**CS 1120 Computer Science II (with Python),** Fall 2020  
Department of Computer Science, Western Michigan University  
Instructor: Wassnaa Al-Mawee, Lab TA: Utkarsha Prashant Kotkar

Andrew Kroll

**SOFTWARE LIFE CYCLE REPORT FOR LAB ASSIGNMENT 4**

**PHASE 1: SPECIFICATION (“What do we build?)**

Write a Python application that provides a literal translation for text entered by a user. To determine how to translate the words, you will use something called a ***parallel corpus****.* A parallel corpus is a large collection of text in two or more languages.

**PHASE 2: DESIGN**

**2.1 Modules (Classes) and Their Structure (Class Hierarchy)**

main:

# Controls the main program logic.

greeting:  
# Displays a greeting to the user and indicates the function of the program.

class UserInterface:

\_\_init\_\_:

Initializes the user interface object

run\_program:  
 # calls necessary methods for displaying a greeting, ask the user what to do, update the corpus file name and text to be translated. Initializes a translator object and loops until the user no longer wants to translate any sentences.

get\_corpus\_filename:

# Displays translated options to the user and sets the name of the corpus file.

get\_source\_text:

# requests the text the user wants to translate

translate:

# Translates the source text and displays the translated text

class Translator:

\_\_init\_\_(file, input):

Initializes the translator object

read\_corpus:

# Reads the input file and stores the data in a two dimensional list for translation

translate:

# Calls necessary methods to perform the translation

\_\_lookup(word):

# Looks up the word in the appropriate corpus and returns the corresponding translation. Returns the word received if not found.

**2.2 Pseudocode for the Modules (Classes)**

class UserInterface:

def \_\_init\_\_(self):

# initialize variables

def run\_program(self):

# get appropriate user information, create translator instance, and do translation

def get\_corpus\_filename(self):

# get translation selection from user and return appropriate file name

def get\_source\_text(self):

# return message to be translated from user

def translate(self):

# read corpus, then translate using the translator

class Translator:

def \_\_init\_\_(self, file: str, source: str):

# initialize variables

def read\_corpus(self):

# Read corpus file to dictionary

def translate(self):

# translate each word in message, capitalize first letter and add trailing punctuation if necessary, then print

def \_\_lookup(self, word: str):

# lookup cleaned word in dictionary and return result or source word if not found

def greeting():

# display a greeting and a brief explanation of what the program does.

def main():

# main program control and loop

**PHASE 3: RISK ANALYSIS (“What can go wrong, and how bad can it be?”)**

Missing corpus files will prevent the program from running properly but will not cause the program to crash.

**PHASE 4: VERIFICATION (“Are the algorithms correct?”)**

All “algorithms” are functional. User input is verified where necessary to prevent catastrophic failure.

**PHASE 5: CODING**

**5a) Code Refinement #1 (class structure with pseudocode only; pseudocode is used as comments)**

class UserInterface:

def \_\_init\_\_(self):

# initialize file, translator, and text variables

def run\_program(self):

# update file variable with get\_corpus\_filename()

# update text variable with get\_source\_text()

# update translator variable with new instance of Translator using file and text variables as arguments

# do translation with translate()

def get\_corpus\_filename(self):

# prompt the user with options

# create selection variable

# get user selection and catch errors until valid

# return 'etf.csv' if selection is 1, otherwise return 'fte.csv'

def get\_source\_text(self):

# return message to be translated from user

def translate(self):

# read\_corpus() in translator object

# translate() in translator object

class Translator:

def \_\_init\_\_(self, file: str, source: str):

# initialize file variable from file argument, source variable from source argument, and an empty dictionary

def read\_corpus(self):

# open file to read

# read lines from file and insert into dictionary

def translate(self):

# create out variable

# create capitalize\_first variable with whether or not to capitalize the first letter

# create last\_chars variable to hold trailing punctuation

# use \_\_lookup(word) to rebuild message into the out variable

# apply capitalization to first letter if necessary

# apply trailing punctuation

# print translated message

def \_\_lookup(self, word: str):

# convert word to lowercase and strip non-word characters

# if word is in dictionary, return value of word, otherwise return source word

def greeting():

# display a greeting and a brief explanation of what the program does.

def main():

# display greeting()

# create running flag variable as True

# continuously run program until user says to quit. Catch FileNotFoundErrors as appropriate

main()

**5b) Code Refinement #2 (still incomplete program: class and constructor/method structure with pseudocode only; pseudocode is used as comments)**

class UserInterface:

def \_\_init\_\_(self):

# initialize file, translator, and text variables

def run\_program(self):

# update file variable with get\_corpus\_filename()

# update text variable with get\_source\_text()

# update translator variable with new instance of Translator using file and text variables as arguments

# do translation with translate()

def get\_corpus\_filename(self):

# prompt the user with options

# create selection variable

# while selection is invalid:

# ask user to select an option

# catch ValueError and notify user

# notify user if selection is invalid

# return 'etf.csv' if selection is 1, otherwise return 'fte.csv'

def get\_source\_text(self):

# create source variable

# while source length is 0:

# update source variable with input from user

# return source variable

def translate(self):

# read\_corpus() in translator object

# translate() in translator object

class Translator:

def \_\_init\_\_(self, file: str, source: str):

# initialize file variable from file argument, source variable from source argument, and an empty dictionary

def read\_corpus(self):

# open file to read

# for each line in file:

# strip newline character

# if line contains ',':

# update dictionary with first part of line as key, second part as value

def translate(self):

# create out variable

# create capitalize\_first variable with whether or not to capitalize the first letter

# create last\_chars variable to hold trailing punctuation

# for each word in source text:

# add a space to out variable if it contains text

# add result of \_\_lookup(word) to out variable

# apply capitalization to first letter if necessary

# apply trailing punctuation

# print translated message

def \_\_lookup(self, word: str):

# convert word to lowercase and strip non-word characters

# if word is in dictionary, return value of word, otherwise return source word

def greeting():

# display a greeting and a brief explanation of what the program does.

def main():

# display greeting()

# create running flag variable as True

# while running flag is True:

# create UserInterface() instance

# run\_program() in UserInterface() instance

# catch FileNotFoundError and display a message for the user

# ask the user if they would like to translate again and update the running flag

main()

**5c) Code Refinement #3 (complete program – with complete fields/properties, code for constructor/methods)**

**LA2Main.py:**

*# Name: Andrew Kroll  
# Date: 2020-10-30  
# Course-Section/LA#: CS1120-951 LA4  
# Description: Translate from English to French and French to English using  
# parallel corpus  
import* re  
  
  
*class* UserInterface:  
 *def \_\_init\_\_*(*self*):  
 *"""  
 Initializes file, translator, and text variables.  
 """  
 self*.file = ""  
 *self*.translator = *None  
 self*.text = ""  
  
 *def* run\_program(*self*):  
 *"""  
 Controls cycle of the program:  
 Retrieves program mode from user.  
 Retrieves text to be translated from user.  
 Executes translation.  
 """  
 self*.file = *self*.get\_corpus\_filename()  
 *self*.text = *self*.get\_source\_text()  
 *self*.translator = Translator(*self*.file, *self*.text)  
 *self*.translate()  
  
 *def* get\_corpus\_filename(*self*):  
 *"""  
 Prompts the user to enter which translation mode they would like to   
 use.* ***:return****: The file name for the appropriate corpus file.   
 """  
 print*("What would you like to translate?:")  
 *print*(" ( 1 ) English to French")  
 *print*(" ( 2 ) French to English")  
 selection = 0  
 *while not* (selection == 1 *or* selection == 2):  
 *try*:  
 selection = *int*(*input*("Select an option: "))  
 *except ValueError*:  
 *print*("\n<Error> Invalid input! Input must be a number.\n")  
 *else*:  
 *if not* (selection == 1 *or* selection == 2):  
 *print*("\n<Error> Invalid input! Must be 1 or 2.\n")  
 *return* "etf.csv" *if* selection == 1 *else* "fte.csv"  
  
 *def* get\_source\_text(*self*):  
 *"""  
 Prompts the user to enter the message they would like to translate.* ***:return****: The message to be translated.  
 """* source = ""  
 *while len*(source) == 0:  
 source = *input*("What would you like to translate? ")  
 *return* source  
  
 *def* translate(*self*):  
 *"""  
 Reads corpus file and executes translation in the Translator object.  
 """  
 self*.translator.read\_corpus()  
 *self*.translator.translate()  
  
  
*class* Translator:  
 *def \_\_init\_\_*(*self*, file: *str*, source: *str*):  
 *"""  
 Initializes the file and source variables from arguments.  
 Initializes an empty dict variable.* ***:param*** *file: The name of the corpus file.* ***:param*** *source: The source text to be translated.  
 """  
 self*.file = file  
 *self*.source = source  
 *self*.dict = {}  
  
 *def* read\_corpus(*self*):  
 *"""  
 Reads the corpus file and enters it into the dict variable.  
 """  
 with open*(*self*.file, 'r') *as* file:  
 *for* line *in* file:  
 line = line.strip()  
 *if* "," *in* line:  
 *self*.dict[line.split(",")[0]] = line.split(",")[1]  
  
 *def* translate(*self*):  
 *"""  
 Translates then displays the source message.  
 """* out = ""  
 capitalize\_first = *self*.source[0] == *self*.source[0].upper()  
 last\_chars = re.sub("[a*-*zA*-*Z]", "",  
 *self*.source.split(" ")[  
 *len*(*self*.source.split(" "))-1])  
 *for* word *in self*.source.split(" "):  
 *if len*(out) > 0:  
 out += " "  
 out += *self*.\_\_lookup(word)  
 *if* capitalize\_first:  
 out = out.capitalize()  
 out += last\_chars  
 *print*(" >> {}".format(out))  
  
 *def* \_\_lookup(*self*, word: *str*):  
 *"""  
 Looks up a word in the dict, then returns the translated word or source  
 word if no translation is present.* ***:param*** *word: The word to be translated.* ***:return****: The translated word, or source word if no translation is   
 present.   
 """* temp = re.sub(r"[^a*-*zA*-*Z]", "", word).lower()  
 *if* temp *in self*.dict:  
 *return self*.dict[temp]  
 *return* temp  
  
  
*def* greeting():  
 *"""  
 Displays a greeting for the user.  
 """  
 print*("Welcome to the Translator! Bienvenue!!")  
 *print*("You can translate sentences from English to French and vice "  
 "versa!")  
 *pass  
  
  
def* main():  
 *"""  
 Main program control. Loops translation until the user no longer wishes  
 to translate anything else.  
 """* greeting()  
 flag = *True  
 while* flag:  
 ui = UserInterface()  
 *try*:  
 ui.run\_program()  
 *except FileNotFoundError*:  
 *print*("\n<Error> Corpus file could not be found.\n")  
 flag = *True if input*("Would you like to translate something else? "  
 "(y/n) ").lower().startswith("y") *else False*main()

**PHASE 6: TESTING (“Did we build it correctly?”)**

Yes. See test output below…

Welcome to the Translator! Bienvenue!!

You can translate sentences from English to French and vice versa!

What would you like to translate?:

( 1 ) English to French

( 2 ) French to English

Select an option: k

<Error> Invalid input! Input must be a number.

Select an option: 2.5

<Error> Invalid input! Input must be a number.

Select an option: 5

<Error> Invalid input! Must be 1 or 2.

Select an option: 1

What would you like to translate? The lazy boy dances with the smelly jellyfish!

>> Le paresseux garcon danse avec le puante meduse!

Would you like to translate something else? (y/n) y

What would you like to translate?:

( 1 ) English to French

( 2 ) French to English

Select an option:

<Error> Invalid input! Input must be a number.

Select an option: 2

What would you like to translate?

What would you like to translate? Le garcon danse.

>> The boy dances.

Would you like to translate something else? (y/n) n

Process finished with exit code 0

**PHASE 7: REFINING THE PROGRAM (“Add bells and whistles to the program”)**

Nothing to add.

**PHASE 8: PRODUCTION**

Uploaded, along with this document, to the dropbox.

**PHASE 9: MAINTENANCE**

If any maintenance is required, I may do so upon receiving feedback.