

Tallest program explanations

```
12 import docx
13 from docx import Document
14 from docx.shared import RGBColor
15 from docx.shared import Inches
16 import tkinter as tk
17 from tkinter import *
18 from tkinter import filedialog
19 from typing import Tuple
20 from tkinter import scrolledtext
21 from tkinter import re
22 import copy
23 import time
24
25
26
27 #This Libraries are for opening word document automatically
28 import os
29 import platform
30 import subprocess
31
32 import xlwings as xw
33 import pandas as pd
34
```

Imports

Read Documents Function

- `readtxt(filename, color: Tuple[int, int, int]):`
- This function Takes the file path to a document (found in a text file) and a color as parameters, in this case red = (255, 0, 0)
- This function also calls the function `getcoloredTxt()` to find the colored text in the document

Read Documents Function

```
35 # reads the text in the document and use the getcoloredTxt function
36 def readTxt(filename, color: Tuple[int, int, int]):
37     doc = docx.Document(filename) # Saves the document to variable "doc"
38     text10 = ""
39     fullText = [] # Stores front and back tags + text
40     new = []
41     global everything # declares it as global to use globally
42     everything = [] # Stores front and back tags + text
43
44     for para in doc.paragraphs:
45         # Getting the colored words from the doc
46         if (getcoloredTxt(para.runs, color)):
47             # Concatenating list of runs between the colored text to single a string
48             sentence = "" .join(r.text for r in para.runs)
49             fullText.append(sentence) # Adds line of text into "fullText"
50             #print(sentence) # Prints everything in the terminal
51             everything.append(sentence)# Adds line of text into "everything"
52             text10 = sentence
53             parent.append("") .join(r.text for r in para.runs))
54             #adds parent tag into "parent" list
55
```

Read Documents Function

```
57     global hasChild # Will store the ones with a child tag
58     global fullText2 # will store everything found
59     global children # will store just child tags
60     # Finds the Lines without a childTag
61     filtered_L = [value for value in fullText if "[" not in value]
62     # removes space
63     filtered_L = [s.replace(":",":") for s in filtered_L]
64     # Finds the Lines with a childTag
65     filtered_LCopy.extend(filtered_L) #adds filtered_L into a copy list
66     #Finds the one with a child
67     hasChild = [value for value in fullText if "[" in value]
68     # Will store everything found
69     fullText2 = [value for value in fullText]
70     # Removes spaces "fullText2" list
71     fullText2 = [s.replace(":",":") for s in fullText2]
72     # Adds "fullText2" into a copy list
73     fullText2Copy.extend(fullText2)
74
75     return fullText, filtered_L, hasChild, filtered_LCopy, fullText2Copy, fullText
76
```

Look for colored word function

- `getcoloredTxt(runs, color):`
- This function takes paragraphs from a document and a color as parameters
- This function will save whole lines with colored words in a list, child tags in another list and just parent tags in another list

Look for colored word function

```
77 def getcoloredTxt(runs, color): # Will Look for colored text
78
79     coloredWords, word = [], "" #declares two Lists
80     for run in runs: # Goes through paragraph and searches for colored word
81         if run.font.color.rgb == RGBColor(*color):
82             word += str(run.text) # Saves everything found
83
84     elif word != "": # This will find the parentTags
85         # Adds the parent tags into these Lists
86         coloredWords.append(word)
87         parentTags.append(word)
88         parents.append(word)
89         word = ""
90
91     if word != "": # This will find the childTags
92         coloredWords.append(word + "\n") # Adds the colored words into list
93         child.append(word) # into list
94         withChild.append(word) # into list
95
96     return coloredWords # return colored tags and text
97
```

`generateReport()` function

- This function lets the user choose their own text file created, via paths to the documents they want the program to use when creating reports
- After choosing a text file, the program will go through each line in the text file and use it as a path to a word document.
- This function also calls the `readtxt()` function, to read each word document and find the red colored text

generateReport() function

```
> def generateReport(): #Will generate the report for tags
  global filepath
  global filepath2
  filepath = filedialog.askopenfilename(initialdir="/",
                                         title="",
                                         filetypes = (("all files", "*.*"), ("word documents", "*.docx")))

  file = open(filepath, 'r')
  #print(filepath)
  file.close()
  # Will store the filepath to the document as a string
  filepath2 = str(filepath)

  a = (filepath2)
  with open(a) as file_in:
    lines = []
    for line in file_in:
      lines.append(line)
    #counters += 1
    #for line in lines:
    #  print(line)
    #for x in counters:
    for line2 in lines:
      print(line2)
    # return filepath2, filtered_L
    line3 = str(line2)

  line4 = line3.replace('\\', '/')
  line5 = line4.replace('..', '..')
  line6 = line5.replace('.\\', '.')
  print(Line6)
  fullText = readtxt(filename=Line6,
                     color=(255, 0, 0))
  print(Line4)
  #filtered_L = readtxt(filename=filepath2, #For future use
  #                      color=(255, 0, 0))
  fullText10 = str(fullText)
  s = ''.join(fullText10)
  w = (s.replace ('[', '[\n'))
```

generateReport() function

```
paragraph = report3.add_paragraph()
filepath3 = str(line4.rsplit('/', 1)[-1]) # change filepath to something.docx
filepath3 = filepath3.split('..', 1)[0] # removes .docx of the file name
print(filepath3 + " added to the report")
nameOfDoc = (filepath3 + " added to the report\n")
T.insert(tk.END, nameOfDoc) #print in GUI

runner = paragraph.add_run("\n" + "Document Name: " + filepath3 + "\n")
runner.bold = True # makes the header bold
# w will be used in the future
w = (w.replace ('[', ''))
w = (w.replace ('', ''))

# creates a table
table = report3.add_table(rows=1, cols=2)

# Adds headers in the 1st row of the table
row = table.rows[0].cells
row[0].text = 'Front Tag'
row[1].text = 'Back Tag/tags'

# Adding style to a table
table.style = 'Colorful List'

# Now save the document to a location
report3.save('report3.docx')
e = 0

child2 = removeAfter(child) #removes everything after the parent tag if there is anything to remove
# while loop until all the parentTags has been added to the report

parents2 = copy.deepcopy(parentTags) # copy of parent tags list
parents2Copy.extend(parents2)
childCopy = copy.deepcopy(child2)
noParent = []
noParent2 = []
orphanChild = []
orphanChildParent = []
parents9000 = []
```

generateReport() function

- This function will also create a table for each of the documents showing all the child tags and parent tags found in the document.
- This function also stores all the text and tags it has extracted, saves the child tags, parent tags and text found, so that the program can use it in the other functions.

generateReport() function

```
193 parents2 = [s.replace(" ", "") for s in parents2] # gets rid of space
194 while parentTags:
195     row = table.add_row().cells # Adding a row and then adding data in it.
196     row[0].text = parentTags[0] # Adds the parentTag to the table
197     noParent.append(parentTags[0])
198
199     if e < len(fullText2): # as long as variable e is not higher than the lines in fullText2
200         if fullText2[e] in filtered_LCopy: #filtered_L contains the parent tags without a child tag
201             orphanChild.append(parentTags[0])
202             parentTags.remove(parentTags[0]) # Removes that tag after use
203             noParent2.append(" ")
204
205             parents9000.append(" ")
206             orphanChildParent.append(" ")
207             row[1].text = " " # No parent tag, so adds empty string to that cell
208             e += 1
209
210     elif fullText2[e] not in filtered_LCopy:
211         parentTags.remove(parentTags[0]) # Removes that tag after use
212         if child2:
213             row[1].text = child2[0] #Adds childTag to table
214             e += 1
215             parents9000.append(child2[0])
216             noParent.append(child2[0])
217             child2.remove(child2[0]) # Removed that tag from the list
218             .....
219             while parentTags: # In case there are any more parent tags left in the list
220                 row = table.add_row().cells # Adding a row and then adding data in it.
221                 row[0].text = parentTags[0]
222                 parentTags.remove(parentTags[0])
223                 while child2: #This is for orphan tags, but not finished
224                     row = table.add_row().cells # Adding a row and then adding data in it.
225                     row[1].text = child2[0]
226                     child2.remove(child2[0])
227                     .....
228
229     parents9.extend(parents9000)
230
231     # Make sure everything is cleared before the program gets the next document
232     child2.clear()
233     parentTags.clear()
234     child.clear()
235     report3.save('report3.docx') #Saves in document "report3"
236
237     global dict11
238     dict11 = dict(zip(parents2, childCopy)) #Creates a dictionary if there is a child tag and parent tag
239     dict1.update(dict11)
240
```

generateReport() function

```
241     noParent = [s.replace(" ", "") for s in noParent]
242     #dicts3 = dict(zip(noParent, noParent2))
243
244
245     orphanChild = [s.replace(" ", "") for s in orphanChild]
246
247     dict9000 = dict(zip(orphanChild, orphanChildParent)) # orphan dictionary
248     orphanDicts.update(dict9000)
249     OrphanChild2.extend([orphanChild])
250
251     text2 = removeParent(everything) # child tag and text
252     #text2 = removeChild(everything) # parent tags and text
253
254     # print(text2)
255     #text3 = removeParent(text2) # only text list
256     text9 = ('.....' + str(text2) + '.....') # child tag and text
257     text3 = removeChild(text2) # only text list
258     # print(text3)
259     text4 = removeText(text2) # child tags
260     # print(text4) #only parent tag list
261     #text7 = [s.replace(" ", "") for s in text3]
262     text8 = [s.replace(" ", "") for s in text4]
263
264     parents9000 = [x.strip(' ') for x in parents9000]
265     #dicts3 = dict(zip(parents2, childCopy))
266     dicts3 = dict(zip(parents2, parents9000))
267
268     dict10.update(dicts3)
269     dict2 = dict(zip(parents2, text3)) # creates a dictionary with child tags and text
270     dict100 = copy.deepcopy(dicts2)
271     sorted(dicts2.keys()) # sorts the keys in the dictionary
272     dict2Copy.update(dicts100)
273     #print(dicts2)
274
275     toggle_state2() # This will enable the generate report button
276     toggle_state3()
277
278     return filepath2, filtered_L
279     return parents2, dict2, dict10, dict2Copy, parents2Copy, fullText2, filtered_LCopy, dict3, orphanDicts, OrphanChild2
```

GenerateReport2() function

- The input data from this function is declared at the beginning count elements.
- By having these count methods, we can sort our dictionary and append our elements into a word document.
- By having these counters, we can compare it to the length of the dictionary and increment or append the elements.

GenerateReport2 function

```
294     def generateReport2():
295         # declaring counters
296         m = 0
297         k = 0
298         i = 0
299         o = 0
300         z = 0
301
302     # Calls remove child function and stores the text into a list called "orphanTagText"
303     orphanTagText = removechild(filtered_LCopy)
304
305     while m < len(dicts2Copy): # While counter m is less than the length of the dict
306         #if fullText2Copy[k] not in filtered_LCopy:
307         if z < len(dicts2Copy) and dicts2Copy:
308             z += 1
309
```

GenerateReport2() function

- As counter elements increment through the function, it then reads child tags from our input data and loops through our parents to our generated report.
- This is the backbone for the functionality for each parent tag. what allows us to display the tag, its requirement text, and children correlated to the parent.
- If we have a parent tag from a document that doesn't have an requirement text associated with the tag, then it will report back "Requirement text not found"

GenerateReport2() function

```
322     for key, value in dicts2Copy.items(): # For all the keys and values in the dict
323         report3.add_paragraph("\n") # Adds a newline to the report
324         m += 1 #Increment counter
325
326         # Checks if counter k is Less than Length of "fullText2Copy"
327         # and not an orphan tag
328         if k < len(fullText2Copy) and fullText2Copy[k] not in filtered_LCopy:
329
330             stringKey = str(key) # Converts the key to a string
331             stringKey2 = (stringKey.replace(' ', '')) # removes spaces in the string
332             # Variable "text" get the key at "stringKey2" in dictionary dicts10
333             text = dicts10[str(stringKey2)]
334             # Splits the text at the seperator "]" and stores it in PTags
335             PTags = text.split(']')
336             #The strip() method removes spaces, then adds "]" to all the elements
337             #in PTags
338             PTags = [s.strip() + ']' for s in PTags]
339             # pop() removes and element
340             PTags.pop()
341
```

GenerateReport2() function

```
344     for x in PTags: # For all the elements in PTags
345         keyCheck = (x.replace('[', '')).replace(']', '') # removes "[" an store in keyCheck
346         keyCheck2 = (keyCheck.replace('[', '')).replace(']', '') # removes "]"
347         keyCheck3 = (keyCheck2.replace('[', '')).replace(']', '') # removes "]" again if there is a nested bracket
348         keyCheck4 = (keyCheck3.replace('[', '')).replace(']', '') # removes spaces
349         report3.add_paragraph(x) # display the parent tag, included brackets
350
351     if keyCheck4 in dicts2Copy: # Checks if text of parent tag is found in dict2
352         report3.add_paragraph(dicts2Copy[str(keyCheck4)]) # add to report
353
354     else: # if text of parent tag is not found
355         report3.add_paragraph("Requirement text not found")
356
```

GenerateReport2() function

```
362     for b in PTags: # Another for loop for PTags, this time for child tags and text
363
364     if b == dict10[str(stringKey2)]: # Checks if b is in "dicts10" also
365         i += 1 # Increment counter
366         hx = dict10[str(stringKey2)] # Assign hx to key in "dicts10"
367         keys = [h for h, v in dict10.items() if v == hx] # finds all the children
368         # print(keys)
369         k += 1 # Increment counter
370         # keys are child tags of hx/the parent tag
371         for item in keys: # for all the child tags in "keys"
372             # Add child tag as a bullet point
373             report3.add_paragraph(item, style='List Bullet')
374             # Add text of child tag
375             para = report3.add_paragraph(dict10Copy[str(item)])
376             # adds indentation of text
377             para.paragraph_format.left_indent = Inches(0.25)
378
```

GenerateReport2() function

- Once the loop reaches a filtered tag, it will correlate it back as orphan tag.
- Once the module is done looping through the counters, the whole document is saved, and a message will be printed out on our console.

GenerateReport2() function

```
390     elif k < len(fullText2Copy) and fullText2Copy[k] in filtered_LCopy:
391         k += 1 # increment counter k
392         report3.add_paragraph("\n") # newLine to the report
393         if i < len(parents2Copy): # Check if counter i is Less than Length of
394             report3.add_paragraph(parents2Copy[i]) # Adds the childtag
395             if o < len(orphanTagText): # Check if counter os is Less than List "orphan"
396                 report3.add_paragraph(orphanTagText[o]) # Adds the text of orphan
397                 o += 1 # Increment counter o
398             if i < len(parents2Copy): # Checks if counter is Less than "parents2Copy"
399                 # Adds text to the report saying that this tag is an orphan tag
400                 report3.add_paragraph(parents2Copy[i] + " is an orphan tag")
401
402         i += 1 # Increment counter i
```

GenerateReport2() function

```
411     msg1 = ("\\nReport Generated\\n") # Adds string message to msg1
412     T.insert(tk.END, msg1) # displays msg1 in GUI
413     msg2 = ("You can now open up your report\\n") # Adds string message
414     T.insert(tk.END, msg2) # displays msg2 in GUI
415     #print("Report Generated")
416     #print("You can now open up your report")
417     report3.save('report3.docx') # Saves report as 'report3.docx'
418     toggle_state() #This will enable the getDoc button
419     toggle_state3() #This will enable the excel report button
420     return dict2Copy
421
```

Removes parent from the text

- `def removeParent(text): #removes tag`
- *This function gets a list from the user as parameter*
- *It removes the parent tags in the elements in the list*
- *Returns just the child tag and text in a newly created list*

Removes parent from the text

```
def removeParent(text): #removes tag
    childAfter = []
    for line in text:
        # removes parent tags
        childAfter = [i.rsplit('`', 1)[0] for i in text]
        # removes parent tags that are left
        childAfter = [re.sub("[\\{\\[ ].*?[\\}\\]]", "", e) for e in childAfter]
        # removes "pass", "fail", etc.
        childAfter = [re.sub("[" + re.escape(str(childAfter)) + "].*?[\\}\\]", "", e) for e in childAfter]
    return childAfter
```

Removes the text between the tags

- `removeText(text6): #this should remove everything before the tag`
 - This function gets a list from the user as parameter
 - It removes the text between the tags in each element in the list
 - #Returns the list without the text

Removes the text between the tags

```
433     def removeText(text6): #this should remove everything before the par  
434         # Goes through the list "text6" and removes all the text inside t  
435         childAfter = [s.split(None, 1)[0] for s in text6]  
436         return childAfter #Returns the list without the text except tag  
437
```

Function to remove "pass", "fail", etc.

- `removeAfter(childtags): #removes everything after the tag, except "pass", "fail"`
- *This function gets a list from the user as parameter*
- *#Returns the list without "pass", "fail", etc.*

Function to remove "pass", "fail", etc.

```
438 def removeAfter(childtags): #removes everything after the tag, example "pas
439     separator = ']' # use ] to look for what to remove
440     # Goes through the list "childtags" and removes all stuff like "pass", "fa
441     childAfter = [i.rsplit(']', 1)[0] + separator for i in childtags]
442     return childAfter #Returns the list without "pass", "fail", etc.
443
```

Remove childTags from paragraphs

- `removeChild(text): # Supposed to remove childTag, this one never fixing possibly`
- *This function gets a list from the user as parameter*
- *# Returns a list of paragraphs without the tags*

Remove childTags from paragraphs

```
444     def removechild(text): # Supposed to remove childTag, this one needs fixing
445         mylst = [] # List of paragraphs after removing tag
446         # Goes through the list "text" and removes tag
447         mylst = [s.split(None, 1)[1] for s in text]
448         return mylst # Returns paragraph without tag
449
```

Function to open word Report automation

- *# This function will open the word report automatically when called.*
- *# Also checks if you are using a Windows PC, Macbook, etc.*

Function to open word Report automation

```
450 # This function will open up the report automatically
451 # Also checks if you are using a Windows PC, Macbook, etc.|
452 def getDocument():
453     if platform.system() == 'Darwin':
454         subprocess.check_call(['open', 'report3.docx'])
455     elif platform.system() == 'Windows':
456         os.startfile('report3.docx')
457     # os.startfile(report3) # try either one for windows if the first option gives
458     else:
459         subprocess.call('xdg-open', report3)
460
```

Function to create an excel report

- *# Creates an excel report using Xlwings and Pandas functions*
- Traditional approach for displaying data with xlwings and having to be filled with these desired elements of child/parent tags and text.
- The autofit() method will help us display the according size of cell to its specific number of characters or text. Basically, helps us the middle requirement text and if there's more than one tag associated with it.

Function to create an excel report

```
461 # Creates an excel report
462 def createExcel():
463     book_arr = xw.App().books
464     wb = book_arr.add()
465     #wb = xw.Book() # Creating an new excel file.
466     # Select the first excel sheet, and rename it
467     excelReport = wb.sheets["Sheet1"]
468     #report = "report"
469     #excelReport.name = report
470     excelReport.range("B1").value = "Report"
471     excelReport.range("B1").api.Font.Size = 18 # Change font size
472     excelReport.range("B1").api.Font.ColorIndex = 2 # Change font color
473     excelReport.range('A1:S1').color = (0, 0, 255) # Change cell background
474
```

Function to create an excel report

```
482     # Inserts the datafram "df" that has the list "dicts2Copy", into the excel report
483     # ChildTag - Text
484     excelReport.range("A3").value = df
485
486     # Adding childTag header and specifies the font size, color and background color
487     excelReport.range("B3").value = 'Child Tag' # This is the heading 'Child Tag'
488     excelReport.range("B3").api.Font.Size = 14 # Change font size
489     excelReport.range("B3").api.Font.ColorIndex = 2 # Change font color
490     excelReport.range('B3:B3').color = (255, 0, 0) # Change cell background color
491
492     # Adding Text header and specifies the font size, color and background color
493     excelReport.range("C3").value = 'Text' # Header
494     excelReport.range("C3").api.Font.Size = 14 # Change font size
495     excelReport.range("C3").api.Font.ColorIndex = 2 # Change font color
496     excelReport.range('C3:C3').color = (0, 255, 0) # Change cell background color
497
498     # Inserts the datafram "df" that has the list "dicts10", into the excel report
499     # childTag - parentTag
500     excelReport.range("D3").value = df2
501
```

Function to create an excel report

```
502     # Adding parentTag header and specifies the font size, color and backgr  
503     excelReport.range("F3").value = 'Parent Tag' # header  
504     excelReport.range("F3").api.Font.Size = 14 # Change font size  
505     excelReport.range("F3").api.Font.ColorIndex = 2 # Change font color  
506     excelReport.range('F3:F3').color = (128, 128, 128) # Change cell background  
507  
508     # Adding childTag header  
509     excelReport.range("E3").value = 'Child Tag' # header  
510     excelReport.range("E3").api.Font.Size = 14 # Change font size  
511     excelReport.range("E3").api.Font.ColorIndex = 2 # Change font color  
512     excelReport.range('E3:E3').color = (255, 0, 0) # Change cell background  
513  
514     wb.sheets["Sheet1"].autofit() # autofit the width of columns  
515     wb.save('report.xlsx') # Saving excel report as 'report.xlsx'  
516  
517
```

Functions to activate the buttons on the

```
518     def toggle_state(): # this will re-enable getDoc button
519         getDoc.config(state="normal") # Change state to "normal"
520
521
522     def toggle_state2(): # this will re-enable generate report
523         genRep.config(state="normal") # Change state to "normal"
524
525
526     def toggle_state3(): # this will re-enable excel report button
527         getExcel.config(state="normal") # Change state to "normal"
528
```

Main

- Creates a word document and saves it as 'report3.docx'
- Declares a bunch of list and dictionaries, before creating the GUI using the Tkinter library

Main

```
530 if __name__ == '__main__':
531     # Creates a word document, saves it as "report_3, and also adds a heading
532     report3 = Document() # Create word document
533     report3.add_heading('Report', 0) # Add heading "report"
534     paragraph = report3.add_paragraph() # Paragraph
535     report3.save('report3.docx') # Saves the word report as "report3.docx"
536     dicts2Copy = {} # This will hold the dicts2 content from all the files provided
537
538     global parents2Copy # copy of parents2 List
539     parents2Copy = [] # List used as a copy
540
541     global filtered_L # Will store the ones without a child tag
542     filtered_L = [] # List to store the ones without a child tags
543
544     global filtered_LCopy # copy of "filtered_L"
545     filtered_LCopy = [] # List stores a copy of "filtered_L"
546
547     global fullText2Copy # copy of list: "fullText2Copy"
548     fullText2Copy = [] # This list will hold a copy of the list "fullText2Copy"
549
550     global parents2 #List of tags
551     parents2 = [] # stores tags in this list
```

Main

```
553     # creates a dictionary for parent and child tags
554     global dicts
555     dicts = {} # This dictionary will hold parent tags and child tags from document
556
557     global OrphanChild2
558     OrphanChild2 = [] # This list will hold orphan tags
559
560     global dicts10
561     dicts10 = {} # Dictionary holding parent tags and child tags
562     global dicts3
563     dicts3 = {} # will hold parentTag and text, Orphan tags
564     global dicts2
565     dicts2 = {} # will hold parentTag and text
566     global orphanDicts
567     orphanDicts = {} # orphan dictionary
568
569     global parents9
570     parents9 = [] # List of parentTags
```

Main

```
572     # declaring different lists that will be used to store, tags and sets
573     parentTags = []
574     parent = [] # This will be used to store everything
575     child = [] # Used to store child tags
576     noChild = [] # Used to store parentTags with no child
577     withChild = [] # Used to store parentTags with child tag
578     parents = [] #Will be used for future function
579
580     global orphanTagText
581     orphanTagText = [] # Will be used to hold text of orphanChildTags
```

GUI

- Creates a window with buttons.
- The buttons links to different functions in the program.
 - *Button 1 lets the user choose a text file, and link to the generateReport()*
 - Button 2 lets the user generate the report, and links to the generateReport function.
This button is disabled, until the toggle Button function is called.
 - *Button 3 lets the user open the word document report created and is linked to the getDocument() function.*
This button is also disabled, until the toggle Button function is called.
 - *Button 4 creates the excel report and is linked to the createExcel() function*
This button is also disabled, until the toggle Button function is called.
- Button 5 ends the program and closes the window.

GUI

```
583      # Creates the gui
584      window = Tk(className=' TARGET v.1.4.x ')
585      # Set window size #
586      window.geometry("380x360")
587      # Creates button 1
588      Button(window, text="Choose Document ", command=generateReport).pack()
589      # Creates button 2, will be disabled untill activate function is called
590      genRep = Button(window, text="Generate Report ", state= DISABLED,
591                      command=generateReport2)
592      genRep.pack()
593      # Creates button 3, will be disabled untill activate function is called
594      getDoc = Button(window, text="Open Generated Report ", state= DISABLED,
595                      command=getDocument)
596      getDoc.pack()
597      # Creates Excel button button 4, will be disabled untill activate function is called
598      getExcel = Button(text="Create Excel Report ", state= DISABLED, command=createExcel)
599      getExcel.pack()
600      # Creates button 5
601      button = Button(text="End Program", command=window.destroy)
602      button.pack()
603
```

GUI

```
608     # Message to user on the GUI
609     msg3 = ('1. Please choose your documents by clicking on \nthe "choose document"
610     T.insert(tk.END, msg3) #print in GUI
611
612     window.mainloop()
613
```