In [ ]:
```
#Assignment2.1
#By: Karim Bangcola Jr.
#Git:
```

In [ ]:
```
"""

Bad Words Censor Program
This program accepts two inputs: 1) a list of bad words, and 2) a sentence t
It will check the sentence against the list, with the final output being the

It is composed of 2 py files main.py and functions,py
main is simply the front-end, looping over the program execution until quit
functions holds all the actual working functions

"""
```

In [ ]:
```
"""
functions.py

functions involved:

    notEmptyCheck(a) = simply checks if input is length > 0. used in inputLo

    inputLoopString = loop that accepts an input prompt. it will keep asking
    if not valid, it will loop until the user does.

    censorSentence = asks the user for a sentence to censor. if the badWords
if not, it will iterate through each element of badWordsList, checking if it
it will slice the sentence and loop through each letter of the bad word, rep

    mainMenu = simple function that asks what option to do and calls the app
    way to quit the program.

    inputWords = accepts inputs from the user and stores it in badWordsList

    displayWords = displays all the contents of badWOrdsList

"""
```

In [ ]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 17 12:12:38 2024

@author: kbangcola
"""

#initialize badWords
badWordsList = []

def notEmptyCheck(a):
    return len(str(a)) > 0
```

```python
def inputLoopString(prompt):

    validInput = False #set flag to retry input if input is invalid

    #begin input looop

    while validInput == False:
        try:
            userInput = str(input(prompt))
            if notEmptyCheck(userInput):
                return userInput
                validInput = True
            else:
                print("ERROR: Values must not be not empty.")
        except ValueError:
            print ("ERROR: Input must be a valid string!") #restrat loop if

    #end input loop

def censor(sentence=''):
    sentence = inputLoopString("Input the sentence you want to censor: ").lc

    if len(badWordsList) <= 0: #check if badwords list is empty
        print("\nError: Bad words blacklist empty! Please input some bad wor
        print("Returning to main menu...")
        mainMenu()
    else: #if blacklist is not empty, proceed
        tempSentenceCensored = sentence #initialize temp variable for censor

        #begin censorship loop

        print("\nCensoring against blacklist...")
        for x in badWordsList: #loop through each word in the blacklist
            #debug
            #print("Now censoring " + x)
            if tempSentenceCensored.find(x) == -1: #if bad word not found ir
                #debug
                print(x + "not found. Skipping.")
                pass
            else:
                #debug
                #print(x + " found. Censoring...")
                #print("Sentence to censor: " + tempSentenceCensored)

                beginBadWord = sentence.find(x) #return start of bad word ir
                endBadWord = beginBadWord+len(x) #ending of bad word in sent

                #debug
                #print("Beginning of bad word: " + str(beginBadWord))
                #print("Ending of bad word: " + str(endBadWord))

                for y in range(len(x)): #loop through each char of the word
                    #print("Substituting character #" + str(y) + " at positi
                    tempSentenceCensored = tempSentenceCensored[:beginBadWor
                    #print(tempSentenceCensored)
                sentence = tempSentenceCensored #sasve the temp sentence as

        #end censorship loop

        print("\nYour censored sentence is: ")
        print(sentence)

            #debug stuff
            #print(beginBadWord) #need to add case when word not found
            #print(endBadWord)
            #print(sentence[:beginBadWord])
            #sentenceCensored = sentence[beginBadWord:(beginBadWord+len(x))]
            #print(sentenceCensored)

def mainMenu():
    print("\nWhat would you like to do today? Input the number of your choic
    print("\n1 to censor a sentence, \n2 to add words to the blacklist, \n3
    menuInput = int(input(("\nPlease select an option: " )))

    if menuInput == 1:
        censor()
    elif menuInput == 2:
        inputWords()
    elif menuInput == 3:
        displayWords()
    elif menuInput == 4:
        raise SystemExit #quit the program
    else:
        print("\nSorry, invalid input. Please try again!")
        mainMenu()

def inputWords():
    print("\n-- Bad word input module --")
    print("Current list of bad words to censor: ")

    displayWords()

    print("\nNow inputting bad words to blacklist. Input your word and press

    #begin input looop

    validInput = False #set flag to continue input loop

    while validInput == False:
        userInput = str(input("\nPlease input bad word [input 'done' to fini
        if notEmptyCheck(userInput): #check first if input is not empty
            if userInput.lower() != 'done': #check if user wants to quit inp
                badWordsList.append(userInput)
                print("Added word " + userInput +" to blacklist.")
            else: #exit loop if input is string and not empty
                validInput = True
        else: #restart loop if input is empty
            print("ERROR: Input must not be not empty.")

    #return to main menu
    print("\nReturning to main menu.")
```

```python
def displayWords():
    print("\n--START OF BAD WORDS BLACKLIST--")

    for x in badWordsList:
        print("\n" + x)

    print("\n--END OF BAD WORDS BLACKLIST--")
```

In [2]:
```python
"""
main.py

simply imports mainMenu and loops until exit.
"""
```

```
What would you like to do today? Input the number of your choice:

1 to censor a sentence,
2 to add words to the blacklist,
3 to view the blacklist,
4 to quit the program.
-- Bad word input module --
Current list of bad words to censor:

--START OF BAD WORDS BLACKLIST--

--END OF BAD WORDS BLACKLIST--

Now inputting bad words to blacklist. Input your word and press Enter. To qu
it input, type 'done' and press enter.
Added word fuck to blacklist.
Added word shit to blacklist.
Added word damn to blacklist.
Returning to main menu.

What would you like to do today? Input the number of your choice:

1 to censor a sentence,
2 to add words to the blacklist,
3 to view the blacklist,
4 to quit the program.
```

```
--START OF BAD WORDS BLACKLIST--

fuck

shit

damn

--END OF BAD WORDS BLACKLIST--

What would you like to do today? Input the number of your choice:

1 to censor a sentence,
2 to add words to the blacklist,
3 to view the blacklist,
4 to quit the program.
Censoring against blacklist...

Your censored sentence is:
lorem impsum **** dolor **** amet **** consecetur

What would you like to do today? Input the number of your choice:

1 to censor a sentence,
2 to add words to the blacklist,
3 to view the blacklist,
4 to quit the program.
```

```
An exception has occurred, use %tb to see the full traceback.

SystemExit

/home/kbangcola/.var/app/org.jupyter.JupyterLab/config/jupyterlab-desktop/jl
ab_server/lib/python3.12/site-packages/IPython/core/interactiveshell.py:358
5: UserWarning: To exit: use 'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

In [1]:
```
%cd bad_words
%pwd
```

```
/home/kbangcola/PSMDSRC103_2425_Bangcola/module2/assignment2.1/bad_words
```

Out[1]:
```
'/home/kbangcola/PSMDSRC103_2425_Bangcola/module2/assignment2.1/bad_words'
```

In [ ]:
```
"""
Quadratic Solver

This program uses the quadratic formula function provided in the last seatwo
As per the instructions, it will accept 3 inputs, solve them, and save the
results and inputs to a text file.

"""
```

In [3]:
```
%cd ..
```

```
/home/kbangcola/PSMDSRC103_2425_Bangcola/module2/assignment2.1
```

In [4]:
```
%cd quadratic
```

In [5]:
```
import math

def notEmptyCheck(a):
    return len(str(a)) > 0

def inputLoopFloat(prompt):

    validInput = False #set flag to retry input if input is invalid

    #begin input looop

    while validInput == False:
        try:
            userInput = float(input(prompt))
            if notEmptyCheck(userInput):
                return userInput
                validInput = True
            else:
                print("ERROR: Values must not be not empty.")
        except ValueError:
            print ("ERROR: Input must be a valid number!") #restrat loop if

def quadratic_formula(a, b, c):
    if b**2 - (4*a*c) < 0:
        x1 = (complex(-b, math.floor(math.sqrt(abs(b**2-(4*a*c)))))) / (2*a)
        x2 = (complex(-b, -1*math.floor(math.sqrt(abs(b**2-(4*a*c)))))) / (2
        return x1, x2
    else:
        x1 = (-b + math.sqrt(b**2 - (4*a*c))) / (2*a)
        x2 = (-b - math.sqrt(b**2 - (4*a*c))) / (2*a)
        return x1, x2

inputA = inputLoopFloat("Please enter the value for variable a: ")
inputB = inputLoopFloat("Please enter the value for variable b: ")
inputC = inputLoopFloat("Please enter the value for variable c: ")
result = quadratic_formula(inputA, inputB, inputC)
print("Now solving...")
print("The values of x are: ")
print(result)
print("\nNow saving results to file 'results.txt'...")

file = open("results.txt", 'w')

file.write("Quadratic Equation solved:")
file.write("\nValues of a, b, c:")
file.write("a = " + str(inputA))
file.write("b = " + str(inputB))
file.write("c = " + str(inputC))
file.write("\nValues of x:")
file.write(str(result))
file.close()

print("Done saving file!")
```

```
Now solving...
The values of x are:
((-1+1.4j), (-1-1.4j))

Now saving results to file 'results.txt'...
Done saving file!
```

In [7]:
```
"""
Output of the above file:
"""

file = open("results.txt", 'r')
data = file.read()
print(data)
file.close()
```

```
Quadratic Equation solved:
Values of a, b, c:a = 10.0b = 20.0c = 30.0
Values of x:((-1+1.4j), (-1-1.4j))
```

In [ ]:
```
"""
Follow-up questions:

Answer the following questions:

Why do built-in functions exist?

    Built-in functions exist to provide a basic level of functionality
within the core Python kernel.

What are the advantages/disadvantages of placing code inside functions
vs. sequential codes?

    By placing code inside functions, we can reduce redundancy within
our program. We can place repeatedly called code in a function and call
the function instea of copy/pasting the same code. We can also take
advantage of the local and global scope of variables - the same variable
can be reused multiple times for different purposes in different functions.
Functions also allow more flexibility in branching our program out to
take different paths - versus sequential code which only runs once top to bo
We can also use functons recursively, further reducing redundancy.

What is the different between a function and a module?
    A module is simply another Python file that can contain functions,
variables, code, etc. We can create a module containing functions and then
we can import that module from another python file. That way, we can
call that function without copy/pasting it again. This way we can separate t
parts of our program - presentation and logic separate.

Discuss the difference between a module and a package.
    A module is one python file but a package can consist of several modules
Popular packages include pandas, plotlib, scipy, etc.
```