as file:  
 file.write(f"{match.strip()}\n")  
 file.write("---\n")  
 print(f"Refined questions for {match\_key} have been written to {output\_file\_path}")  
  
 print("\nRefined Question Numbers:")  
 print(refined\_question\_numbers)  
  
 # Writing the refined\_questions dictionary to a file  
 """dict\_file\_path = os.path.join(output\_dir, 'refined\_questions\_dictionary.py')  
 dict\_name = 'refined\_questions\_dictionary'  
 write\_dictionary(refined\_questions\_dictionary, dict\_file\_path, dict\_name)  
 print("Refined questions dictionary has been written to:", dict\_file\_path)  
 print("Press Enter to continue.")  
 input()  
  
 return refined\_questions\_dictionary, refined\_question\_numbers"""  
  
def create\_refined\_questions\_dict(output\_dir):  
 # Path to the combined refined questions file  
 combined\_file\_path = os.path.join(output\_dir, "combined\_refined\_questions.txt")  
 only\_refined\_questions = {}  
 question\_id = 1 # Start question ID from 1  
  
 with open(combined\_file\_path, 'r', encoding='utf-8') as file:  
 content = file.read()  
 # Split the content by the separator "---\n"  
 questions = content.split("---\n")  
  
 # Iterate over each block extracted and treat it as a separate question  
 for question in questions:  
 if question.strip(): # Check if the question block is not empty  
 number\_match = re.search(r'^\s\*(\d+)\.', question)  
 question\_number = number\_match.group(1)  
 question\_number = int(question\_number)  
 only\_refined\_questions[str(question\_id)] = {  
 'question\_id': str(question\_id),  
 'question\_number': question\_number, # Use question\_id as question\_number  
 'question': question.strip(),  
 'source\_file\_hyperlink': "No hyperlink" # Placeholder if no hyperlink is available  
 }  
 question\_id += 1  
  
 # Define the file path and dictionary name for writing the refined\_questions dictionary  
 dict\_file\_path = os.path.join(output\_dir, 'only\_refined\_questions.py')  
 dict\_name = 'only\_refined\_questions' # This should match the dictionary name used in the script  
  
 # Call to write the dictionary to a file  
 write\_dictionary(only\_refined\_questions, dict\_file\_path, dict\_name)  
 print("Only refined questions dictionary has been written to:", dict\_file\_path)  
  
 return only\_refined\_questions  
  
def create\_2nd\_round\_dict(questions\_dict, only\_refined\_questions, output\_dir, base\_filename):  
 # Initialize an empty dictionary for the second round  
 second\_round\_dict = {}  
  
 # Extract question numbers from both dictionaries  
 questions\_numbers = set(q['question\_number'] for q in questions\_dict.values())  
 refined\_question\_numbers = set(q['question\_number'] for q in only\_refined\_questions.values())  
  
 # Create a set of all question numbers from both sets  
 all\_question\_numbers = questions\_numbers.union(refined\_question\_numbers)  
  
 # Iterate over the union of question numbers  
 for question\_number in all\_question\_numbers:  
 original\_info = next((v for k, v in questions\_dict.items() if v['question\_number'] == question\_number), None)  
 refined\_info = next((v for k, v in only\_refined\_questions.items() if v['question\_number'] == question\_number),  
 None)  
  
 if refined\_info:  
 # Prefer refined info if available  
 second\_round\_dict[str(question\_number)] = {  
 'question\_id': question\_number,  
 'question\_number': question\_number,  
 'question': refined\_info['question'].strip(),  
 'source\_file\_hyperlink': refined\_info.get('source\_file\_hyperlink', "No hyperlink")  
 }  
 elif original\_info:  
 # Use original info if refined info is not available  
 second\_round\_dict[str(question\_number)] = {  
 'question\_id': question\_number,  
 'question\_number': question\_number,  
 'question': original\_info['question'].strip(),  
 'source\_file\_hyperlink': original\_info['source\_file\_hyperlink']  
 }  
  
 # Write the updated dictionary to a Python file  
 dict\_file\_path = os.path.join(output\_dir, f"{base\_filename}\_second\_round\_dict.py")  
 with open(dict\_file\_path, 'w', encoding='utf-8') as file:  
 file.write(f"{base\_filename}\_second\_round\_dict = " + repr(second\_round\_dict))  
 print(f"Second round dictionary has been created and written to {dict\_file\_path}")  
  
 # Optionally, create a text file from the updated dictionary with questions appearing sequentially  
 text\_file\_path = os.path.join(output\_dir, f"{base\_filename}\_extracted\_questions\_second\_round.txt")  
 # Inside the create\_2nd\_round\_dict function, when you need to sort the keys:  
 sorted\_keys = sorted(second\_round\_dict.keys(), key=lambda x: str(x)) # Convert keys to strings before sorting  
 with open(text\_file\_path, 'w', encoding='utf-8') as file:  
 for question\_id in sorted\_keys:  
 question\_info = second\_round\_dict[question\_id]  
 file.write(f"Question {question\_id}: {question\_info['question']}\n\n---\n\n")  
 print(f"Extracted questions text file for the second round has been created and written to {text\_file\_path}")  
  
 return second\_round\_dict  
  
def extract\_answer\_options(second\_dict, filename\_prefix, base\_dir\_path):  
 answer\_options\_extracted\_dict\_name = f"{filename\_prefix}answer\_options\_extracted\_dictionary"  
 answer\_options\_extracted\_dictionary = {}  
  
 pattern = r'\((a|b|c|d)\)\s(.\*?)\s\*(?=\([abcd]\)|$)'  
  
 for question\_id, question\_details in second\_dict.items():  
 question\_part = question\_details.get('question', '')  
 answer\_part = question\_details.get('answer', '')  
 source\_file\_hyperlink = question\_details.get('source\_file\_hyperlink', '')  
  
 matches = re.findall(pattern, answer\_part, flags=re.DOTALL)  
 options = {match[0]: match[1].strip() for match in matches}  
  
 answer\_options\_extracted\_dictionary[question\_id] = {  
 "question\_id": question\_id,  
 "question\_number": question\_details.get('question\_number', 'N/A'),  
 "question": question\_part,  
 "answer": answer\_part,  
 "answer\_option\_a": options.get('a', ''),  
 "answer\_option\_b": options.get('b', ''),  
 "answer\_option\_c": options.get('c', ''),  
 "answer\_option\_d": options.get('d', ''),  
 "source\_file\_hyperlink": source\_file\_hyperlink,  
 }  
  
 file\_path = base\_dir\_path / f"{answer\_options\_extracted\_dict\_name}.py"  
  
 # Using existing write\_dictionary function  
 write\_dictionary(answer\_options\_extracted\_dictionary, file\_path, answer\_options\_extracted\_dict\_name)  
  
 print(f"The file '{answer\_options\_extracted\_dict\_name}.py' has been created in the directory {base\_dir\_path}")  
 return answer\_options\_extracted\_dictionary  
  
  
def main(input\_file\_path):  
 base\_dir\_path = input\_file\_path.parent  
 base\_filename = input\_file\_path.stem # Ensure base\_filename is properly defined here  
  
 # Dynamic filename prefix for more flexibility  
 filename\_prefix = f"{base\_filename}\_"  
  
 # Step 1: Read the original text from the input file  
 with open(input\_file\_path, 'r', encoding='utf-8') as file:  
 original\_text = file.read()  
  
 # Step 2: Clean the text and write to a new cleaned file  
 cleaned\_text = clean\_and\_correct\_pre\_process\_text(original\_text, input\_file\_path)  
 cleaned\_file\_path = input\_file\_path.parent / (input\_file\_path.stem + '\_cleaned.txt')  
 with open(cleaned\_file\_path, 'w', encoding='utf-8') as file:  
 file.write(cleaned\_text)  
 print(f"Cleaned text file '{cleaned\_file\_path.name}' created.")  
  
 # Step 3: Replace undesirable characters in the cleaned file and save it as fully cleaned  
 fully\_cleaned\_file\_path = replace\_undesirable\_characters(cleaned\_file\_path)  
 print(f"Fully cleaned text file '{fully\_cleaned\_file\_path.name}' created.")  
  
 # Step 4: Extract questions from the fully cleaned file  
 print(f"Press Enter to continue with the extraction of all questions.")  
 input()  
 with open(fully\_cleaned\_file\_path, 'r', encoding='utf-8') as file:  
 fully\_cleaned\_text = file.read()  
  
 # extract\_all\_questions(fully\_cleaned\_text, input\_file\_path.parent, input\_file\_path.stem)  
 questions\_dict, question\_numbers = extract\_all\_questions(fully\_cleaned\_text, input\_file\_path.parent, input\_file\_path.stem)  
  
 # Step 5: Take user input for target number of questions and proceed with assessment  
 print("Press Enter to Continue for assessing extraction")  
 input()  
 user\_provided\_target = int(input("Enter the number of questions you are targeting to extract: "))  
 # Now you would call the assess\_extraction function and pass the necessary arguments, including the target.  
 performance\_data = assess\_extraction(questions\_dict, user\_provided\_target)  
  
 # You will also need to pass a file path for writing the performance data to an Excel file  
 performance\_file\_path = base\_dir\_path / f"{input\_file\_name}\_extraction\_performance.xlsx"  
 # Ensure we use the same file to append the updated data  
 write\_performance\_excel(performance\_data, performance\_file\_path)  
  
 # New Step 6: First refine process  
 print("Press Enter to continue with the first refinement of questions.")  
 input()  
 # refined\_questions\_dictionary, refined\_question\_numbers = first\_refine(questions\_dict, base\_dir\_path) # Capture refined numbers  
 first\_refine(questions\_dict, base\_dir\_path) # Only call the function, no need to capture returns  
  
 # Combine all first refined questions into a single file  
 print("Press Enter to continue combining all refined questions into a single file.")  
 input()  
 combine\_refined\_questions(base\_dir\_path)  
  
 # New Step 7: Create the only\_refined\_questions dictionary and generate the .py file  
 print("Press Enter to continue creating the only\_refined\_questions dictionary and generate the .py file.")  
 input()  
 only\_refined\_questions = create\_refined\_questions\_dict(base\_dir\_path)  
  
 # New Step 8: Create the second round dictionary and generate the associated text file  
 print("Press Enter to continue creating the second round dictionary and generate the .py file.")  
 input()  
  
 second\_round\_dict = create\_2nd\_round\_dict(questions\_dict, only\_refined\_questions, base\_dir\_path, base\_filename)  
  
 # New Step 9: Assess extraction after first refine  
 print("Press Enter to Continue for assessing extraction after first refine.")  
 input()  
 refined\_performance\_data = assess\_extraction(second\_round\_dict, user\_provided\_target)  
 write\_performance\_excel(refined\_performance\_data, performance\_file\_path)  
  
 print(f"Updated extraction performance has been assessed and saved in the same file.")  
  
 # New Step 10: Extract answer options  
 print("Press Enter to continue extracting answer options.")  
 input()  
 answer\_options\_extracted\_dictionary = extract\_answer\_options(second\_round\_dict, filename\_prefix, base\_dir\_path)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 input\_file\_name = input("Enter the name of the text file (without the .txt extension): ")  
 parts = input\_file\_name.split('\_')  
 subfolder\_name = parts[6] + ' ' + parts[7]  
 base\_dir\_path = Path(r'C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Area') / subfolder\_name.lower()  
  
 # Create subfolder if it doesn't exist  
 base\_dir\_path.mkdir(parents=True, exist\_ok=True)  
  
 print(f"Parts: {parts}")  
 for i, part in enumerate(parts):  
 print(f"Part {i + 1}: {part}")  
 print(f"Subfolder name: {subfolder\_name}")  
 print(f"Base directory path: {base\_dir\_path}")  
  
 input\_file\_path = base\_dir\_path / (input\_file\_name + '.txt')  
 print(f"Input file path: {input\_file\_path}")  
  
 if input\_file\_path.exists():  
 print(f"Input file found in {input\_file\_path.parent}")  
 main(input\_file\_path)  
 else:  
 print(f"File not found: {input\_file\_path}")

1. **fresh\_extract\_parts.py (earlier named as fresh\_extract\_answer\_options.py)** running successfully as on mid-day of 19 Apr, 2024 is:

import re  
from pathlib import Path  
import pandas as pd  
import ast # Import ast module for safer evaluation  
import importlib.util  
from pathlib import Path  
import datetime  
  
  
def load\_dictionary(file\_name, base\_dir\_path):  
 *""" Load a dictionary from a Python file within the specified directory. """* dict\_path = base\_dir\_path / f"{file\_name}.py"  
 if not dict\_path.exists():  
 print(f"File {dict\_path} not found.")  
 return None  
  
 spec = importlib.util.spec\_from\_file\_location(file\_name, dict\_path)  
 module = importlib.util.module\_from\_spec(spec)  
 spec.loader.exec\_module(module)  
  
 # Assuming the dictionary is named the same as file\_name but with an underscore instead of dot  
 dict\_name = f"{file\_name}"  
  
 if hasattr(module, dict\_name):  
 return getattr(module, dict\_name)  
 else:  
 print(f"No dictionary named '{dict\_name}' found in {file\_name}.py")  
 return None  
  
"""def extract\_answer\_options(sample\_text, filename\_prefix, base\_dir\_path):  
 answer\_options\_extracted\_dict\_name = f"{filename\_prefix}answer\_options\_extracted\_dictionary"  
 answer\_options\_extracted\_dictionary = {}  
 options\_temp\_storage = {}  
  
 pattern = r'\((a|b|c|d)\)\s+(.\*?)(?=\s\*\(\w\)\s|\s\*$)'  
 matches = re.findall(pattern, sample\_text, flags=re.DOTALL)  
  
 print(f"Found {len(matches)} matches:")  
 for index, (option, content) in enumerate(matches, start=1):  
 print(f"Part {index}:")  
 print(f"Option: {option}, Content: {content.strip()}")  
 print("---")  
  
 number\_match = re.search(r'^\s\*(\d+)\.', content) # Assuming each option starts with the question number  
 if number\_match:  
 question\_number = int(number\_match.group(1))  
 # Initialize dictionary for new questions  
 if question\_number not in options\_temp\_storage:  
 options\_temp\_storage[question\_number] = {  
 "question\_id": question\_number,  
 "question\_number": question\_number,  
 "answer\_option\_a": "{content.strip()}",  
 "answer\_option\_b": "",  
 "answer\_option\_c": "",  
 "answer\_option\_d": ""  
 }  
 # Assign content to the correct option  
 options\_temp\_storage[question\_number][f"answer\_option\_{option}"] = content.strip()  
  
 # Now populate the main dictionary with the processed data from temp storage  
 for key, value in options\_temp\_storage.items():  
 answer\_options\_extracted\_dictionary[key] = value  
  
 file\_path = base\_dir\_path / f"{answer\_options\_extracted\_dict\_name}.py"  
 # Using existing write\_dictionary function  
 write\_dictionary(answer\_options\_extracted\_dictionary, file\_path, answer\_options\_extracted\_dict\_name)  
  
 print(f"The file '{answer\_options\_extracted\_dict\_name}.py' has been created in the directory {base\_dir\_path}")  
 return answer\_options\_extracted\_dictionary"""  
  
  
def write\_dictionary(data\_dict, file\_path, dict\_name):  
 with open(file\_path, 'w', encoding='utf-8') as file:  
 file.write(f'{dict\_name} = ' + repr(data\_dict))  
 print(f"Dictionary has been written to {file\_path}")  
  
  
def create\_excel\_for\_dict(data\_dict, output\_dir, filename\_prefix):  
 # Convert dictionary to DataFrame  
 df = pd.DataFrame(data\_dict.values())  
  
 # Specify the file path to match the dictionary name  
 file\_path = output\_dir / f"{filename\_prefix}.xlsx"  
  
 # Inform the user about the process starting  
 print("Press Enter to Continue to create the Excel file.")  
 input()  
  
 # Write DataFrame to Excel  
 with pd.ExcelWriter(file\_path, engine='openpyxl') as writer:  
 df.to\_excel(writer, index=False)  
  
 # Inform the user about the creation of the file  
 print(f"Excel file has been created and saved in '{file\_path}'.")  
  
  
def extract\_question\_and\_answer\_parts\_and\_answer\_options(input\_dictionary, dict\_name, base\_dir\_path):  
 output\_dict\_name = '\_'.join(dict\_name.split('\_')[:-3]) + "\_all\_parts\_extracted\_dictionary"  
 output\_dictionary = {}  
  
 options\_pattern = re.compile(r'\((a|b|c|d)\)\s+(.\*?)(?=\s\*\([abcd]\)\s+|\s\*$)', re.DOTALL)  
 year\_pattern = re.compile(r'\[(\d{4}(-\d+)?)\]') # Enhanced pattern to capture years like "2020-1"  
 current\_year = datetime.datetime.now().year  
  
 for key, entry in input\_dictionary.items():  
 question\_text = entry['question']  
 # Extract year and update the dictionary  
 year\_match = year\_pattern.search(question\_text)  
 if year\_match:  
 year = int(year\_match.group(1))  
 # Check if the year is within the valid range  
 if 1900 <= year <= current\_year:  
 exam\_year = year  
 else:  
 exam\_year = "Invalid" # Mark as invalid if outside the range  
  
 question\_text = year\_pattern.sub("", question\_text) # Remove the year from the question text  
 else:  
 exam\_year = "Unknown" # Default value if no year is found  
  
 question\_part = re.split(r'\([abcd]\)', question\_text)[0].strip()  
 question\_part = re.sub(r'^\d+\.\s\*', '', question\_part) # Remove the leading question number  
  
 if question\_part:  
 if question\_part in question\_text:  
 answer\_part = question\_text.split(question\_part)[-1].strip() # Corrected to handle the delimiter issue  
 else:  
 answer\_part = "" # Fallback in case the question part is not properly split  
 else:  
 answer\_part = question\_text # Fallback to use the whole text if question part is empty  
  
 output\_dictionary[key] = entry.copy() # Copy existing data  
 output\_dictionary[key]['question\_part'] = question\_part  
 output\_dictionary[key]['answer\_part'] = answer\_part  
 output\_dictionary[key]['exam\_year'] = exam\_year # Add the exam year to the dictionary  
  
 # Extract and assign options  
 options = options\_pattern.findall(question\_text)  
 for option, content in options:  
 output\_dictionary[key][f'answer\_option\_{option}'] = content.strip()  
  
 file\_path = base\_dir\_path / f"{output\_dict\_name}.py"  
 write\_dictionary(output\_dictionary, file\_path, output\_dict\_name)  
  
 print(f"Updated dictionary is written to {file\_path}")  
 return output\_dictionary, output\_dict\_name  
  
def main():  
 base\_dir\_path = Path(  
 r'C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Python Programming\Trials\Regex Trials\answer\_options')  
 base\_dir\_path.mkdir(parents=True, exist\_ok=True) # Ensure the directory exists  
  
 print("Choose an option:")  
 print("1. Use sample text")  
 print("2. Use dictionary from a file")  
 choice = input("Enter your choice (1 or 2): ")  
  
 processed\_dict = None # Initialize to None to handle scope  
 dict\_name\_used\_for\_excel = None # Initialize a variable to hold the dictionary name  
  
 if choice == '1':  
 sample\_text = """  
 119. With reference to forced labour(Vishti) in India during the  
 Gupta period, which one of the following statements is  
 correct? [2019-I]  
 (a) It was considered a source of income for the State,  
 a sort of tax paid by the people.  
 (b) It was totally absent in the Madhya Pradesh and  
 Kathiawar regions of the Gupta Empire.  
 (c) The forced labourer as entitled to weekly wages.  
 (d) The eldest son of the labourer was sent as the forced  
 labourer.  
 ---  
 120. Building 'Kalyaana Mandapas' was a notable feature in  
 the temple construction in the kingdom of [2019-I]  
 (a) Chalukya (b) Chandela  
 (c) Rashtrakuta  
 ---  
 121. With reference to the cultural history of India, which one  
 of the following is the correct description of the term  
 'paramitas'? [2020-I]  
 (a) The earliest Dharmashastra texts written in  
 aphoristic (sutra) style.  
 (b) Philosophical schools that did not accept the  
 authority of Vedas.  
 (c) Perfections whose attainment led to the Bodhisattva  
 path  
 (d) Powerful merchant guilds of early medieval south  
 India.  
 ---  
 122. With reference to the scholars/litterateurs of ancient  
 India, consider the following statements: [2020-I]  
 1. Panini is associated with Pushyamitra Shunga  
 2. Amarasimha is associated with Harshavardhana  
 3. Kalidasa is associated with Chandra Gupta-II  
 Which of the statements given above is/are correct?  
 (a) 1 and 2 only (b) 2 and 3 only  
 (c) 3 only  
 ---  
 """  
 filename\_prefix = 'test\_'  
 # Assuming the function can handle text directly; otherwise, you need to adjust the implementation  
 processed\_dict = extract\_question\_and\_answer\_parts\_and\_answer\_options(sample\_text, filename\_prefix,  
 base\_dir\_path)  
 elif choice == '2':  
 dict\_name = input("Enter the dictionary file name (without '.py' extension): ")  
 data\_dict = load\_dictionary(dict\_name, base\_dir\_path)  
 if data\_dict:  
 print(f"Dictionary '{dict\_name}' loaded successfully.")  
 # Correct handling of the function output  
 processed\_dict, dict\_name\_used\_for\_excel = extract\_question\_and\_answer\_parts\_and\_answer\_options(data\_dict,  
 dict\_name,  
 base\_dir\_path)  
 else:  
 print("Error processing the dictionary.")  
  
 if processed\_dict:  
 print(f"The Excel file will now be created from the dictionary '{dict\_name\_used\_for\_excel}'.")  
 input("Press Enter to Continue...")  
 # Correctly pass the dictionary and its name to the function  
 create\_excel\_for\_dict(processed\_dict, base\_dir\_path, dict\_name\_used\_for\_excel)  
 else:  
 print("Invalid choice or failed operation. Exiting.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

1. **fresh\_identify\_question\_****types.py** running successfully as on mid-day of 19 Apr, 2024 is:

import re  
from pathlib import Path  
import pandas as pd  
import ast # Import ast module for safer evaluation  
import importlib.util  
from pathlib import Path  
  
def load\_dictionary(file\_name, base\_dir\_path):  
 *""" Load a dictionary from a Python file within the specified directory. """* dict\_path = base\_dir\_path / f"{file\_name}.py"  
 if not dict\_path.exists():  
 print(f"File {dict\_path} not found.")  
 return None  
  
 spec = importlib.util.spec\_from\_file\_location(file\_name, dict\_path)  
 module = importlib.util.module\_from\_spec(spec)  
 spec.loader.exec\_module(module)  
  
 # Assuming the dictionary is named the same as file\_name but with an underscore instead of dot  
 dict\_name = f"{file\_name}"  
  
 if hasattr(module, dict\_name):  
 return getattr(module, dict\_name)  
 else:  
 print(f"No dictionary named '{dict\_name}' found in {file\_name}.py")  
 return None  
  
  
def write\_dictionary(data\_dict, file\_path, dict\_name):  
 with open(file\_path, 'w', encoding='utf-8') as file:  
 file.write(f'{dict\_name} = ' + repr(data\_dict))  
 print(f"Dictionary has been written to {file\_path}")  
  
def create\_text\_from\_dictionary(loaded\_dictionary, dictionary\_name, output\_folder\_path):  
 # Construct the output text file name  
 output\_text\_file\_name = f"{dictionary\_name}\_text\_created.txt"  
 output\_text\_file\_path = output\_folder\_path / output\_text\_file\_name  
  
 # Inform the user about the file name and path  
 print(f"The text file will be created as: {output\_text\_file\_name}")  
 print(f"In the folder: {output\_folder\_path}")  
 input("Press Enter to continue...")  
  
 # Create the text file from the dictionary  
 with open(output\_text\_file\_path, 'w', encoding='utf-8') as file:  
 for question\_id, question\_info in loaded\_dictionary.items():  
 file.write(f"Question ID: {question\_id}\n")  
 file.write(f"Question\_number: {question\_info['question\_number']}\n")  
 file.write(f"Question\_part: {question\_info['question\_part']}\n")  
 # commenting out only to get the question\_part  
 # if 'answer' in question\_info:  
 # file.write(f"Answer: {question\_info['answer']}\n")  
 # file.write("\n---\n\n")  
  
 print(f"Text file created: {output\_text\_file\_path}")  
 return output\_text\_file\_name # Return the name of the created text file  
  
def create\_excel\_for\_dict(data\_dict, output\_dir, filename\_prefix):  
 # Convert dictionary to DataFrame  
 df = pd.DataFrame(data\_dict.values())  
  
 # Specify the file path to match the dictionary name  
 file\_path = output\_dir / f"{filename\_prefix}.xlsx"  
  
 # Inform the user about the Excel file creation details before creating it  
 print(f"Excel file will be created with the name '{filename\_prefix}.xlsx' in the directory '{output\_dir}'.")  
 input("Press Enter to Continue to create the Excel file.")  
  
 # Write DataFrame to Excel  
 with pd.ExcelWriter(file\_path, engine='openpyxl') as writer:  
 df.to\_excel(writer, index=False)  
  
 # Confirm the creation of the file  
 print(f"Excel file has been created and saved in '{file\_path}'.")  
  
  
def identify\_r\_and\_a\_type(input\_dictionary, dict\_name, base\_dir\_path):  
 *"""Identify Reason and Assertion type questions."""* # Generate the output dictionary name by removing the last two parts and adding '\_r\_and\_a\_extracted\_dictionary'  
 new\_dict\_name = '\_'.join(dict\_name.split('\_')[:-2]) + "\_r\_and\_a\_extracted\_dictionary"  
 output\_dictionary = {key: {\*\*val, 'question\_type': ''} for key, val in input\_dictionary.items()}  
 r\_and\_a\_questions\_list = [] # Initialize the list to store matching questions  
  
 # Regex pattern to find Reason and Assertion type questions  
 # pattern\_for\_r\_and\_a\_type = re.compile(r'\d+\.\s+Assertion\s\*\([A]\):.\*?Reason\s\*\([R]\):.\*?(?=\d+\.|$)', re.DOTALL)  
 pattern\_for\_r\_and\_a\_type = re.compile(r'Assertion\s\*\([A]\):.\*?Reason\s\*\([R]\):.\*?(?=\Z)', re.DOTALL)  
  
 # Iterate through dictionary items  
 for key, question\_info in input\_dictionary.items():  
 question\_text = question\_info['question\_part'] # Only check the question part  
 matches = pattern\_for\_r\_and\_a\_type.findall(question\_text)  
  
 if matches:  
 output\_dictionary[key]['question\_type'] = 'r\_and\_a\_type' # Update question type for matches  
 # for match in matches:  
 # r\_and\_a\_questions\_list.append(match) # Add to list for output and counting  
 # output\_dictionary[key] = question\_info.copy() # Copy the whole entry if it's a match  
  
 # Print the number and details of matched questions  
 print(f"Found {len(r\_and\_a\_questions\_list)} 'Reason and Assertion' type questions:")  
 for question in r\_and\_a\_questions\_list:  
 print(question)  
  
 # Writing matched questions to a text file  
 text\_file\_path = base\_dir\_path / f"{new\_dict\_name}\_text.txt"  
 with open(text\_file\_path, 'w', encoding='utf-8') as file:  
 for question in r\_and\_a\_questions\_list:  
 file.write(question + '\n\n')  
 print(f"Text file created: {text\_file\_path}")  
  
 # Inform user and ask to continue before writing the dictionary  
 print(f"Dictionary will be created with the name '{new\_dict\_name}.py' in the directory '{base\_dir\_path}'.")  
 input("Press Enter to continue...")  
  
 # Writing matched questions to the dictionary  
 # Write dictionary and inform user  
 dict\_file\_path = base\_dir\_path / f"{new\_dict\_name}.py"  
 write\_dictionary(output\_dictionary, dict\_file\_path, new\_dict\_name)  
 # print(f"Dictionary has been written to {dict\_file\_path}")  
  
 # Inform user and ask to continue before creating the Excel file  
 print(f"Excel file will be created with the name '{new\_dict\_name}.xlsx' in the directory '{base\_dir\_path}'.")  
 input("Press Enter to continue...")  
 create\_excel\_for\_dict(output\_dictionary, base\_dir\_path, new\_dict\_name)  
  
 return output\_dictionary, text\_file\_path  
  
def main():  
 base\_dir\_path = Path(  
 r'C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Python Programming\Trials\Regex Trials\answer\_options')  
 base\_dir\_path.mkdir(parents=True, exist\_ok=True) # Ensure the directory exists  
  
 print("Choose an option:")  
 print("1. Use sample text")  
 print("2. Use dictionary from a file")  
 print("3. Create text file from dictionary") # New option  
 choice = input("Enter your choice (1, 2, or 3): ")  
  
 processed\_dict = None # Initialize to None to handle scope  
 dict\_name\_used\_for\_excel = None # Initialize a variable to hold the dictionary name  
  
 if choice == '1':  
 sample\_text = """  
 119. With reference to forced labour(Vishti) in India during the  
 Gupta period, which one of the following statements is  
 correct? [2019-I]  
 (a) It was considered a source of income for the State,  
 a sort of tax paid by the people.  
 (b) It was totally absent in the Madhya Pradesh and  
 Kathiawar regions of the Gupta Empire.  
 (c) The forced labourer as entitled to weekly wages.  
 (d) The eldest son of the labourer was sent as the forced  
 labourer.  
 ---  
 120. Building 'Kalyaana Mandapas' was a notable feature in  
 the temple construction in the kingdom of [2019-I]  
 (a) Chalukya (b) Chandela  
 (c) Rashtrakuta  
 ---  
 121. With reference to the cultural history of India, which one  
 of the following is the correct description of the term  
 'paramitas'? [2020-I]  
 (a) The earliest Dharmashastra texts written in  
 aphoristic (sutra) style.  
 (b) Philosophical schools that did not accept the  
 authority of Vedas.  
 (c) Perfections whose attainment led to the Bodhisattva  
 path  
 (d) Powerful merchant guilds of early medieval south  
 India.  
 ---  
 122. With reference to the scholars/litterateurs of ancient  
 India, consider the following statements: [2020-I]  
 1. Panini is associated with Pushyamitra Shunga  
 2. Amarasimha is associated with Harshavardhana  
 3. Kalidasa is associated with Chandra Gupta-II  
 Which of the statements given above is/are correct?  
 (a) 1 and 2 only (b) 2 and 3 only  
 (c) 3 only  
 ---  
 """  
 filename\_prefix = 'test\_'  
 processed\_dict, text\_file\_path = identify\_r\_and\_a\_type(sample\_text, filename\_prefix, base\_dir\_path)  
 dict\_name\_used\_for\_excel = filename\_prefix.rstrip('\_') + "\_r\_and\_a\_extracted\_dictionary"  
  
 elif choice == '2':  
 dict\_name = input("Enter the dictionary file name (without '.py' extension): ")  
 data\_dict = load\_dictionary(dict\_name, base\_dir\_path)  
 if data\_dict:  
 print(f"Dictionary '{dict\_name}' loaded successfully.")  
 processed\_dict, text\_file\_path = identify\_r\_and\_a\_type(data\_dict, dict\_name, base\_dir\_path)  
 dict\_name\_used\_for\_excel = '\_'.join(dict\_name.split('\_')[:-2]) + "\_r\_and\_a\_extracted\_dictionary"  
 if processed\_dict:  
 print("Reason and Assertion type questions have been identified and extracted.")  
 else:  
 print("Error loading or processing the dictionary.")  
  
 elif choice == '3': # Assuming you add a third option to create text files  
 dict\_name = input("Enter the dictionary file name (without '.py' extension): ")  
 data\_dict = load\_dictionary(dict\_name, base\_dir\_path)  
 if data\_dict:  
 print(f"Dictionary '{dict\_name}' loaded successfully.")  
 created\_file\_name = create\_text\_from\_dictionary(data\_dict, dict\_name, base\_dir\_path)  
 print(f"Text file '{created\_file\_name}' has been successfully created.")  
 else:  
 print("Error loading the dictionary.")  
  
 if processed\_dict:  
 if dict\_name\_used\_for\_excel:  
 print(f"The Excel file will now be created from the dictionary '{dict\_name\_used\_for\_excel}'.")  
 input("Press Enter to Continue...")  
 create\_excel\_for\_dict(processed\_dict, base\_dir\_path, dict\_name\_used\_for\_excel)  
 else:  
 print("Invalid choice or failed operation. Exiting.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

1. **identify\_list\_type.py** running successfully as on mid-day of 19 Apr, 2024 (including producing the performance excel) is:

import re  
from pathlib import Path  
import pandas as pd  
import ast # Import ast module for safer evaluation  
import importlib.util  
from datetime import datetime # Include this at the top of your script  
  
# Regex patterns  
pattern\_for\_list\_type\_question\_part = re.compile(  
 r'(List[-\s]?I|List[-\s]?II|List[-\s]?1|List[-\s]?2|Codes|following pairs|following statements|pairs|given below)',  
 re.IGNORECASE  
)  
"""pattern\_for\_list\_type\_question\_part = re.compile(  
 r'(?=.\*\b1\b)(?=.\*\b2\b)(?=.\*\b3\b)(List[-\s]?I|List[-\s]?II|List[-\s]?1|List[-\s]?2|Codes|following pairs|following statements|pairs|given below)',  
 re.IGNORECASE  
)"""  
"""pattern\_for\_list\_type\_question\_part = re.compile(  
 r'(?:following\s+(?:pairs|statements))?'  
 r'(?:(?(1)(?=.\*\b[1-3]\b)|))(List[-\s]?I|List[-\s]?II|List[-\s]?1|List[-\s]?2|Codes|given below)',  
 re.IGNORECASE  
)"""  
  
  
# pattern\_for\_list\_type\_answer\_part = re.compile(r'\b(?:1\s\*(?:and|or|,)\s\*2(?:\s\*,\s\*3)?)\b.\*?\b(?:and|or|both|none|either|neither|)\b')  
# pattern\_for\_list\_type\_answer\_part = re.compile(r'\b(?:(?:A|B|C|D)\s\*-\s\*[1-5]\s\*;\s\*){2,4}(?:A|B|C|D)\s\*-\s\*[1-5]\b')  
# pattern\_for\_list\_type\_answer\_part = re.compile(  
# r'(\b(?:1\s\*(?:and|or|,)\s\*2(?:\s\*,\s\*3)?)\b.\*?\b(?:and|or|both|none|either|neither)\b)|'  
# r'(\b(?:(?:A|B|C|D)\s\*-\s\*[1-4]\s\*;\s\*){3}(?:A|B|C|D)\s\*-\s\*[1-4]\b)'  
#)  
#pattern\_for\_list\_type\_answer\_part = re.compile(  
# r'(\b(?:1\s\*(?:and|or|,)?\s\*2\s\*(?:and|or|,)?\s\*(?:3)?\s\*(?:and|or|,)?\b.\*?\b(?:and|or|both|none|either|neither)\b))|'  
# r'(\b(?:(?:A|B|C|D)\s\*-\s\*[1-4]\s\*;?\s\*){3}(?:A|B|C|D)\s\*-\s\*[1-4]\b)'  
#)  
pattern\_for\_list\_type\_answer\_part = re.compile(  
 r'(\b(?:1\s\*(?:and|or|,)?\s\*2\s\*(?:and|or|,)?\s\*(?:3)?\s\*(?:and|or|,)?\b.\*?\b(?:and|or|both|none|either|neither)\b))|'  
 r'(\b(?:(?:A|B|C|D)\s\*-\s\*[1-4]\s\*;?\s\*){3}(?:A|B|C|D)\s\*-\s\*[1-4]\b)|'  
 r'(\b[1-4],\s\*[1-4],\s\*[1-4],\s\*[1-4]\b)'  
)  
  
  
def identify\_r\_and\_a\_type(input\_dictionary, dict\_name, base\_dir\_path):  
 *"""Identify Reason and Assertion type questions."""* # Generate the output dictionary name by removing the last two parts and adding '\_r\_and\_a\_extracted\_dictionary'  
 new\_dict\_name = '\_'.join(dict\_name.split('\_')[:-2]) + "\_r\_and\_a\_extracted\_dictionary"  
 output\_dictionary = {key: {\*\*val, 'question\_type': ''} for key, val in input\_dictionary.items()}  
 r\_and\_a\_questions\_list = [] # Initialize the list to store matching questions  
  
 # Regex pattern to find Reason and Assertion type questions  
 # pattern\_for\_r\_and\_a\_type = re.compile(r'\d+\.\s+Assertion\s\*\([A]\):.\*?Reason\s\*\([R]\):.\*?(?=\d+\.|$)', re.DOTALL)  
 pattern\_for\_r\_and\_a\_type = re.compile(r'Assertion\s\*\([A]\):.\*?Reason\s\*\([R]\):.\*?(?=\Z)', re.DOTALL)  
  
 # Iterate through dictionary items  
 for key, question\_info in input\_dictionary.items():  
 question\_text = question\_info['question\_part'] # Only check the question part  
 matches = pattern\_for\_r\_and\_a\_type.findall(question\_text)  
  
 if matches:  
 output\_dictionary[key]['question\_type'] = 'r\_and\_a\_type' # Update question type for matches  
 # for match in matches:  
 # r\_and\_a\_questions\_list.append(match) # Add to list for output and counting  
 # output\_dictionary[key] = question\_info.copy() # Copy the whole entry if it's a match  
  
 # Print the number and details of matched questions  
 print(f"Found {len(r\_and\_a\_questions\_list)} 'Reason and Assertion' type questions:")  
 for question in r\_and\_a\_questions\_list:  
 print(question)  
  
 # Writing matched questions to a text file  
 text\_file\_path = base\_dir\_path / f"{new\_dict\_name}\_text.txt"  
 with open(text\_file\_path, 'w', encoding='utf-8') as file:  
 for question in r\_and\_a\_questions\_list:  
 file.write(question + '\n\n')  
 print(f"Text file created: {text\_file\_path}")  
  
 # Inform user and ask to continue before writing the dictionary  
 print(f"Dictionary will be created with the name '{new\_dict\_name}.py' in the directory '{base\_dir\_path}'.")  
 input("Press Enter to continue...")  
  
 # Writing matched questions to the dictionary  
 # Write dictionary and inform user  
 dict\_file\_path = base\_dir\_path / f"{new\_dict\_name}.py"  
 write\_dictionary(output\_dictionary, dict\_file\_path, new\_dict\_name)  
 # print(f"Dictionary has been written to {dict\_file\_path}")  
  
 # Inform user and ask to continue before creating the Excel file  
 print(f"Excel file will be created with the name '{new\_dict\_name}.xlsx' in the directory '{base\_dir\_path}'.")  
 input("Press Enter to continue...")  
 create\_excel\_for\_dict(output\_dictionary, base\_dir\_path, new\_dict\_name)  
  
 return output\_dictionary, text\_file\_path  
  
"""def load\_dictionary(file\_name, base\_dir\_path):  
 # Load a dictionary from a Python file within the specified directory.  
 dict\_path = base\_dir\_path / f"{file\_name}.py"  
 if not dict\_path.exists():  
 print(f"File {dict\_path} not found.")  
 return None  
  
 spec = importlib.util.spec\_from\_file\_location(file\_name, dict\_path)  
 module = importlib.util.module\_from\_spec(spec)  
 spec.loader.exec\_module(module)  
  
 # Assuming the dictionary is named the same as file\_name but with an underscore instead of dot  
 dict\_name = f"{file\_name}"  
  
 if hasattr(module, dict\_name):  
 return getattr(module, dict\_name)  
 else:  
 print(f"No dictionary named '{dict\_name}' found in {file\_name}.py")  
 return None"""  
  
def load\_dictionary(file\_name, base\_dir\_path):  
 *""" Load a dictionary from a Python file within the specified directory. """* dict\_path = base\_dir\_path / f"{file\_name}.py"  
 if not dict\_path.exists():  
 print(f"File {dict\_path} not found.")  
 return {} # Return an empty dictionary instead of None  
  
 spec = importlib.util.spec\_from\_file\_location(file\_name, dict\_path)  
 module = importlib.util.module\_from\_spec(spec)  
 spec.loader.exec\_module(module)  
  
 # Ensure the dictionary name does not include the '.py' extension  
 dict\_name = file\_name if not file\_name.endswith('.py') else file\_name[:-3]  
  
 # Attempt to get the dictionary  
 loaded\_dict = getattr(module, dict\_name, {})  
  
 # Check if the dictionary is empty and print a message  
 if not loaded\_dict:  
 print(f"Loaded dictionary named '{dict\_name}' is empty.")  
 else:  
 print(f"Loaded dictionary named '{dict\_name}' successfully.")  
  
 return loaded\_dict  
  
def write\_dictionary(data\_dict, file\_path, dict\_name):  
 with open(file\_path, 'w', encoding='utf-8') as file:  
 file.write(f'{dict\_name} = ' + repr(data\_dict))  
 print(f"Dictionary has been written to {file\_path}")  
  
def create\_text\_from\_dictionary(loaded\_dictionary, dictionary\_name, output\_folder\_path):  
 *"""  
 Create text files from the input dictionary for question and answer parts.  
 """* output\_text\_file\_name\_question\_part = f"{dictionary\_name}\_text\_created\_question\_part.txt"  
 output\_text\_file\_name\_answer\_part = f"{dictionary\_name}\_text\_created\_answer\_part.txt"  
 output\_text\_file\_path\_question\_part = output\_folder\_path / output\_text\_file\_name\_question\_part  
 output\_text\_file\_path\_answer\_part = output\_folder\_path / output\_text\_file\_name\_answer\_part  
  
 # Create the text file for question parts from the dictionary  
 with open(output\_text\_file\_path\_question\_part, 'w', encoding='utf-8') as file\_question\_part:  
 for question\_id, question\_info in loaded\_dictionary.items():  
 file\_question\_part.write(f"Question ID: {question\_id}\n")  
 file\_question\_part.write(f"Question Number: {question\_info['question\_number']}\n")  
 file\_question\_part.write(f"Content: {question\_info['question\_part']}\n---\n\n")  
  
 # Create the text file for answer parts from the dictionary  
 with open(output\_text\_file\_path\_answer\_part, 'w', encoding='utf-8') as file\_answer\_part:  
 for question\_id, question\_info in loaded\_dictionary.items():  
 file\_answer\_part.write(f"Question ID: {question\_id}\n")  
 file\_answer\_part.write(f"Question Number: {question\_info['question\_number']}\n")  
 file\_answer\_part.write(f"Content: {question\_info['answer\_part']}\n")  
 file\_answer\_part.write(f"a) {question\_info.get('answer\_option\_a', 'Not available')}\n")  
 file\_answer\_part.write(f"b) {question\_info.get('answer\_option\_b', 'Not available')}\n")  
 file\_answer\_part.write(f"c) {question\_info.get('answer\_option\_c', 'Not available')}\n")  
 file\_answer\_part.write(f"d) {question\_info.get('answer\_option\_d', 'Not available')}\n---\n\n")  
  
 print(f"Text files created: {output\_text\_file\_path\_question\_part} and {output\_text\_file\_path\_answer\_part}")  
  
 return output\_text\_file\_name\_question\_part, output\_text\_file\_name\_answer\_part  
  
def create\_excel\_for\_dict(data\_dict, output\_dir, filename\_prefix):  
 # Convert dictionary to DataFrame  
 df = pd.DataFrame(data\_dict.values())  
  
 # Specify the file path to match the dictionary name  
 file\_path = output\_dir / f"{filename\_prefix}.xlsx"  
  
 # Inform the user about the Excel file creation details before creating it  
 print(f"Excel file will be created with the name '{filename\_prefix}.xlsx' in the directory '{output\_dir}'.")  
 input("Press Enter to Continue to create the Excel file.")  
  
 # Write DataFrame to Excel  
 with pd.ExcelWriter(file\_path, engine='openpyxl') as writer:  
 df.to\_excel(writer, index=False)  
  
 # Confirm the creation of the file  
 print(f"Excel file has been created and saved in '{file\_path}'.")  
  
def create\_performance\_excel(question\_dict, answer\_dict, base\_dir\_path, original\_dict\_name):  
 try:  
 target = float(input("Please enter the target number for extraction calculation: "))  
 if target == 0: # Prevent division by zero  
 print("Target cannot be zero. Please enter a valid number.")  
 return  
 except ValueError:  
 print("Invalid input. Please enter a numeric value.")  
 return  
  
 # Extract base file name for new Excel file  
 parts = original\_dict\_name.split('\_')  
 new\_file\_name = '\_'.join(parts[:-7]) + "\_list\_type\_identification\_performance"  
 excel\_file\_path = base\_dir\_path / f"{new\_file\_name}.xlsx"  
  
 # Prepare data for DataFrame  
 # Prepare data for DataFrame  
 data = {  
 'sl\_num': range(1, max(len(question\_dict), len(answer\_dict)) + 1),  
 'question\_id\_in\_que': [info['question\_id'] for info in question\_dict.values()],  
 'question\_number\_in\_que': [info['question\_number'] for info in question\_dict.values()],  
 'set\_in\_que': [', '.join(question\_dict.keys())],  
 'question\_id\_in\_ans': [info['question\_id'] for info in answer\_dict.values()],  
 'question\_number\_in\_ans': [info['question\_number'] for info in answer\_dict.values()],  
 'set\_in\_ans': [', '.join(answer\_dict.keys())],  
 '% extraction for list type (Questions)': [len(question\_dict.keys()) / target \* 100],  
 '% extraction for list type (Answers)': [len(answer\_dict.keys()) / target \* 100]  
 }  
  
 # Adjust the length of set\_in\_que and set\_in\_ans to match the maximum length  
 data['set\_in\_que'] \*= len(data['sl\_num'])  
 data['set\_in\_ans'] \*= len(data['sl\_num'])  
 data['% extraction for list type (Questions)'] \*= len(data['sl\_num'])  
 data['% extraction for list type (Answers)'] \*= len(data['sl\_num'])  
  
 # Fill in gaps if one dictionary is larger than the other  
 for key in ['question\_id\_in\_que', 'question\_number\_in\_que', 'question\_id\_in\_ans', 'question\_number\_in\_ans']:  
 while len(data[key]) < max(len(question\_dict), len(answer\_dict)):  
 data[key].append(None)  
  
 # Create DataFrame and write to Excel  
 performance\_df = pd.DataFrame(data)  
  
 # Add a row at the end for the current date and time  
 # Add a row at the end for the current date and time  
 current\_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")  
 performance\_summary = pd.DataFrame({'sl\_num': ["Extraction Performance at " + current\_time]})  
  
 # Use concat to add the new row  
 performance\_df = pd.concat([performance\_df, performance\_summary], ignore\_index=True)  
  
 # Write DataFrame to Excel  
 with pd.ExcelWriter(excel\_file\_path, engine='openpyxl', mode='w') as writer:  
 performance\_df.to\_excel(writer, index=False)  
  
 # Inform user that the Excel file has been created  
 print(f"The {new\_file\_name}.xlsx has been created in the {base\_dir\_path} folder.")  
# Modify your identify\_list\_type function to call create\_performance\_excel  
def identify\_list\_type(data\_dict, pattern, dict\_name, base\_dir\_path, file\_type):  
 matched\_dict = {} # This will store all matches with their original data  
 match\_count = 0  
  
 # Iterate through each item in the dictionary to apply the pattern  
 for question\_id, info in data\_dict.items():  
 if file\_type == "question\_part":  
 current\_text = info['question\_part']  
 else:  
 current\_text = "\n".join([info.get(f'answer\_option\_{chr(97 + i)}', 'Not available') for i in range(4)])  
  
 if pattern.search(current\_text):  
 match\_count += 1  
 # Retrieve the question\_number from the dictionary entry  
 question\_number = info.get('question\_number', 'Not available')  
  
 print(f"Match found in {file\_type} file for Question ID {question\_id} and Question Number {question\_number}:")  
 print(current\_text + "\n---\n")  
 matched\_dict[question\_id] = info  
  
 print(f"Found {match\_count} matches in {file\_type} file.")  
  
 dict\_file\_name = f"{dict\_name}\_list\_type\_questions\_{file\_type}\_dictionary"  
  
 dict\_path = base\_dir\_path / f"{dict\_file\_name}.py"  
 excel\_path = base\_dir\_path / f"{dict\_file\_name}.xlsx"  
  
 # write the basic dictionaries created  
 write\_dictionary(matched\_dict, dict\_path, dict\_file\_name)  
  
 # write the excel files corresponding to the basic dictionaries created  
 df = pd.DataFrame.from\_dict(matched\_dict, orient='index')  
 df.to\_excel(excel\_path, index=False)  
 print(f"Dictionary and Excel file created for {file\_type} at {dict\_path} and {excel\_path}")  
  
 # write the text files corresponding to the basic dictionaries created  
 # Create additional text files after dictionaries and Excels are done  
 question\_text\_file\_path = base\_dir\_path / f"{dict\_name}\_{file\_type}\_from\_question\_parts.txt"  
 answer\_text\_file\_path = base\_dir\_path / f"{dict\_name}\_{file\_type}\_from\_answer\_parts.txt"  
  
 with open(question\_text\_file\_path, 'w', encoding='utf-8') as q\_file, \  
 open(answer\_text\_file\_path, 'w', encoding='utf-8') as a\_file:  
 for id, content in matched\_dict.items():  
 # Write to question parts text file  
 q\_file.write(f"Question ID: {id}\n")  
 q\_file.write(f"Question Number: {content['question\_number']}\n")  
 q\_file.write(f"Content: {content['question\_part']}\n")  
 q\_file.write("---\n\n")  
 # Write to answer parts text file if applicable  
 if 'answer\_part' in content:  
 a\_file.write(f"Question ID: {id}\n")  
 a\_file.write(f"Question Number: {content['question\_number']}\n")  
 a\_file.write(f"Content: {content['answer\_part']}\n")  
 a\_file.write("---\n\n")  
  
 print(f"Text files created at {question\_text\_file\_path} and {answer\_text\_file\_path}")  
  
 # create the performance excel  
 if file\_type == 'answer\_part':  
 question\_dict\_file = f"{dict\_name}\_list\_type\_questions\_question\_part\_dictionary"  
 print(f"the dict\_file\_name is :{question\_dict\_file}")  
 input("Press Enter to continue...")  
 answer\_dict\_file = f"{dict\_name}\_list\_type\_questions\_answer\_part\_dictionary"  
 print(f"the dict\_file\_name is :{answer\_dict\_file}")  
 input("Press Enter to continue...")  
 question\_dict = load\_dictionary(question\_dict\_file, base\_dir\_path)  
 answer\_dict = load\_dictionary(answer\_dict\_file, base\_dir\_path)  
 create\_performance\_excel(question\_dict, answer\_dict, base\_dir\_path, dict\_name)  
  
def main():  
 base\_dir\_path = Path(r'C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Python Programming\Trials\Regex Trials\question types')  
 base\_dir\_path.mkdir(parents=True, exist\_ok=True) # Ensure the directory exists  
  
 print("Choose an option:")  
 print("1. Use sample text")  
 print("2. Use dictionary from a file")  
 print("3. Create text files and extract list type questions from dictionary")  
 choice = input("Enter your choice (1, 2, or 3): ")  
  
 processed\_dict = None # Initialize to None to handle scope  
 dict\_name\_used\_for\_excel = None # Initialize a variable to hold the dictionary name  
  
 if choice == '1':  
 sample\_text = """  
 119. With reference to forced labour(Vishti) in India during the  
 Gupta period, which one of the following statements is  
 correct? [2019-I]  
 (a) It was considered a source of income for the State,  
 a sort of tax paid by the people.  
 (b) It was totally absent in the Madhya Pradesh and  
 Kathiawar regions of the Gupta Empire.  
 (c) The forced labourer as entitled to weekly wages.  
 (d) The eldest son of the labourer was sent as the forced  
 labourer.  
 ---  
 120. Building 'Kalyaana Mandapas' was a notable feature in  
 the temple construction in the kingdom of [2019-I]  
 (a) Chalukya (b) Chandela  
 (c) Rashtrakuta  
 ---  
 121. With reference to the cultural history of India, which one  
 of the following is the correct description of the term  
 'paramitas'? [2020-I]  
 (a) The earliest Dharmashastra texts written in  
 aphoristic (sutra) style.  
 (b) Philosophical schools that did not accept the  
 authority of Vedas.  
 (c) Perfections whose attainment led to the Bodhisattva  
 path  
 (d) Powerful merchant guilds of early medieval south  
 India.  
 ---  
 122. With reference to the scholars/litterateurs of ancient  
 India, consider the following statements: [2020-I]  
 1. Panini is associated with Pushyamitra Shunga  
 2. Amarasimha is associated with Harshavardhana  
 3. Kalidasa is associated with Chandra Gupta-II  
 Which of the statements given above is/are correct?  
 (a) 1 and 2 only (b) 2 and 3 only  
 (c) 3 only  
 ---  
 """  
 filename\_prefix = 'test\_'  
 processed\_dict, text\_file\_path = identify\_r\_and\_a\_type(sample\_text, filename\_prefix, base\_dir\_path)  
 dict\_name\_used\_for\_excel = filename\_prefix.rstrip('\_') + "\_r\_and\_a\_extracted\_dictionary"  
  
 elif choice == '2':  
 dict\_name = input("Enter the dictionary file name (without '.py' extension): ")  
 data\_dict = load\_dictionary(dict\_name, base\_dir\_path)  
 if data\_dict:  
 print(f"Dictionary '{dict\_name}' loaded successfully.")  
 processed\_dict, text\_file\_path = identify\_r\_and\_a\_type(data\_dict, dict\_name, base\_dir\_path)  
 dict\_name\_used\_for\_excel = '\_'.join(dict\_name.split('\_')[:-2]) + "\_r\_and\_a\_extracted\_dictionary"  
 if processed\_dict:  
 print("Reason and Assertion type questions have been identified and extracted.")  
 else:  
 print("Error loading or processing the dictionary.")  
  
 elif choice == '3': # Assuming you add a third option to create text files  
 dict\_name = input("Enter the dictionary file name (without '.py' extension): ")  
 data\_dict = load\_dictionary(dict\_name, base\_dir\_path)  
 if data\_dict:  
 print(f"Dictionary '{dict\_name}' loaded successfully.")  
 created\_file\_name\_question\_part, created\_file\_name\_answer\_part = create\_text\_from\_dictionary(data\_dict,  
 dict\_name,  
 base\_dir\_path)  
 print(  
 f"Text files '{created\_file\_name\_question\_part}' and '{created\_file\_name\_answer\_part}' have been successfully created.")  
  
 # Extract list type from loaded dictionary  
 print("Processing question part for list type questions...")  
 input("Press Enter to continue...")  
 # Convert dictionary question parts into a single string for regex processing  
 identify\_list\_type(data\_dict, pattern\_for\_list\_type\_question\_part, dict\_name, base\_dir\_path, "question\_part")  
  
 print("Processing answer part for list type answers...")  
 input("Press Enter to continue...")  
 # Convert dictionary answer parts into a single string for regex processing  
 identify\_list\_type(data\_dict, pattern\_for\_list\_type\_answer\_part, dict\_name, base\_dir\_path, "answer\_part")  
  
 else:  
 print("Error loading the dictionary.")  
  
 else:  
  
 print("Invalid choice or failed operation. Exiting.")  
 return # Add return here to handle any other unexpected cases  
  
 # Additional logic for creating Excel files if a valid dictionary was processed  
 if processed\_dict and dict\_name\_used\_for\_excel:  
 print(f"The Excel file will now be created from the dictionary '{dict\_name\_used\_for\_excel}'.")  
 input("Press Enter to Continue...")  
 create\_excel\_for\_dict(processed\_dict, base\_dir\_path, dict\_name\_used\_for\_excel)  
 elif choice in ['1', '2', '3']: # Ensure this only triggers if one of the valid choices was selected  
 print("Operation completed successfully.")  
 else:  
 print("No further action required or no data to process.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

1. **fresh\_extract\_question\_types.py** running successfully as on mid-day of 17 Apr, 2024 is:

import re  
from pathlib import Path  
import pandas as pd  
import ast # Import ast module for safer evaluation  
import importlib.util  
from pathlib import Path  
  
def load\_dictionary(file\_name, base\_dir\_path):  
 *""" Load a dictionary from a Python file within the specified directory. """* dict\_path = base\_dir\_path / f"{file\_name}.py"  
 if not dict\_path.exists():  
 print(f"File {dict\_path} not found.")  
 return None  
  
 spec = importlib.util.spec\_from\_file\_location(file\_name, dict\_path)  
 module = importlib.util.module\_from\_spec(spec)  
 spec.loader.exec\_module(module)  
  
 # Assuming the dictionary is named the same as file\_name but with an underscore instead of dot  
 dict\_name = f"{file\_name}"  
  
 if hasattr(module, dict\_name):  
 return getattr(module, dict\_name)  
 else:  
 print(f"No dictionary named '{dict\_name}' found in {file\_name}.py")  
 return None  
  
  
def write\_dictionary(data\_dict, file\_path, dict\_name):  
 with open(file\_path, 'w', encoding='utf-8') as file:  
 file.write(f'{dict\_name} = ' + repr(data\_dict))  
 print(f"Dictionary has been written to {file\_path}")  
  
def create\_text\_from\_dictionary(loaded\_dictionary, dictionary\_name, output\_folder\_path):  
 # Construct the output text file name  
 output\_text\_file\_name = f"{dictionary\_name}\_text\_created.txt"  
 output\_text\_file\_path = output\_folder\_path / output\_text\_file\_name  
  
 # Inform the user about the file name and path  
 print(f"The text file will be created as: {output\_text\_file\_name}")  
 print(f"In the folder: {output\_folder\_path}")  
 input("Press Enter to continue...")  
  
 # Create the text file from the dictionary  
 with open(output\_text\_file\_path, 'w', encoding='utf-8') as file:  
 for question\_id, question\_info in loaded\_dictionary.items():  
 file.write(f"Question ID: {question\_id}\n")  
 file.write(f"Question\_number: {question\_info['question\_number']}\n")  
 file.write(f"Question\_part: {question\_info['question\_part']}\n")  
 # commenting out only to get the question\_part  
 # if 'answer' in question\_info:  
 # file.write(f"Answer: {question\_info['answer']}\n")  
 # file.write("\n---\n\n")  
  
 print(f"Text file created: {output\_text\_file\_path}")  
 return output\_text\_file\_name # Return the name of the created text file  
  
def create\_excel\_for\_dict(data\_dict, output\_dir, filename\_prefix):  
 # Convert dictionary to DataFrame  
 df = pd.DataFrame(data\_dict.values())  
  
 # Specify the file path to match the dictionary name  
 file\_path = output\_dir / f"{filename\_prefix}.xlsx"  
  
 # Inform the user about the Excel file creation details before creating it  
 print(f"Excel file will be created with the name '{filename\_prefix}.xlsx' in the directory '{output\_dir}'.")  
 input("Press Enter to Continue to create the Excel file.")  
  
 # Write DataFrame to Excel  
 with pd.ExcelWriter(file\_path, engine='openpyxl') as writer:  
 df.to\_excel(writer, index=False)  
  
 # Confirm the creation of the file  
 print(f"Excel file has been created and saved in '{file\_path}'.")  
  
  
def identify\_r\_and\_a\_type(input\_dictionary, dict\_name, base\_dir\_path):  
 *"""Identify Reason and Assertion type questions."""* # Generate the output dictionary name by removing the last two parts and adding '\_r\_and\_a\_extracted\_dictionary'  
 new\_dict\_name = '\_'.join(dict\_name.split('\_')[:-2]) + "\_r\_and\_a\_extracted\_dictionary"  
 output\_dictionary = {key: {\*\*val, 'question\_type': ''} for key, val in input\_dictionary.items()}  
 r\_and\_a\_questions\_list = [] # Initialize the list to store matching questions  
  
 # Regex pattern to find Reason and Assertion type questions  
 # pattern\_for\_r\_and\_a\_type = re.compile(r'\d+\.\s+Assertion\s\*\([A]\):.\*?Reason\s\*\([R]\):.\*?(?=\d+\.|$)', re.DOTALL)  
 pattern\_for\_r\_and\_a\_type = re.compile(r'Assertion\s\*\([A]\):.\*?Reason\s\*\([R]\):.\*?(?=\Z)', re.DOTALL)  
  
 # Iterate through dictionary items  
 for key, question\_info in input\_dictionary.items():  
 question\_text = question\_info['question\_part'] # Only check the question part  
 matches = pattern\_for\_r\_and\_a\_type.findall(question\_text)  
  
 if matches:  
 output\_dictionary[key]['question\_type'] = 'r\_and\_a\_type' # Update question type for matches  
 # for match in matches:  
 # r\_and\_a\_questions\_list.append(match) # Add to list for output and counting  
 # output\_dictionary[key] = question\_info.copy() # Copy the whole entry if it's a match  
  
 # Print the number and details of matched questions  
 print(f"Found {len(r\_and\_a\_questions\_list)} 'Reason and Assertion' type questions:")  
 for question in r\_and\_a\_questions\_list:  
 print(question)  
  
 # Writing matched questions to a text file  
 text\_file\_path = base\_dir\_path / f"{new\_dict\_name}\_text.txt"  
 with open(text\_file\_path, 'w', encoding='utf-8') as file:  
 for question in r\_and\_a\_questions\_list:  
 file.write(question + '\n\n')  
 print(f"Text file created: {text\_file\_path}")  
  
 # Inform user and ask to continue before writing the dictionary  
 print(f"Dictionary will be created with the name '{new\_dict\_name}.py' in the directory '{base\_dir\_path}'.")  
 input("Press Enter to continue...")  
  
 # Writing matched questions to the dictionary  
 # Write dictionary and inform user  
 dict\_file\_path = base\_dir\_path / f"{new\_dict\_name}.py"  
 write\_dictionary(output\_dictionary, dict\_file\_path, new\_dict\_name)  
 # print(f"Dictionary has been written to {dict\_file\_path}")  
  
 # Inform user and ask to continue before creating the Excel file  
 print(f"Excel file will be created with the name '{new\_dict\_name}.xlsx' in the directory '{base\_dir\_path}'.")  
 input("Press Enter to continue...")  
 create\_excel\_for\_dict(output\_dictionary, base\_dir\_path, new\_dict\_name)  
  
 return output\_dictionary, text\_file\_path  
  
def main():  
 base\_dir\_path = Path(  
 r'C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Python Programming\Trials\Regex Trials\answer\_options')  
 base\_dir\_path.mkdir(parents=True, exist\_ok=True) # Ensure the directory exists  
  
 print("Choose an option:")  
 print("1. Use sample text")  
 print("2. Use dictionary from a file")  
 print("3. Create text file from dictionary") # New option  
 choice = input("Enter your choice (1, 2, or 3): ")  
  
 processed\_dict = None # Initialize to None to handle scope  
 dict\_name\_used\_for\_excel = None # Initialize a variable to hold the dictionary name  
  
 if choice == '1':  
 sample\_text = """  
 119. With reference to forced labour(Vishti) in India during the  
 Gupta period, which one of the following statements is  
 correct? [2019-I]  
 (a) It was considered a source of income for the State,  
 a sort of tax paid by the people.  
 (b) It was totally absent in the Madhya Pradesh and  
 Kathiawar regions of the Gupta Empire.  
 (c) The forced labourer as entitled to weekly wages.  
 (d) The eldest son of the labourer was sent as the forced  
 labourer.  
 ---  
 120. Building 'Kalyaana Mandapas' was a notable feature in  
 the temple construction in the kingdom of [2019-I]  
 (a) Chalukya (b) Chandela  
 (c) Rashtrakuta  
 ---  
 121. With reference to the cultural history of India, which one  
 of the following is the correct description of the term  
 'paramitas'? [2020-I]  
 (a) The earliest Dharmashastra texts written in  
 aphoristic (sutra) style.  
 (b) Philosophical schools that did not accept the  
 authority of Vedas.  
 (c) Perfections whose attainment led to the Bodhisattva  
 path  
 (d) Powerful merchant guilds of early medieval south  
 India.  
 ---  
 122. With reference to the scholars/litterateurs of ancient  
 India, consider the following statements: [2020-I]  
 1. Panini is associated with Pushyamitra Shunga  
 2. Amarasimha is associated with Harshavardhana  
 3. Kalidasa is associated with Chandra Gupta-II  
 Which of the statements given above is/are correct?  
 (a) 1 and 2 only (b) 2 and 3 only  
 (c) 3 only  
 ---  
 """  
 filename\_prefix = 'test\_'  
 processed\_dict, text\_file\_path = identify\_r\_and\_a\_type(sample\_text, filename\_prefix, base\_dir\_path)  
 dict\_name\_used\_for\_excel = filename\_prefix.rstrip('\_') + "\_r\_and\_a\_extracted\_dictionary"  
  
 elif choice == '2':  
 dict\_name = input("Enter the dictionary file name (without '.py' extension): ")  
 data\_dict = load\_dictionary(dict\_name, base\_dir\_path)  
 if data\_dict:  
 print(f"Dictionary '{dict\_name}' loaded successfully.")  
 processed\_dict, text\_file\_path = identify\_r\_and\_a\_type(data\_dict, dict\_name, base\_dir\_path)  
 dict\_name\_used\_for\_excel = '\_'.join(dict\_name.split('\_')[:-2]) + "\_r\_and\_a\_extracted\_dictionary"  
 if processed\_dict:  
 print("Reason and Assertion type questions have been identified and extracted.")  
 else:  
 print("Error loading or processing the dictionary.")  
  
 elif choice == '3': # Assuming you add a third option to create text files  
 dict\_name = input("Enter the dictionary file name (without '.py' extension): ")  
 data\_dict = load\_dictionary(dict\_name, base\_dir\_path)  
 if data\_dict:  
 print(f"Dictionary '{dict\_name}' loaded successfully.")  
 created\_file\_name = create\_text\_from\_dictionary(data\_dict, dict\_name, base\_dir\_path)  
 print(f"Text file '{created\_file\_name}' has been successfully created.")  
 else:  
 print("Error loading the dictionary.")  
  
 if processed\_dict:  
 if dict\_name\_used\_for\_excel:  
 print(f"The Excel file will now be created from the dictionary '{dict\_name\_used\_for\_excel}'.")  
 input("Press Enter to Continue...")  
 create\_excel\_for\_dict(processed\_dict, base\_dir\_path, dict\_name\_used\_for\_excel)  
 else:  
 print("Invalid choice or failed operation. Exiting.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

1. **fresh\_extract\_questions\_all\_types.py** running successfully as on morning of 17 Apr, 2024 is:

from pathlib import Path  
import re  
import pandas as pd  
import importlib.util  
import os  
from openpyxl import load\_workbook  
from openpyxl.styles import Font  
from datetime import datetime  
from bs4 import BeautifulSoup  
import argparse  
from fresh\_text\_cleaner import clean\_and\_correct\_pre\_process\_text, replace\_undesirable\_characters  
import glob  
  
# refined\_questions = {}  
questions\_dict = {}  
def get\_refined\_files(directory):  
 *"""Retrieve a list of all files ending with '\_first\_refined.txt' in the specified directory."""* pattern = os.path.join(directory, '\*\_first\_refined.txt')  
 return glob.glob(pattern)  
  
  
def combine\_refined\_questions(output\_dir):  
 *"""Combine all refined question files into a single file."""* refined\_files = get\_refined\_files(output\_dir)  
 combined\_file\_path = os.path.join(output\_dir, "combined\_refined\_questions.txt")  
  
 with open(combined\_file\_path, 'w', encoding='utf-8') as combined\_file:  
 for file\_path in refined\_files:  
 with open(file\_path, 'r', encoding='utf-8') as file:  
 combined\_file.write(file.read())  
 # combined\_file.write("\n---\n") # Add a separator after each file's content  
  
 print(f"All refined questions have been combined and written to {combined\_file\_path}")  
  
  
"""def clean\_and\_preprocess\_text(text, file\_path):  
 # This is a placeholder function to handle text cleaning and preprocessing  
 # Implement or import your actual cleaning functions here  
 return text # Return cleaned text for now"""  
  
def clean\_file(input\_file\_path):  
 cleaned\_file\_path = input\_file\_path.parent / (input\_file\_path.stem + '\_cleaned.txt')  
 with open(input\_file\_path, 'r', encoding='utf-8') as file:  
 text = file.read()  
 cleaned\_text = clean\_and\_correct\_pre\_process\_text(text, input\_file\_path)  
 with open(cleaned\_file\_path, 'w', encoding='utf-8') as cleaned\_file:  
 cleaned\_file.write(cleaned\_text)  
 print(f"Cleaned text file '{cleaned\_file\_path.name}' created in '{cleaned\_file\_path.parent}'. Press Enter to continue with removing undesirable characters.")  
 input()  
 return cleaned\_file\_path  
  
  
def write\_dictionary(data\_dict, file\_path, dict\_name):  
 with open(file\_path, 'w', encoding='utf-8') as file:  
 # Write the dictionary with a specific name that matches the expected attribute  
 file.write(f'{dict\_name} = ' + repr(data\_dict))  
 print(f"Dictionary has been written to {file\_path}")  
  
  
def load\_replacement\_dictionary(file\_path):  
 with open(file\_path, 'r', encoding='utf-8') as file:  
 dictionary = eval(file.read())  
 return dictionary  
  
def extract\_question\_number(questions):  
 question\_numbers\_first\_round = []  
 for question in questions:  
 # Extract the number from the start of the question  
 match = re.match(r"^(\d+)\.", question)  
 if match:  
 # If a number is found, append it to the list of question numbers  
 question\_number = int(match.group(1))  
 question\_numbers\_first\_round.append(question\_number)  
 else:  
 # If no number is found, append None or some placeholder to indicate a missing number  
 question\_numbers\_first\_round.append(None)  
 return question\_numbers\_first\_round  
  
def assess\_extraction(questions\_dict, target):  
 # Extract question numbers directly from the dictionary keys, which represent the question numbers  
 question\_numbers = [int(q\_num) for q\_num in questions\_dict.keys() if q\_num is not None]  
  
 missing\_numbers = sorted(set(range(1, target + 1)) - set(question\_numbers))  
 valid\_numbers = sorted(set(question\_numbers))  
  
 percentage\_valid = len(valid\_numbers) / target \* 100 if target != 0 else 0  
  
 return {  
 "date\_time": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),  
 "target": target,  
 "valid\_question\_numbers": valid\_numbers,  
 "missing\_question\_numbers": missing\_numbers,  
 "percentage\_valid\_questions": percentage\_valid  
 }  
  
def write\_performance\_excel(data, file\_path):  
 # Convert the data to a DataFrame  
 new\_data\_df = pd.DataFrame([data])  
  
 # If the file already exists, read it, append the new data, and save it  
 if os.path.exists(file\_path):  
 try:  
 # Load the existing data  
 existing\_df = pd.read\_excel(file\_path)  
 # Append the new data  
 updated\_df = pd.concat([existing\_df, new\_data\_df], ignore\_index=True)  
 except Exception as e:  
 print(f"Failed to read the existing Excel file. Error: {e}")  
 print("New data will be written to a new file.")  
 updated\_df = new\_data\_df  
 else:  
 updated\_df = new\_data\_df  
  
 # Use XlsxWriter to write the DataFrame to the Excel file  
 with pd.ExcelWriter(file\_path, engine='xlsxwriter') as writer:  
 updated\_df.to\_excel(writer, index=False)  
  
 print(f"Performance data updated and saved in '{file\_path}'. The file is located in the folder: '{os.path.dirname(file\_path)}'.")  
  
def extract\_all\_questions(text, output\_dir, base\_filename):  
  
 # pattern\_for\_all\_questions = r'(\d+\.\s+(?:.\* ?(?:\n\s \*)?) \* ?(?=\(d\)[ ^ a - d] \* \n \*\d +\.\s |\Z))'  
 # pattern\_for\_all\_questions = r'(\d+\.\s+(?:.\*?(?:\n\s\*)?)\*?(?=\(d\)(?:(?!\([abcd]\)).)\*\n\*\d+\.\s|\Z))'  
 pattern\_for\_all\_questions = r'(\d+\.\s+[\s\S]+?(?=\s\*\(\w\)\s+[\s\S]\*?){3,4}\s\*\(\w\)\s+[\s\S]\*?(?=\d+\.\s+|\Z))'  
  
 questions = re.findall(pattern\_for\_all\_questions, text)  
  
 if not questions:  
 print("No questions extracted. Please check the file content and format.")  
 return {}, []  
  
 print(f"Number of questions found initially: {len(questions)}") # Initial count right after findall  
  
 # After extracting questions  
 question\_numbers\_first\_round = extract\_question\_number(questions)  
 # Uncommenting as initialised already at the beginning of the script  
 # questions\_dict = {}  
 output\_file\_path = output\_dir / (base\_filename + '\_extracted\_questions.txt')  
  
 with open(output\_file\_path, 'w', encoding='utf-8') as file:  
 for i, question in enumerate(questions, start=1):  
 question\_num = question\_numbers\_first\_round[i - 1]  
 if question\_num is not None:  
 questions\_dict[question\_num] = {  
 'question\_id': question\_num,  
 'question\_number': question\_num,  
 'question': question.strip(),  
 'source\_file\_hyperlink': str(output\_file\_path)  
 }  
 file.write(f"Question {question\_num}:\n{question.strip()}\n\n---\n\n")  
  
 print(f"Questions have been written to {output\_file\_path}")  
  
 # Write dictionary to a .py file  
 dict\_file\_path = output\_dir / (base\_filename + '\_first\_round\_dict.py')  
 dict\_name = base\_filename + '\_first\_round\_dict' # This should match the filename without '.py'  
 write\_dictionary(questions\_dict, dict\_file\_path, dict\_name)  
  
 # return questions\_dict, question\_numbers # Return both the dictionary and the list of question numbers  
 return questions\_dict, question\_numbers\_first\_round  
  
  
def write\_questions\_to\_file(questions, file\_path):  
 with open(file\_path, 'w', encoding='utf-8') as file:  
 for question in questions:  
 file.write(question.strip() + '\n\n---\n\n')  
 print(f"Questions have been written to {file\_path}")  
  
def first\_refine(questions\_dict, output\_dir):  
 pattern\_for\_breaking\_at\_a = r'(\d+\.\s+(?:[\s\S]\*?\([a-d]\)[\s\S]\*?)(?=\d+\.\s|\Z))'  
 refined\_questions\_dictionary = {}  
 refined\_question\_numbers = [] # Initialize to track which questions are refined  
  
 for question\_id, question\_info in questions\_dict.items():  
 matches = re.findall(pattern\_for\_breaking\_at\_a, question\_info['question'], re.DOTALL)  
 if len(matches) > 1:  
 for index, match in enumerate(matches, start=1):  
 match\_key = f"{question\_id}\_{index}" # Creating a unique key for each match  
 """refined\_questions\_dictionary[match\_key] = {  
 'question\_number': question\_info['question\_number'], # Keeping the original question number  
 'question': f"\n{match.strip()}\n" # Formatting the question with new lines  
 }  
 refined\_question\_numbers.append(match\_key)"""  
 file\_name = f"{match\_key}\_first\_refined.txt"  
 output\_file\_path = os.path.join(output\_dir, file\_name)  
 with open(output\_file\_path, 'w', encoding='utf-8') as file:  
 file.write(f"{match.strip()}\n")  
 file.write("---\n")  
 print(f"Refined questions for {match\_key} have been written to {output\_file\_path}")  
  
 print("\nRefined Question Numbers:")  
 print(refined\_question\_numbers)  
  
 # Writing the refined\_questions dictionary to a file  
 """dict\_file\_path = os.path.join(output\_dir, 'refined\_questions\_dictionary.py')  
 dict\_name = 'refined\_questions\_dictionary'  
 write\_dictionary(refined\_questions\_dictionary, dict\_file\_path, dict\_name)  
 print("Refined questions dictionary has been written to:", dict\_file\_path)  
 print("Press Enter to continue.")  
 input()  
  
 return refined\_questions\_dictionary, refined\_question\_numbers"""  
  
def create\_refined\_questions\_dict(output\_dir):  
 # Path to the combined refined questions file  
 combined\_file\_path = os.path.join(output\_dir, "combined\_refined\_questions.txt")  
 only\_refined\_questions = {}  
 question\_id = 1 # Start question ID from 1  
  
 with open(combined\_file\_path, 'r', encoding='utf-8') as file:  
 content = file.read()  
 # Split the content by the separator "---\n"  
 questions = content.split("---\n")  
  
 # Iterate over each block extracted and treat it as a separate question  
 for question in questions:  
 if question.strip(): # Check if the question block is not empty  
 number\_match = re.search(r'^\s\*(\d+)\.', question)  
 question\_number = number\_match.group(1)  
 question\_number = int(question\_number)  
 only\_refined\_questions[str(question\_id)] = {  
 'question\_id': str(question\_id),  
 'question\_number': question\_number, # Use question\_id as question\_number  
 'question': question.strip(),  
 'source\_file\_hyperlink': "No hyperlink" # Placeholder if no hyperlink is available  
 }  
 question\_id += 1  
  
 # Define the file path and dictionary name for writing the refined\_questions dictionary  
 dict\_file\_path = os.path.join(output\_dir, 'only\_refined\_questions.py')  
 dict\_name = 'only\_refined\_questions' # This should match the dictionary name used in the script  
  
 # Call to write the dictionary to a file  
 write\_dictionary(only\_refined\_questions, dict\_file\_path, dict\_name)  
 print("Only refined questions dictionary has been written to:", dict\_file\_path)  
  
 return only\_refined\_questions  
  
def create\_2nd\_round\_dict(questions\_dict, only\_refined\_questions, output\_dir, base\_filename):  
 # Initialize an empty dictionary for the second round  
 second\_round\_dict = {}  
  
 # Extract question numbers from both dictionaries  
 questions\_numbers = set(q['question\_number'] for q in questions\_dict.values())  
 refined\_question\_numbers = set(q['question\_number'] for q in only\_refined\_questions.values())  
  
 # Create a set of all question numbers from both sets  
 all\_question\_numbers = questions\_numbers.union(refined\_question\_numbers)  
  
 # Iterate over the union of question numbers  
 for question\_number in all\_question\_numbers:  
 original\_info = next((v for k, v in questions\_dict.items() if v['question\_number'] == question\_number), None)  
 refined\_info = next((v for k, v in only\_refined\_questions.items() if v['question\_number'] == question\_number),  
 None)  
  
 if refined\_info:  
 # Prefer refined info if available  
 second\_round\_dict[str(question\_number)] = {  
 'question\_id': question\_number,  
 'question\_number': question\_number,  
 'question': refined\_info['question'].strip(),  
 'source\_file\_hyperlink': refined\_info.get('source\_file\_hyperlink', "No hyperlink")  
 }  
 elif original\_info:  
 # Use original info if refined info is not available  
 second\_round\_dict[str(question\_number)] = {  
 'question\_id': question\_number,  
 'question\_number': question\_number,  
 'question': original\_info['question'].strip(),  
 'source\_file\_hyperlink': original\_info['source\_file\_hyperlink']  
 }  
  
 # Write the updated dictionary to a Python file  
 dict\_file\_path = os.path.join(output\_dir, f"{base\_filename}\_second\_round\_dict.py")  
 with open(dict\_file\_path, 'w', encoding='utf-8') as file:  
 file.write(f"{base\_filename}\_second\_round\_dict = " + repr(second\_round\_dict))  
 print(f"Second round dictionary has been created and written to {dict\_file\_path}")  
  
 # Optionally, create a text file from the updated dictionary with questions appearing sequentially  
 text\_file\_path = os.path.join(output\_dir, f"{base\_filename}\_extracted\_questions\_second\_round.txt")  
 # Inside the create\_2nd\_round\_dict function, when you need to sort the keys:  
 sorted\_keys = sorted(second\_round\_dict.keys(), key=lambda x: str(x)) # Convert keys to strings before sorting  
 with open(text\_file\_path, 'w', encoding='utf-8') as file:  
 for question\_id in sorted\_keys:  
 question\_info = second\_round\_dict[question\_id]  
 file.write(f"Question {question\_id}: {question\_info['question']}\n\n---\n\n")  
 print(f"Extracted questions text file for the second round has been created and written to {text\_file\_path}")  
  
 return second\_round\_dict  
  
def extract\_answer\_options(second\_dict, filename\_prefix, base\_dir\_path):  
 answer\_options\_extracted\_dict\_name = f"{filename\_prefix}answer\_options\_extracted\_dictionary"  
 answer\_options\_extracted\_dictionary = {}  
  
 pattern = r'\((a|b|c|d)\)\s(.\*?)\s\*(?=\([abcd]\)|$)'  
  
 for question\_id, question\_details in second\_dict.items():  
 question\_part = question\_details.get('question', '')  
 answer\_part = question\_details.get('answer', '')  
 source\_file\_hyperlink = question\_details.get('source\_file\_hyperlink', '')  
  
 matches = re.findall(pattern, answer\_part, flags=re.DOTALL)  
 options = {match[0]: match[1].strip() for match in matches}  
  
 answer\_options\_extracted\_dictionary[question\_id] = {  
 "question\_id": question\_id,  
 "question\_number": question\_details.get('question\_number', 'N/A'),  
 "question": question\_part,  
 "answer": answer\_part,  
 "answer\_option\_a": options.get('a', ''),  
 "answer\_option\_b": options.get('b', ''),  
 "answer\_option\_c": options.get('c', ''),  
 "answer\_option\_d": options.get('d', ''),  
 "source\_file\_hyperlink": source\_file\_hyperlink,  
 }  
  
 file\_path = base\_dir\_path / f"{answer\_options\_extracted\_dict\_name}.py"  
  
 # Using existing write\_dictionary function  
 write\_dictionary(answer\_options\_extracted\_dictionary, file\_path, answer\_options\_extracted\_dict\_name)  
  
 print(f"The file '{answer\_options\_extracted\_dict\_name}.py' has been created in the directory {base\_dir\_path}")  
 return answer\_options\_extracted\_dictionary  
  
  
def main(input\_file\_path):  
 base\_dir\_path = input\_file\_path.parent  
 base\_filename = input\_file\_path.stem # Ensure base\_filename is properly defined here  
  
 # Dynamic filename prefix for more flexibility  
 filename\_prefix = f"{base\_filename}\_"  
  
 # Step 1: Read the original text from the input file  
 with open(input\_file\_path, 'r', encoding='utf-8') as file:  
 original\_text = file.read()  
  
 # Step 2: Clean the text and write to a new cleaned file  
 cleaned\_text = clean\_and\_correct\_pre\_process\_text(original\_text, input\_file\_path)  
 cleaned\_file\_path = input\_file\_path.parent / (input\_file\_path.stem + '\_cleaned.txt')  
 with open(cleaned\_file\_path, 'w', encoding='utf-8') as file:  
 file.write(cleaned\_text)  
 print(f"Cleaned text file '{cleaned\_file\_path.name}' created.")  
  
 # Step 3: Replace undesirable characters in the cleaned file and save it as fully cleaned  
 fully\_cleaned\_file\_path = replace\_undesirable\_characters(cleaned\_file\_path)  
 print(f"Fully cleaned text file '{fully\_cleaned\_file\_path.name}' created.")  
  
 # Step 4: Extract questions from the fully cleaned file  
 print(f"Press Enter to continue with the extraction of all questions.")  
 input()  
 with open(fully\_cleaned\_file\_path, 'r', encoding='utf-8') as file:  
 fully\_cleaned\_text = file.read()  
  
 # extract\_all\_questions(fully\_cleaned\_text, input\_file\_path.parent, input\_file\_path.stem)  
 questions\_dict, question\_numbers = extract\_all\_questions(fully\_cleaned\_text, input\_file\_path.parent, input\_file\_path.stem)  
  
 # Step 5: Take user input for target number of questions and proceed with assessment  
 print("Press Enter to Continue for assessing extraction")  
 input()  
 user\_provided\_target = int(input("Enter the number of questions you are targeting to extract: "))  
 # Now you would call the assess\_extraction function and pass the necessary arguments, including the target.  
 performance\_data = assess\_extraction(questions\_dict, user\_provided\_target)  
  
 # You will also need to pass a file path for writing the performance data to an Excel file  
 performance\_file\_path = base\_dir\_path / f"{input\_file\_name}\_extraction\_performance.xlsx"  
 # Ensure we use the same file to append the updated data  
 write\_performance\_excel(performance\_data, performance\_file\_path)  
  
 # New Step 6: First refine process  
 print("Press Enter to continue with the first refinement of questions.")  
 input()  
 # refined\_questions\_dictionary, refined\_question\_numbers = first\_refine(questions\_dict, base\_dir\_path) # Capture refined numbers  
 first\_refine(questions\_dict, base\_dir\_path) # Only call the function, no need to capture returns  
  
 # Combine all first refined questions into a single file  
 print("Press Enter to continue combining all refined questions into a single file.")  
 input()  
 combine\_refined\_questions(base\_dir\_path)  
  
 # New Step 7: Create the only\_refined\_questions dictionary and generate the .py file  
 print("Press Enter to continue creating the only\_refined\_questions dictionary and generate the .py file.")  
 input()  
 only\_refined\_questions = create\_refined\_questions\_dict(base\_dir\_path)  
  
 # New Step 8: Create the second round dictionary and generate the associated text file  
 print("Press Enter to continue creating the second round dictionary and generate the .py file.")  
 input()  
  
 second\_round\_dict = create\_2nd\_round\_dict(questions\_dict, only\_refined\_questions, base\_dir\_path, base\_filename)  
  
 # New Step 9: Assess extraction after first refine  
 print("Press Enter to Continue for assessing extraction after first refine.")  
 input()  
 refined\_performance\_data = assess\_extraction(second\_round\_dict, user\_provided\_target)  
 write\_performance\_excel(refined\_performance\_data, performance\_file\_path)  
  
 print(f"Updated extraction performance has been assessed and saved in the same file.")  
  
 # New Step 10: Extract answer options  
 print("Press Enter to continue extracting answer options.")  
 input()  
 answer\_options\_extracted\_dictionary = extract\_answer\_options(second\_round\_dict, filename\_prefix, base\_dir\_path)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 input\_file\_name = input("Enter the name of the text file (without the .txt extension): ")  
 parts = input\_file\_name.split('\_')  
 subfolder\_name = parts[6] + ' ' + parts[7]  
 base\_dir\_path = Path(r'C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Area') / subfolder\_name.lower()  
  
 # Create subfolder if it doesn't exist  
 base\_dir\_path.mkdir(parents=True, exist\_ok=True)  
  
 print(f"Parts: {parts}")  
 for i, part in enumerate(parts):  
 print(f"Part {i + 1}: {part}")  
 print(f"Subfolder name: {subfolder\_name}")  
 print(f"Base directory path: {base\_dir\_path}")  
  
 input\_file\_path = base\_dir\_path / (input\_file\_name + '.txt')  
 print(f"Input file path: {input\_file\_path}")  
  
 if input\_file\_path.exists():  
 print(f"Input file found in {input\_file\_path.parent}")  
 main(input\_file\_path)  
 else:  
 print(f"File not found: {input\_file\_path}")

1. **fresh\_extract\_answer\_options.py** successfully as on mid-day of 17 Apr, 2024 **i**s:

import re  
from pathlib import Path  
import pandas as pd  
import ast # Import ast module for safer evaluation  
import importlib.util  
from pathlib import Path  
import datetime  
  
  
def load\_dictionary(file\_name, base\_dir\_path):  
 *""" Load a dictionary from a Python file within the specified directory. """* dict\_path = base\_dir\_path / f"{file\_name}.py"  
 if not dict\_path.exists():  
 print(f"File {dict\_path} not found.")  
 return None  
  
 spec = importlib.util.spec\_from\_file\_location(file\_name, dict\_path)  
 module = importlib.util.module\_from\_spec(spec)  
 spec.loader.exec\_module(module)  
  
 # Assuming the dictionary is named the same as file\_name but with an underscore instead of dot  
 dict\_name = f"{file\_name}"  
  
 if hasattr(module, dict\_name):  
 return getattr(module, dict\_name)  
 else:  
 print(f"No dictionary named '{dict\_name}' found in {file\_name}.py")  
 return None  
  
"""def extract\_answer\_options(sample\_text, filename\_prefix, base\_dir\_path):  
 answer\_options\_extracted\_dict\_name = f"{filename\_prefix}answer\_options\_extracted\_dictionary"  
 answer\_options\_extracted\_dictionary = {}  
 options\_temp\_storage = {}  
  
 pattern = r'\((a|b|c|d)\)\s+(.\*?)(?=\s\*\(\w\)\s|\s\*$)'  
 matches = re.findall(pattern, sample\_text, flags=re.DOTALL)  
  
 print(f"Found {len(matches)} matches:")  
 for index, (option, content) in enumerate(matches, start=1):  
 print(f"Part {index}:")  
 print(f"Option: {option}, Content: {content.strip()}")  
 print("---")  
  
 number\_match = re.search(r'^\s\*(\d+)\.', content) # Assuming each option starts with the question number  
 if number\_match:  
 question\_number = int(number\_match.group(1))  
 # Initialize dictionary for new questions  
 if question\_number not in options\_temp\_storage:  
 options\_temp\_storage[question\_number] = {  
 "question\_id": question\_number,  
 "question\_number": question\_number,  
 "answer\_option\_a": "{content.strip()}",  
 "answer\_option\_b": "",  
 "answer\_option\_c": "",  
 "answer\_option\_d": ""  
 }  
 # Assign content to the correct option  
 options\_temp\_storage[question\_number][f"answer\_option\_{option}"] = content.strip()  
  
 # Now populate the main dictionary with the processed data from temp storage  
 for key, value in options\_temp\_storage.items():  
 answer\_options\_extracted\_dictionary[key] = value  
  
 file\_path = base\_dir\_path / f"{answer\_options\_extracted\_dict\_name}.py"  
 # Using existing write\_dictionary function  
 write\_dictionary(answer\_options\_extracted\_dictionary, file\_path, answer\_options\_extracted\_dict\_name)  
  
 print(f"The file '{answer\_options\_extracted\_dict\_name}.py' has been created in the directory {base\_dir\_path}")  
 return answer\_options\_extracted\_dictionary"""  
  
  
def write\_dictionary(data\_dict, file\_path, dict\_name):  
 with open(file\_path, 'w', encoding='utf-8') as file:  
 file.write(f'{dict\_name} = ' + repr(data\_dict))  
 print(f"Dictionary has been written to {file\_path}")  
  
  
def create\_excel\_for\_dict(data\_dict, output\_dir, filename\_prefix):  
 # Convert dictionary to DataFrame  
 df = pd.DataFrame(data\_dict.values())  
  
 # Specify the file path to match the dictionary name  
 file\_path = output\_dir / f"{filename\_prefix}.xlsx"  
  
 # Inform the user about the process starting  
 print("Press Enter to Continue to create the Excel file.")  
 input()  
  
 # Write DataFrame to Excel  
 with pd.ExcelWriter(file\_path, engine='openpyxl') as writer:  
 df.to\_excel(writer, index=False)  
  
 # Inform the user about the creation of the file  
 print(f"Excel file has been created and saved in '{file\_path}'.")  
  
  
def extract\_question\_and\_answer\_parts\_and\_answer\_options(input\_dictionary, dict\_name, base\_dir\_path):  
 output\_dict\_name = '\_'.join(dict\_name.split('\_')[:-3]) + "\_all\_parts\_extracted\_dictionary"  
 output\_dictionary = {}  
  
 options\_pattern = re.compile(r'\((a|b|c|d)\)\s+(.\*?)(?=\s\*\([abcd]\)\s+|\s\*$)', re.DOTALL)  
 year\_pattern = re.compile(r'\[(\d{4}(-\d+)?)\]') # Enhanced pattern to capture years like "2020-1"  
 current\_year = datetime.datetime.now().year  
  
 for key, entry in input\_dictionary.items():  
 question\_text = entry['question']  
 # Extract year and update the dictionary  
 year\_match = year\_pattern.search(question\_text)  
 if year\_match:  
 year = int(year\_match.group(1))  
 # Check if the year is within the valid range  
 if 1900 <= year <= current\_year:  
 exam\_year = year  
 else:  
 exam\_year = "Invalid" # Mark as invalid if outside the range  
  
 question\_text = year\_pattern.sub("", question\_text) # Remove the year from the question text  
 else:  
 exam\_year = "Unknown" # Default value if no year is found  
  
 question\_part = re.split(r'\([abcd]\)', question\_text)[0].strip()  
 question\_part = re.sub(r'^\d+\.\s\*', '', question\_part) # Remove the leading question number  
  
 if question\_part:  
 if question\_part in question\_text:  
 answer\_part = question\_text.split(question\_part)[-1].strip() # Corrected to handle the delimiter issue  
 else:  
 answer\_part = "" # Fallback in case the question part is not properly split  
 else:  
 answer\_part = question\_text # Fallback to use the whole text if question part is empty  
  
 output\_dictionary[key] = entry.copy() # Copy existing data  
 output\_dictionary[key]['question\_part'] = question\_part  
 output\_dictionary[key]['answer\_part'] = answer\_part  
 output\_dictionary[key]['exam\_year'] = exam\_year # Add the exam year to the dictionary  
  
 # Extract and assign options  
 options = options\_pattern.findall(question\_text)  
 for option, content in options:  
 output\_dictionary[key][f'answer\_option\_{option}'] = content.strip()  
  
 file\_path = base\_dir\_path / f"{output\_dict\_name}.py"  
 write\_dictionary(output\_dictionary, file\_path, output\_dict\_name)  
  
 print(f"Updated dictionary is written to {file\_path}")  
 return output\_dictionary, output\_dict\_name  
  
def main():  
 base\_dir\_path = Path(  
 r'C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Python Programming\Trials\Regex Trials\answer\_options')  
 base\_dir\_path.mkdir(parents=True, exist\_ok=True) # Ensure the directory exists  
  
 print("Choose an option:")  
 print("1. Use sample text")  
 print("2. Use dictionary from a file")  
 choice = input("Enter your choice (1 or 2): ")  
  
 processed\_dict = None # Initialize to None to handle scope  
 dict\_name\_used\_for\_excel = None # Initialize a variable to hold the dictionary name  
  
 if choice == '1':  
 sample\_text = """  
 119. With reference to forced labour(Vishti) in India during the  
 Gupta period, which one of the following statements is  
 correct? [2019-I]  
 (a) It was considered a source of income for the State,  
 a sort of tax paid by the people.  
 (b) It was totally absent in the Madhya Pradesh and  
 Kathiawar regions of the Gupta Empire.  
 (c) The forced labourer as entitled to weekly wages.  
 (d) The eldest son of the labourer was sent as the forced  
 labourer.  
 ---  
 120. Building 'Kalyaana Mandapas' was a notable feature in  
 the temple construction in the kingdom of [2019-I]  
 (a) Chalukya (b) Chandela  
 (c) Rashtrakuta  
 ---  
 121. With reference to the cultural history of India, which one  
 of the following is the correct description of the term  
 'paramitas'? [2020-I]  
 (a) The earliest Dharmashastra texts written in  
 aphoristic (sutra) style.  
 (b) Philosophical schools that did not accept the  
 authority of Vedas.  
 (c) Perfections whose attainment led to the Bodhisattva  
 path  
 (d) Powerful merchant guilds of early medieval south  
 India.  
 ---  
 122. With reference to the scholars/litterateurs of ancient  
 India, consider the following statements: [2020-I]  
 1. Panini is associated with Pushyamitra Shunga  
 2. Amarasimha is associated with Harshavardhana  
 3. Kalidasa is associated with Chandra Gupta-II  
 Which of the statements given above is/are correct?  
 (a) 1 and 2 only (b) 2 and 3 only  
 (c) 3 only  
 ---  
 """  
 filename\_prefix = 'test\_'  
 # Assuming the function can handle text directly; otherwise, you need to adjust the implementation  
 processed\_dict = extract\_question\_and\_answer\_parts\_and\_answer\_options(sample\_text, filename\_prefix,  
 base\_dir\_path)  
 elif choice == '2':  
 dict\_name = input("Enter the dictionary file name (without '.py' extension): ")  
 data\_dict = load\_dictionary(dict\_name, base\_dir\_path)  
 if data\_dict:  
 print(f"Dictionary '{dict\_name}' loaded successfully.")  
 # Correct handling of the function output  
 processed\_dict, dict\_name\_used\_for\_excel = extract\_question\_and\_answer\_parts\_and\_answer\_options(data\_dict,  
 dict\_name,  
 base\_dir\_path)  
 else:  
 print("Error processing the dictionary.")  
  
 if processed\_dict:  
 print(f"The Excel file will now be created from the dictionary '{dict\_name\_used\_for\_excel}'.")  
 input("Press Enter to Continue...")  
 # Correctly pass the dictionary and its name to the function  
 create\_excel\_for\_dict(processed\_dict, base\_dir\_path, dict\_name\_used\_for\_excel)  
 else:  
 print("Invalid choice or failed operation. Exiting.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

1. **test\_pattern\_with\_direct\_answer\_options.py** running successfully as on morning of 17 Apr, 2024 **i**s:

import re  
from pathlib import Path  
import pandas as pd  
import ast # Import ast module for safer evaluation  
import importlib.util  
from pathlib import Path  
  
  
def load\_dictionary(file\_name, base\_dir\_path):  
 *""" Load a dictionary from a Python file within the specified directory. """* dict\_path = base\_dir\_path / f"{file\_name}.py"  
 if not dict\_path.exists():  
 print(f"File {dict\_path} not found.")  
 return None  
  
 spec = importlib.util.spec\_from\_file\_location(file\_name, dict\_path)  
 module = importlib.util.module\_from\_spec(spec)  
 spec.loader.exec\_module(module)  
  
 # Assuming the dictionary is named the same as file\_name but with an underscore instead of dot  
 dict\_name = f"{file\_name}"  
  
 if hasattr(module, dict\_name):  
 return getattr(module, dict\_name)  
 else:  
 print(f"No dictionary named '{dict\_name}' found in {file\_name}.py")  
 return None  
  
"""def extract\_answer\_options(sample\_text, filename\_prefix, base\_dir\_path):  
 answer\_options\_extracted\_dict\_name = f"{filename\_prefix}answer\_options\_extracted\_dictionary"  
 answer\_options\_extracted\_dictionary = {}  
 options\_temp\_storage = {}  
  
 pattern = r'\((a|b|c|d)\)\s+(.\*?)(?=\s\*\(\w\)\s|\s\*$)'  
 matches = re.findall(pattern, sample\_text, flags=re.DOTALL)  
  
 print(f"Found {len(matches)} matches:")  
 for index, (option, content) in enumerate(matches, start=1):  
 print(f"Part {index}:")  
 print(f"Option: {option}, Content: {content.strip()}")  
 print("---")  
  
 number\_match = re.search(r'^\s\*(\d+)\.', content) # Assuming each option starts with the question number  
 if number\_match:  
 question\_number = int(number\_match.group(1))  
 # Initialize dictionary for new questions  
 if question\_number not in options\_temp\_storage:  
 options\_temp\_storage[question\_number] = {  
 "question\_id": question\_number,  
 "question\_number": question\_number,  
 "answer\_option\_a": "{content.strip()}",  
 "answer\_option\_b": "",  
 "answer\_option\_c": "",  
 "answer\_option\_d": ""  
 }  
 # Assign content to the correct option  
 options\_temp\_storage[question\_number][f"answer\_option\_{option}"] = content.strip()  
  
 # Now populate the main dictionary with the processed data from temp storage  
 for key, value in options\_temp\_storage.items():  
 answer\_options\_extracted\_dictionary[key] = value  
  
 file\_path = base\_dir\_path / f"{answer\_options\_extracted\_dict\_name}.py"  
 # Using existing write\_dictionary function  
 write\_dictionary(answer\_options\_extracted\_dictionary, file\_path, answer\_options\_extracted\_dict\_name)  
  
 print(f"The file '{answer\_options\_extracted\_dict\_name}.py' has been created in the directory {base\_dir\_path}")  
 return answer\_options\_extracted\_dictionary"""  
  
  
def write\_dictionary(data\_dict, file\_path, dict\_name):  
 with open(file\_path, 'w', encoding='utf-8') as file:  
 file.write(f'{dict\_name} = ' + repr(data\_dict))  
 print(f"Dictionary has been written to {file\_path}")  
  
  
def create\_excel\_for\_dict(data\_dict, output\_dir, filename\_prefix):  
 # Convert dictionary to DataFrame  
 df = pd.DataFrame(data\_dict.values())  
  
 # Specify the file path to match the dictionary name  
 file\_path = output\_dir / f"{filename\_prefix}.xlsx"  
  
 # Inform the user about the process starting  
 print("Press Enter to Continue to create the Excel file.")  
 input()  
  
 # Write DataFrame to Excel  
 with pd.ExcelWriter(file\_path, engine='openpyxl') as writer:  
 df.to\_excel(writer, index=False)  
  
 # Inform the user about the creation of the file  
 print(f"Excel file has been created and saved in '{file\_path}'.")  
  
  
def extract\_question\_and\_answer\_parts\_and\_answer\_options(input\_dictionary, dict\_name, base\_dir\_path):  
 output\_dict\_name = '\_'.join(dict\_name.split('\_')[:-3]) + "\_all\_parts\_extracted\_dictionary"  
 output\_dictionary = {}  
  
 options\_pattern = re.compile(r'\((a|b|c|d)\)\s+(.\*?)(?=\s\*\([abcd]\)\s+|\s\*$)', re.DOTALL)  
  
 for key, entry in input\_dictionary.items():  
 question\_text = entry['question']  
 question\_part = re.split(r'\([abcd]\)', question\_text)[0].strip()  
 answer\_part = question\_text.split(question\_part)[-1].strip()  
  
 output\_dictionary[key] = entry.copy() # Copy existing data  
 output\_dictionary[key]['question\_part'] = question\_part  
 output\_dictionary[key]['answer\_part'] = answer\_part  
  
 # Extract and assign options  
 options = options\_pattern.findall(question\_text)  
 for option, content in options:  
 output\_dictionary[key][f'answer\_option\_{option}'] = content.strip()  
  
 file\_path = base\_dir\_path / f"{output\_dict\_name}.py"  
 write\_dictionary(output\_dictionary, file\_path, output\_dict\_name)  
  
 print(f"Updated dictionary is written to {file\_path}")  
 return output\_dictionary, output\_dict\_name  
  
  
def main():  
 base\_dir\_path = Path(  
 r'C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Python Programming\Trials\Regex Trials\answer\_options')  
 base\_dir\_path.mkdir(parents=True, exist\_ok=True) # Ensure the directory exists  
  
 print("Choose an option:")  
 print("1. Use sample text")  
 print("2. Use dictionary from a file")  
 choice = input("Enter your choice (1 or 2): ")  
  
 processed\_dict = None # Initialize to None to handle scope  
 dict\_name\_used\_for\_excel = None # Initialize a variable to hold the dictionary name  
  
 if choice == '1':  
 sample\_text = """  
 119. With reference to forced labour(Vishti) in India during the  
 Gupta period, which one of the following statements is  
 correct? [2019-I]  
 (a) It was considered a source of income for the State,  
 a sort of tax paid by the people.  
 (b) It was totally absent in the Madhya Pradesh and  
 Kathiawar regions of the Gupta Empire.  
 (c) The forced labourer as entitled to weekly wages.  
 (d) The eldest son of the labourer was sent as the forced  
 labourer.  
 ---  
 120. Building 'Kalyaana Mandapas' was a notable feature in  
 the temple construction in the kingdom of [2019-I]  
 (a) Chalukya (b) Chandela  
 (c) Rashtrakuta  
 ---  
 121. With reference to the cultural history of India, which one  
 of the following is the correct description of the term  
 'paramitas'? [2020-I]  
 (a) The earliest Dharmashastra texts written in  
 aphoristic (sutra) style.  
 (b) Philosophical schools that did not accept the  
 authority of Vedas.  
 (c) Perfections whose attainment led to the Bodhisattva  
 path  
 (d) Powerful merchant guilds of early medieval south  
 India.  
 ---  
 122. With reference to the scholars/litterateurs of ancient  
 India, consider the following statements: [2020-I]  
 1. Panini is associated with Pushyamitra Shunga  
 2. Amarasimha is associated with Harshavardhana  
 3. Kalidasa is associated with Chandra Gupta-II  
 Which of the statements given above is/are correct?  
 (a) 1 and 2 only (b) 2 and 3 only  
 (c) 3 only  
 ---  
 """  
 filename\_prefix = 'test\_'  
 # Assuming the function can handle text directly; otherwise, you need to adjust the implementation  
 processed\_dict = extract\_question\_and\_answer\_parts\_and\_answer\_options(sample\_text, filename\_prefix,  
 base\_dir\_path)  
 elif choice == '2':  
 dict\_name = input("Enter the dictionary file name (without '.py' extension): ")  
 data\_dict = load\_dictionary(dict\_name, base\_dir\_path)  
 if data\_dict:  
 print(f"Dictionary '{dict\_name}' loaded successfully.")  
 # Correct handling of the function output  
 processed\_dict, dict\_name\_used\_for\_excel = extract\_question\_and\_answer\_parts\_and\_answer\_options(data\_dict,  
 dict\_name,  
 base\_dir\_path)  
 else:  
 print("Error processing the dictionary.")  
  
 if processed\_dict:  
 print(f"The Excel file will now be created from the dictionary '{dict\_name\_used\_for\_excel}'.")  
 input("Press Enter to Continue...")  
 # Correctly pass the dictionary and its name to the function  
 create\_excel\_for\_dict(processed\_dict, base\_dir\_path, dict\_name\_used\_for\_excel)  
 else:  
 print("Invalid choice or failed operation. Exiting.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

1. **test\_breaking\_with\_direct\_text.py** running successfully as on morning 15 Apr, 2024 is:

import re  
import os  
  
sample\_text = """  
  
69. With reference to ancient Jainism, which one of the following  
statements is correct? [2004]  
(a) Jainism was spread in South India under the leadership  
of Sthalabahu.  
(b) The Jainas who remained under the leadership of  
Bhadrabahu were called Shvetambaras after the council  
held at Pataliputra.  
(c) Jainism enjoyed the patronage of the Kalinga king  
Kharaavela in the first century BC.  
(d) In the initial stage of Jainism, the Jainas worshipped  
images unlike the Buddhist.  
70. Which one of the following four Vedas contains an account  
of magical charms and spells? [2004]  
(a) Rigveda (b) Y ajurveda  
(c) Atharvaveda (d) Samaveda  
 71. Who among the following was not a contemporary of the  
other three? [2005]  
(a) Bimbsara (b) Gautama Buddha  
(c) Milinda  
  
  
  
"""  
  
# Define a regex pattern to break at each question number  
# pattern\_for\_breaking\_at\_a = r'(\d+\.\s+[\s\S]\*?(?=\d+\.\s|\Z))'  
# pattern\_for\_breaking\_at\_a = r'(\d+\.\s+[\s\S]\*?)(?=\n\d+\.\s+[^1-9]|\Z)'  
pattern\_for\_breaking\_at\_a = r'(\d+\.\s+(?:[\s\S]\*?\([a-d]\)[\s\S]\*?)(?=\d+\.\s|\Z))'  
  
  
# Find all matches for the defined pattern in the sample text.  
matches = re.findall(pattern\_for\_breaking\_at\_a, sample\_text, re.DOTALL)  
  
# Define the output directory and filename  
output\_dir = r"C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Area\history ancient"  
file\_name = "combined\_questions1.txt"  
output\_file\_path = os.path.join(output\_dir, file\_name)  
  
# Open the output file once and write all matches  
with open(output\_file\_path, 'w', encoding='utf-8') as file:  
 for index, match in enumerate(matches, start=1):  
 # question\_text = f"Question {index}:\n{match.strip()}\n"  
 pattern\_for\_breaking\_at\_a = r'(\d+\.\s+(?:[\s\S]\*?\([a-d]\)[\s\S]\*?)(?=\d+\.\s|\Z))'  
 matches = re.findall(pattern\_for\_breaking\_at\_a, sample\_text, re.DOTALL)  
 question\_text = f"\n{match.strip()}\n"  
 # Write each question to the output file  
 file.write(question\_text)  
 file.write("---\n") # Add a separator after each question  
  
 # Print the question text to the terminal  
 print(question\_text)  
 print("---\n") # Print a separator after each question in the terminal  
  
print(f"First time all questions have been combined and written to {output\_file\_path}")  
  
# Define the output directory and filename  
output\_dir = r"C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Area\history ancient"  
file\_name = "combined\_questions2.txt"  
output\_file\_path = os.path.join(output\_dir, file\_name)  
  
# Open the output file once and write all matches  
# Open the output file once and write all matches  
with open(output\_file\_path, 'w', encoding='utf-8') as file:  
 for match in matches:  
 # Write each question directly to the output file without the "Question X:" prefix  
 file.write(f"{match.strip()}\n")  
 file.write("---\n") # Add a separator after each question  
  
print(f"Second time all questions have been combined and written to {output\_file\_path}")

1. **test\_pattern\_with\_direct\_text.py** running successfully as on morning 15 Apr, 2024 is:

import re  
  
sample\_text = """  
  
  
Question 61: 61. In ancient Indian Buddhist monasteries, a ceremony called  
Pavarana used to be held. It was the: [2002]  
(a) occasion to elect the Sanghparinayaka and two  
speakers one on Dhamma and the other on Vinaya  
(b) confession by monks of their offences committed  
during their stay in the monasteries during the rainy  
season  
(c) ceremony of initiation of new person into the Buddhist  
Sangha in which the head is shaved and when yellow  
robes are offered  
(d) gathering of Buddhist monks on the next day to the  
full-moon day of Ashadha when they take up a fixed  
abode for the next four months of the rainy season  
62. The ancient Indian play Mudrarakshasa of Visakhadutt has  
its subject on: [2002]  
(a) a conflict between Gods and Demons of ancient Hindu  
lore  
(b) a romantic story of an Aryan prince and a tribal woman  
(c) the story of the power struggle between two Aryan .  
  
  
  
"""  
  
  
  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?Select the correct answer using the codes given below:[\s\S]\*?\(d\)[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?:Select|Codes|Answer).\*?(?:\(a\)[\s\S]\*?\(b\)[\s\S]\*?\(c\)[\s\S]\*?\(d\)[\s\S]\*?))(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?Select the correct answer using the codes given below:[\s\S]\*?\(d\)[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?\(d\)[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)[\s\S]\*?\(b\)[\s\S]\*?\(c\)[\s\S]\*?\(d\))[ \t]\*(?:\n|$))'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|$)'  
# pattern\_for\_list\_type1 = r'(\d+\..+?)\s+(\d+\.\s+.+?\d+\.\s+.+?)\s+(.\*)'  
# pattern\_for\_list\_type1 = r'(?<!\n)(\b\d+\.\s+)([A-Z\(].\*?)(?=\s\*\b\d+\.|$)'  
# pattern\_for\_list\_type1 = r'(\b\d+\.\s+)([A-Z][^\.\n]+)(?=\s\*\b\d+\.|$)'  
# pattern\_for\_list\_type1 = r'\b\d+\.\s+[A-Z][^\.\n]+'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?)(?=\n+\d+\.|$)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?=\n+\d+\.|$))'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|$)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?)(?=\n+\d+\.|$)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\([a-d]\)|\([1-4]\))[ \t]\*[\s\S]\*?)(?=\n+\d+\.|$)'  
# pattern\_for\_question = r'(\d+\.\s+[\s\S]\*?\[.\*?\])'  
# pattern\_for\_options = r'(?:Select the correct answer using the codes given below:\n[a-d]\))([\s\S]\*?)(?=\n{2,}|$)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!^\d+\.\s|\Z)[\s\S])+)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!(?:\d+\.\s|\Z))[\s\S])+?)\n(?=\d+\.\s|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!\d+\.\s)[\s\S])+?)(?=\n\d+\.\s|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!\d+\.\s)[\s\S])+)(?=\n\d+\.\s|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!\d+\.\s)[\s\S](?!\n\d+\.\s))+)(?=\n\d+\.\s|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!\n\d+\.\s)[\s\S])+)(?=\n\d+\.\s|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!\n\d+\.\s)[\s\S])+(?=\n\d+\.\s|\Z))'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!\n\d+\.\s)[\s\S])+)(?=\n\d+\.\s|\Z|\n{2,})'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!\n\d+\.\s|^\d+\.\s)[\s\S])+)(?=\n\d+\.\s|\Z|\n{2,})'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?)\s\*((?:(?!\n\d+\.\s)[^\n])+(?=\n\d+\.\s|\Z|\n{2,}))'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?[\s\S]\*?)(?=\n+\d+\.\s+\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select)[\s\S]\*?)(?=\n+\d+\.\s+|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select)[\s\S]\*?)(?=\s\*\d+\.\s+|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?[Select|correct answer|codes][\s\S]\*?\(d\)[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)' # 2/5  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[^\n]\*(?:\n(?! \*\(\w\)).\*)\*?\n(?:\(a\)|\(b\)|\(c\)|\(d\))[^\n]\*(?i:select|codes|answer)[^\n]\*[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[^\n?]\*(?:\n(?! \*\(\w\)).\*)\*?\n(?:\(a\)|\(b\)|\(c\)|\(d\))[^\n]\*(?i:select|codes|answer)[^\n]\*[\s\S]\*?\?\n)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z$)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\d+\.|\Z$)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
#pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\. [\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\d+\. |\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\. [\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n\d+\. |\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n\d+\. |\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\([abcd]\))[\s\S]\*?)(?=\n\d+\. |\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n\d+\. |\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n\d+\. |\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+(?:(?!\d+\.\s).)\*?(?i:select|codes|answer)(?:(?!\d+\.\s).)\*?(?:\(a\)\s+.\*?\(b\)\s+.\*?\(c\)\s+.\*?\(d\)\s+.\*?))(?=\s\*\n+\d+\.|\s\*\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+(?:(?!\d+\.\s)[\s\S])\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)\s\*[\s\S]\*?\(b\)\s\*[\s\S]\*?\(c\)\s\*[\s\S]\*?\(d\)\s\*[\s\S]\*?))(?=\n+\d+\.\s|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n\d+\. |\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+(?:(?!\d+\.\s)[\s\S])\*?(?i:select|codes|answer)\s\*(?:(?!\d+\.\s)[\s\S])\*?(?:\(a\)\s\*[\s\S]\*?\(b\)\s\*[\s\S]\*?\(c\)\s\*[\s\S]\*?\(d\)\s\*[\s\S]\*?))(?=\n+\d+\.\s\*|\Z)'  
# pattern = r'(Unit - A: History of India\n1 \n\nA 2)'  
# pattern\_for\_all\_questions = r'(\d+\.\s+(?:.\*?(?:\n\s\*)?)\*?(?=\(d\)(?:(?!\([abcd]\)).)\*\n\*\d+\.\s|\Z))'  
pattern\_for\_all\_questions = r'(\d+\.\s+(?:.\*?(?:\n\s\*)?)\*?(?=\(d\)(?:(?![\s\S]\*?\([abcd]\)).)\*\n\*\d+\.\s|\Z))'  
  
  
  
"""pattern\_for\_list\_type1 = (  
 r'(\d+\.\s+[\s\S]\*?'  
 r'(?i:select|codes|answer)[\s\S]\*?'  
 r'(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)'  
 r'(?=\n+\d+\.|\Z)'  
)"""  
  
  
print(f"Using pattern for all questions: {pattern\_for\_all\_questions}")  
  
matches = re.findall(pattern\_for\_all\_questions, sample\_text)  
print(f"Found {len(matches)} matches:")  
"""for match in matches:  
 print(match.strip())  
 print("---")  
 # question = match[0].strip()  
 # options = match[1].strip()  
 # print(question)  
 # print(options)  
 # print("---")"""  
  
# Get the total number of parts  
# total\_matches = len(matches)  
print(f"Found {len(matches)} parts:")  
for index, match in enumerate(matches, start=1):  
 print(f"Part {index}:")  
 print(match.strip())  
 print("---")  
  
# Print each part with its reverse index number  
for index, match in enumerate(matches, start=1):  
 # reverse\_index = total\_matches - index + 1  
 # print(f"Match {reverse\_index}: {match}")  
 print(f"Match {index}: {match}")  
  
# Get the total number of parts  
"""total\_parts = len(parts)  
  
# Print each part with its reverse index number  
for index, part in enumerate(parts, start=1):  
 reverse\_index = total\_parts - index + 1  
 print(f"Part {reverse\_index}: {part}")"""  
  
"""specific\_question\_number = "1."  
specific\_question\_pattern = re.escape(specific\_question\_number) + r'\s+.\*?(?=\n\d+\.|\Z)'  
specific\_question\_match = re.search(specific\_question\_pattern, sample\_text, re.DOTALL)  
if specific\_question\_match:  
 specific\_question\_text = specific\_question\_match.group().strip()  
 print(f"Specific Question {specific\_question\_number}:")  
 print(specific\_question\_text)"""  
  
  
text\_after\_question\_1 = re.search(r'1\.(.\*?)(?=\n\d+\.|\Z)', sample\_text, re.DOTALL)  
if text\_after\_question\_1:  
 text\_after\_question\_1 = text\_after\_question\_1.group(1).strip()  
 print("Text after question 1:")  
 print(text\_after\_question\_1)

1. **fresh\_excel\_creator.py** which running successfully as on evening of 15 Apr, 2024 is:

import pandas as pd  
from pathlib import Path  
import importlib.util  
import os  
from openpyxl import load\_workbook  
from openpyxl.styles import Font  
  
def load\_module(file\_name, directory):  
 module\_name = file\_name.rstrip('.py') # Remove .py to get the module name  
 file\_path = os.path.join(directory, file\_name) # Use the provided directory  
  
 spec = importlib.util.spec\_from\_file\_location(module\_name, file\_path)  
 module = importlib.util.module\_from\_spec(spec)  
 spec.loader.exec\_module(module)  
 return module  
  
print("Select the dictionary to process:")  
print("1. list\_type2\_disha\_26\_2020\_upsc\_pre\_gs1\_history\_ancient\_que")  
print("2. r\_and\_a\_type\_disha\_26\_2020\_upsc\_pre\_gs1\_history\_ancient\_que")  
print("3. second\_stage\_combined\_disha\_26\_2020\_upsc\_pre\_gs1\_history\_ancient\_que")  
print("4. hia\_list\_of\_undesirable\_characters\_for\_question\_bank")  
print("5. third\_stage\_combined\_disha\_26\_2020\_upsc\_pre\_gs1\_history\_ancient\_que")  
print("6. first\_round\_dict\_disha\_26\_2020\_upsc\_pre\_gs1\_history\_ancient\_que") # New option  
print("7. second\_round\_dict\_disha\_26\_2020\_upsc\_pre\_gs1\_history\_ancient\_que") # New option  
print("8. refined\_questions\_dictionary") # New option  
print("9. only\_refined\_questions") # New option  
print("11. disha\_26\_2020\_upsc\_pre\_gs1\_history\_ancient\_que\_answer\_options\_extracted\_dictionary") # New option  
  
user\_choice = input("Enter your choice (1/2/3/4/5/6/7/8/9/11): ") # Update the input options  
  
file\_map\_dict = {  
 "1": {  
 "file\_name": "list\_type2\_disha\_26\_2020\_upsc\_pre\_gs1\_history\_ancient\_que.py",  
 "directory": r"C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Area\history ancient"  
 },  
 "2": {  
 "file\_name": "r\_and\_a\_type\_disha\_26\_2020\_upsc\_pre\_gs1\_history\_ancient\_que.py",  
 "directory": r"C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Area\history ancient"  
 },  
 "3": {  
 "file\_name": "second\_stage\_combined\_disha\_26\_2020\_upsc\_pre\_gs1\_history\_ancient\_que.py",  
 "directory": r"C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Area\history ancient"  
 },  
 "4": {  
 "file\_name": "hia\_list\_of\_undesirable\_characters\_for\_question\_bank.py",  
 "directory": r"C:\Users\PRATIK HAJELA\PycharmProjects\question\_bank\_creation\_from\_pdf"  
 },  
 "5": {  
 "file\_name": "third\_stage\_combined\_disha\_26\_2020\_upsc\_pre\_gs1\_history\_ancient\_que.py",  
 "directory": r"C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Area\history ancient"  
 },  
 "6": { # New entry for the newly created dictionary  
 "file\_name": "disha\_26\_2020\_upsc\_pre\_gs1\_history\_ancient\_que\_first\_round\_dict.py",  
 "directory": r"C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Area\history ancient"  
 },  
 "7": { # New entry for the second round dictionary  
 "file\_name": "disha\_26\_2020\_upsc\_pre\_gs1\_history\_ancient\_que\_second\_round\_dict.py",  
 "directory": r"C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Area\history ancient"  
 },  
 "8": { # New entry for the second round dictionary  
 "file\_name": "refined\_questions\_dictionary.py",  
 "directory": r"C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Area\history ancient"  
 },  
 "9": { # New entry for the second round dictionary  
 "file\_name": "only\_refined\_questions.py",  
 "directory": r"C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Area\history ancient"  
 },  
 "11": { # New entry for the second round dictionary  
 "file\_name": "disha\_26\_2020\_upsc\_pre\_gs1\_history\_ancient\_que\_answer\_options\_extracted\_dictionary.py",  
 "directory": r"C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Area\history ancient"  
 }  
}  
  
  
if user\_choice in file\_map\_dict:  
 selected\_file = file\_map\_dict[user\_choice]["file\_name"]  
 directory = file\_map\_dict[user\_choice]["directory"]  
 selected\_module = load\_module(selected\_file, directory)  
 questions\_dict = getattr(selected\_module, selected\_file[:-3]) # Use the file name without '.py' as the dictionary name  
  
 if user\_choice == "6": # Specific handling for first\_round\_dict  
 question\_details = [v for v in questions\_dict.values()] # Extract values to a list of dictionaries  
 df = pd.DataFrame(question\_details) # Convert list of dictionaries directly to DataFrame  
  
 elif user\_choice == "4": # Handling for undesirable characters  
 df = pd.DataFrame([(k,  
 v.get('character\_update\_number', 0),  
 v.get('character\_add\_source', ''),  
 v.get('character\_add\_date', ''),  
 v.get('replacement', ''),  
 v.get('replacement\_update\_number', 0),  
 v.get('replacement\_add\_source', ''),  
 v.get('replacement\_add\_date', ''))  
 for k, v in questions\_dict['unwanted\_characters'].items()],  
 columns=['unwanted\_character',  
 'character\_update\_number',  
 'character\_add\_source',  
 'character\_add\_date',  
 'replacement',  
 'replacement\_update\_number',  
 'replacement\_add\_source',  
 'replacement\_add\_date'])  
  
 else:  
 df = pd.DataFrame.from\_dict(questions\_dict, orient='index')  
  
 # Define the output Excel file path  
 output\_excel\_file\_path = Path(directory) / (selected\_file[:-3] + ".xlsx")  
 df.to\_excel(output\_excel\_file\_path, index=False) # Save the DataFrame to an Excel file  
  
 # Format hyperlinks, if applicable  
 if user\_choice in ["6", "7", "8", "9"]: # Add hyperlinks for first and second round dicts  
 hyperlink\_column = 'D' if user\_choice in ["6", "7", "8", "9"] else 'I'  
 wb = load\_workbook(output\_excel\_file\_path)  
 ws = wb.active  
 for row in range(2, ws.max\_row + 1):  
 cell = ws[f"{hyperlink\_column}{row}"]  
 if cell.value and isinstance(cell.value, str): # Ensure the value is a string before setting the hyperlink  
 cell.hyperlink = cell.value  
 cell.font = Font(color='0000FF', underline='single')  
 cell.value = "Click Here"  
 wb.save(output\_excel\_file\_path)  
  
 print(f"Excel file '{output\_excel\_file\_path.name}' has been created in {directory}")  
else:  
 print("Invalid choice. Exiting.")

1. **fresh\_text\_cleaner.py** which while running successfully as on morning of 15 Apr, 2024 is:

import re  
from bs4 import BeautifulSoup  
import os  
from pathlib import Path  
  
import warnings  
# Suppress BeautifulSoup's MarkupResemblesLocatorWarning  
warnings.filterwarnings("ignore", category=UserWarning, module='bs4')  
warnings.filterwarnings("ignore")  
  
  
unwanted\_text\_pattern = r'(?:^Page \d+:[^\n]\*\n|^\s\*Page \d+\s\*\n|^\s\*Unit - A: History of India\s\*\n|^\s\*Ancient History\s\*\n|^\s\*edieval History|^\s\*Topicwise Solved Papers|^\s\*Medieval History |^\s\*Gq |^\s\*GED |^\s\*GE»|\s\*\|\s\*Unit - A: Histo|\s\*\|\s\*Unit - A: History)'  
  
def load\_dictionary(file\_path):  
 with open(file\_path, 'r', encoding='utf-8') as file:  
 content = file.read()  
 dictionary = eval(content.split('=', 1)[1].strip())  
 return dictionary  
  
def clean\_and\_correct\_question\_text(text):  
 # Check for empty strings and return immediately to avoid unnecessary processing  
 if text is None:  
 return None # Or return an empty string if that's more appropriate for your use case  
  
 # Use BeautifulSoup to remove HTML tags and entities  
 soup = BeautifulSoup(text, "html.parser")  
 cleaned\_text = soup.get\_text()  
  
 # Encode and decode to remove non-ASCII characters added on 7 Apr 2024  
 cleaned\_text = cleaned\_text.encode('ascii', 'ignore').decode('ascii')  
  
 # General replacement for variations of "l." with "1."  
 # cleaned\_text = re.sub(r'\b[l1]\.\s\*', '1. ', cleaned\_text)  
  
 return cleaned\_text  
def clean\_and\_correct\_pre\_process\_text(text, input\_file\_path):  
 # Rest of the function remains the same  
  
 # Check for empty strings and return immediately to avoid unnecessary processing  
 if not text:  
 return text  
  
 # Use BeautifulSoup to remove HTML tags and entities  
 soup = BeautifulSoup(text, "html.parser")  
 cleaned\_text = soup.get\_text()  
  
 # Define a list of tuples where each tuple is a (pattern, replacement)  
 replacements = [  
 ('\u2014', '-'),  
 ('&amp;', '&'),  
 ('&lt;', '<'),  
 ('&gt;', '>'),  
 ('&quot;', '"'),  
 ('&#39;', "'"),  
 ('&nbsp;', ' '),  
 ('@d);', '(d)'),  
 ('Il.', '1.'),  
 ('land2', '1. and 2.'),  
 ('@d)', '(d)'),  
 ('(@) \_)', '(d)'),  
 ('@)', '(d)'),  
 ('()', '(d)'),  
 ('(ce)', '(c)'),  
 ('(dd)', '(d)'),  
 ('(@ (i)', '(d) (i)'),  
 ('(() (a))', '(d) (i)'),  
 ('((11) C)', '(ii) C'),  
 ('(dG))', '(d)'),  
 ('â€˜', "'"),  
 ('â€™', "'"),  
 ('â€œ', '"'),  
 ('â€', '"'),  
 (r'Unit - A: History of India\s\*1\s\*\n\s\*A 2', ''),  
 (r'\|\s\*', ''), # Remove '|' followed by optional whitespace  
 ('EBD\_8342', ''), # Add this line to remove 'EBD\_8342'  
 (r'EBD\_8342\s\*', ''), # Remove 'EBD\_8342' followed by any whitespace including new lines  
 (r'Ancient History [A-Z]\d+', ''), # Remove patterns like Ancient History A3  
 (r'\(Cc\)', '(c)'), # Replace (Cc) with (c)  
 ('lndia', 'India'), # Type 1 correction  
  
 (r'Unit - A: History of India\n1 Ancient History\n\nTopicwise Solved PapersA2', '\n'),  
 (r'EBD\_8342\n\nAncient History A3\n', '\n'),  
 (r'Topicwise Solved PapersA4\n', '\n'),  
 (r'EBD\_8342\n\nAncient History A5\n', '\n'),  
 (r'Topicwise Solved PapersA6\n', '\n'),  
 (r'EBD\_8342\n\nAncient History A7\n', '\n'),  
 (r'Topicwise Solved PapersA8\n', '\n'),  
 (r'EBD\_8342\n\nAncient History A9\n', '\n'),  
 ('Ancient History', ''), # Remove 'Ancient History'  
 ('A 3', ''), # Remove 'A 3'  
 (r'Topicwise Solved PapersA 2', ''), # Remove 'Topicwise Solved PapersA' followed by a number  
 (r'Topicwise Solved PapersA4', ''), # Remove ''Topicwise Solved PapersA' followed by a number'Topicwise Solved PapersA4'  
 (r'Ancient History A5', ''), # Remove 'Ancient History A5'  
 (r'Topicwise Solved PapersA6', ''), # Remove 'Topicwise Solved PapersA6'  
 (r'Ancient History A7', ''), # Remove 'Ancient History A7'  
 (r'Topicwise Solved PapersA8', ''), # Remove 'Topicwise Solved PapersA8'  
 (r'Ancient History A9', ''), # Remove 'Ancient History A9'  
 (r'Ancient History A3', ''), # Remove 'Ancient History A3'  
 (r'Topicwise Solved PapersA4', ''), # Remove 'Topicwise Solved PapersA4'  
 (r'Topicwise Solved Papers', ''), # Remove 'Topicwise Solved Papers'  
 (r'Topicwise Solved PapersA2', ''), # Remove 'Topicwise Solved PapersA2'  
 (r'Ancient History A9\s\*', ''), # Remove 'Ancient History A9' followed by any whitespace including new lines  
 (r'EBD\_\d+', ''), # Assuming 'EBD\_' followed by any number  
 (r'Ancient History [A-Z]\d+', ''), # Catch 'Ancient History A5', 'Ancient History B3', etc.  
 (r'\n+', '\n') # Normalize newlines in case multiple replacements cause extra line breaks  
  
 ]  
  
 # replacements.append((r'Unit - A: History of India\s\*1\s\*\n\s\*A 2', ''))  
  
 replaced\_characters = set()  
  
 # Apply replacements  
 for pattern, replacement in replacements:  
 if pattern in cleaned\_text:  
 replaced\_characters.add((pattern, replacement))  
 print(f"Replacing {pattern} with {replacement}")  
 cleaned\_text = cleaned\_text.replace(pattern, replacement)  
 input("Press Enter to continue...")  
  
 # Correction rules for numeric and list patterns  
 correction\_rules = [  
 (r'(\d),(\d)and(\d)', r'\1, \2 and \3'),  
 (r'(\d),(\d)', r'\1, \2'),  
 (r'(\d),(\d),(\d),(\d)', r'\1, \2, \3, \4'),  
 (r'(\d)\s\*and(\d)', r'\1 and \2'), # Separate 'and' from numbers  
 (r'(\d)([a-zA-Z])', r'\1 \2'), # Separate numbers that stick to letters  
 (r'([a-zA-Z])(\d)', r'\1 \2'), # Separate letters that stick to numbers  
 (r'land(\d)', r'1 and \1'), # Replace 'land' with '1 and'  
 (r'(\d)and(\d)', r'\1 and \2'), # Add space between number and 'and'  
 (r'(\d);([ABCDabcd])', r'\1; \2'), # Add a space after the semicolon  
 ]  
  
 # Apply each correction rule to the cleaned text  
 for pattern, replacement in correction\_rules:  
 cleaned\_text = re.sub(pattern, replacement, cleaned\_text)  
  
 # After replacements are done  
 if replaced\_characters:  
 print("The following unwanted characters were replaced:")  
 for char, rep in replaced\_characters:  
 print(f"Character: {char} - Replaced with: {rep}")  
  
 print(  
 f"These characters were saved in the file 'unwanted\_characters.txt' in the folder '{os.path.dirname(input\_file\_path)}'")  
  
 with open(os.path.join(os.path.dirname(input\_file\_path), 'unwanted\_characters.txt'), 'a',  
 encoding='utf-8') as f:  
 for char, rep in replaced\_characters:  
 f.write(f"{char} - {rep}\n")  
  
 return cleaned\_text  
  
def replace\_undesirable\_characters(input\_file\_path):  
 dictionary\_file\_path = Path(r'C:\Users\PRATIK HAJELA\PycharmProjects\question\_bank\_creation\_from\_pdf\hia\_list\_of\_undesirable\_characters\_for\_question\_bank.py')  
 dictionary = load\_dictionary(dictionary\_file\_path)  
 with open(input\_file\_path, 'r', encoding='utf-8') as file:  
 text = file.read()  
  
 for char, details in dictionary['unwanted\_characters'].items():  
 if details['replacement'] != "":  
 text = text.replace(char, details['replacement'])  
  
 output\_file\_path = input\_file\_path.parent / f"{input\_file\_path.stem}\_fully\_cleaned.txt"  
 with open(output\_file\_path, 'w', encoding='utf-8') as output\_file:  
 output\_file.write(text)  
  
 print(f"Fully cleaned text file '{output\_file\_path.name}' created in '{output\_file\_path.parent}'")  
 return output\_file\_path  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 folder\_path = r"C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Python Programming\Trials\Regex Trials"  
 file\_name = input("Enter the name of the text file (without the .txt extension): ")  
 input\_file\_path = os.path.join(folder\_path, file\_name + '.txt')  
  
 with open(input\_file\_path, 'r', encoding='utf-8') as file:  
 original\_text = file.read()  
  
 cleaned\_text = clean\_and\_correct\_pre\_process\_text(original\_text)  
  
 print("Cleaned text:", cleaned\_text)  
  
 # Save the cleaned text to a new file  
 output\_file\_path = os.path.join(folder\_path, f"{file\_name}\_cleaned.txt")  
 with open(output\_file\_path, 'w', encoding='utf-8') as output\_file:  
 output\_file.write(cleaned\_text)  
 print(f"Cleaned text saved to {output\_file\_path}")

1. **fresh\_questions\_organiser.py** which running successfully as on 10 Apr, 2024 is:

import re  
from datetime import datetime  
from pathlib import Path  
from fresh\_text\_cleaner import clean\_and\_correct\_question\_text, clean\_and\_correct\_pre\_process\_text  
from fresh\_text\_cleaner import unwanted\_text\_pattern  
from fresh\_text\_cleaner import clean\_and\_correct\_question\_text, clean\_and\_correct\_pre\_process\_text, replace\_undesirable\_characters  
  
  
def extract\_matches(pattern, text):  
 question\_pattern = re.compile(pattern)  
 matches = question\_pattern.findall(text)  
 match\_dict = {match.split('.')[0]: match for match in matches}  
 return match\_dict  
  
def clean\_file(input\_file\_path):  
 with open(input\_file\_path, 'r', encoding='utf-8') as file:  
 text = file.read()  
 #cleaned\_text = clean\_and\_correct\_pre\_process\_text(text)  
 cleaned\_text = clean\_and\_correct\_pre\_process\_text(text, input\_file\_path)  
 cleaned\_file\_path = input\_file\_path.parent / ('cleaned\_' + input\_file\_path.name)  
 with open(cleaned\_file\_path, 'w', encoding='utf-8') as cleaned\_file:  
 cleaned\_file.write(cleaned\_text)  
 print(f"Cleaned text file '{cleaned\_file\_path.name}' created in '{cleaned\_file\_path.parent}'")  
 return cleaned\_file\_path  
  
  
def write\_matches\_to\_file(matches, output\_file\_path):  
 with open(output\_file\_path, 'w', encoding='utf-8') as file:  
 for match in matches:  
 file.write(match.strip() + '\n\n---\n\n')  
  
  
def write\_combined\_matches\_to\_file(combined\_dict, output\_file\_path):  
 with open(output\_file\_path, 'w', encoding='utf-8') as file:  
 for key in sorted(combined\_dict.keys(), key=int):  
 file.write(combined\_dict[key].strip() + '\n\n---\n\n')  
  
def write\_dictionary(dictionary, dictionary\_name, output\_file\_path):  
 with open(output\_file\_path, 'w', encoding='utf-8') as file:  
 file.write(f"{dictionary\_name} = {{\n")  
 for key, value in dictionary.items():  
 file.write(f" '{key}': {value},\n")  
 file.write("}\n")  
 print(f"Dictionary file '{output\_file\_path.name}' created in '{output\_file\_path.parent}' with dictionary name '{dictionary\_name}'")  
  
def merge\_dictionaries(first\_dict, second\_dict):  
 merged\_dict = first\_dict.copy()  
 for key, value in second\_dict.items():  
 if key not in merged\_dict:  
 merged\_dict[key] = value  
 return merged\_dict  
  
  
def extract\_list\_type2\_questions(cleaned\_file\_path, input\_file\_name, base\_dir\_path):  
  
 with open(cleaned\_file\_path, 'r', encoding='utf-8') as file:  
 cleaned\_text = file.read()  
 print("Starting extraction...") # Debugging statement  
  
 # Debugging statement to check the beginning of the text  
 print("First 100 characters of the text for pattern matching:")  
 print(cleaned\_text[:100])  
  
 input("Press Enter to continue with pattern processing...")  
  
 first\_pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list)[\s\S]\*?(?i:list)[\s\S]\*?(?i:codes)[\s\S]\*?)(?=\n\*\d+\.\s+|\Z)'  
  
 # second\_pattern\_for\_list\_type2 = r'(\d+\.\s+(?=.\*(?:codes|pairs|matched|With reference to|consider))(?=.\*(?:correctly matched|given above|Which of the|Match|following pairs|consider the following pairs|pairs given above))(?!.\*(?:not form part of|not a part of|not belong to|not correct|not a Harappan Site|phrases defines the nature of))[\s\S]\*?\s\*(?:\([a-d]\)\s\*[\s\S]\*?){2,4})(?=\n+\d+\.|\Z)|(\d+\.\s+With reference to.\*?consider the following pairs:[\s\S]\*?\s\*(?:\([a-d]\)\s\*[\s\S]\*?){2,4})(?=\n+\d+\.|\Z)' # second\_pattern\_for\_list\_type2 = r'(\d+\.\s+(?:(?i:match\s+)?(?:(?:(?i:list)[\s\S]\*?(?i:list))|(?i:lists))[\s\S]\*?(?i:codes)|(?i:pairs))[\s\S]\*?(?:\([a-d]\)\s\*[\s\S]\*?){4}(?=\n\*\d+\.\s+|\Z))'  
 # second\_pattern\_for\_list\_type2 = r'(\d+\.\s+(?=.\*(?:codes|pairs|matched))[\s\S]\*?(?:pairs above|following pairs)[\s\S]\*?\s\*(?:\([a-d]\)\s\*[\s\S]\*?){3,4})(?=\n+\d+\.|\Z)'  
 second\_pattern\_for\_list\_type2 = r'(\d+\.\s+(?=.\*(?:codes|pairs|matched))[\s\S]\*?(?:pairs above|following pairs)[\s\S]\*?\s\*(?:\([a-d]\)\s\*[\s\S]\*?){3,4})(?=\n+\d+\.|\Z)'  
  
 # processing for first\_pattern  
 question\_pattern = re.compile(first\_pattern\_for\_list\_type2)  
 print(f"Using first pattern: {first\_pattern\_for\_list\_type2}")  
 matches = re.findall(question\_pattern, cleaned\_text)  
 print(f"Found {len(matches)} matches.")  
 # Create the output file path based on the first\_pattern  
 output\_file\_path = base\_dir\_path.parent / f"first\_pattern\_for\_list\_type2\_{base\_dir\_path.name}"  
 with open(output\_file\_path, 'w', encoding='utf-8') as file:  
 for match in matches:  
 print("Match object:", match) # Print the entire match object  
 print("Writing match:", match[0]) # Print the match before writing  
 file.write(match.strip() + '\n\n---\n\n') # Write the entire match  
  
 print(f"Processed questions file '{output\_file\_path.name}' created in '{output\_file\_path.parent}'")  
 print(f"Number of matches: {len(matches)}")  
  
 # create and write dictionary  
 first\_pattern\_dict = extract\_matches(first\_pattern\_for\_list\_type2, cleaned\_text)  
 first\_pattern\_dict = {match.split('.')[0]: {'question\_number': match.split('.')[0], 'question': match,  
 'source\_file\_hyperlink': str(cleaned\_file\_path)} for match in matches}  
 first\_pattern\_dict = {match.split('.')[0]: {'question\_number': match.split('.')[0], 'question': match,  
 'source\_file\_hyperlink': str(cleaned\_file\_path),  
 'question\_type': 'list\_type2'} for match in matches}  
 first\_pattern\_file\_path = base\_dir\_path / f"first\_pattern\_for\_list\_type2\_{input\_file\_name}.py"  
 write\_dictionary(first\_pattern\_dict, f"first\_pattern\_for\_list\_type2\_{input\_file\_name}", first\_pattern\_file\_path)  
  
 # processing for second\_pattern  
 question\_pattern = re.compile(second\_pattern\_for\_list\_type2)  
 print(f"Using second pattern: {second\_pattern\_for\_list\_type2}")  
 matches = re.findall(question\_pattern, cleaned\_text)  
 print(f"Found {len(matches)} matches.")  
 # Create the output file path based on the second\_pattern  
 output\_file\_path = base\_dir\_path.parent / f"second\_pattern\_for\_list\_type2\_{base\_dir\_path.name}"  
 with open(output\_file\_path, 'w', encoding='utf-8') as file:  
 for match in matches:  
 if isinstance(match, tuple):  
 match\_str = next(filter(None, match)) # Extract the non-empty string from the tuple  
 else:  
 match\_str = match  
 print("Match object:", match\_str) # Print the entire match object  
 file.write(match\_str.strip() + '\n\n---\n\n') # Write the entire match  
  
 print(f"Processed questions file '{output\_file\_path.name}' created in '{output\_file\_path.parent}'")  
 print(f"Number of matches: {len(matches)}")  
  
 # create and write dictionary  
 second\_pattern\_dict = extract\_matches(second\_pattern\_for\_list\_type2, cleaned\_text)  
 second\_pattern\_dict = {match.split('.')[0]: {'question\_number': match.split('.')[0], 'question': match,  
 'source\_file\_hyperlink': str(cleaned\_file\_path)} for match in matches}  
 second\_pattern\_dict = {match.split('.')[0]: {'question\_number': match.split('.')[0], 'question': match,  
 'source\_file\_hyperlink': str(cleaned\_file\_path),  
 'question\_type': 'list\_type2'} for match in matches}  
  
 second\_pattern\_file\_path = base\_dir\_path / f"second\_pattern\_for\_list\_type2\_{input\_file\_name}.py"  
 write\_dictionary(second\_pattern\_dict, f"second\_pattern\_for\_list\_type2\_{input\_file\_name}", second\_pattern\_file\_path)  
  
 # Combine dictionaries  
 list\_type2\_dict = {\*\*first\_pattern\_dict, \*\*second\_pattern\_dict}  
 list\_type2\_dict\_file\_path = base\_dir\_path / f"list\_type2\_{input\_file\_name}.py"  
 write\_dictionary(list\_type2\_dict, f"list\_type2\_{input\_file\_name}", list\_type2\_dict\_file\_path)  
  
 return list\_type2\_dict  
  
 # At the end of the function, before returning:  
 return list\_type2\_dict  
  
def extract\_r\_and\_a\_type\_questions(cleaned\_file\_path, input\_file\_name, base\_dir\_path):  
  
 with open(cleaned\_file\_path, 'r', encoding='utf-8') as file:  
 cleaned\_text = file.read()  
 print("Starting extraction of R and A type questions...")  
  
 pattern\_for\_r\_and\_a\_type = r'(\d+\.\s+Assertion\s+\(A\):[\s\S]\*?Reason\s+\(R\):[\s\S]\*?\(a\)[\s\S]\*?\(b\)[\s\S]\*?\(c\)[\s\S]\*?\(d\)[\s\S]\*?)(?=\n\*\d+\.\s+|\Z)'  
  
 question\_pattern = re.compile(pattern\_for\_r\_and\_a\_type)  
 print(f"Using pattern: {pattern\_for\_r\_and\_a\_type}")  
 matches = re.findall(question\_pattern, cleaned\_text)  
 print(f"Found {len(matches)} matches.")  
  
 output\_file\_path = base\_dir\_path / f"r\_and\_a\_type\_{input\_file\_name}.txt"  
 with open(output\_file\_path, 'w', encoding='utf-8') as file:  
 for match in matches:  
 file.write(match.strip() + '\n\n---\n\n')  
 print(f"Processed R and A type questions file '{output\_file\_path.name}' created in '{output\_file\_path.parent}'")  
  
 r\_and\_a\_type\_dict = extract\_matches(pattern\_for\_r\_and\_a\_type, cleaned\_text)  
 r\_and\_a\_type\_dict = {match.split('.')[0]: {'question\_number': match.split('.')[0], 'question': match,  
 'source\_file\_hyperlink': str(cleaned\_file\_path)} for match in matches}  
 r\_and\_a\_type\_dict = {match.split('.')[0]: {'question\_number': match.split('.')[0], 'question': match,  
 'source\_file\_hyperlink': str(cleaned\_file\_path),  
 'question\_type': 'r\_and\_a\_type'} for match in matches}  
  
 r\_and\_a\_type\_dict\_file\_path = base\_dir\_path / f"r\_and\_a\_type\_{input\_file\_name}.py"  
 write\_dictionary(r\_and\_a\_type\_dict, f"r\_and\_a\_type\_{input\_file\_name}", r\_and\_a\_type\_dict\_file\_path)  
  
 return r\_and\_a\_type\_dict  
  
def extract\_list\_type1\_questions(cleaned\_file\_path, input\_file\_name, base\_dir\_path):  
 with open(cleaned\_file\_path, 'r', encoding='utf-8') as file:  
 cleaned\_text = file.read()  
 print("Starting extraction of list type 1 questions...")  
  
 # pattern\_for\_list\_type1 = r'(\d+\.\s+(?=.\*\b[1-5]\.\s)[\s\S]\*?\(a\)[\s\S]\*?\(b\)[\s\S]\*?\(c\)[\s\S]\*?\(d\)[\s\S]\*?)(?=\n+\d+\.|\Z)'  
 # pattern\_for\_list\_type1 = r'(\d+\.\s+(?:\d+\.\s+[\w\s]+)+.\*?(?:\(a\)[\s\S]\*?\(b\)[\s\S]\*?\(c\)[\s\S]\*?\(d\)[\s\S]\*?))(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type1 = r'(\d+\.\s+(?:\d+\.\s+[\w\s,]+[\.\;])+.\*?(?:\(a\)[\s\S]\*?\(b\)[\s\S]\*?\(c\)[\s\S]\*?\(d\)[\s\S]\*?))(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type1 = r'(\d+\.\s+.\*?(?:\d+\.\s+[\w\s,]+)+.\*?Select the correct.\*?(?:\(a\)[\s\S]\*?\(b\)[\s\S]\*?\(c\)[\s\S]\*?\(d\)[\s\S]\*?))(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type1 = r'(\d+\.\s+(?:[\w\s,]+[\d]+\.\s+)+.\*?(?:Select|Which|Consider).\*?(?:\(a\)[\s\S]\*?\(b\)[\s\S]\*?\(c\)[\s\S]\*?\(d\)[\s\S]\*?))(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type1 = r'(\d+\.\s+(?=.\*\b(?:[1-5]|1[0-5])\.\s)[\s\S]\*?(?:Select|Which|Consider|Select the correct|What|The correct|Which of the following).\*?(?:\(a\)[\s\S]\*?\(b\)[\s\S]\*?\(c\)[\s\S]\*?\(d\)[\s\S]\*?))(?=\n+\d+\.|\Z)'  
 # pattern\_for\_list\_type1 = r'(\d+\.\s+(?=.\*\d+\.\s.\*\d+\.\s)[\s\S]\*?(?:Select|Which).\*?(?:\(a\)[\s\S]\*?\(b\)[\s\S]\*?\(c\)[\s\S]\*?\(d\)[\s\S]\*?))(?=\n+\d+\.|\Z)'  
 # pattern\_for\_list\_type1 = r'(\d+\.\s+(?=.\* ?\d+\.\s.\* ?\d+\.\s)[\s\S] \* ?(?:Select | Which). \*?(?:\([a-d]\)\s \* [\s\S] \* ?){3, 4})(?=\n+\d+\.| \Z)'  
 # pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
 pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)' # 2/5  
  
 question\_pattern = re.compile(pattern\_for\_list\_type1)  
 print(f"Using pattern for list\_type1 : {pattern\_for\_list\_type1}")  
 matches = re.findall(question\_pattern, cleaned\_text)  
 print(f"Found {len(matches)} matches.")  
 for match in matches:  
 print(match.strip())  
 print("---")  
  
 output\_file\_path = base\_dir\_path / f"list\_type1\_{input\_file\_name}.txt"  
 with open(output\_file\_path, 'w', encoding='utf-8') as file:  
 for match in matches:  
 file.write(match.strip() + '\n\n---\n\n')  
 print(f"Processed list type 1 questions file '{output\_file\_path.name}' created in '{output\_file\_path.parent}'")  
  
 list\_type1\_dict = extract\_matches(pattern\_for\_list\_type1, cleaned\_text)  
 list\_type1\_dict = {match.split('.')[0]: {'question\_number': match.split('.')[0], 'question': match,  
 'source\_file\_hyperlink': str(cleaned\_file\_path),  
 'question\_type': 'list\_type1'} for match in matches}  
  
 list\_type1\_dict\_file\_path = base\_dir\_path / f"list\_type1\_{input\_file\_name}.py"  
 write\_dictionary(list\_type1\_dict, f"list\_type1\_{input\_file\_name}", list\_type1\_dict\_file\_path)  
  
 return list\_type1\_dict  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 input\_file\_name = input("Enter the name of the text file (without the .txt extension): ")  
 parts = input\_file\_name.split('\_')  
 subfolder\_name = parts[6] + ' ' + parts[7]  
 base\_dir\_path = Path(r'C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Area') / subfolder\_name.lower()  
  
 print(f"Input file name: {input\_file\_name}")  
 print(f"Parts: {parts}")  
 for i, part in enumerate(parts):  
 print(f"Part {i + 1}: {part}")  
 print(f"Subfolder name: {subfolder\_name}")  
 print(f"Base directory path: {base\_dir\_path}")  
  
 input\_file\_path = base\_dir\_path / (input\_file\_name + '.txt')  
 print(f"Input file path: {input\_file\_path}")  
  
 if input\_file\_path.exists():  
 print(f"Input file found in {input\_file\_path.parent}")  
 cleaned\_file\_path = clean\_file(input\_file\_path) # Call the clean\_file function here  
 print(f"Cleaned file path: {cleaned\_file\_path}")  
 # cleaned\_file\_path = input\_file\_path  
  
 input("Informing that the replace\_undesirable\_characters module will be used next. Press Enter to continue...")  
 fully\_cleaned\_file\_path = replace\_undesirable\_characters(cleaned\_file\_path)  
 print(f"Fully cleaned file path: {fully\_cleaned\_file\_path}")  
  
 # Rest of the code...  
 list\_type2\_dict = extract\_list\_type2\_questions(cleaned\_file\_path, input\_file\_name, base\_dir\_path)  
 # Prompt for extracting R and A type questions  
 r\_and\_a\_type\_dict = extract\_r\_and\_a\_type\_questions(cleaned\_file\_path, input\_file\_name, base\_dir\_path)  
 # Combine dictionaries for second stage  
 second\_stage\_combined\_dict = {\*\*list\_type2\_dict, \*\*r\_and\_a\_type\_dict}  
 second\_stage\_combined\_dict\_file\_path = base\_dir\_path / f"second\_stage\_combined\_{input\_file\_name}.py"  
 write\_dictionary(second\_stage\_combined\_dict, f"second\_stage\_combined\_{input\_file\_name}",  
 second\_stage\_combined\_dict\_file\_path)  
 # Extract list type 1 questions  
 list\_type1\_dict = extract\_list\_type1\_questions(cleaned\_file\_path, input\_file\_name, base\_dir\_path)  
 # Combine dictionaries for third stage  
 third\_stage\_combined\_dict = {\*\*second\_stage\_combined\_dict, \*\*list\_type1\_dict}  
 third\_stage\_combined\_dict\_file\_path = base\_dir\_path / f"third\_stage\_combined\_{input\_file\_name}.py"  
 write\_dictionary(third\_stage\_combined\_dict, f"third\_stage\_combined\_{input\_file\_name}",  
 third\_stage\_combined\_dict\_file\_path)  
  
 else:  
 print(f"Input file not found in {input\_file\_path.parent}. Please check the file name and location.")

1. earlier version of **test\_pattern\_with\_direct\_text.py** kept only for reference

import re  
  
sample\_text = """  
35. What is the correct chronological order in which the following appeared in India? [1998]  
1. Gold coins  
2. Punch marked silver coins  
3. Iron plough  
4. Urban culture  
Select the correct answer using the codes given below:  
(a) 3, 4, 1, 2 (b) 3, 4, 2, 1  
(c) 4, 3, 1, 2 (d) 4, 3, 2, 1  
  
14. Which of the following were common to both Buddhism  
and Jainism? [1996]  
1. Avoidance of extremities of penance and enjoyment  
2. Indifference to the authority of the Vedas  
3. Denial of efficacy of rituals  
4. Non-injury to animal life  
Select the answer using the codes given below:  
Codes:  
(a) 1, 2, 3 and 4 (b) 2, 3 and 4  
(c) 1, 3, and 4 (d) 1 and 2  
  
35. What is the correct chronological order in which the  
following appeared in India? [1998]  
1. Gold coins  
2. Punch marked silver coins  
3. Iron plough  
4. Urban culture  
Select the correct answer using the codes given below:  
(a) 3, 4, 1, 2 (b) 3, 4, 2, 1  
(c) 4, 3, 1, 2 (d) 4, 3, 2, 1  
  
42. The following persons came to India at one time or another:  
1. Fa-Hien 2. I-Tsing  
3. Megasthanese 4. Hieun-Tsang  
The correct chronological sequence of their visits is:  
(a) 3, 1, 2, 4 (b) 3, 1, 4, 2 [1999]  
(c) 1, 3, 2, 4 (d) 1, 3, 4, 2  
  
  
64. Consider the following statements:  
1. The Cholas defeated Pandya and Chera rulers and  
established their domination over peninsular India in  
the early medieval times.  
2. The Cholas sent an expedition against Sailendra empire  
of South-East Asia and conquered some of the areas.  
Which of these statements is/are correct? [2003]  
(a) Only 1 (b) Only 2  
(c) Both 1 and 2 (d) Neither 1 nor 2  
  
66. Consider the following statements: [2003]  
1. V ardhaman Mahavira's mother was the daughter of  
Lichchhavi chief Chetaka  
2. Gautama Buddha's mother was a princess from the  
Koshalan dynasty  
3. Parshvanatha, the twenty-third Tirthankara, belonged  
to Benaras  
Which of these statements is/are correct?  
(a) Only 1 (b) Only 2  
(c) 2 and 3 (d) 1, 2 and 3  
  
67. Consider the following statements: [2003]  
1. The last Mauryan ruler, Brihadratha was assassinated  
by his commander-in-chief, Pushyamitra Sunga.  
2. The last Sunga king, Devabhuti was assassinated by  
his Brahmana Minister Vasudeva Kanva who usurped  
the throne.  
3 . The last ruler of the Kanva dynasty was deposed by  
the Andhras.  
Which of these statements is/are correct?  
(a) 1 and 2 (b) Only 2  
(c) Only 3 (d) 1, 2 and 3  
  
68. Consider the following statements:  
1. The Chinese pilgrim Fa-Hien attended the fourth Great  
Buddhist Council held by Kanishka.  
2. The Chinese pilgrim Hiuen-Tsang met Harsha and  
found him to be antagonistic to Buddhism.  
Which of the statements given above is/are correct?  
[2004]  
(a) 1 only (b) 2 only  
(c) Both 1 and 2 (d) Neither 1 nor 2  
  
75. Consider the following statements: [2006]  
1 . The Ikshvaku rulers of Southern India were  
antagonistic towards Buddhism.  
2. The Pala rulers of Eastern India were patrons of  
Buddhism.  
Which of the statement(s) given above is/ are correct?  
(a) 1 only (b) 2 only  
(c) Both 1 and 2 (d) Neither 1 nor   
  
82. Regarding the Indus V alley Civilization, consider the  
following statements :  
1. It was predominantly a secular civilization and the  
religious element, though present, did not dominate  
the scene  
2. During this period, cotton was used for manufacturing  
textiles in India [2011 - I]  
Which of the statements given above is/are correct?  
(a) 1 only (b) 2 only  
(c) Both 1 and 2 (d) Neither 1 nor 2  
  
83. The “Dharma” and “Rita” depict a central Idea of ancient  
V edic civilization of India. In this context, consider the  
following statements : [2011 - I]  
1. Dharma was a conception of obligations and of the  
discharge of one’s duties to oneself and to others.  
2. Rita was the fundamental moral law governing the  
functioning of the universe and all it contained.  
Which of the statements given above is/are correct?  
(a) 1 only (b) 2 only  
(c) Both 1 and 2 (d) Neither 1 nor 2  
  
  
  
"""  
  
  
  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?Select the correct answer using the codes given below:[\s\S]\*?\(d\)[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?:Select|Codes|Answer).\*?(?:\(a\)[\s\S]\*?\(b\)[\s\S]\*?\(c\)[\s\S]\*?\(d\)[\s\S]\*?))(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?Select the correct answer using the codes given below:[\s\S]\*?\(d\)[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?\(d\)[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)[\s\S]\*?\(b\)[\s\S]\*?\(c\)[\s\S]\*?\(d\))[ \t]\*(?:\n|$))'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|$)'  
# pattern\_for\_list\_type1 = r'(\d+\..+?)\s+(\d+\.\s+.+?\d+\.\s+.+?)\s+(.\*)'  
# pattern\_for\_list\_type1 = r'(?<!\n)(\b\d+\.\s+)([A-Z\(].\*?)(?=\s\*\b\d+\.|$)'  
# pattern\_for\_list\_type1 = r'(\b\d+\.\s+)([A-Z][^\.\n]+)(?=\s\*\b\d+\.|$)'  
# pattern\_for\_list\_type1 = r'\b\d+\.\s+[A-Z][^\.\n]+'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?)(?=\n+\d+\.|$)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?=\n+\d+\.|$))'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|$)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?)(?=\n+\d+\.|$)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\([a-d]\)|\([1-4]\))[ \t]\*[\s\S]\*?)(?=\n+\d+\.|$)'  
# pattern\_for\_question = r'(\d+\.\s+[\s\S]\*?\[.\*?\])'  
# pattern\_for\_options = r'(?:Select the correct answer using the codes given below:\n[a-d]\))([\s\S]\*?)(?=\n{2,}|$)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!^\d+\.\s|\Z)[\s\S])+)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!(?:\d+\.\s|\Z))[\s\S])+?)\n(?=\d+\.\s|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!\d+\.\s)[\s\S])+?)(?=\n\d+\.\s|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!\d+\.\s)[\s\S])+)(?=\n\d+\.\s|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!\d+\.\s)[\s\S](?!\n\d+\.\s))+)(?=\n\d+\.\s|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!\n\d+\.\s)[\s\S])+)(?=\n\d+\.\s|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!\n\d+\.\s)[\s\S])+(?=\n\d+\.\s|\Z))'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!\n\d+\.\s)[\s\S])+)(?=\n\d+\.\s|\Z|\n{2,})'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?\[.\*?\])\s\*((?:(?!\n\d+\.\s|^\d+\.\s)[\s\S])+)(?=\n\d+\.\s|\Z|\n{2,})'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?)\s\*((?:(?!\n\d+\.\s)[^\n])+(?=\n\d+\.\s|\Z|\n{2,}))'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?[\s\S]\*?)(?=\n+\d+\.\s+\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select)[\s\S]\*?)(?=\n+\d+\.\s+|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select)[\s\S]\*?)(?=\s\*\d+\.\s+|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?[Select|correct answer|codes][\s\S]\*?\(d\)[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)' # 2/5  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[^\n]\*(?:\n(?! \*\(\w\)).\*)\*?\n(?:\(a\)|\(b\)|\(c\)|\(d\))[^\n]\*(?i:select|codes|answer)[^\n]\*[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[^\n?]\*(?:\n(?! \*\(\w\)).\*)\*?\n(?:\(a\)|\(b\)|\(c\)|\(d\))[^\n]\*(?i:select|codes|answer)[^\n]\*[\s\S]\*?\?\n)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z$)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\d+\.|\Z$)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n+\d+\.|\Z)'  
#pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n\d+\.|\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\. [\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\d+\. |\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\. [\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n\d+\. |\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n\d+\. |\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\([abcd]\))[\s\S]\*?)(?=\n\d+\. |\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n\d+\. |\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n\d+\. |\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+(?:(?!\d+\.\s).)\*?(?i:select|codes|answer)(?:(?!\d+\.\s).)\*?(?:\(a\)\s+.\*?\(b\)\s+.\*?\(c\)\s+.\*?\(d\)\s+.\*?))(?=\s\*\n+\d+\.|\s\*\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+(?:(?!\d+\.\s)[\s\S])\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)\s\*[\s\S]\*?\(b\)\s\*[\s\S]\*?\(c\)\s\*[\s\S]\*?\(d\)\s\*[\s\S]\*?))(?=\n+\d+\.\s|\Z)'  
pattern\_for\_list\_type1 = r'(\d+\.\s+[\s\S]\*?(?i:select|codes|answer)[\s\S]\*?(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)(?=\n\d+\. |\Z)'  
# pattern\_for\_list\_type1 = r'(\d+\.\s+(?:(?!\d+\.\s)[\s\S])\*?(?i:select|codes|answer)\s\*(?:(?!\d+\.\s)[\s\S])\*?(?:\(a\)\s\*[\s\S]\*?\(b\)\s\*[\s\S]\*?\(c\)\s\*[\s\S]\*?\(d\)\s\*[\s\S]\*?))(?=\n+\d+\.\s\*|\Z)'  
  
"""pattern\_for\_list\_type1 = (  
 r'(\d+\.\s+[\s\S]\*?'  
 r'(?i:select|codes|answer)[\s\S]\*?'  
 r'(?:\(a\)|\(b\)|\(c\)|\(d\))[\s\S]\*?)'  
 r'(?=\n+\d+\.|\Z)'  
)"""  
  
  
print(f"Using pattern for list type1: {pattern\_for\_list\_type1}")  
  
matches = re.findall(pattern\_for\_list\_type1, sample\_text)  
print(f"Found {len(matches)} matches:")  
for match in matches:  
 print(match.strip())  
 print("---")  
 # question = match[0].strip()  
 # options = match[1].strip()  
 # print(question)  
 # print(options)  
 # print("---")  
  
specific\_question\_number = "1."  
specific\_question\_pattern = re.escape(specific\_question\_number) + r'\s+.\*?(?=\n\d+\.|\Z)'  
specific\_question\_match = re.search(specific\_question\_pattern, sample\_text, re.DOTALL)  
if specific\_question\_match:  
 specific\_question\_text = specific\_question\_match.group().strip()  
 print(f"Specific Question {specific\_question\_number}:")  
 print(specific\_question\_text)  
  
  
text\_after\_question\_1 = re.search(r'1\.(.\*?)(?=\n\d+\.|\Z)', sample\_text, re.DOTALL)  
if text\_after\_question\_1:  
 text\_after\_question\_1 = text\_after\_question\_1.group(1).strip()  
 print("Text after question 1:")  
 print(text\_after\_question\_1)

1. **fresh\_update\_hia\_list\_of\_undesirable\_characters.py**

import os  
import datetime  
from pathlib import Path  
  
# Load the existing dictionary  
def load\_dictionary(file\_path):  
 with open(file\_path, 'r', encoding='utf-8') as file:  
 content = file.read()  
 dictionary = eval(content.split('=', 1)[1].strip())  
 return dictionary  
  
# Write the updated dictionary to the file  
def write\_dictionary(dictionary, file\_path):  
 with open(file\_path, 'w', encoding='utf-8') as file:  
 file.write('hia\_list\_of\_undesirable\_characters\_for\_question\_bank = ')  
 file.write(repr(dictionary))  
 print(f"Dictionary file updated at '{file\_path}'")  
  
# Add new unwanted characters to the dictionary  
def add\_unwanted\_characters(input\_file\_path, dictionary\_file\_path):  
 dictionary = load\_dictionary(dictionary\_file\_path)  
 with open(input\_file\_path, 'r', encoding='utf-8') as file:  
 text = file.read()  
  
 # Create a set to store all characters found in the input file  
 all\_characters = set()  
  
 # Open the output file for writing the characters and their context  
 output\_file\_path = Path(  
 r"C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Python Programming\Unwanted Characters etc") / f"unwanted\_characters\_{input\_file\_path.stem}.txt"  
 with open(output\_file\_path, 'w', encoding='utf-8') as output\_file:  
 for char in text:  
 if ord(char) > 127:  
 all\_characters.add(char)  
 # Extract the sentence or words around the character  
 context = extract\_context(text, char)  
 print(f"Character: {char} - Unicode: {ord(char)} - Context: {context}")  
 output\_file.write(f"Character: {char} - Unicode: {ord(char)} - Context: {context}\n")  
  
 if char not in dictionary['unwanted\_characters']:  
 dictionary['unwanted\_characters'][char] = {  
 "replacement": "",  
 "date\_added": datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),  
 "file\_name": input\_file\_path.name,  
 "source\_of\_replacement": "",  
 "character\_update\_number": 1, # Increment this for each update  
 "replacement\_update\_number": 0 # Keep this as 0 for new characters  
 }  
  
 write\_dictionary(dictionary, dictionary\_file\_path)  
  
 print(f"All characters found in the input file have been saved to '{output\_file\_path}'")  
  
# Add replacements to the dictionary  
def add\_replacements(dictionary\_file\_path):  
 dictionary = load\_dictionary(dictionary\_file\_path)  
 for char, details in dictionary['unwanted\_characters'].items():  
 if details['replacement'] == "":  
 print(f"Character: {char} - Unicode: {ord(char)}")  
 replacement = input(f"Enter the replacement for '{char}' (leave blank to skip): ")  
 if replacement:  
 details['replacement'] = replacement  
 details['date\_added'] = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")  
 details['source\_of\_replacement'] = "Manual Update"  
 details['replacement\_update\_number'] += 1 # Increment this for each replacement update  
 else:  
 print(f"Character: {char} - Unicode: {ord(char)} - Existing replacement: {details['replacement']}")  
 input("Press Enter to replace with the existing replacement or type a new replacement: ")  
 # If the user enters a new replacement, update the dictionary  
 new\_replacement = input().strip()  
 if new\_replacement:  
 details['replacement'] = new\_replacement  
 details['date\_added'] = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")  
 details['source\_of\_replacement'] = "Manual Update"  
 details['replacement\_update\_number'] += 1 # Increment this for each replacement update  
  
 write\_dictionary(dictionary, dictionary\_file\_path)  
  
def extract\_context(text, char, window=4):  
 index = text.find(char)  
 if index == -1:  
 return "Character not found in text"  
  
 # Extract words around the character  
 words = text.split()  
 for i, word in enumerate(words):  
 if char in word:  
 start = max(0, i - window)  
 end = min(len(words), i + window + 1)  
 return ' '.join(words[start:end])  
 return "Context not found"  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 action = input("Choose action - 2: Add Unwanted Characters, 3: Add Replacements: ")  
 dictionary\_file\_path = Path(r'C:\Users\PRATIK HAJELA\PycharmProjects\question\_bank\_creation\_from\_pdf\hia\_list\_of\_undesirable\_characters\_for\_question\_bank.py')  
  
 if action == "2":  
 input\_file\_name = input("Enter the name of the text file (without the .txt extension): ")  
 input\_file\_path = Path(r"C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Python Programming\Trials\Regex Trials") / (input\_file\_name + '.txt')  
 add\_unwanted\_characters(input\_file\_path, dictionary\_file\_path)  
 elif action == "3":  
 add\_replacements(dictionary\_file\_path)  
 else:  
 print("Invalid action selected.")

1. **first\_time\_create\_hia\_list\_of\_undesirable\_characters.py:**

import os  
import datetime  
from pathlib import Path  
  
# Load the existing dictionary  
def load\_dictionary(file\_path):  
 with open(file\_path, 'r', encoding='utf-8') as file:  
 content = file.read()  
 dictionary = eval(content.split('=', 1)[1].strip())  
 return dictionary  
  
# Write the updated dictionary to the file  
def write\_dictionary(dictionary, file\_path):  
 with open(file\_path, 'w', encoding='utf-8') as file:  
 file.write('hia\_list\_of\_undesirable\_characters\_for\_question\_bank = ')  
 file.write(repr(dictionary))  
 print(f"Dictionary file updated at '{file\_path}'")  
  
# Add new fields to the dictionary  
def add\_new\_fields(dictionary\_file\_path):  
 dictionary = load\_dictionary(dictionary\_file\_path)  
 for char, details in dictionary['unwanted\_characters'].items():  
 details['character\_update\_number'] = 0  
 details['character\_add\_source'] = details.get('file\_used', 'ChatGPT4')  
 details['character\_add\_date'] = details.get('date\_added', datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))  
 details['replacement\_update\_number'] = 0 if char != '¬' else 1 # Set to 1 only for the character "¬"  
 details['replacement\_add\_source'] = details.get('file\_used', 'ChatGPT4')  
 details['replacement\_add\_date'] = details.get('date\_added', datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))  
 del details['file\_used']  
 del details['date\_added']  
  
 write\_dictionary(dictionary, dictionary\_file\_path)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 input("Press Enter to continue to add new fields to the existing dictionary 'hia\_list\_of\_undesirable\_characters\_for\_question\_bank.py'...")  
 dictionary\_file\_path = Path(r'C:\Users\PRATIK HAJELA\PycharmProjects\question\_bank\_creation\_from\_pdf\hia\_list\_of\_undesirable\_characters\_for\_question\_bank.py')  
 add\_new\_fields(dictionary\_file\_path)

1. **text\_extraction\_pypdf2.py**

import PyPDF2  
import os  
from pathlib import Path  
  
  
def extract\_text\_from\_pdf(pdf\_path):  
 with open(pdf\_path, 'rb') as file:  
 reader = PyPDF2.PdfReader(file)  
 text = ''  
 for page\_number, page in enumerate(reader.pages, start=1):  
 page\_text = page.extract\_text()  
 if not page\_text:  
 print(f"Warning: No text found on page {page\_number}")  
 text += page\_text + '\n'  
 return text  
  
  
def create\_directories\_and\_extract\_text(pdf\_file\_name):  
 input\_path = Path(r'C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Input')  
 base\_path = Path(r'C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Area')  
  
 parts = pdf\_file\_name.split(' ')  
 print("Parts of the file name:")  
 for i, part in enumerate(parts):  
 print(f"Part {i + 1}: {part}")  
  
 # Let's use the fifth and sixth parts for the subfolder name  
 subfolder\_name = parts[4] + ' ' + parts[5]  
 subfolder\_path = base\_path / subfolder\_name.lower()  
 subfolder\_path.mkdir(parents=True, exist\_ok=True)  
  
 # Create the text file name without the "Pages X - Y" part  
 text\_file\_name = '\_'.join(parts[:7]).lower().replace(' ', '\_') + '.txt'  
 text\_file\_path = subfolder\_path / text\_file\_name  
  
 pdf\_path = input\_path / (pdf\_file\_name + '.pdf')  
 pdf\_text = extract\_text\_from\_pdf(pdf\_path)  
  
 with open(text\_file\_path, 'w', encoding='utf-8') as text\_file:  
 text\_file.write(pdf\_text)  
  
 print(f"Text extracted and saved to {text\_file\_path}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 pdf\_file\_name = input("Enter the name of the PDF file (without the .pdf extension): ")  
 create\_directories\_and\_extract\_text(pdf\_file\_name)

1. **test\_for\_invisible\_characters.py** running successfully as on 10 Apr, 2024 is**:**

import re  
import os  
  
def check\_and\_replace\_invisible\_characters(input\_file\_path, output\_file\_path, replacements\_file\_path):  
 with open(input\_file\_path, 'r', encoding='utf-8') as file:  
 text = file.read()  
  
 replaced\_characters = set()  
  
 with open(output\_file\_path, 'w', encoding='utf-8') as file:  
 for char in text:  
 if ord(char) > 127:  
 file.write(f"Character: {char} - Unicode: {ord(char)}\n")  
 replacement = input(f"Enter the replacement for '{char}' (press Enter to skip): ")  
 if replacement:  
 text = text.replace(char, replacement)  
 replaced\_characters.add((char, replacement))  
  
 if replaced\_characters:  
 with open(replacements\_file\_path, 'a', encoding='utf-8') as file:  
 for char, replacement in replaced\_characters:  
 file.write(f"{char} - {replacement}\n")  
  
 with open(input\_file\_path, 'w', encoding='utf-8') as file:  
 file.write(text)  
  
 return replaced\_characters  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 input\_file\_name = input("Enter the name of the text file (without the .txt extension): ")  
 input\_file\_path = os.path.join(r"C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Python Programming\Trials\Regex Trials", input\_file\_name + '.txt')  
 output\_file\_path = os.path.join(r"C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Python Programming\Trials\Regex Trials", "output\_invisible\_characters.txt")  
 replacements\_file\_path = os.path.join(output\_file\_path, f'invisible\_characters\_found\_in\_{input\_file\_name}.txt')  
  
  
  
 replaced\_characters = check\_and\_replace\_invisible\_characters(input\_file\_path, output\_file\_path, replacements\_file\_path)  
  
 if replaced\_characters:  
 print("The following characters were replaced and saved to the replacements file:")  
 for char, replacement in replaced\_characters:  
 print(f"Character: {char} - Replaced with: {replacement}")  
 else:  
 print("No characters were replaced.")

1. **trying\_various\_regex.py**

import re  
from datetime import datetime  
from pathlib import Path  
from fresh\_text\_cleaner import clean\_and\_correct\_question\_text, clean\_and\_correct\_pre\_process\_text  
from fresh\_text\_cleaner import unwanted\_text\_pattern  
  
# old\_question\_pattern\_1 = re.compile(r'\n(?=\d+\.\s(?!\d+\.))')  
# old\_question\_pattern\_2 = re.compile(r'\n(?=\d+\.\s\*\S)')  
  
def extract\_questions(text, base\_output\_path, pattern\_type):  
 print("Starting extraction...") # Debugging statement  
  
 # Debugging statement to check the beginning of the text  
 print("First 100 characters of the text for pattern matching:")  
 print(text[:100])  
  
 cleaned\_text = clean\_and\_correct\_pre\_process\_text(text)  
 # cleaned\_text = text # not cleaning for debugging  
  
 # Save the cleaned text to a new file  
 cleaned\_file\_path = base\_output\_path.parent / f"cleaned\_{base\_output\_path.name}"  
 with open(cleaned\_file\_path, 'w', encoding='utf-8') as cleaned\_file:  
 cleaned\_file.write(cleaned\_text)  
  
 # Inform the user that the cleaned file has been created  
 print(  
 f"Cleaned text file '{cleaned\_file\_path.name}' created in '{cleaned\_file\_path.parent}'. Please inspect it before continuing.")  
  
 # Wait for user input to continue  
 input("Press Enter to continue with pattern processing...")  
  
 # pattern\_for\_r\_and\_a\_type = r'(\d+\.\s\*Assertion\s\*\((A)\):.\*?Reason\s\*\((R)\):.\*?)(?=\n\*(\d+\.\s\*Assertion|\Z))'  
 # pattern\_for\_r\_and\_a\_type = r'(\d+\.\s\*Assertion\s\*\((A)\):.\*?Reason\s\*\((R)\):.\*?)(?=\d+\.\s\*|\Z)'  
 # pattern\_for\_r\_and\_a\_type = r'(\d+\.\s\*Assertion\s\*\((A)\):.\*?Reason\s\*\((R)\):.\*?)(?=\n\*\d+\.\s\*|\Z)'  
 # pattern\_for\_r\_and\_a\_type = r'(\d+\.\s\*Assertion\s\*\((A)\):.\*?Reason\s\*\((R)\):[\s\S]\*?)(?=\d+\.\s|\Z)'  
 pattern\_for\_r\_and\_a\_type = r'(\d+\.\s+Assertion\s\*\((A)\):\s\*[\s\S]+?Reason\s\*\((R)\):\s\*[\s\S]+?)(?=\n\*\d+\.\s+|\Z)'  
 pattern\_for\_list\_type1 = r'\d+\.\s\*[A-Z]'  
 # pattern\_for\_list\_type2 = r'List-I.\*?List-II.\*?Codes:'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+[\s\S]+?(?:List|list).+?(?:List|list).+?(?:Codes|codes).+?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?:[\s\S](?!List|list|Codes|codes))\*?(List|list)[\s\S]\*?(List|list)[\s\S]\*?(Codes|codes)[\s\S]\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+[\s\S]\*?(List|list)[\s\S]\*?(List|list)[\s\S]\*?(Codes|codes)[\s\S]\*?)(?=\n+\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+.\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?:(?!List|list|Codes|codes).)\*?(List|list).\*?(List|list).\*?(Codes|codes).\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?:(?!List|list|Codes|codes).)\*(?:List|list).\*(?:List|list).\*(?:Codes|codes).+?(?=\n\*\d+\.\s+|\Z))'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+[\s\S]\*?(List|list)\s\*I[\s\S]\*?(List|list)\s\*II[\s\S]\*?(Codes|codes)[\s\S]\*?)(?=\n\*\d+\.\s+|\Z)'  
 pattern\_for\_list\_type2 = r'(\d+\.\s+[\s\S]\*?(List\s\*[I1])[\s\S]\*?(List\s\*[II2])[\s\S]\*?(Codes|codes)[\s\S]\*?)(?=\n\*\d+\.\s+|\Z)'  
 pattern\_for\_simple\_question = r'(\d+)\.\s\*(.\*?)\[(\d{4})\]\s\*\((a)\)\s\*(.\*?)\s\*\((b)\)\s\*(.\*?)\s\*\((c)\)\s\*(.\*?)\s\*\((d)\)\s\*(.\*?)'  
  
 patterns = {  
 "1": ("r\_and\_a\_type", pattern\_for\_r\_and\_a\_type),  
 "2": ("list\_type1", pattern\_for\_list\_type1),  
 "3": ("list\_type2", pattern\_for\_list\_type2),  
 "4": ("simple\_question", pattern\_for\_simple\_question)  
 }  
  
 pattern\_name, selected\_pattern = patterns[pattern\_type]  
 question\_pattern = re.compile(selected\_pattern)  
  
 # Split the text into raw questions  
 # raw\_questions = re.split(r'\n(?=\d+\.\s)', text)  
  
 # Find matches using the pattern  
 # matches = [q for q in raw\_questions if re.search(question\_pattern, q)]  
 print(f"Using pattern: {selected\_pattern}") # Debugging statement  
  
 # Find all matches using the pattern  
 matches = re.findall(question\_pattern, cleaned\_text)  
  
 # Print the matches  
 # for match in matches:  
 # print(match)  
 # print("---") # Divider between matches  
  
 # Debugging statement to confirm matches were found  
 print(f"Found {len(matches)} matches.")  
  
 # if matches:  
 # print("Sample match:", matches[0]) # Print first match for review  
  
 # Create the output file path based on the pattern name  
 output\_file\_path = base\_output\_path.parent / f"{pattern\_name}\_processed\_{base\_output\_path.name}"  
  
 # Write the matches to the output file with dividers  
 # Print and write the matches  
 with open(output\_file\_path, 'w', encoding='utf-8') as file:  
 for match in matches:  
 file.write(match[0].strip() + '\n\n---\n\n')  
  
 print(f"Processed questions file '{output\_file\_path.name}' created in '{output\_file\_path.parent}'")  
 print(f"Number of matches: {len(matches)}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 text\_file\_name = input("Enter the name of the text file (without the .txt extension): ")  
 text\_file\_path = Path(r'C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Python Programming\Trials\Regex Trials') / (text\_file\_name + '.txt')  
 pattern\_type = input("Enter the pattern type (1/2/3/4): ")  
  
 # debugging statements  
 # with open(text\_file\_path, 'rb') as file:  
 # binary\_content = file.read()  
 # text = binary\_content.decode('utf-8')  
  
 with open(text\_file\_path, 'r', encoding='utf-8') as file:  
 text = file.read()  
 print(text[:500]) # Print the first 500 characters of the text  
 extract\_questions(text, text\_file\_path, pattern\_type)

1. trying\_list2\_regex.py

import re  
from datetime import datetime  
from pathlib import Path  
from fresh\_text\_cleaner import clean\_and\_correct\_question\_text, clean\_and\_correct\_pre\_process\_text  
from fresh\_text\_cleaner import unwanted\_text\_pattern  
  
def extract\_questions\_using\_pattern(text, pattern, output\_file\_name):  
 matches = re.findall(pattern, text)  
 matches\_dict = {match[0]: match for match in matches}  
 with open(output\_file\_name, 'w', encoding='utf-8') as file:  
 for match in matches:  
 file.write(match.strip() + '\n\n---\n\n')  
 return matches\_dict  
  
def extract\_list\_type2\_questions(text, base\_output\_path):  
 print("Starting extraction...") # Debugging statement  
  
 # Debugging statement to check the beginning of the text  
 print("First 100 characters of the text for pattern matching:")  
 print(text[:100])  
  
 cleaned\_text = clean\_and\_correct\_pre\_process\_text(text)  
 cleaned\_file\_path = base\_output\_path.parent / f"cleaned\_{base\_output\_path.name}"  
 with open(cleaned\_file\_path, 'w', encoding='utf-8') as cleaned\_file:  
 cleaned\_file.write(cleaned\_text)  
 print(  
 f"Cleaned text file '{cleaned\_file\_path.name}' created in '{cleaned\_file\_path.parent}'. Please inspect it before continuing.")  
  
 input("Press Enter to continue with pattern processing...")  
  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+[\s\S]\*?(List\s\*[I1]|list\s\*[I1])[\s\S]\*?(List\s\*[II2]|list\s\*[II2])[\s\S]\*?(Codes|codes)[\s\S]\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+[\s\S]\*?(List\s\*[I1]|list\s\*[I1])[\s\S]\*?(List\s\*[II2]|list\s\*[II2])[\s\S]\*?(Codes|codes)[\s\S]\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+[\s\S]\*?(List\s\*[I1]|list\s\*[I1])[\s\S]\*?(List\s\*[II2]|list\s\*[II2])[\s\S]\*?)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+.\*?(List\s\*[I1]).\*?(List\s\*[II2]).\*?Codes:.\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+[\s\S]\*?(?:List\s\*[I1]|list\s\*[I1])[\s\S]\*?(?:List\s\*[II2]|list\s\*[II2])[\s\S]\*?Codes:[\s\S]\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(List\s\*[I1].\*?List\s\*[II2])'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+[\s\S]\*?(List|list)[\s\S]\*?(List|list)[\s\S]\*?(Codes|codes)[\s\S]\*?)(?=\n+\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?:(?!\d+\.\s).)\*List\s\*[I1](?:(?!\d+\.\s).)\*List\s\*[II2](?:(?!\d+\.\s).)\*Codes:(?:(?!\d+\.\s).)\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?:(?!List\s\*[I1]|list\s\*[I1]|List\s\*[II2]|list\s\*[II2]|Codes:|\d+\.\s).)\*(List|list)\s\*[I1](?:(?!List\s\*[I1]|list\s\*[I1]|List\s\*[II2]|list\s\*[II2]|Codes:|\d+\.\s).)\*(List|list)\s\*[II2](?:(?!List\s\*[I1]|list\s\*[I1]|List\s\*[II2]|list\s\*[II2]|Codes:|\d+\.\s).)\*Codes:(?:(?!List\s\*[I1]|list\s\*[I1]|List\s\*[II2]|list\s\*[II2]|Codes:|\d+\.\s).)\*?)(?=\n+\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+.\*?List\s\*[I1].\*?List\s\*[II2].\*?Codes:.\*?)(?=\n+\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+.\*?(List|Codes).\*?)(?=\n+\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+List-I\s+\(\)\s+List-II\s+\(\)\s+:\s+\[\]\s+List-I\s+List-II\s+\(\)\s+\(\)\s+A\.\s+1\.\s+B\.\s+2\.\s+C\.\s+3\.\s+D\.\s+4\.\s+\s\*Codes:\s+\(a\)\s+\(b\)\s+\(c\)\s+\(d\))'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+?List-I\s+?List-II\s+?\([^)]\*?\)\s\*?:\s+?List\s+-\s+?I\s+?List\s+-\s+?II\s+?\([^)]\*?\)\s+?\([^)]\*?\)\s+?[A-Za-z]\.\s+?\d+\.\s+?[A-Za-z]\.\s+?\d+\.\s+?[A-Za-z]\.\s+?\d+\.\s+?Codes:\s+?\(a\)\s+?\w+\s+?\(b\)\s+?\w+\s+?\(c\)\s+?\w+\s+?\(d\)\s+?\w+)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+[\s\S]\*?(List|list)[\s\S]\*?(List|list)[\s\S]\*?(Codes|codes)[\s\S]\*?)(?=\n+\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list)[\s\S]\*?(?i:list)[\s\S]\*?(?i:codes)[\s\S]\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list)\s\*-\s\*I.\*?(?i:list)\s\*-\s\*II.\*?(?i:codes):\s\*\([a-d]\)\s\*-\s\*\d+;\s\*\([a-d]\)\s\*-\s\*\d+;\s\*\([a-d]\)\s\*-\s\*\d+;\s\*\([a-d]\)\s\*-\s\*\d+)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list).\*?(?i:list).\*?(?i:codes).\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list)\s+I.\*?(?i:list)\s+II.\*?(?i:codes).\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:List-I).\*?(?i:List-II).\*?(?i:codes).\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list\s+I).\*?(?i:list\s+II).\*?(?i:codes).\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list)\s\*-\s\*(I|II).\*?(?i:codes)[\s\S]\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?:(?!\d+\.\s).)\*?\bList\s\*-\s\*I\b.\*?\bList\s\*-\s\*II\b.\*?\bCodes:\b.\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?:[\s\S]\*?)List\s\*-\s\*I(?:[\s\S]\*?)List\s\*-\s\*II(?:[\s\S]\*?)Codes:(?:[\s\S]\*?))(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list)\s\*-\s\*I.\*?(?i:list)\s\*-\s\*II.\*?(?i:codes):\s\*\([a-d]\)\s\*-\s\*\d+;\s\*\([a-d]\)\s\*-\s\*\d+;\s\*\([a-d]\)\s\*-\s\*\d+;\s\*\([a-d]\)\s\*-\s\*\d+)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list)\s\*-\s\*I.\*?(?i:list)\s\*-\s\*II.\*?(?i:codes):\s\*(?:\([a-d]\)\s\*-\s\*\d+;\s\*){3,4}\([a-d]\)\s\*-\s\*\d+)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list)\s\*-\s\*I.\*?(?i:list)\s\*-\s\*II.\*?(?i:codes):.\*?\([a-d]\)\s\*-\s\*\d+.\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match).\*?(?i:list).\*?I.\*?(?i:list).\*?II.\*?(?i:codes):.\*?\([a-d]\).\*?\d+.\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list)\s\*-\s\*I.\*?(?i:list)\s\*-\s\*II.\*?(?:(?i:codes|pairs):\s\*(?:\([a-d]\)\s\*-\s\*\d+;\s\*){0,4}\([a-d]\)\s\*-\s\*\d+)?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match).\*?(?i:list).\*?I.\*?(?i:list).\*?II.\*?(?:(?i:codes|pairs):.\*?\([a-d]\).\*?\d+.\*?))(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match).\*?(?i:list).\*?I.\*?(?i:list).\*?II.\*?(?i:codes|pairs)?.\*?\([a-d]\).\*?\d+.\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list)[\s\S]\*?(?i:list)[\s\S]\*?(?i:codes|pairs)[\s\S]\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match.\*?list).\*?(?i:list).\*?I.\*?(?i:list).\*?II.\*?(?i:codes|pairs).\*?(?:\([a-d]\)\s\*-\s\*\d+;?\s\*){3,4}\([a-d]\)\s\*-\s\*\d+)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match.\*?list).\*?(?i:list).\*?I.\*?(?i:list).\*?II.\*?(?:(?i:codes):\s\*(?:\([a-d]\)\s\*-\s\*\d+;?\s\*){3,4}\([a-d]\)\s\*-\s\*\d+|(?i:pairs).\*?))(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list).\*?I.\*?(?i:list).\*?II.\*?(?:(?i:codes):\s\*(?:\([a-d]\)\s\*-\s\*\d+;?\s\*){3,4}\([a-d]\)\s\*-\s\*\d+|(?i:pairs).\*?))(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match).\*?(?i:list).\*?I.\*?(?i:list).\*?II.\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match).\*?(?i:list).\*?I.\*?(?i:list).\*?II.\*?(?:(?i:codes|pairs):\s\*(?:\([a-d]\)\s\*-\s\*\d+;?\s\*)\*\([a-d]\)\s\*-\s\*\d+))(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list)[\s\S]\*?(?i:list)[\s\S]\*?(?:(?i:codes)[\s\S]\*?|\b(?i:pairs)\b))(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list)[\s\S]\*?(?i:list)[\s\S]\*?(?:(?i:codes|pairs))[\s\S]\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list)[\s\S]\*?(?i:list)[\s\S]\*?(?:(?i:codes|pairs))[\s\S]\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list)[\s\S]\*?(?i:list)[\s\S]\*?(?:(?i:codes|pairs))[\s\S]\*?(?:\([a-d]\)[\s\S]\*?\d+[\s\S]\*?))(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list)[\s\S]\*?(?i:list)[\s\S]\*?(?:(?i:codes|pairs))[\s\S]\*?)(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list)[\s\S]\*?(?i:list)[\s\S]\*?(?:(?i:codes|pairs))[\s\S]\*?(?:\([a-d]\)[\s\S]\*?\d+[\s\S]\*?|\b\d+\b[\s\S]\*?\([a-d]\)[\s\S]\*?))(?=\n\*\d+\.\s+|\Z)'  
 # pattern\_for\_list\_type2 = r'(\d+\.\s+(?i:match\s+list).\*?(?i:list)[\s\S]\*?(?i:list)[\s\S]\*?(?:(?i:codes|pairs))[\s\S]\*?(?:\([a-d]\)[\s\S]\*?\d+|\b\d+\b[\s\S]\*?\([a-d]\))[\s\S]\*?)(?=\n\*\d+\.\s+|\Z)'  
 pattern\_for\_list\_type2 = r'(\d+\.\s+(?:(?i:match\s+)?(?:(?:(?i:list)[\s\S]\*?(?i:list))|(?i:lists))[\s\S]\*?(?i:codes)|(?i:pairs))[\s\S]\*?)(?=\n\*\d+\.\s+|\Z)'  
  
  
 """pattern\_for\_list\_type2 = (  
 r'(\d+\.\s+(?i:match\s+list).\*?'  
 r'(?i:list\s\*[-–]\s\*I).\*?'  
 r'(?i:list\s\*[-–]\s\*II).\*?'  
 r'(?i:codes):.\*?'  
 r'(?=\n\*\d+\.\s+|\Z)'  
 )"""  
  
 question\_pattern = re.compile(pattern\_for\_list\_type2)  
  
 print(f"Using pattern: {pattern\_for\_list\_type2}")  
  
 matches = re.findall(question\_pattern, cleaned\_text)  
  
 print(f"Found {len(matches)} matches.")  
  
 # Create the output file path based on the pattern name  
 output\_file\_path = base\_output\_path.parent / f"list\_type2\_processed\_{base\_output\_path.name}"  
 with open(output\_file\_path, 'w', encoding='utf-8') as file:  
 for match in matches:  
 print("Match object:", match) # Print the entire match object  
 print("Writing match:", match[0]) # Print the match before writing  
 file.write(match.strip() + '\n\n---\n\n') # Write the entire match  
  
 print(f"Processed questions file '{output\_file\_path.name}' created in '{output\_file\_path.parent}'")  
 print(f"Number of matches: {len(matches)}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 text\_file\_name = input("Enter the name of the text file (without the .txt extension): ")  
 text\_file\_path = Path(  
 r'C:\Users\PRATIK HAJELA\Desktop\Question Bank Trials\Python Programming\Trials\Regex Trials') / (  
 text\_file\_name + '.txt')  
  
 with open(text\_file\_path, 'r', encoding='utf-8') as file:  
 text = file.read()  
 print(text[:500])  
  
 extract\_list\_type2\_questions(text, text\_file\_path)