Filetype Identification
Summary and Execution Flow

Problem Statement:

With the enormous number of languages and file types used for writing logical source or for data purposes, it is very important for a product like BlueOptima to effectively identify and categorize a file into its type. And this has to be done solely based on Extension and Name of the file itself.
This work sample requires you to identify different sources that could be used to identify details of a file type like following (but not limited to)

1. Short Description (explaining the usage of the file type)
2. Language Family (Java, Python, Perl, etc.)

A summary of how the project has been implemented along with figures of input and its respective output and also a general execution flow of the project.

Execution Flow:

1. Extract data from four different data sources using python script and store it in the **DFileIdentification.csv** file.
2. Implement the program file **classOutput.py** and take filenames as input from the user. Make a list of filenames and create an **input.csv** file.
3. Read **input.csv** and store all extensions in a list. Check each extension whether it is valid or not.
4. After validating an extension, search each in the **DFileIdentification.csv** file.

5. After the particular row has been found from data, write it in the **output.txt** file in a user-readable format.
6. Create custom unit test cases of each method of **classOutput.py** by **unittest** framework in python.
7. Validate each test case by running the testing script and check if any of the test case fails or not.

Input:

```
E:\DESTOP\fileTypeIdentification>python classOutput.py
Enter a list of extensions separated by comma binaryTree.cpp,output1.a$,ml.r,roman.php,avc.c


user list of extensions is ['binaryTree.cpp', 'output1.a$', 'ml.r', 'roman.php', 'avc.c']

E:\DESTOP\fileTypeIdentification>
```

Taking input of file extensions from users in CLI.

```
file_name
binaryTree.cpp
output1.a$
ml.r
roman.php
avc.c
```

The input file is found in **input.csv** and above is the sample.

Output:

This is a readable form of data stored in the **output.txt** file.

```
Title : .cpp
Category :  Developer Files
Type :  C++ Source Code File
Description :  A CPP file is a source code file written in C++. a popular programming
Application :  Microsoft Visual Studio Code|Eclipse IDE for C Developers|BloodshedSof
Reference  :  https://fileinfo.com//extension/cpp

Not valid extension!

Title : .r
Category :  Developer Files
Type :  R Script File
Description :  An R file is a script written in R. a programming language used for st
Application :  Ratfor|Ratfor|Ratfor
Reference  :  https://fileinfo.com//extension/r
```

Steps to Run Program:

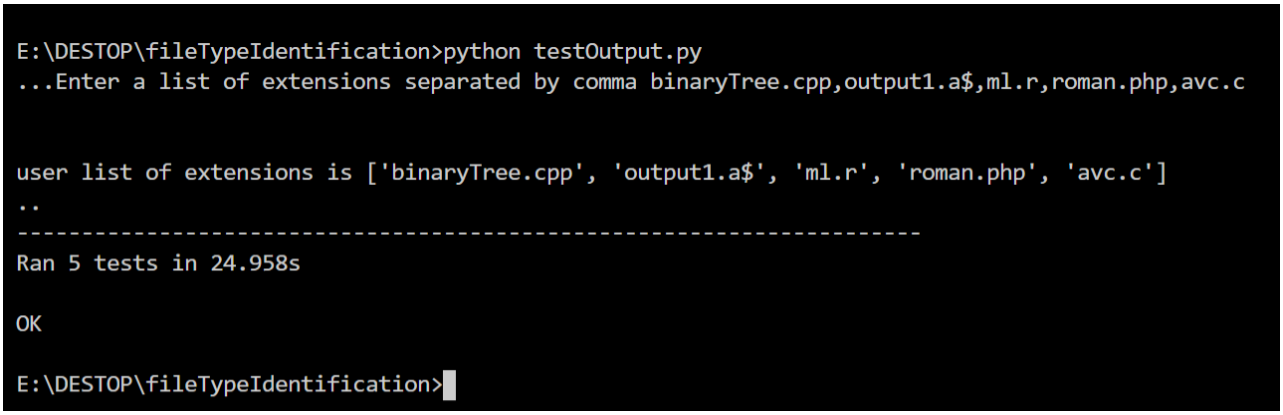
1. Implement **classOutput.py** and take input from user in the console.
2. Check the **output.txt** file for the results.
3. To test the **classOutput.py** script implement **testOutput.py** and check the results in the console.

Unit Testing:

Unit Testing is done on Output(**classOutput.py**) and Scraping (**scrapClass.py**) scripts. Classes are made in testing script to check each and every method of scripts. Each method contains edge unit test cases to check whether the output of particular method is correct according to our test case or not. All methods are implemented by unittest framework in python.

In **testOutput.py** we have tested the following methods -

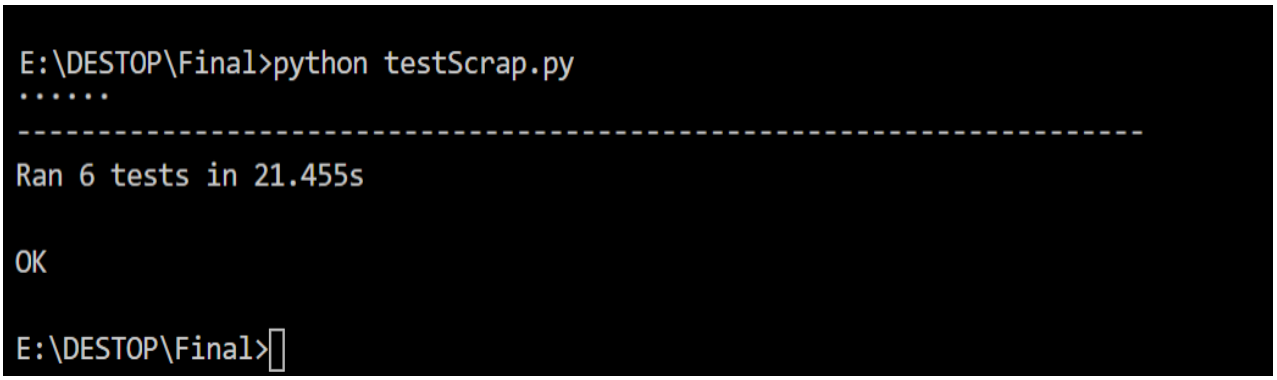
1. **Input.csv** has the correct input of filenames given by the user or not.
2. Input.csv has the same number of rows and columns as provided by the user respectively. We have also tested some rows as a part of edge cases where data is matched with custom input.
3. Each extension given by the user is in valid format i.e should contain only alphabets and numbers.
4. Each extension which we have searched in the whole dataset is producing output as expected or not.



Unit test of **testOutput.py** file

In **testScrap.py** we have tested the following methods -

1. Check whether the title of each data source to be the same as the custom expected output given by us.
2. Check some content of each data source to be the same as expected output is given by us.
3. Random rows of each data source to be as same as the expected output is given by us.



Unit test of **testScrap.py** file

Libraries and Frameworks Used:

- **Beautiful Soup**: Python library for pulling out data from HTML web repositories.
- **requests**: Python library allows us to send HTTP requests and maintaining HTTP connection.
- **urllib**: Python package that collects several modules for working with URLs.
- **re**: Python library provides regular expression operations.
- **NumPy**: Python library used for working with arrays.
- **csv**: Python module allows users to read/write tabular data in csv format.
- **Pandas**: Python package provides data structures designed to make working with structured data easy.
- unittest: Python framework supports test automation, aggregation of tests into collections, and independence of the tests from the reporting framework.

Result

According to the problem statement, we are able to identify different sources as well as provided the required output. We have taken the input filenames from user and stored it in **input.csv** file. We have searched each extension of input.csv file and provided the output in **output.txt** in a readable format so that we will get information about each extension in one place. We have also tested our scraping and main scripts by using unittest framework and they are providing the results as expected.