ELO TRANSLATION PROJECT

SARAH ****

SOME VOCAB

- Errors
  - Logic Errors
  - Runtime Errors
- Strings – ex. “This is a string”
- Data Structure – a particular way of organizing data for computers
  - Dictionary – type of data structure sorted by keyword
  - List – type of data structure sorted by index
- Library – Folders of folders or files (in this instance)

RESEARCH QUESTIONS

- What is involved in converting syntactic and semantics into a computer program?
- How can irregularities and inconsistencies in language rules affect a translation and the coding behind it?

LEARNING GOALS

- To improve programming skills
- To learn the programming language Python
- To improve French skills
- To study and research Linguistic Algorithms in relation to Computer Programming

RESEARCH

TYPES OF MACHINE TRANSLATION

- Rule-Based (Rule-Based Machine Translation, RBMT)
  - Transfer-Based Machine Translation (TBMT/TBLT)
  - Inter-lingual Machine Translation
- Example-Based Machine Translation

INTERLINGUISTICS

- Study of interlinguae, “neutral language” (independent of any language).
- Interlingual Machine Translation
  - Source Language → Interlingual Language → Target Language

TRANSFER-BASED MACHINE TRANSLATION

- To make a translation, it is necessary to have an intermediate representation that captures the "meaning" of the original sentence in order to generate the correct translation.
- Inter-lingual → intermediate language can be *independent*.
- Transfer-Based → some dependence on language pair.

BASICS OF TBMT

Original Text →

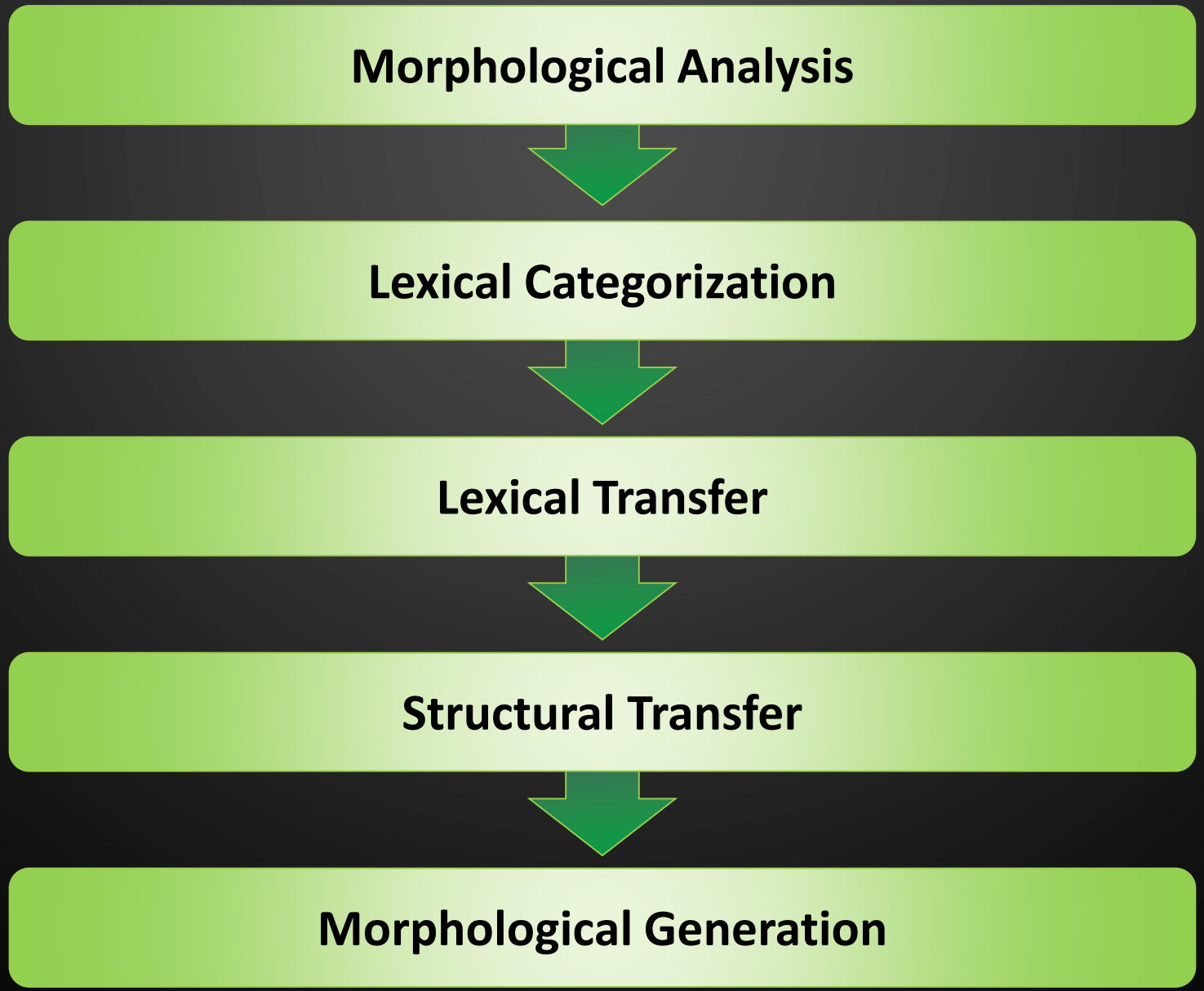
1st Intermediate Representation in Original Language →

2nd Intermediate Representation in Target Language →

Final Text

TMBT'S MOST COMMON STAGES

- Morphological Analysis
- Lexical Categorization
- Lexical Transfer
- Structural Transfer
- Morphological Generation


Morphological Analysis
Lexical Categorization
Lexical Transfer
Structural Transfer
Morphological Generation

MORPHOLOGICAL ANALYSIS

- Surface forms of the input text are classified as to part-of-speech (e.g. noun, verb, etc) and subcategory (number, gender, tense, etc).
- All of the possible analyses of surface form are typically outputted at this stage along with *lemma* of each word.

LEMMA

- Canonical form, dictionary form, or citation form of a set of words.
- Ex: run, runs, ran, running → all the same *lexeme*, the *lemma* is run.
- *Lemma* refers to a particular form that is chosen to represent the lexeme.
- Lemmatization → determining (using an algorithm) the lemma for a given word.

LEXICAL CATEGORIZATION

- Looks at the context of a word to try to determine the correct meaning in the context of the input.
- Can involve part-of-speech tagging and word sense disambiguation.

LEXICAL TRANSFER

- Basically dictionary translation
- Source language lemma is looked up in a bilingual dictionary and the translation is chosen.

STRUCTURAL TRANSFER

- Deals with phrases and chunks, typical features include concordance of gender, number, and re-ordering of words or phrases.

MORPHOLOGICAL GENERATION

- From output of structural transfer stage, the target language surface forms are generated.

TBMT'S TWO TYPES

- Superficial Transfer (or syntactic):
  - This level is categorized by transferring "syntactic structures" between the source language and the target language.
  - Suitable for languages in the same family or type (ex. The Romance Languages).
- Deep Transfer (or semantic):
  - This level constructs a semantic representation that is dependent on the source language. The representation can consist of a series of structures which represent the meaning.
  - This level is used to translate more distantly related languages (ex. Spanish-English).

RESEARCHED RESOURCES

- Python and its packages
- WordNet Database
- Python's Natural Language Tool Kit (NLTK) package
  - Contains Linguistic tools and methods for analysis

WHY PYTHON?

- Flexible, intuitive language
- Works extremely well with strings
- Had the NLTK
- WordNet works well with it

OBJECTIVE OF PROJECT

- To create a translation program for French to English that is able to take in user input in French and convert and manipulate it into an English translation

CODE WALKTHROUGH

SAMPLE RUNS

Translating a phrase

```
Enter a phrase: la fleur bleue

 the blue flower

Go Again? (Y or N) n

Press Enter to quit...
```

A user adding a word

```
Enter a phrase: add
Do you want to add a word to libraries? (Y or N) y
French word? vert
English translation? green
Part of speech? (NOUN, VERB, ADJECTIVE, OTHER) adjective
Is this an adjective for beauty, age, size, or quality? (Y or N) n
Word Added

Go Again? (Y or N) n

Press Enter to quit...
```

Admin accepting a word into libraries

```
Enter a phrase: admin
Do you want to admin? (Y or N) y
Password: jedi
Access Granted
['vert', 'green', 'ADJ']
Add word to libraries? (Y or N) y
Word Added

Go Again? (Y or N) n

Press Enter to quit...
```

```
pathroot = 'C:\\Users\\Sarah.coffen\\AppData\\Local\\Programs\\Python\\Python35-32\\code'
pathDET = '\\LIBRARY'
pathNN = '\\LIBRARY\\NOUNS\\NOUNS'
pathVR = '\\LIBRARY\\VERBSREG\\V'
pathVIR = '\\LIBRARY\\VERBSIRREG\\V'
pathADJREG = '\\LIBRARY\\ADJECTIVESREG\\ADJ REG '
pathADJBAGS = '\\LIBRARY\\ADJECTIVESBAGS\\ADJ REG '
pathWORDS = '\\LIBRARY\\WORDS\\'
pathHOLD ='\\LIBRARY'
```

Directories to library files

```
alpha = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
```

```
determiners = {}
os.chdir(pathroot + pathDET)

determiners01 = open("DETERMINERS.txt", "r+")

os.chdir(pathroot + pathHOLD)
holding01 = open("HOLDINGFILE.txt", "r").close()

hold01 = []
hold = []

os.chdir(pathroot)
```

```
phrase_source1 = "null"
wordLIST = "null"
phrase_POS = "null"
phrase_src_list = []
```

Declaring data structures and strings here expand their scope

```
loop_run()
```

The rest of the program runs from inside this method

```
def loop_run():
    fill_dictionaries()
    repeat = True
    phrase_source1 = "null"
    while(repeat == True):
        phrase_source1 = get_input(phrase_source1)
        admw = admin_phrase(phrase_source1)
        addw = user_test_holding(phrase_source1)
        if(admw == True):
            run_admin()
        elif(addw == True):
            user_add_holding()
        else:
            if(phrase_source1 == "admin" or phrase_source1 == "add"):
                phrase_source1 = get_input(phrase_source1)
            if(phrase_source1 != "admin" and phrase_source1 != "add"):
                phrase_source2 = tokenize_and_print(phrase_source1)
                print('\n' + display_translation(phrase_source2))
        repeat = go_again()
    ending()
```

There are 3 main tasks this allows the user to do:

1. **Translate a phrase**
2. **Add a word to a holding file**
3. **Add words in holding file to libraries through admin control**

```
def tokenize_and_print(phr1):
    whereL = []
    para1 = tokenize_sent_only(phr1)
    #print(para1)
    for index, sentence in enumerate(para1):
        sent1 = tokenize_words_only(sentence)
        (whereIis, sent1) = replace_token(sent1)
        sent1 = reorder_structure(whereIis, sent1)
        para1[index] = sent1
    #print(para1)
    return para1
```

Breaks up input into list format and sends it to be translated and restructured

```
def display_translation(phr1):
    phrase = ""
    for every in phr1:
        for eve in every:
            phrase = phrase + " " + str(eve)
    return phrase
```

Formats phrase as a simple string

```
def check_dict_quiet(word, lang):
    d = enchant.Dict(lang)
    d.check(word)
    returned = check_OTHER(word)
    working = False
    where = ''
    if (returned['Other'] == True):
        where = returned['Where']
        working == True
    elif (d.check(word) == True):
        where = 'wn'
        working == True
    else:
        while (d.check(word) == False and working == False):
            returned = check_OTHER(word)
            if (returned['Other'] == True):
                where = returned['Where']
                working = True
            else:
                sugg = d.suggest(word)
                print(word)
                print(sugg)
                word = input("What did you mean? ")
                where = 'wn'
    return {'Word':word, 'Where':where}
```

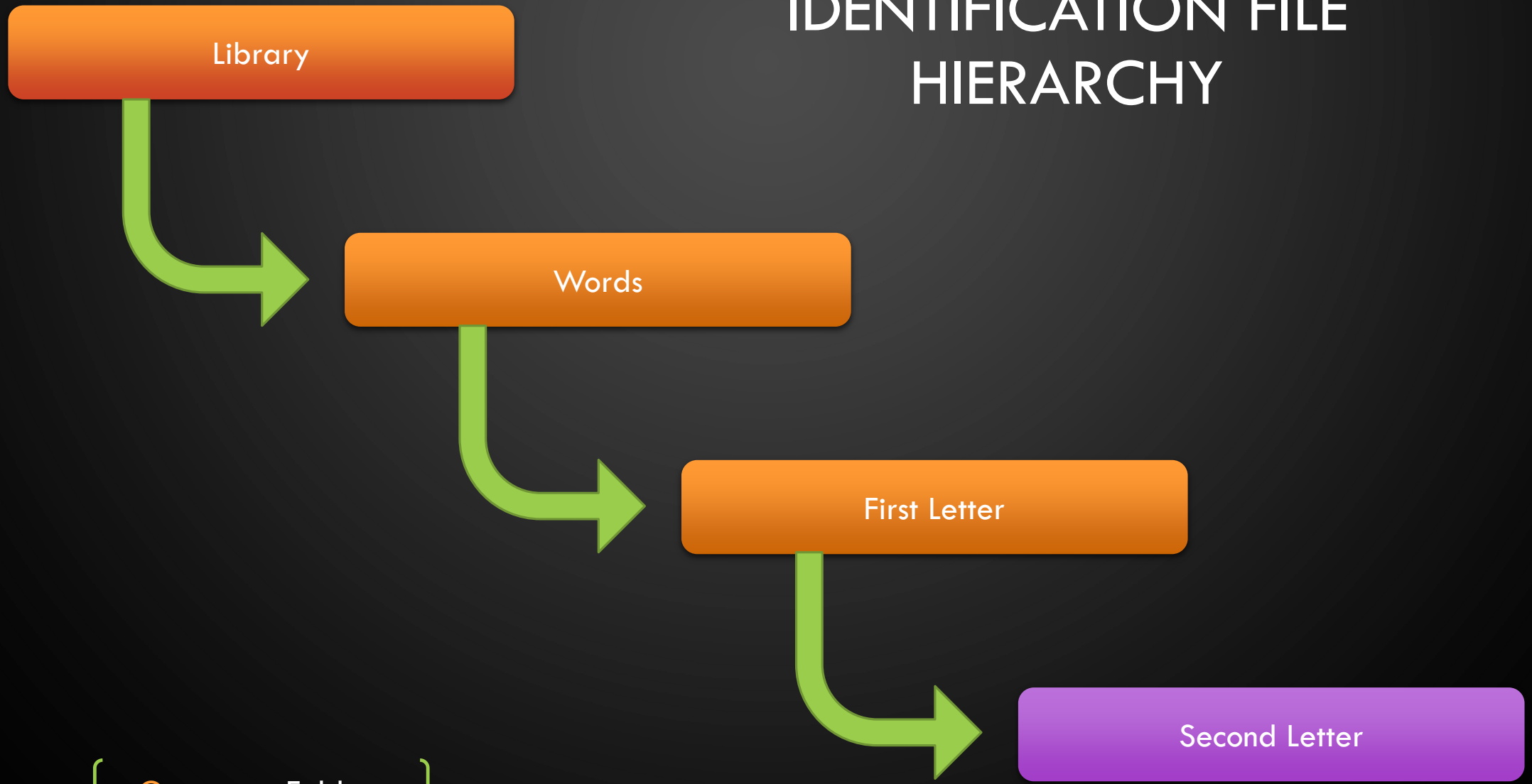
Then checks WordNet libraries for word

Checks libraries for word

```
def check_OTHER(word):
    isOTHER = False
    where = "null"
    let1 = word[0].upper()
    let2 = word[1].upper()
    wordSEARCH = {}
    path = pathroot + pathWORDS + '\\' + let1
    os.chdir(path)
    file01 = open(let2 + '.txt', "r")
    wordList = file01.read().split()
    inde = 0
    while(inde < len(wordList) - 1):
        (key, val) = wordList[inde:inde+2]
        wordSEARCH[key] = val
        inde = inde + 2
    for key in wordSEARCH:
        if(key == word):
            isOTHER = True
            where = wordSEARCH[word]
    os.chdir(pathroot)
    file01.close()
    return {'Other':isOTHER,'Where':where}
```

Returns the word and where it was found

IDENTIFICATION FILE HIERARCHY



Orange = Folders
Purple = .txt file

```
def replace_token(sent1):
    whereLIST = []
    for index, word in enumerate(sent1):
        if(word == "'" or word == '"'):
            sent1[index] = ""
        else:
            worddd = check_dict_quiet(word, "fr_FR")
            wrd = worddd['Word']
            if wrd in word:
                word = wrd
            else:
                word = word.replace(word, wrd)
            worddd['Word'] = word
            if (worddd['Where'] == 'wn'):
                word = lemmatizer_from_french(word)
                whereLIST.append('wn')
            else:
                where = worddd['Where']
                whereLIST.append(where)
                let1 = word[0].upper()
                let2 = word[1].upper()
                os.chdir(get_path(where, let1))
                filename = get_filename(where, let1, let2)
                file01 = open(filename + '.txt', "r")
                contents = {}
                posLIST = file01.read().split()
                inde = 0
                while(inde < len(posLIST) - 1):
                    (key, val) = posLIST[inde:inde+2]
                    contents[key] = val
                    inde = inde + 2
                word = contents[word]
                os.chdir(pathroot)
                file01.close()
            sent1[index] = word
    return whereLIST, sent1
```

Sends word to be found, then directs it to either the lemmatizer or the libraries to get a definition.

```
def lemmatizer_from_french(wrd1):
    wrd2 = "null"
    if(len(wn.lemmas(wrd1, lang='fra')) >=1):
        lemma_most_common = []
        lemmalist = []
        lems = ""
        lemmatized_word_formatted = wn.lemmas(wrd1, lang='fra')
        for lemm in lemmatized_word_formatted:
            lemst = str(lemm)
            to_ind = lemst.find('.')
            lems = lemst[7:to_ind]
            lemmalist.append(lems)
        lemma_most_common = most_likely_mode2(lemmalist)
        return lemma_most_common
    else:
        word2 = '"' + wrd1 + '"'
        return word2
```

Uses the lemmatizing methods that are a part of the WordNet package to return the most likely English lemma of a French word

```
def most_likely_mode(lst):
    counter = Counter(lst)
    _,val = counter.most_common(1)[0]
    return [x for x,y in counter.items() if y == val]

def most_likely_mode2(lst):
    vals = Counter(lst).most_common(1)
    most_common_formatted = str(vals)
    to_ind = most_common_formatted.find(',')
    most_common = most_common_formatted[3:(to_ind - 1)]
    return most_common
```

Uses the Counter class to find the most returned lemma of that word

French

Noun

Adjective

Noun

Adjective

English

Adjective

Noun

```
def reorder_structure(wherelist, sentence):
    indice = []
    working = True
    while(working == True):
        if('ADJ' in wherelist):
            inde02 = wherelist.index('ADJ')
            inde01 = inde02 - 1
            if(inde01 >= 0 and inde01 not in indice):
                if(wherelist[inde01] == 'NN'):
                    indice.append(inde01)
                    indice.append(inde02)
                    temp01 = sentence[inde01]
                    sentence.remove(temp01)
                    sentence.insert(inde02,temp01)
                else:
                    working = False
            else:
                working = False
        else:
            working = False
    return sentence
```

```
def get_input(phrase_source1):
    phrase_source1 = input("Enter a phrase: ")
    phrase_source1 = phrase_source1.lower()
    return phrase_source1
```

Gets input

```
def valid_input_TF(quest):
    use = input(quest)
    use = use.upper()
    rep = True
    answer = ""
    while(rep == True):
        if(use == "Y" or use == "N"):
            rep = False
            if(use == "Y"):
                answer = True
            else:
                answer = False
        else:
            use = input(quest)
            use = use.upper()
    return answer
```

Returns True/False based on user's response

```
def go_again():
    going = True
    rep = True
    print("")
    while(rep == True):
        go_again = input("Go Again? (Y or N) ")
        go_again = go_again.upper()
        if(go_again == "Y" or go_again == "N"):
            rep = False
            if(go_again == "Y"):
                going = True
            else:
                going = False
    print("")
    return going
```

Returns True/False to repeat program based on user's response

```
def fill_holding():
    os.chdir(pathroot + pathHOLD)
    holding01 = open("HOLDINGFILE.txt", "r")
    hold = holding01.read()
    holdLIST = hold.split()
    inde = 0
    while(inde < len(holdLIST) - 1):
        wordLI = holdLIST[inde:inde + 3]
        hold01.append(wordLI)
        inde = inde + 3
    holding01.close()
    os.chdir(pathroot)
```

These methods are used to place a user entered value—based on certain questions—into a holding file and to fill a list with the holding file's values

```
def user_test_holding(phrase_in):
    if(phrase_in == "add"):
        use = input("Do you want to add a word to libraries? (Y or N) ")
        use = use.upper()
        rep = True
        while(rep == True):
            if(use == "Y" or use == "N"):
                rep = False
                if(use == "Y"):
                    addmore = True
                else:
                    addmore = False
            else:
                use = input("Do you want to add a word to libraries? (Y or N) ")
                use = use.upper()
        return addmore
    else:
        return False

def user_add_holding():
    wordFR = input("French word? ")
    wordFR = wordFR.lower()
    wordEN = input("English translation? ")
    wordEN = wordEN.lower()
    whereW = False
    while(whereW == False):
        where = input("Part of speech? (NOUN, VERB, ADJECTIVE, OTHER) ")
        where = where.upper()
        if(where == "NOUN" or where == "VERB" or where == "ADJECTIVE" or
           where == "OTHER"):
            whereW = True
        else:
            whereW = False
    tag = where_get_tag(where)
    os.chdir(pathroot + pathHOLD)
    holding01 = open("HOLDINGFILE.txt", "a")
    holding01.write(wordFR + '\t' + wordEN + '\t' + tag + '\n')
    os.chdir(pathroot)
    holding01.close()
    print("Word Added")
```

```
def where_get_tag(wher):
    where = wher
    if(where == "NOUN"):
        tag = "NN"
    elif(where == "VERB"):
        isIR = input("Is this an irregular verb? (Y or N) ")
        isIR = isIR.upper()
        rep = True
        isir = False
        while(rep == True):
            if(isIR == "Y" or isIR == "N"):
                rep = False
                if(isIR == "Y"):
                    isis = True
                else:
                    isir = False
            else:
                isIR = input("Is this an irregular verb? (Y or N) ")
                isIR = isIR.upper()
        if(isir == True):
            tag = "VIR"
        else:
            tag = "V"
    elif(where == "ADJECTIVE"):
        isBAGS = input("Is this an adjective for beauty, age, size, or quality? (Y or N) ")
        isBAGS = isBAGS.upper()
        rep = True
        isbags = False
        while(rep == True):
            if(isBAGS == "Y" or isBAGS == "N"):
                rep = False
                if(isBAGS == "Y"):
                    isbags = True
                else:
                    isbags = False
            else:
                isBAGS = input("Is this an adjective for beauty, age, size, or quality? (Y or N) ")
                isBAGS = isBAGS.upper()
        if(isbags == True):
            tag = "ADJBAGS"
        else:
            tag = "ADJ"
    else:
        tag = "DET"
    return tag
```

This method uses information entered by user to get a part of speech tag for the word

```
def admin_phrase(phrase_in):
    if(phrase_in == "admin"):
        use = input("Do you want to admin? (Y or N) ")
        use = use.upper()
        rep = True
        while(rep == True):
            if(use == "Y" or use == "N"):
                rep = False
                if(use == "Y"):
                    adm = True
                else:
                    adm = False
            else:
                use = input("Do you want to admin? (Y or N) ")
                use = use.upper()
        return adm
    else:
        return False
```

Brings user to an administrative setting if they have the passcode

```
def run_admin():
    passed = pass_admin()
    if(passed == True):
        print("Access Granted")
        review_holding()
    else:
        print("Access Denied")
```

```
def pass_admin():
    passcode = input("Password: ")
    passcode = passcode.lower()
    if (passcode == "jedi"):
        return True
    else:
        return False
```

Asks admin what they wish to do with each word in the holding file

```
def review_holding():
    del hold[:]
    fill_holding()
    for eachLI in hold01:
        print(eachLI)
        doAdd = valid_input_TF("Add word to libraries? (Y or N) ")
        if(doAdd == True):
            add_to_files(eachLI)
            hold01.remove(eachLI)
        else:
            doRemove = valid_input_TF("Remove word from waiting list? (Y or N) ")
            if(doRemove == True):
                hold01.remove(eachLI)
    os.chdir(pathroot + pathHOLD)
    holding01 = open("HOLDINGFILE.txt", "w").close()
    holding01 = open("HOLDINGFILE.txt", "a+")
    for eachList in hold01:
        holding01.write(eachList[0] + '\t' + eachList[1] + '\t' + eachList[2] + '\n')
    holding01.close()
```

```
def add_to_files(wordL):
    word = wordL[0]
    eng = wordL[1]
    where = wordL[2]
    wh1 = word[0].upper()
    wh2 = word[1].upper()
    path = get_path(where, wh1)
    filename = get_filename(where, wh1, wh2)
    os.chdir(path)
    file01 = open(filename+'.txt', 'a+')
    file01.seek(0,0)
    infile = file01.read().split()
    contents = {}
    inde = 0
    while(inde < len(infile) - 1):
        (key, val) = infile[inde:inde+2]
        contents[key] = val
        inde = inde + 2
    isNEW = True
    for each in contents:
        if(each == word):
            isNEW = False
    if(isNEW == False):
        print("Already in libraries")
    else:
        file01.write(word + '\t' + eng + '\n')
        os.chdir(pathroot + pathWORDS + wh1)
        file02 = open(wh2 + '.txt', 'a+')
        file02.write(word + '\t' + where + '\n')
        file02.close()
        print("Word Added")
    os.chdir(pathroot)
    file01.close()
```

Adds word to file in library

Gets file name based on it's tag and it's first two letters

```
def get_filename(where, l1, l2):
    whe = where
    let1 = l1.upper()
    let2 = l2.upper()
    name = ""
    fromAlpha = alpha.index(let2) + 1
    if(where == 'NN'):
        name = "NOUNS" + let1 + ' (' + str(fromAlpha) + ')'
    elif(where == 'VIR'):
        name = "V" + let1 + ' (' + str(fromAlpha) + ')'
    elif(where == 'V'):
        name = "V" + let1 + ' (' + str(fromAlpha) + ')'
    elif(where == 'ADJBAGS'):
        name = "ADJREG" + let1 + ' (' + str(fromAlpha) + ')'
    elif(where == 'ADJ'):
        name = "ADJREG" + let1 + ' (' + str(fromAlpha) + ')'
    else:
        name = "DETERMINERS"
    return name
```

Gets directory path based on it's tag and it's first letter

```
def get_path(where, l1):
    path = pathroot
    let1 = l1.upper()
    if(where == 'NN'):
        path = path + pathNN + let1
    elif(where == 'VIR'):
        path = path + pathVIR + let1
    elif(where == 'V'):
        path = path + pathVR + let1
    elif(where == 'ADJBAGS'):
        path = path + pathADJBAGS + let1
    elif(where == 'ADJ'):
        path = path + pathADJREG + let1
    else:
        path = path + pathDET
    return path
```

PART OF SPEECH BASED FILE HIERARCHY

Orange = Folders
Purple = .txt file

FUTURE PLANS FOR PROJECT

- Continue to evolve the tree-file system for libraries
- Develop a more in depth verb stemmer than the one provided by NLTK
- Use knowledge gained here in future projects and research

OBSTACLES AND OVERCOMING THEM

- Learning a new programming language
  - Spent time just learning the basics
- Spent a month trying to adapt a NLTK type package to fit my needs that ultimately failed
  - Used methods as an example to work on own library design
- Hitting errors left and right
  - Debug as always

GOALS MET

- Researched Linguistics
- Learned a new programming language
- Reviewed French grammar, syntactic patterns

FUTURE PLANS FOR MYSELF

- Majoring in Computer Engineering and Computer Science
- Now I understand what independent research is like and can take what I've learned from this experience to future research opportunities.

REFERENCES AND ACKNOWLEDGEMENTS

- Bird, Steven, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. Beijing: O'Reilly, 2009. Print.
- Burton, Strang, PhD, Rose-Marie Déchaine, PhD, and Eric Vatikiotis-Bateson, PhD. *Linguistics for Dummies*. Toronto: J. Wiley & Sons Canada, 2012. Print.
- Goldman, Neil M., and Christopher K. Riesbeck. *A Conceptually Based Sentence Paraphraser*. Advanced Research Projects Agency, May 1973. Print.
- "How Does Google Translate Work?" *The Mary Sue How Does Google Translate Work Comments*. Web. 19 Nov. 2015.
- "Learn to Code." *Codecademy*. Web. 24 Nov. 2015. <https://www.codecademy.com/>.
- "Machine Translations." *Wikipedia*. Wikimedia Foundation. Web. 24 Nov. 2015. <https://en.wikipedia.org/wiki/Machine_translation>.
- "Programming Languages and Their Pros and Cons, Thoughts from a Biologist." *WordPress.com*. WordPress. Web. 24 Nov. 2015. <http://www.sarahflanagan.wordpress.com/2015/02/05/programming-languages-and-their-pros-cons-thoughts-from-a-biologist/>.
- Schank, Roger C. *The Fourteen Primitive Actions and Their Inferences*. National Institutes of Mental Health, Advanced Research Projects Agency, Mar. 1973. Print.
- Sturges, Hale, Linda Cregg. Nielsen, and Henry L. Herbst. *Une Fois Pour Toutes: Une Révision Des Structures Essentielles De La Langue Française*. White Plains, NY: Longman, 1992. Print.

Acknowledgements:

- Mrs. Barbara Reid
- Mrs. Donna Couture
- Mr. David Hobbs
- Professor Jim Weiner, UNH
- Professor Sylvia Weber Russell, UNH
- Stephanie Simmons and Walter Coffen

QUESTIONS?