SI 224 2023 PROJECT

Section A - Implementation:

Please refer to Section B for the program introduction and help documentation.

Importing Libraries

In [176…

```
import sys
!{sys.executable} -m pip install --upgrade pip
!{sys.executable} -m pip install omdb
!{sys.executable} -m pip install keyboard
```

```
Requirement already satisfied: pip in c:\users\dalen\anaconda3\lib\site-packages
(23.1.2)
Requirement already satisfied: omdb in c:\users\dalen\anaconda3\lib\site-packages
(0.10.1)
Requirement already satisfied: requests>=2.0.1 in c:\users\dalen\anaconda3\lib\sit
e-packages (from omdb) (2.28.1)
Requirement already satisfied: charset-normalizer<3,>=2 in c:\users\dalen\anaconda
3\lib\site-packages (from requests>=2.0.1->omdb) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\dalen\anaconda3\lib\site-p
ackages (from requests>=2.0.1->omdb) (3.3)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\dalen\anaconda3\l
ib\site-packages (from requests>=2.0.1->omdb) (1.26.11)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\dalen\anaconda3\lib
\site-packages (from requests>=2.0.1->omdb) (2022.9.14)
Requirement already satisfied: keyboard in c:\users\dalen\anaconda3\lib\site-packa
ges (0.13.5)
Requirement already satisfied: pandas in c:\users\dalen\anaconda3\lib\site-package
s (1.3.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\dalen\anaconda3\lib\site-
packages (from pandas) (1.21.5)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\dalen\anaconda3
\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in c:\users\dalen\anaconda3\lib\site-p
ackages (from pandas) (2022.1)
Requirement already satisfied: six>=1.5 in c:\users\dalen\anaconda3\lib\site-packa
ges (from python-dateutil>=2.7.3->pandas) (1.16.0)
```

Section A - Final Code

In [53]:

```
## Importing libraries
import requests
import json
import omdb
import keyboard
import pandas as pd

#Initialising variables
stop=False #Boolean variable
choice=0 #Integer variable
validSelection=True #Boolean variable
viewDFValid=False #Boolean variable
viewDFChoice=0 #Integer variable
discontinue=0 #Integer variable
```

```
#Initialising global variable
global title

#Opening textfiles
searchHistoryFile= open('Search History.txt','r+',encoding="UTF-8") #Textfile which
lastSearchFile= open('Last Searched.txt','r+',encoding="UTF-8") #Textfile which hol

#Defining search + print function
def search_title(title):
    parameters = {
        "t": title #Setting title as parameters for API search
    }
    response = requests.get('http://www.omdbapi.com/?i=tt3896198&apikey=64621bf0',p
    info= response.json() #Receive respnse as json data
    movieInfo=[] #Initialise movieInfo list
    newInfo='Result:'+info['Title']+'\n' #Formatting output from json data to movie
    searchHistoryFile.write(newInfo) #Writing formatted output of most recent searc
    newInfo='Title:'+info['Title'] +'\n' #As above
    movieInfo.append(newInfo) #As above
    newInfo='Genre:'+info['Genre'] +'\n'
    movieInfo.append(newInfo)
    newInfo= 'Year:'+ info['Year'] +'\n'
    movieInfo.append(newInfo)
    newInfo= 'Runtime:'+ info['Runtime'] +'\n'
    movieInfo.append(newInfo)
    newInfo= 'Director:'+ info['Director'] +'\n'
    movieInfo.append(newInfo)
    newInfo= 'Awards:'+ info['Awards'] +'\n'
    movieInfo.append(newInfo)
    newInfo= 'Actors:'+ info['Actors'] +'\n'
    movieInfo.append(newInfo)
    for j in range(len(movieInfo)): #Print contents of movieInfo list
        print(movieInfo[j])
        lastSearchFile.write(movieInfo[j]) #Write contents of movieInfo list to all
    lastSearchFile.write('\n') #Add extra line to text file entry for formatting

#Defining dataframe view of recent search function
def view_df():
    parameters = {
        "t": title #Setting title as parameters for API search
    }
    response = requests.get('http://www.omdbapi.com/?i=tt3896198&apikey=64621bf0',p
    info= response.json() #Receive responnse as json data
    dfData = { #Entering data
    'Title': info['Title'],
    'Genre': info['Genre'],
    'Year': info['Year'],
    'Runtime': info['Runtime'],
    'Director': info['Director'],
    'Awards': info['Awards'],
    'Actors': info['Actors']
    }
    df_searchResult = pd.DataFrame(dfData, index=['1']) #Create df using pandas
    display(df_searchResult) #Output df

response=requests.get('http://www.omdbapi.com/?i=tt3896198&apikey=64621bf0') #Check
print(response) #200 - OK. The request was successful.

print('\t This program enables you to look up and receive information about movies
for line in searchHistoryFile: #Prints titles of previous searches
    print(line)
print('\n Please press "n" to continue') #Waits for user interaction before proceed
keyboard.wait('n')
```

```
print('What would you like to do first?') #Presents navigation options to user
try: #try / except statement allowing for user error in response
    choice=int(input('\t 1: Search by title. \n \t 2: View previous search informat
except ValueError: #Accounts for non-numeric input from user
    print('Please enter a valid selection')
    print('\n Please press "n" to continue') #Waits for user interaction before pro
    keyboard.wait('n')
    choice=int(input('\t 1: Search by title. \n \t 2: View previous search informat
if choice>3: #Accounts for input outside of option 1-3 range
    validSelection=False #Boolean variable to determine validity of navigation sele
    while validSelection==False: #While navigation input is invalid LOOP
        print('Please make a valid selection.')
        print('\n Please press "n" to continue')
        keyboard.wait('n') #Wait for user interaction
        choice=int(input('\t 1: Search by title. \n \t 2: View previous search info
        if choice== 1 or choice==2 or choice==3:
            validSelection=True #Reevaluates if navigation input is valid and while
if choice==1: #Navigation input option 1
    while stop==False:
        try: #Allowing for user input errors
            title= input('Enter a title: ') #User enters title they wish to search
            search_title(title) #Implement SEARCH_TITLE function
            while viewDFValid==False:
                print('\n Would you like to view this information as a data frame?
                viewDFChoice=int(input('\t 1: Yes \n \t 2: No \n'))
                try: #Catch user input errors
                    if viewDFChoice==1:
                        view_df() #Display df
                        viewDFValid=True #Exit WHILE loop
                        break
                    elif viewDFChoice==2:
                        viewDFValid=True #Exit WHILE loop
                        break
                except ValueError:
                    print('Please enter a valid selection')
                    print('\n Please press "n" to continue')
                    keyboard.wait('n') #Wait for user interaction
            print('\n What would you like to do next?') #Ask user for further optio
            try: #Further allowing for user input errors
                choice=int(input('\t 1: Search by title. \n \t 2: View previous sea
            except ValueError: #Catch user error
                print('Please enter a valid selection')
                choice=int(input('\t 1: Search by title. \n \t 2: View previous sea
            if choice==1: #If new user input is navigation option 1
                continue #Continue back to while loop
            elif choice==2: #If new user input is naviagtion option 2
                for line in lastSearchFile: #Print text file contents: all informat
                    print(line)
                print('\n The above information contains all previously searched ti
            else:
                print('\t You have exited.') #Exiting program
                searchHistoryFile.close()
                lastSearchFile.close()
                stop=True
                break
        except KeyError: #IF MOVIE NOT FOUND
            print('\n \t Movie not found!')
            try: #catching user error
                choice=int(input('\t 1: Search by title. \n \t 2: View previous sea
            except ValueError:
                print('Please enter a valid selection')
                discontinue= int(input('Would you like to \n \t 1:Enter a different
            if discontinue != 1 and discontinue != 2: #if input is invalid- neither
```

```
                print('\t Please enter 1 or 2 \n press "n" to continue \n')
                keyboard.wait('n') #Wait for user interaction
                discontinue= int(input('\n Would you like to \n \t 1:Enter a differ
            elif discontinue==2: #Exiting program
                print('\t You have exited.')
                searchHistoryFile.close()
                lastSearchFile.close()
                stop=True
            #else:
                #title= input('Enter a title: ') #User enters title they wish to se
                #search_title(title)
elif choice==2: #If navigation option 2 selected
    for line in lastSearchFile: #Print text file contents: all information from pre
        print(line)
    searchHistoryFile.close()
    lastSearchFile.close()
    print('\n The above contains all information about previously searched titles.'
elif choice==3:
    searchHistoryFile.close() #Exiting program
    lastSearchFile.close()
    print('\t You have exited.')
```

<Response [200]>
	This program enables you to look up and receive information about movies using the Open Movie Database API
	Here is a list of previously searched titles:

Result:Rick and Morty

Result:The Mandalorian

Result:The Book of Boba Fett

Result:The Office

 Please press "n" to continue
What would you like to do first?
	1: Search by title.
	2: View previous search information.
	3: Exit.
2
Title:Rick and Morty

Genre:Animation, Adventure, Comedy

Year:2013–

Runtime:23 min

Director:N/A

Awards:Won 2 Primetime Emmys. 18 wins & 37 nominations total

Actors:Justin Roiland, Chris Parnell, Spencer Grammer

Title:The Mandalorian

Genre:Action, Adventure, Fantasy

Year:2019–

Runtime:40 min

Director:N/A

Awards:Won 14 Primetime Emmys. 58 wins & 122 nominations total

Actors:Pedro Pascal, Chris Bartlett, Katee Sackhoff

Title:The Book of Boba Fett

Genre:Action, Adventure, Sci-Fi

Year:2021–

Runtime:N/A

Director:N/A

Awards:Won 1 Primetime Emmy. 4 wins & 12 nominations total

```
Actors:Temuera Morrison, Ming-Na Wen, Frank Trigg


Title:The Office

Genre:Comedy

Year:2005–2013

Runtime:22 min

Director:N/A

Awards:Won 5 Primetime Emmys. 51 wins & 195 nominations total

Actors:Steve Carell, Jenna Fischer, John Krasinski


 The above contains all information about previously searched titles.
```

Section B - Design documentation:

Introduction

This program enables the user to search for various movies, series or episodes using the OMDb (Open Movie Database) API. The user is presented with different navigation options and receives information in an interactive manner in order to minimise user error. When the user enters a title, the program first confirms the title exists on the OMDb, and then displays all relevant information for the entry. At the same time, the title of the search is written to a textfile 'Search History', which contains all titles from previous searches. Additionally, all information from each previously searched title is also written to a textfile 'Last Searched', so that information for all previous searches can also be viewed. After a navigation option is selected and enacted, the user is again presented with the other navigation options so they may choose to search another title, view previous search information, or exit. Step by step information is provided for the user once the program is run in an attempt to minimise user error.

The code of the program includes the following sections:

- Importing libraries
- Declaring and initialising variables
- Opening text files
- Defining functions
- Checking the status of the API connection
- Navigation
- Main code

Demonstration

The most integral aspect of the code is the function search_title. The function gets the title to search for from the user as input and uses the OMDb API to find the data associated with the title. The json library is then utilised to output this data as a dictionary. The information from the dictionary is then transferred to a list in order to make it easier to output and interact with. The function then uses a for-loop to output the data from the list and writes the relevant formatted information to each text file so the information may be accessed when later recalled.

In [ ]:
```
#Defining search + print function
def search_title(title):
    parameters = {
        "t": title #Setting title as parameters for API search
    }
    response = requests.get('http://www.omdbapi.com/?i=tt3896198&apikey=64621bf0',p
    info= response.json() #Receive respnse as json data
    movieInfo=[] #Initialise movieInfo list
    newInfo='Result:'+info['Title']+'\n' #Formatting output from json data to movie
    searchHistoryFile.write(newInfo) #Writing formatted output of most recent searc
    newInfo='Title:'+info['Title'] +'\n' #As above
    movieInfo.append(newInfo) #As above
    newInfo='Genre:'+info['Genre'] +'\n'
    movieInfo.append(newInfo)
    newInfo= 'Year:'+ info['Year'] +'\n'
    movieInfo.append(newInfo)
    newInfo= 'Runtime:'+ info['Runtime'] +'\n'
    movieInfo.append(newInfo)
    newInfo= 'Director:'+ info['Director'] +'\n'
    movieInfo.append(newInfo)
    newInfo= 'Awards:'+ info['Awards'] +'\n'
    movieInfo.append(newInfo)
    newInfo= 'Actors:'+ info['Actors'] +'\n'
    movieInfo.append(newInfo)
    for j in range(len(movieInfo)): #Print contents of movieInfo list
        print(movieInfo[j])
        lastSearchFile.write(movieInfo[j]) #Write contents of movieInfo list to all
    lastSearchFile.write('\n') #Add extra line to text file entry for formatting
```

The second function view_df is less integral to the functioning of the code, and instead serves as an extra feature available to the user to view the data. Similar to the search_title function, the function uses the title given by the user and uses the OMDb API to find the data associated with the title. The json library is then utilised to output this data as a dictionary. However, view_df instead utilises a dataframe in order to display the dictionary information received from the API.

In [ ]:
```
def view_df():
    parameters = {
        "t": title #Setting title as parameters for API search
    }
    response = requests.get('http://www.omdbapi.com/?i=tt3896198&apikey=64621bf0',p
    info= response.json() #Receive responnse as json data
    dfData = { #Entering data
    'Title': info['Title'],
    'Genre': info['Genre'],
    'Year': info['Year'],
    'Runtime': info['Runtime'],
    'Director': info['Director'],
    'Awards': info['Awards'],
    'Actors': info['Actors']
```

```
    }
    df_searchResult = pd.DataFrame(dfData, index=['1']) #Create df using pandas
    display(df_searchResult) #Output df
```

The remainder of the code mainly deals with navigation options and catching user errors. This is done through a series of if statements in conjunction with try except statements to catch errors. While many of these structures contain more nested if statements and try except statements within them which can get repetitive, the use of the above functions helps to minimise this.

The example section of code below demonstrates the use of an if statement for navigation options:

In [ ]:
```
if choice>3: #Accounts for input outside of option 1-3 range
    validSelection=False #Boolean variable to determine validity of navigation sele
    while validSelection==False: #While navigation input is invalid LOOP
        print('Please make a valid selection.')
        print('\n Please press "n" to continue')
        keyboard.wait('n') #Wait for user interaction
        choice=int(input('\t 1: Search by title. \n \t 2: View previous search info
        if choice== 1 or choice==2 or choice==3:
            validSelection=True #Reevaluates if navigation input is valid and while
```

The example section of code below demonstrates the use of a try except statement to catch various user errors:

In [ ]:
```
try: #Catch user input errors
    if viewDFChoice==1:
        view_df() #Display df
        viewDFValid=True #Exit WHILE loop
        break
    elif viewDFChoice==2:
        viewDFValid=True #Exit WHILE loop
        break
except ValueError:
    print('Please enter a valid selection')
    print('\n Please press "n" to continue')
    keyboard.wait('n') #Wait for user interaction
```

The use of these structures enables the user to navigate the program accoring to what they would like to do.

When the program is first run, the user is presented with a brief description of the function of the program as well as the output from the text file 'Search History', so that the user can see the titles previously searched using the program.

The use of the keyboard.wait function is to wait for user interaction before proceeding, in the hope of decreasing the likliness of user error when entering their navigation selection or other input.

The relevant code to demonstrate the use of the keyboard.wait function can be seen below:

In [ ]:
```
print('\n Please press "n" to continue') #Waits for user interaction before proceed
keyboard.wait('n')

print('What would you like to do first?') #Presents navigation options to user
```

```
try: #try / except statement allowing for user error in response
    choice=int(input('\t 1: Search by title. \n \t 2: View previous search informat
except ValueError: #Accounts for non-numeric input from user
    print('Please enter a valid selection')
    print('\n Please press "n" to continue') #Waits for user interaction before pro
    keyboard.wait('n')
    choice=int(input('\t 1: Search by title. \n \t 2: View previous search informat
```

The rest of the code is mainly different iterations of this, giving the user new navigation options according to what option was previously chosen.

Key Design Decisions

This program includes many key design decisions. These include the use of functions to incorporate the main features of the program as well the decision to base the program on a selection based navigation system, due to the lack of GUI. Additionally, formatting the output data in a neat, coherent manner was a priority- and many design decisions were made with that in mind.

The Use of Functions for Key Design Decisions

I initially opted not to use a funcion for the section of code now defined under the search_title function, and instead wrote the code where it was applicable to the navigation choice made. However, as I incorperated more options through the navigation options, I realised the best decision would be for the search_title process to be a function. This was due to factors such as the need for the action to be recalled multiple times throughout the code, as well as the amount of lines of code required for the action and the time it took to develop the action and get it working correctly.

Once I realised the effectiveness of the search_title function and the ways it improved the coding of the overall program, it became clear that the view_df process would also be most useful as a function for the same factors: it's need to be called in multiple instances throughout the code due to the nature of the navigation system used.

The Use of a Navigation System as a Key Design Decision

Due to my previous coding experience in Pascal, I was accustomed to utilising a GUI to get input from the user. This influenced my decision to use a numbered navigation system for this project, as I wanted the user to have different options for input that the program would be able to interpret in a specific way with minimal user error.

However, the navigation system does have a few functional issues. Due to the different navigational branches the user is able to choose, and the lack of a GUI, the navigation options don't always link back to the three main options presented to the user throughout the program:

1: Enter a title to search for 2: View previous search information 3: Exit

I wanted the navigation system to always link back to these three options no matter which of the first two options were selected. Unfortunately, between trying to catch user errors with try except statements and accounting for user choice, this became difficult to keep up

with. While I did use loops in many of these instances in an attempt to keep the navigation as cohesive as possible, the limited options for the project interface hindered this.

The Format of the Output Data as a Key Design Decision

Due to the nature of the project scope and the requirement of various forms of text output such as a text file, I decideded I wanted to prioritise the neat-ness and and organisation of the output. I did this through utilising '\n' for adding a new line and '\t' for tab, which made the text output easier to read with the limited formatting options available. This is demonstrated by the code excerpt below:

In [ ]:
```
choice=int(input('\t 1: Search by title. \n \t 2: View previous search information.
```

Key Logic Aspects

The key logic aspects of the program include the numbered navigation system which utilises a series of if statements to determine which action should take place in accordance with which numbered option was selected.

An example of this can be viewed below:

In [ ]:
```
try: #try / except statement allowing for user error in response
    choice=int(input('\t 1: Search by title. \n \t 2: View previous search informat
except ValueError: #Accounts for non-numeric input from user
    print('Please enter a valid selection')
    print('\n Please press "n" to continue') #Waits for user interaction before pro
    keyboard.wait('n')
    choice=int(input('\t 1: Search by title. \n \t 2: View previous search informat
if choice>3: #Accounts for input outside of option 1-3 range
    validSelection=False #Boolean variable to determine validity of navigation sele
    while validSelection==False: #While navigation input is invalid LOOP
        print('Please make a valid selection.')
        print('\n Please press "n" to continue')
        keyboard.wait('n') #Wait for user interaction
        choice=int(input('\t 1: Search by title. \n \t 2: View previous search info
        if choice== 1 or choice==2 or choice==3:
            validSelection=True #Reevaluates if navigation input is valid and while
```

This excerpt of code is just a small example of how if statements are utilised in the program for navigation, which demonstrates the use of logic gates in the program.

The code above begins with asking the user for their navigation selection, which may only be the options 1, 2, or 3- which is outlined in the input question posed to the user. This input function is also nested in a try except statement, in an attempt to catch any ValueErrors the user's input may result in.

If the input the user enters does result in a ValueError, the program alerts the user to their error and asks them to reattempt with valid input. The user must then press the key 'n' to continue, this feature was added in an effort to minimise user error and reemphasise the incorret input the user has previosuly entered. The program then repeats the original naviagtion options.

Here the first if statement of the code excerpt is encountered. It's purpose is to also catch errors of invalid input. For example, entering the number 4 would not invoke a ValueError in the try except statement, as 4 is an acceptable integer and not any kind of string or char. However, 4 is not a valid naviagtional option presented to the user. The if statement therefore checks that the integer the user enters is within the range of navigational choices, 1-3, and if it is not, the boolean variable validSelection is set from True to false and the following while loop which uses validSelection as its conditional is entered.

The while loop first alert's the user through a print statement that their input is again invalid, and to please make a valid selection and press 'n' to continue. The whie loop then asks the user to again enter their navigational selection. This is followed by an if statement which requires the conditions that the choice variable from the user's input either be equal to 1,2 or 3 to be valid. If the choice variable is equal to 1,2 or 3, the boolean variable validSelection is set to True and the while loop ends due to the change in conditional.

The code of the rest of the program, which deals similarly with matters of navigation and the processes that each navigational option enacts, is included below. It demonstrates how the main body of the code for the program relys on if statements as logic gates for which navigational option the user has chosen.

In [ ]:
```
if choice==1: #Navigation input option 1
    while stop==False:
        try: #Allowing for user input errors
            title= input('Enter a title: ') #User enters title they wish to search
            search_title(title) #Implement SEARCH_TITLE function
            while viewDFValid==False:
                print('\n Would you like to view this information as a data frame?
                viewDFChoice=int(input('\t 1: Yes \n \t 2: No \n'))
                try: #Catch user input errors
                    if viewDFChoice==1:
                        view_df() #Display df
                        viewDFValid=True #Exit WHILE loop
                        break
                    elif viewDFChoice==2:
                        viewDFValid=True #Exit WHILE loop
                        break
                except ValueError:
                    print('Please enter a valid selection')
                    print('\n Please press "n" to continue')
                    keyboard.wait('n') #Wait for user interaction
            print('\n What would you like to do next?') #Ask user for further optic
            try: #Further allowing for user input errors
                choice=int(input('\t 1: Search by title. \n \t 2: View previous sea
            except ValueError: #Catch user error
                print('Please enter a valid selection')
                choice=int(input('\t 1: Search by title. \n \t 2: View previous sea
            if choice==1: #If new user input is navigation option 1
                continue #Continue back to while loop
            elif choice==2: #If new user input is naviagtion option 2
                for line in lastSearchFile: #Print text file contents: all informat
                    print(line)
                print('\n The above information contains all previously searched ti
            else:
                print('\t You have exited.') #Exiting program
                searchHistoryFile.close()
                lastSearchFile.close()
                stop=True
```

```
                break
        except KeyError: #IF MOVIE NOT FOUND
            print('\n \t Movie not found!')
            try: #catching user error
                choice=int(input('\t 1: Search by title. \n \t 2: View previous sea
            except ValueError:
                print('Please enter a valid selection')
                discontinue= int(input('Would you like to \n \t 1:Enter a different
            if discontinue != 1 and discontinue != 2: #if input is invalid- neither
                print('\t Please enter 1 or 2 \n press "n" to continue \n')
                keyboard.wait('n') #Wait for user interaction
                discontinue= int(input('\n Would you like to \n \t 1:Enter a differ
            elif discontinue==2: #Exiting program
                print('\t You have exited.')
                searchHistoryFile.close()
                lastSearchFile.close()
                stop=True
            #else:
                #title= input('Enter a title: ') #User enters title they wish to se
                #search_title(title)
elif choice==2: #If navigation option 2 selected
    for line in lastSearchFile: #Print text file contents: all information from pre
        print(line)
    searchHistoryFile.close()
    lastSearchFile.close()
    print('\n The above contains all information about previously searched titles.'
elif choice==3:
    searchHistoryFile.close() #Exiting program
    lastSearchFile.close()
    print('\t You have exited.')
```

The above code uses an if statement with three options, if choice is equal to 1,2 or 3. Each of these if / elif statements represnets a navigational option the user may choose. The first option, the most detailed option, contains multiple other if statements and try except statements in order to catch user errors and navigate the program. The functions search_title and view_df are also used in option 1 presented here, demonstrating how the use of functions for the proccesses that would need to be called repeatedly all in all made the code easier to read, less cumbersome and increased efficiency.

The above code does however contain a lot of repetition, and is at times not the most efficient. These are issues I will reflect on in the 'Retrospective Section' below.

Data Collection

The program collects data in two ways. Program navigational data is collected through input from the user, while the data assosiated with the searched title is collected throught the OMDb API.

An API (Application Programming Interface) is a set of defined rules or software that enables different applications, computers, or software to communicate with each other. It acts as an intermediary between these different applications or systems to transfer data. A common example of how it achieves this, and relevant to this program, is through retrieving data.

In terms of this program, the user asks the program for information about a search title, this information which can be found on the OMDb. The OMDb API then retrieves the data from

the OMDb for the user, and outputs it back to the user through this program. This illustrates how the API collects data from the OMDb.

The OMDb specifically is interacted with through the dedicated OMDb API, which takes a set of parameters relevant to the required information and searches the database according to these parameters, such as 'i','t','type' or 's'.

Data is also collected from the user directly through the use of the input function, this includes data related to navigational choices as well as titles to search the OMDBb for, and retreive information about.

Retrospective Section

The program initially seemed quite straight forward and I didn't encounter any major errors that stopped the program from running that I felt I wouldn't be able to figure out, or sit with for over a few hours. My main issues were trying to make the navigational options as efficient as possible and ensuring the user input valid data. Aside from the text file issue I describe below, coding the program and making improvments as I progressed through it was swift. The biggest aspect of this was restructuring the code after over half the program was completed to rework the functions which overall made the code easier to work with, more efficient and less cumbersome.

There was a bit of a learning curve to practically implement the python libraries I was more unfamilair with, such as json and pandas. However any errors I encountered merely made me more familiar with the workings and details of these libraries, especially pandas as I opted to add an extra feature to my program using a dataframe.

I detail the aspects of the project which could be improved upon below:

Problems Encountered and Improvments

I did encounter a few problems while coding this project. These were mainly due to the limited nature of the navigational system and the limited way in which data could be collected from the user, as well as processing the direct output from the OMDb itself into a text file.

Navigation System

Ideally, the navigational system used would always return to a kind of 'home screen', in which the three main appropriate navigational features would always be returned to at the end of a different navigational selection. This would entail this 'home screen' being shown after the user has searched for their desired title, or viewed the previous information available or search another title. However, this constant return to 3 options became tricky to code due to the never ending possibilities of which combination of navigational choices the user could opt for, as well as the lack of a comprehensive GUI which would present more options.

While it was possible to achieve this to an extent. However, the amount of while loops, if statements, nested loops, nested if statements, nested try except statements, and normal try except statements started to become cluttered and confusing to follow, all on the small

chance the user would want to repeat certain navigational options in a certain order. While I'm sure there was a more comprehensive and efficent way to achieve this kind of 'home screen' system, I decided due to time constraints that mine was comprehenisve enough for the scope of the project and the user's intentions.

Using functions for the key design decisions did remedy this to an extent, and I beleive the use of Object Orientated Coding could have assisted as well. But again, unfortunately due to time constratints I did not have enough time left to completely redeisgn my program based around these revelations, and instead implemented them as much as I could, to a reasonable degree which I beleved would help the functionality of the navigational system.

Data Collection

Due to the lack of GUI, I found it difficult collecting data from the user solely through the input function. This meant requesting any information from the user one line at a time and only being able to flag the input data as invalid once it could be processed through the code. This would have been much easier with a comprehenisve GUI which allowed multiple types of input data for different parameters through features such as text boxes, input boxes, or multi select boxes.

Text File Output

This was the biggest issue I encountered, as I simply did not have the depth of knowledge required to prevent it from happening, but understoof how to reset it each time it happened. I also struggled to find any internet resources on the matter which was in the scope of my understanding.

Due to certain characters in the data output by OMDb, such as '/' or '-', the text file I used to keep track of information from previously searched titless kept encoding itself into a format Jupyter Notebook could not work with. Jupyter Notebook apparently requires textfiles to be 'UTF-8' encoded, a format which apparently did not support the use of characters such as '/' or '-'. Now each time the text file changed to 'ANSI' encoding to cope with the characters from the OMDb output, Jupyter Notebook was unable to work with it. I could fix this through opening the text file in my laptop Notepad and resaving it as 'UTF-8' encoded, and thus reaccess the information of the previously searched titles, however I was unable to find a way to stop the text file from becoming 'ANSI' encoded in the first place.

I researched the matter heavily and did attempt to remedy it by specifying the text file should be 'UTF-8' encoded when the file was opened, but I still occassionally encountered the problem and the text file would then be unreadable by Jupyter Notebook until resaved as 'UTF-8' encoded.

Help Documentation

The program should be used according to the instructions presented to the user at each navigational option.

The program makes use of a navigational system with 3 options to present the user with different actions, many of these actions may be repeated more than once.

In the event that the user may like to select a naviagtional option which is unavailable or not presented, it is advised to best restart the program, as all information from previous searches will be saved and stored on the associated text file accordingly.

In the event the 'Last Searched' text file presents a written error within the text file about 'UTF-8' encoding, the best course of procedure is to open the text file in computer's default text file viewer, and resave the text file with 'Encoding:' set to 'UTF-8' instead of 'ANSI'. The default text file viewer should ask whether you would like to overwrite the previosuly existing text file, to which the answer is yes. This will ensure no data is lost from the text file of previous searches due to the encoding issues present.

In the case of any kind of ValueError or KeyError, please ensure the input entered is in the correct format required. For navigational options, this is mainly limited to an integer between 1-3.