Each line in the test file given below contains a date of the form 08-MAR-85. You are to author a program that builds a dictionary that can be used to translate the dates of the form found in the file (the keys) to the MM-DD-YYYY format (the values).

Author a program that:

1. Opens a file for reading
2. Uses a **UDF** to convert a string given in the file's date format to a date in the required format
3. Uses the data returned from the function to build/populate a dictionary suitable for decoding dates from the DD-MMM-YY format into the MM-DD-YYYY format
4. Displays the resulting dictionary

**Function Requirements**

Your function will:

1. Require that you construct a dictionary to translate MMM strings found in the argument string into their corresponding 2-digit month numbers (as strings). Your dictionary should be able to translate all 12 month values (JAN-DEC to 01-12, respectively).
2. Accept a single string argument in the DD-MMM-YY format
3. Split the string into its day, month and year components
4. Translate/map the MMM component (using the dictionary from step 1) to its corresponding 2-digit month string.
5. Create a 4-digit year using the 2-digit year component. You may assume all years are from the year 2000 onward
6. Return a string in the MM-DD-YYYY format

**Function Example:**

* *Given Input:* '05-JAN-08'
* *Expected Output:* '01-05-2008'
* *Explanation:* Input is split into dd, mmm, yy elements. 'JAN'(MMM) is translated/mapped to 01 using the dictionary from step 1. '08' (YY) is expanded to 4 digits. 05 (DD) is unchanged. New date elements are used to construct a date string of the required form.

**Required Function Signature**

**def** parse\_dates(dt: str) **->** str:

**Final Dictionary Example:**

* { '08-MAR-05' : '03-08-2005', '09-FEB-19' : '02-09-2019', ...}

Author a program that:

1. Opens a file in read mode
2. Iterates over the file's lines splitting each line into words.
3. Skip words with less than 2 characters.
4. Normalize each word to lower case
5. Uses a Boolean **UDF** to determine if a word's last letter alphabetically follows the word's first letter. *See example.*
6. Displays the word if the function returns True

**Function Requirements**

Your function must:

1. accept a string as an argument
2. Return True if a word's last letter immediately follows the word's first letter in the alphabet. Otherwise, return False.

**Note:** 'a' is considered to follow the letter 'z'. That is, 'z' wraps around to 'a'.

**Function Example:**

* *Given Input:* 'could'
* *Expected Output:* True
* *Explanation:* The word's last letter (d) immediately follows (alphabetically) the word's first letter (c). That is, (d) immediately follows (c) in the alphabet

**Required Function Signature**

**def** first\_last(in\_str: str) **->** bool:

Author your solution using the test data provided in the code-cell below.

Q3:

Author a program that:

1. Opens a file for reading
2. Uses a **UDF** to read the file's lines and return a list of those lines in reverse order.
3. Prints the first 10 lines of the returned list. Each of the 10 lines must appear on its own line. That is, do not simply print a slice of the returned list.

**Function Requirements**

Your function must:

1. accept an open file handle as an argument
2. Ingest the file's contents into a list
3. Skip any lines that are blank. Ie, lines that contain only a newline character.
4. Reverse the order of the file's contents (ie., the lines in the file). For example, the file's last line should be first, the second to the last line next, etc.
5. Return the resulting reversed list

**Function Example:**

* *Given Input:*  
  line1  
  line2  
  line3
* *Expected Output:* [ line3, line2, line1 ]

**Required Function Signature**:

Code written for this question without file handling:

import io  
fh = open('lines\_file.txt')  
def reverse\_file\_lines(fh: io.IOBase) -> list:  
 data = fh.readlines()  
 newdata = []  
 for line in data:  
 newdata.append(line.strip())  
 newdata.reverse()  
 return(newdata[0:11])  
  
res = reverse\_file\_lines(fh)  
print(res)  
  
# question 3 without uDF  
fh = open('lines\_file.txt')  
data = fh.readlines()  
newdata = []  
for line in data:  
 newdata.append(line.strip())  
 newdata.reverse()  
print (newdata[0:11])

the without UDF returns the desired results but is not in a function.