```python
import re

class Data:

    # list of possible type of questions ~ truncated for flexibility [non-exhaustive]
    possibleList = [["capital", "distanc", "weather", "movi", "forecast", "cit", "length", "climat", "humidit", "director", "actor"], # direct-answer question
                    ["exercis", "diet", "cook", "workout", "routin", "gym", "activit", "nutri", "wellness", "recipi", "fitnes", "yoga", "meditat", "stretch", "cardio", "strength", "vitamin", "calori", "symptom"], # health-related questions
                    ["calendar", "remind", "task", "schedul", "event", "deadlin", "project", "checklist", "alert", "notif", "organ", "priorit", "goal", "plann", "timelin", "focus", "track", "habit", "workflow"], # productivity questions
                    ["scor", "gam", "jok", "song", "challeng", "puzzl", "music", "lyric", "match", "adventur", "humor", "quiz", "fun", "comed", "story", "celebr", "sport", "trend"], # entertainment questions
                    ["plu", "minus", "multipl", "divid", "formula", "concept", "ratio", "algebra", "geometr", "calculus", "integrat", "deriv", "vector", "probabil", "statist", "measur", "equation", "matrix", "quantit"], # mathematical questions
                    ["pric", "mean", "fact", "happen", "latest", "explain", "differenc", "orig", "reason", "impact", "histor", "overview", "background"], # knowledge-building question
                    ["best", "advic", "help", "tip", "plan", "stuck", "assist", "recommend", "suggest", "guid", "strategy", "solv", "improv", "overcom", "choic", "option"], # advice-seeking questions
                    ["cost", "valu", "budget", "cheap", "expens", "discount", "sale", "offer", "stock", "inventor", "demand", "suppl", "quot", "deal", "order", "purchas", "rent", "bill"]] # economic questions

    # list of possible cities that the user might reference [non-exhaustive]
    specificPlaceList = ["Los Angeles", "Chicago", "San Francisco", "Miami", "Austin", "Las Vegas", "Paris", "London", "Tokyo", "Sydney", "Rome", "Barcelona",
    "Berlin", "Dubai", "Toronto", "Seoul", "Bangkok", "Mexico City", "Riverside", "Cape Town", "California", "Florida", "Texas", "New York", "Nevada", "Hawaii", "Colorado", "Alaska", "Arizona",
    "Utah", "Illinois", "Michigan", "Washington", "Georgia", "North Carolina", "Tennessee", "South Carolina", "Oregon", "New Jersey", "Virginia", "United States", "Canada", "United Kingdom",
    "France", "Italy", "Spain", "Mexico", "Germany", "Australia", "Brazil", "Japan", "India", "South Korea", "Thailand", "South Africa", "China", "Russia",  "Egypt", "Argentina", "New Zealand", "Pakistan"]

    # list of possible pop-culture references [non-exhaustive]
    specificPopCultureList = ["The Avengers", "Star Wars", "The Matrix", "Harry Potter", "Jurassic Park", "Titanic", "The Godfather", "Pulp Fiction", "Back to the Future", "The Lion King", # movies
    "Apple", "Nike", "Tesla", "Coca-Cola", "McDonald’s", "Amazon", "Google", "Adidas", "Disney", "Microsoft", "Nvidia", # brands
    "Friends", "Game of Thrones", "The Office", "Stranger Things", "Breaking Bad", "The Simpsons", "The Mandalorian", "The Crown", "The Walking Dead", "Westworld", # tv shows
    "The Beatles", "Beyonce", "Kanye West", "Taylor Swift", "Elvis Presley", "Michael Jackson", "Ariana Grande", "Drake", "Lady Gaga", "Eminem", # artists
    "Super Mario Bros.", "Minecraft", "Fortnite", "The Legend of Zelda", "Call of Duty", "Grand Theft Auto", "Pokémon", "League of Legends", "FIFA", "The Witcher", # video games
    "Rolls-Royce", "Ferrari", "Lamborghini", "Porsche", "Maserati", "Bentley", "Aston Martin", "Bugatti", "McLaren", "Mercedes-Benz", "Lexus", "BMW", "Audi"] # automotive brands

    # words that are autocorrected
    correctedWord = []

    # method that removes accidental duplicates found by regular expression
    def __remove_duplicate(self, list):
        accList = []
        for i in list:
            if i not in accList:
                accList.append(i)
        return accList

    # uses Python Regular Expressions to derive key data in a structural format, replicating a basic version of Natural Language Processing
    # "QT" = Question Type && "I" = Identifier
    def parsedData(self, userInput, original_input):
        question_type = self.__remove_duplicate(re.findall(r"(what|who|why|where|when|how|will|can|play|lets|let|should|is|tell|give|if)", userInput))
        identifiers = self.__remove_duplicate(re.findall(r"(capital|best|cit|length|climat|humidit|director|actor|task|schedul|event|deadlin|project|checklist|alert|notif|organ|advic|stuck|help|tip|distanc|plan|weather|forecast|latest"
                                                         r"|happen|movi|exercis|song|diet|workout|explain|differenc|routin|gym|activit|nutri|wellness|recipi|fitnes|calendar|remind|cook|scor|pric|mean|plu|ratio|minus|multipl|divid|"
                                                         r"jok|gam|fact|formula|concept|algebra|geometr|challeng|puzzl|music|lyric|match|adventur|humor|yoga|meditat|stretch|cardio|"
                                                         r"strength|vitamin|calori|priorit|goal|plann|timelin|focus|track|habit|workflow|quiz|fun|comedi|story|celebr|sport|trend|"
                                                         r"calculus|integrat|deriv|vector|probabil|statist|measur|equation|symptom|matrix|quantit|orig|reason|impact|histor|overview|background|assist|recommend|suggest|guid|strategy|solv|improv|overcom|choic|option"
                                                         r"cost|valu|budget|cheap|expens|discount|sale|offer|stock|inventor|demand|suppl|quot|deal|order|purchas|rent|bill)", userInput))

        if self.autoCorrect(userInput, self.possibleList) is not None:
            identifiers.extend(self.autoCorrect(userInput, self.possibleList))
            identifiers = self.__remove_duplicate(identifiers)
        result =  {"QT": question_type, "I": identifiers}
        self.correctedWord.extend(identifiers)
        return result

    # takes the original sentence inputted by the user and then removes suffixes; local change not global
    def stemWord(self, userInput):
        return re.sub(r'\b(?!(is\b))(e|es|ing|ed|s|se|ication|ization|isation|ized|ised|ied|ous|y|ies|tion|ent|ents|er|ers|ic|ation|ating|ize|ian|ate|ative|atives|ity|ics|in|inate|ance|ive)\b', '', userInput)

    # finds any key data similar to the list above that might be misspelled to reinterpret input ~ similarity >= 70%
    def autoCorrect(self, userInput, c_list):
        inputList = userInput.split()
        count = 0
        result = []
        for r in range(len(c_list)):
            for c in range(len(c_list[r])):
                for n in inputList:
                    iter = min(len(n), len(c_list[r][c]))
                    for i in range(iter):
                        if list(n).__getitem__(i) == list(c_list[r][c]).__getitem__(i):
                            count += 1
                        elif (i + 1) < len(c_list[r][c]):
                            if list(n).__getitem__(i) == list(c_list[r][c]).__getitem__(i + 1):
                                count += 1
                    if (count / max(len(n), len(c_list[r][c]))) * 100 >= 70:
                        result.append(c_list[r][c])
                    count = 0
        return result if result else None

    # prints the content by returning them
    def printContent(self, items):
        if len(items) == 1:
            return items[0]  # Return the single item as a string
        else:
            return ", ".join(items)
```