





**Table of Contents**

[**1.**](#_heading=h.30j0zll) **Introduction: 2**

[**1.1.**](#_heading=h.1fob9te) **Objectives if Projects 2**

[**1.2.**](#_heading=h.3znysh7) **Deliverables 2**

[**2.**](#_heading=h.2et92p0) **Design a scraper to scrape information from a website 2**

[**2.1.**](#_heading=h.tyjcwt) **Coding a scraper 2**

[**2.2.**](#_heading=h.3dy6vkm) **Testing a scraper 2**

[**3.**](#_heading=h.1t3h5sf) **Design a formatter to format output data in CSV file 2**

[**3.1.**](#_heading=h.4d34og8) **Coding a formatter 2**

[**3.2.**](#_heading=h.2s8eyo1) **Testing a formatter 2**

[**4.**](#_heading=h.17dp8vu) **Design a cracker to crack a password 3**

[**4.1.**](#_heading=h.3rdcrjn) **Coding to generate the combination of password 3**

[**4.2.**](#_heading=h.26in1rg) **Testing password generation 3**

[**4.3.**](#_heading=h.lnxbz9) **Coding to classify the generated passwords 3**

[**4.4.**](#_heading=h.35nkun2) **Testing password classification 3**

[**5.**](#_heading=h.44sinio) **Reflection 3**

[**5.1.**](#_heading=h.2jxsxqh) **Write the reflection of student1 3**

[**5.2.**](#_heading=h.z337ya) **Write the reflection of student2 3**

[**5.3.**](#_heading=h.3j2qqm3) **Write the reflection of student3 3**

[**6.**](#_heading=h.1y810tw) **Reference Page 3**

[**7.**](#_heading=h.4i7ojhp) **Appendix Page(s). 4**

1. **Introduction:**

Write a paragraph (less than 100 words) to describe what is project all about

* 1. **Objectives if Projects**

Write a paragraph/list (100-150 words) to describe the Objective of the project

* 1. **Deliverables**

Write a paragraph/list (100-150 words) to describe What are the deliverables of the project.

1. **Design a scraper to scrape information from a website**

Web scraping is a simple means of collecting data from different websites. It allows the Users to collect and manage data as per their requirements. It has wide applications in domains such as price monitoring and collecting huge datasets for various machine learning tasks.

In this component of the project, a Python based scraping tool Scrapy is used to scrape information from the first page of Amazon website's search on earphones. The information extracted includes product name, the link to the product image and the ratings of the product. The output is displayed on the terminal along with being stored in a JSON file.

* 1. **Coding a scraper**

The code for the **website\_spider.py** of the scraper along with it’s **items.py** is as follows:

**website\_spider.py**

import scrapy

import csv

from ..items import WebsitescraperItem

class WebsiteSpiderSpider(scrapy.Spider):

name = 'website\_spider'

start\_urls = ['https://www.amazon.in/s?k=earphones+with+mic&i=electronics&rh=p\_72%3A1318476031&dc&crid=2N58U5A6XCGT2&qid=1586958396&rnid=1318475031&sprefix=ear%2Celectronics%2C355&ref=sr\_nr\_p\_72\_1']

def parse(self, response):

product = WebsitescraperItem()

product\_name = response.css('.a-color-base.a-text-normal').css('::text').extract()

product\_image\_link = response.css('.s-image-fixed-height .s-image::attr(src)').extract()

product\_rating = response.css('.a-icon-alt::text').extract()

product['product\_name'] = product\_name

product['product\_image\_link'] = product\_image\_link

product['product\_rating'] = product\_rating[4:]

yield product

pass

**items.py**

import scrapy

class WebsitescraperItem(scrapy.Item):

# define the fields for your item here like:

product\_name = scrapy.Field()

product\_image\_link = scrapy.Field()

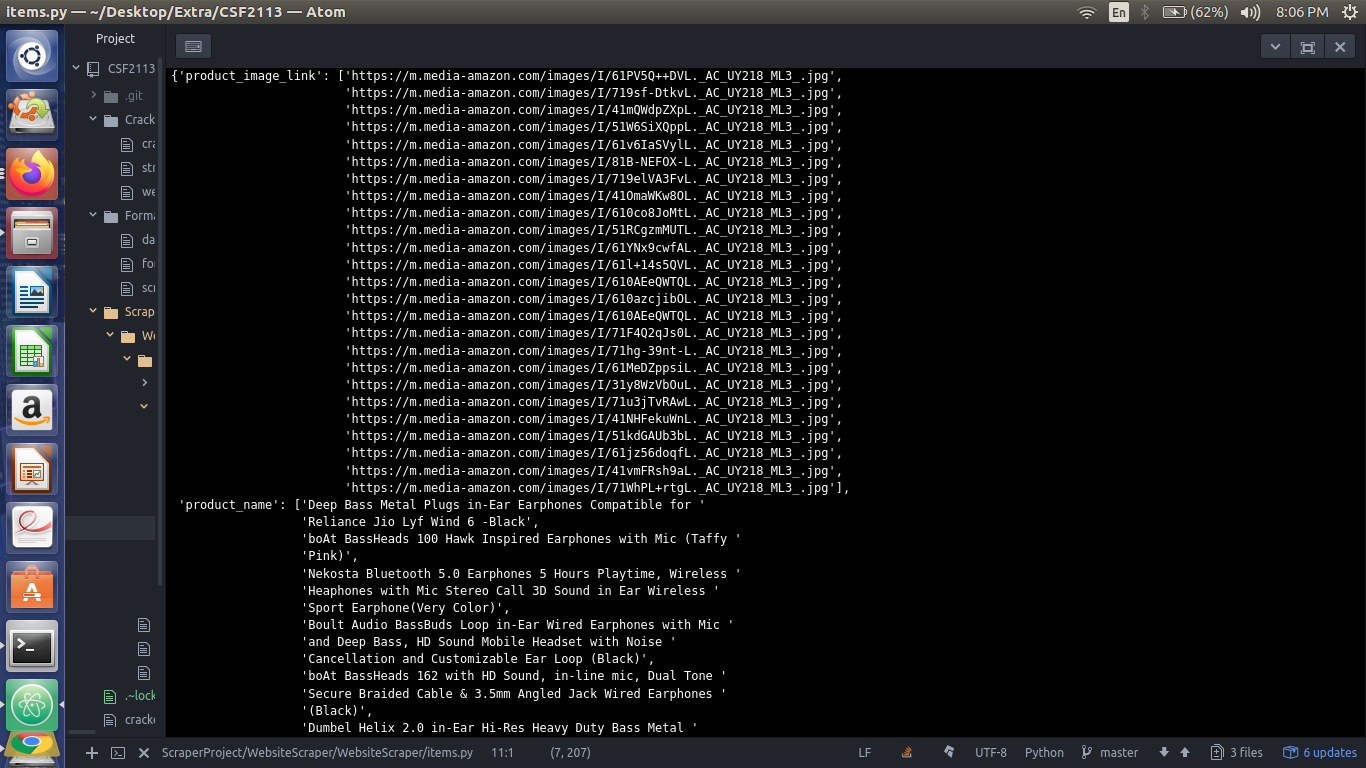
product\_rating = scrapy.Field()

pass

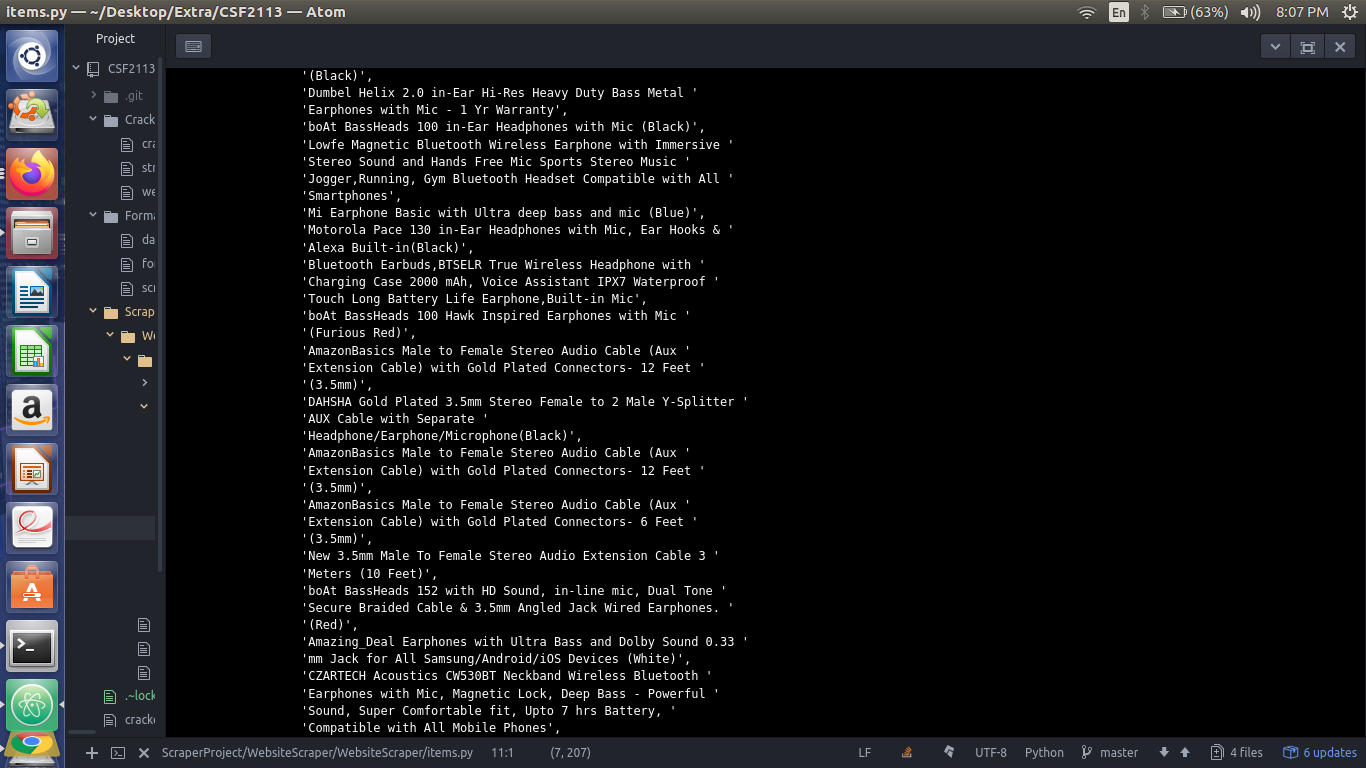
For the entire working code, refer to the code submitted.

* 1. **Testing a scraper**

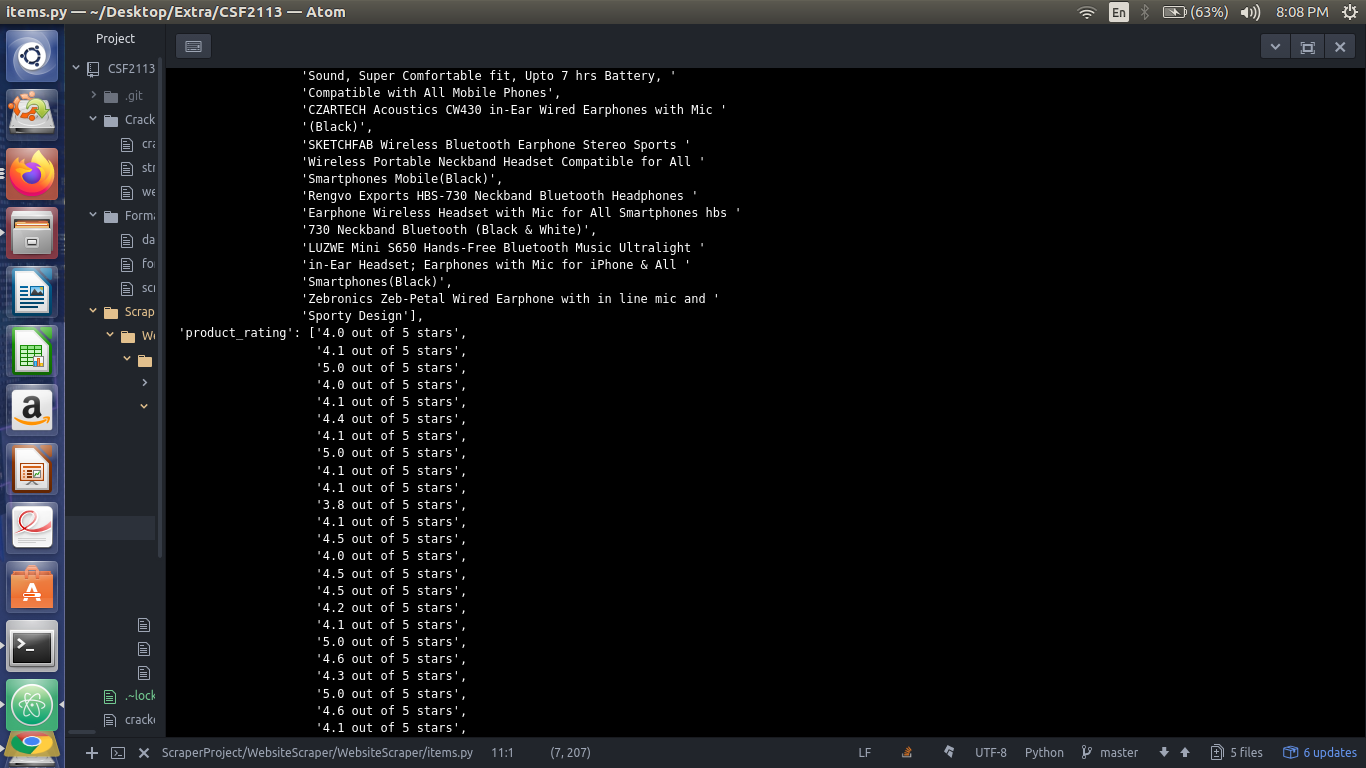
The output displayed on the terminal is as follows:



Terminal Output Part 1



Terminal Output Part 2



Terminal Output Part 3

The output is also stored in a JSON file:



Output JSON file

1. **Design a formatter to format output data in CSV file**

The formatter takes the “json” file generated by the scrapper as its input and returns the data in a “tsv” file named “scrapped\_data.tsv”, in a human friendly format. The json file obtained through scrapping does not have data in a human friendly format. Hence, the data needs to be formatted according to the various attributes that it represents. For our present project, we have decided to use scrap “amazon.in” and we obtain data corresponding to three different tags, which are :

1. Product name

2. Product image url

3. Product rating

The formatter parses the json file and creates a data-frame using the python library pandas. The columns in the data-frame correspond to the above mentioned attributes. All scrapped data-instances are arranged according to the same and then stored in a tab-seperated-value manner (as the instances contained strings which contain commas, we could not use csv file format for seperation). The formatter saves the dataframe in the “scrapped\_data.tsv”.

* 1. **Coding a formatter**

import sys

import pandas as pd

""" The script takes the input in the following format :

-> python formatter.py <path for the json file obtained through scrapper>

and the script produces the output in a file named scrapped\_data.tsv """

# the following piece checks for the availability of the input

if len(sys.argv) == 1:

print("Need Filename!")

sys.exit(-1)

else:

path = sys.argv[1]

# opening the file in read\_only mode

try :

file = open(path, "r")

except :

print("File failed to open! Please verify if the path-name is correctly specified")

quit()

# the input is expected as a json file and hence, the following piece of code is to extract all the relevant data from a json file

tag = 0

for each in file:

if tag == 1 :

data = each[:-1]

break

tag = tag + 1

file.close()

# formatting the data

i = 0

columns = []

dataset = []

while i < len(data):

if data[i] == ":":

if data[i-1] == '"':

name = ""

j = -2

while data[i+j] != '"':

name = name + data[i+j]

j = j - 1

columns.append(name[::-1])

aux = ""

tag = 0

while data[i] != "]":

if data[i] == "[":

tag = 1

i = i + 1

if tag == 1:

aux = aux + data[i]

i = i + 1

dataset.append(aux)

i = i + 1

column\_data = [[] for i in range(len(columns))]

tag = 0

for each in dataset:

i = 0

while i < len(each):

if each[i] == '"':

word = ""

i = i + 1

while each[i] != '"':

word = word + each[i]

i = i + 1

column\_data[tag].append(word)

i = i + 1

tag = tag + 1

# encoding the data into a pandas data\_frame

dataset = [[] for i in range(len(column\_data[0]))]

i = 0

while i < len(column\_data[0]):

j = 0

aux = []

while j < len(columns):

aux.append(column\_data[j][i])

j = j + 1

dataset[i] = aux

i = i + 1

# saving the data as a tab-seperated file using pandas

df = pd.DataFrame(dataset, columns = columns)

df.to\_csv("scrapped\_data.tsv", sep = "\t", index = False)

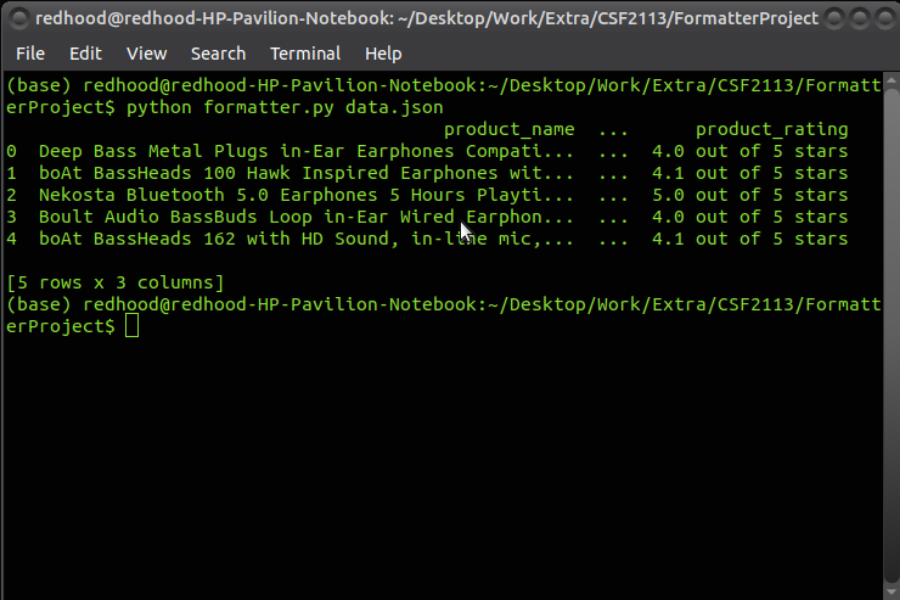
* 1. **Testing a formatter**

\* The screenshots have also been added in the “Images” folder.

\* The file generated as the output has also been included in the ‘FormatterProject’ folder.

**Fig: The above picture shows the snapshot of the ‘scrapped\_data.tsv’ file**





**Fig: The above figure shows the output in the terminal. The top 5 rows of the**

**dataframe has been printed (dataframe.head())**

1. **Design a cracker to crack a password**

The cracker designed in the current project performs a variety of tasks.

To start with, we have defined passwords into two categories :

1. Strong – passwords which contain at least one small-alphabet, one capital-alphabet, one numeric instance and one special character.

2. Weak – all other strings consisting of valid characters but not satisfying at least one of the above criteria.

The cracker is then designed to generate passwords randomly. These can be either strong or weak and of any length that the user desires. For this, generate\_strong\_password() and generate\_weak\_password() functions can be used respectively.

The check\_pass\_word\_type() function in the cracker takes a candidate password as its input and determines whether the password is valid or invalid, and if it is valid, whether it is strong or weak. The above mentioned conditions are utilized for the decision making process. The generate\_password\_dictionary() function takes in the number of strong passwords and the number of weak passwords to be generated, it generates these passwords randomly (all of length = 8) and stores in a python dictionary. The dictionary has keys as the string representing the password and the values as it’s corresponding md5 hash. The save\_dictionary() function takes in a password\_dictionary (as described in the previous step) and saves all the passwords in the dictionary in two separate files, which are - ‘strong\_passwords.txt’ and ‘weak\_passwords.txt’. All the passwords in the dictionary are classified on the run into the two types, and saved in the corresponding text files respectively.

Additionally, the cracker file also provides for :

1. Loading a dictionary from a text-file (where each line corresponds to a password string).

2. Using a dictionary to perform brute-password cracking (it uses a dictionary and attempts to crack a password(provided as an md5 hash input) by comparing the md5 hash values of all the passwords stored in the dictionary.

* 1. **Coding to generate the combination of password and to classify the generated passwords**

import random

import hashlib

import numpy as np

""" The passwords can be of two types, strong or weak.

Strong passwords are made up of a combination of at least one capital letter, small letter, number and a special character.

Weak passwords do not cater to all the above conditions. """

# 33 - 47 and 58 - 64 and 91 - 96 and 123 - 126: special characters

# 48 - 57 : numbers

# 65 - 90 : capital letters

# 97 - 122 : small letters

# declaring the various types of allowed characters in the password. If a password contains any other characters it is deemed invalid

special = [i for i in range(33,48)]

special.extend([i for i in range(58,65)])

special.extend([i for i in range(91,97)])

special.extend([i for i in range(123,127)])

numbers = [i for i in range(48,58)]

capital = [i for i in range(65,91)]

small = [i for i in range(97,123)]

# gives a random numric character

def get\_random\_number():

return chr(numbers[np.random.randint(low = 0, high = len(numbers), size = 1)[0]])

# gives a random small-alphabet character

def get\_random\_small():

return chr(small[np.random.randint(low = 0, high = len(small), size = 1)[0]])

# gives a random capital-alphabet character

def get\_random\_capital():

return chr(capital[np.random.randint(low = 0, high = len(capital), size = 1)[0]])

# gives a random special character

def get\_random\_special():

return chr(special[np.random.randint(low = 0, high = len(special), size = 1)[0]])

# function to print a dictionary in 'key : value' format

def print\_dict(d):

for each in d.keys():

print(each, end = ": ")

print(d[each])

# the following function generates a random strong password (the default length of the generated password is 8)

def generate\_strong\_password(size = 8):

if size < 4:

print("Error! A strong password needs to be at least 4 characters in length")

return "-1"

num = get\_random\_number()

small = get\_random\_small()

cap = get\_random\_capital()

spec = get\_random\_special()

out = np.random.randint(low = 33, high = 127, size = size-4)

pwd = num + small + cap + spec

for each in out:

pwd = pwd + chr(each)

pwd = ''.join(random.sample(pwd, len(pwd)))

return pwd

# the following function generates a random weak password (the default length of the generated password is 8)

def generate\_weak\_password(size = 8):

rand = np.random.randint(low = 0, high = 4, size = 3)

pwd = ""

while len(pwd) < size:

rand2 = np.random.randint(low = 0, high = 3, size = 1)

if rand[rand2][0] == 0:

pwd = pwd + get\_random\_capital()

elif rand[rand2][0] == 1:

pwd = pwd + get\_random\_number()

elif rand[rand2][0] == 2:

pwd = pwd + get\_random\_small()

else :

pwd = pwd + get\_random\_special()

return pwd

# returns 0 for a weak password and returns 1 for a strong password and returns -1 if the password is invalid

def check\_pass\_word\_type(password):

arr = [0 for i in range(4)]

i = 0

while i < len(password):

if ord(password[i]) in special:

arr[0] = 1

elif ord(password[i]) in numbers:

arr[1] = 1

elif ord(password[i]) in capital:

arr[2] = 1

elif ord(password[i]) in small:

arr[3] = 1

else :

return -1

i = i + 1

for each in arr :

if each == 0:

return 0

return 1

# takes in the number of strong passwords and weak passwords to be generated and generates a dictionary

# the dictionary contains the passwords as its key value and it's md5 hash as its value

def generate\_password\_dict(num\_strong = 10, num\_weak = 10):

passwords = {}

i = 0

while i < (num\_strong):

\_ = generate\_strong\_password()

if \_ not in passwords.keys():

passwords[\_] = hashlib.md5(\_.encode("utf-8")).hexdigest()

i = i + 1

i = 0

while i < (num\_weak):

\_ = generate\_weak\_password()

if \_ not in passwords.keys():

passwords[\_] = hashlib.md5(\_.encode("utf-8")).hexdigest()

i = i + 1

return passwords

# takes a dictionary as the input and stores the passwords in the dictionary in two seperate files. The seperation is based on the basis of password type (strong and weak)

def save\_dictionary(dic):

f = open("weak\_passwords.txt", "w")

for each in dic.keys():

if check\_pass\_word\_type(each) == 0:

f.write(each)

f.write("\n")

f.close()

f = open("strong\_passwords.txt", "w")

for each in dic.keys():

if check\_pass\_word\_type(each) == 1:

f.write(each)

f.write("\n")

f.close()

return 1

# takes in path name as the input and generates a dictionary with all the passwords present in the path name

def load\_dictionary(path):

try :

f = open(path, "r")

except :

print("Invalid path name!")

quit()

d = {}

for each in f :

each = each[:-1]

if check\_pass\_word\_type(each) in [0,1]:

d[each] = hashlib.md5(each.encode("utf-8")).hexdigest()

f.close()

return d

# takes in a dictionary and a password and adds the password to the dictionary

def add\_password\_to\_dict(dic, password):

if check\_pass\_word\_type(password) == -1:

print("Invalid password!")

return -1

if password in dic.keys():

print("Password already exists!")

return -1

dic[password] = hashlib.md5(password.encode("utf-8")).hexdigest()

return 1

# takes in a dictionary as the input and gives information regarding the passwords stored in the dictionary

def analyze\_dic(dic):

strong\_pass = 0

weak\_pass = 0

invalid\_pass = 0

for each in dic.keys():

if check\_pass\_word\_type(each) == 1:

strong\_pass = strong\_pass + 1

elif check\_pass\_word\_type(each) == 0 :

weak\_pass = weak\_pass + 1

else :

invalid\_pass = invalid\_pass + 1

print("The dictionary contains: ")

print(strong\_pass, " strong passwords")

print(weak\_pass, " weak passwords")

print(invalid\_pass, " invalid passwords")

return 1

# takes a dictionary of passwords and a md5 hash as the input and attempts to crack the password.

# if successful, it returns the password from the dictionary

def crack\_pass\_using\_dic(dic, inp):

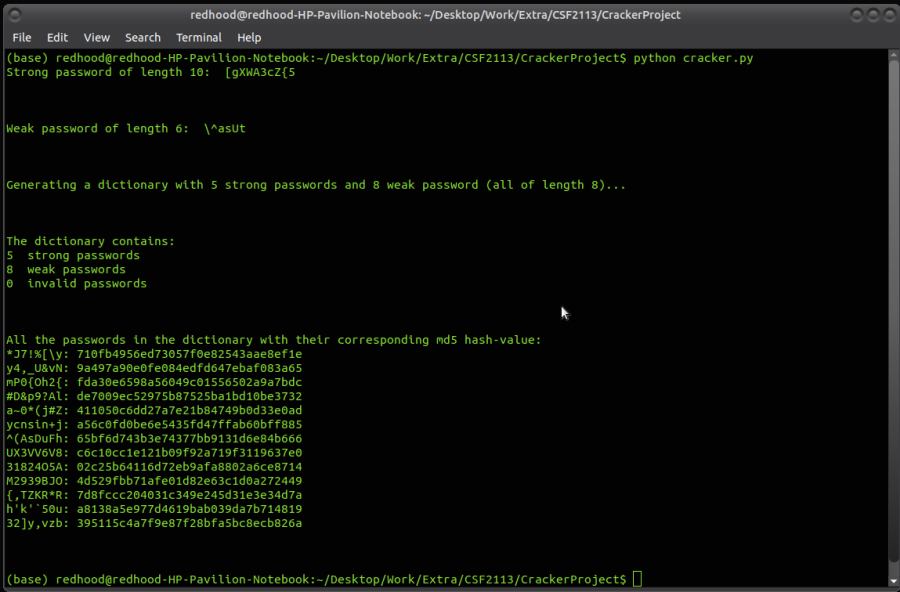
for each in dic.keys():

if dic[each] == inp:

return (1, each)

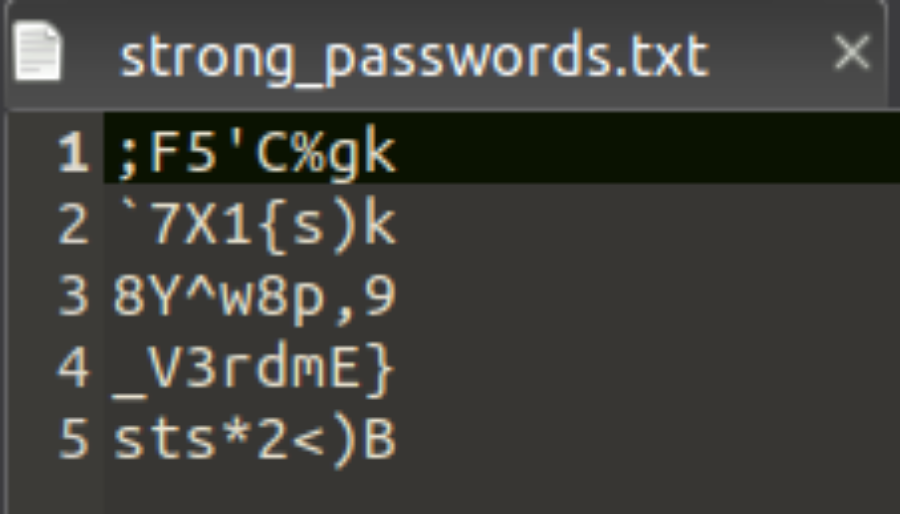
return (0, "No match found")

* 1. **Testing password generation**

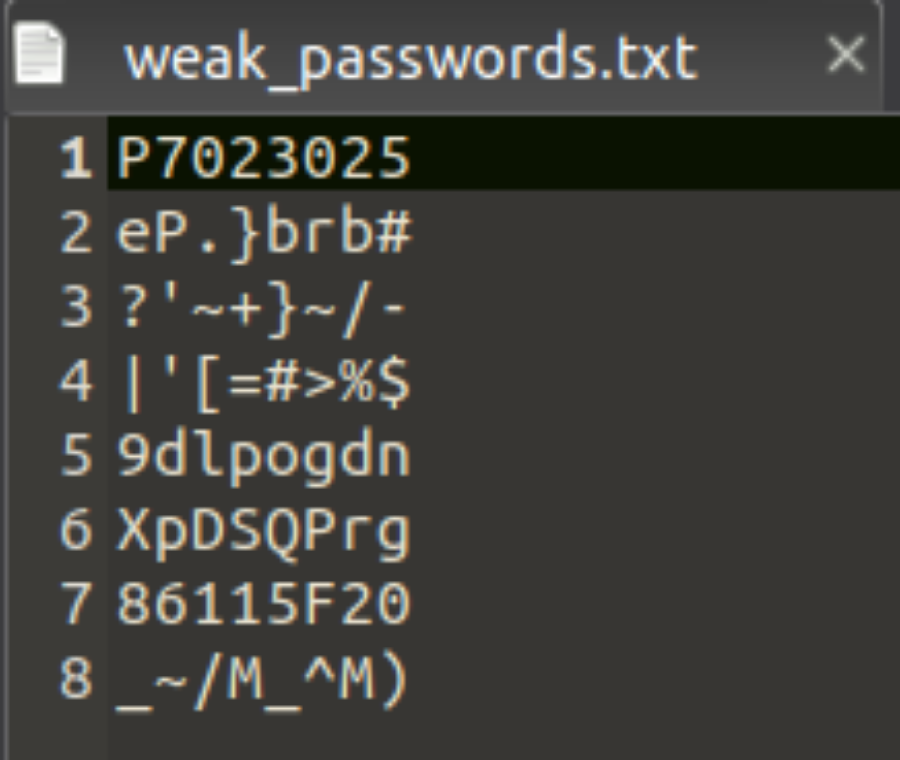


**Fig: The above figure shows the terminal output of the cracker.**

* 1. **Testing password classification**



**Fig: The above figure shows the text-file where the generated strong passwords have been stored**



**Fig: The above figure shows the text-file where the generated weak passwords have been stored**

1. **Reflection**
   1. **Write the reflection of student1**

Write down the reflection of student1

* 1. **Write the reflection of student2**

Write down the reflection of student1

* 1. **Write the reflection of student3**

Write down the reflection of student1

1. **Reference Page**

Include all the external references that you might have used. Use MLA or APA referencing style (Any One).

1. **Appendix Page(s).**

Include any other documents/code/information related to the project.