ENSF 338
Laboratory 1

Lab Section 02

Anhad Wander
UCID: 30208390

Haseeb Tahir
UCID: 10190846

Manmohit SIngh
UCID: 30216112

January 29, 2025

Part 1:
(Input Script)

```
Users > haseebtahir > ENSF338 > 338lab1.py > ...
/Users
hello_world(number, name):
    print(f"Hello, world! My group number is {number}. My name is {name}.")

hello_world(11, "Haseeb Tahir")
```

(Output)

```
haseebtahir@macbookair ENSF338 %  cd /Users/haseebtahir/ENSF338 ; /usr/bin/env /opt/homebrew/bi
n/python3 /Users/haseebtahir/.vscode/extensions/ms-python.debugpy-2024.14.0-darwin-arm64/bundle
d/libs/debugpy/adapter/../../debugpy/launcher 63643 -- /Users/haseebtahir/ENSF338/338lab1.py
Hello, world! My group number is 11. My name is Haseeb Tahir.
```

Part 2.1:

```
Lab 1 > Lab1OG.py > ...
'''
i)  The function of this code is to find the roots of a quadratic equation. It checks if
    the value under the square root for the quadratic formula is above zero and if so
    initiates the code accordingly.

ii) The error in the code was during the print statements where the opening quotation (')
    was closed with an apostrophe (') instead of a closing quotation and therefore the code
    could not run due to the syntax error.
'''

import sys
import math
def do_stuff():
    a = float(sys.argv[1])
    b = float(sys.argv[2])
    c = float(sys.argv[3])
    d = b**2 - 4*a*c
    if d > 0:
        root1 = (-b + math.sqrt(d)) / (2*a)
        root2 = (-b - math.sqrt(d)) / (2*a)
        print(f'The solutions are: {root1}, {root2}')
    elif d == 0:
        root = -b / (2*a)
        print(f'The solution is: {root}')
    else:
        print('There are no real solutions.')

do_stuff()
```

Part 2.2:

Part 2.3:
Private - repo

Part 2.4:

Part 2.5:

Script

ex2.5.py > ...

```python
import json
import timeit

def size_42(n):
    if type(n) == dict:
        for key, value in n.items():
            if key == "size":
                n[key] = 42
            else:
                size_42(value)
    elif type(n) == list:
        for i in n:
            size_42(i)

with open('large-file.json', 'r') as lrg_data:
    data = json.load(lrg_data)

time_taken = timeit.timeit(lambda: [size_42(i) for i in data], number=10)

avrg_time = time_taken / 10
print(f"Average time to modify 'size' values: {avrg_time:.6f} seconds")

rvrs_data = data[::-1]
with open('output.2.3.json', 'w') as lrg_data_updt:
    json.dump(rvrs_data, lrg_data_updt, indent=2)
```

Output

TERMINAL — Python

```
/usr/local/bin/python3 /Users/manmohitsingh/Desktop/ENSF_338/Lab_1/Group_5_Lab_1/ex2.5.py
(base) manmohitsingh@Manmohits-MacBook-Pro Group_5_Lab_1 % /usr/local/bin/python3 /Users/manmohitsingh/Desktop/ENSF_338/Lab_1/Group_5_Lab_1/ex2.5.py
Average time to modify 'size' values: 0.036206 seconds
(base) manmohitsingh@Manmohits-MacBook-Pro Group_5_Lab_1 %
```

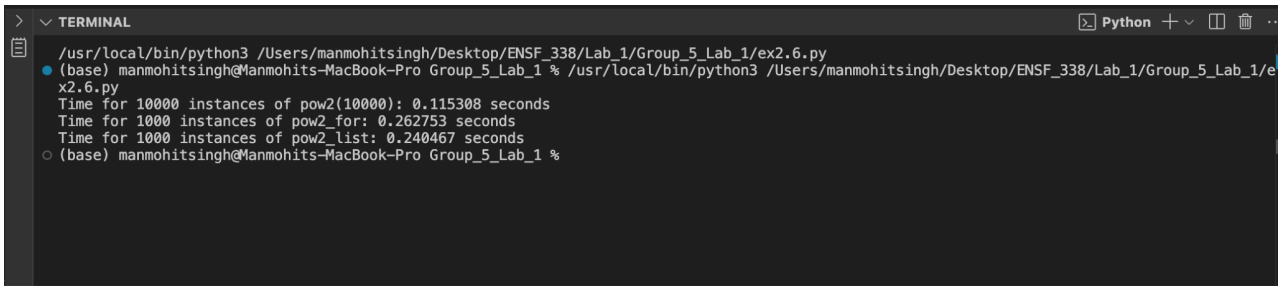
Part 2.6:

Script

ex2.6.py > ...

```python
import timeit

# Function to compute the n-th power of 2
def pow2(n):
    return 2 ** n

# Timing the execution of 10000 instances of pow2(10000)
pow2_10000 = timeit.timeit(lambda: pow2(10000), number=10000)
print(f"Time for 10000 instances of pow2(10000): {pow2_10000:.6f} seconds")

def pow2_for():
    result = []
    for n in range(1001):
        result.append(2 ** n)
    return result

def pow2_list():
    return [2 ** n for n in range(1001)]

time_for = timeit.timeit(lambda: pow2_for(), number=1000)
time_list = timeit.timeit(lambda: pow2_list(), number=1000)

print(f"Time for 1000 instances of pow2_for: {time_for:.6f} seconds")
print(f"Time for 1000 instances of pow2_list: {time_list:.6f} seconds")
```

Output



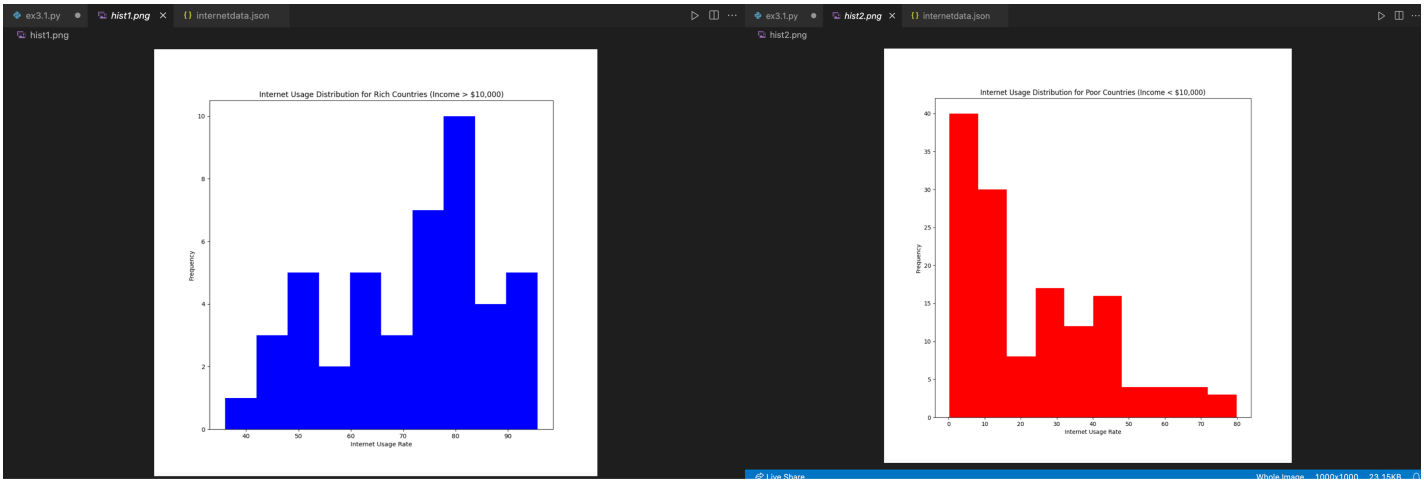
Part 3.1:

Script

ex3.1.py

```python
import json #importing json library
import matplotlib.pyplot as plt #importing matplot library

with open("internetdata.json", "r") as internet: #Opening json file in reading mode
    dtls = json.load(internet) #Loading the json file content in a variable



rich_country = [] #Creating an empty list
poor_country = [] #Creating an empty list

for i in dtls: #Iterating in dtls
    if i["incomeperperson"] != None: #Filtering null incomeperperson values
        if i["incomeperperson"] > 10000: #Filtering if the income is above 10000
            rich_country.append(i) #If income is above 10000 then append the entire dictionary to rich_country list
        else: #Else statement
            poor_country.append(i) #If income is not above 10000 then append the entire dictionary to rich_country list

rich_internet = [j['internetuserate'] for j in rich_country if j['internetuserate']!= None] #Storing the internet usage for rich countries in a variable using list comprehension
poor_internet = [k['internetuserate'] for k in poor_country if k['internetuserate']!= None] #Storing the internet usage for poor countries in a variable using list comprehension


plt.figure(figsize=(10, 10)) #Setting figsize for plot
plt.hist(rich_internet, color="blue") #Using rich_internet for the histogram and giving the colour
plt.title("Internet Usage Distribution for Rich Countries (Income > $10,000)") #Histogram title
plt.xlabel("Internet Usage Rate") #Histogram x-axis label
plt.ylabel("Frequency") #Histogram y-axis label
plt.savefig("hist1.png") #Saving the histogram as a png file

plt.figure(figsize=(10, 10)) #Setting figsize for plot
plt.hist(poor_internet, color="red") #Using poor_internet for the histogram and giving the colour
plt.title("Internet Usage Distribution for Poor Countries (Income < $10,000)") #Histogram title
plt.xlabel("Internet Usage Rate") #Histogram x-axis label
plt.ylabel("Frequency") #Histogram y-axis label
plt.savefig("hist2.png") #Saving the histogram as a png file

```

Output



Part 3.2:

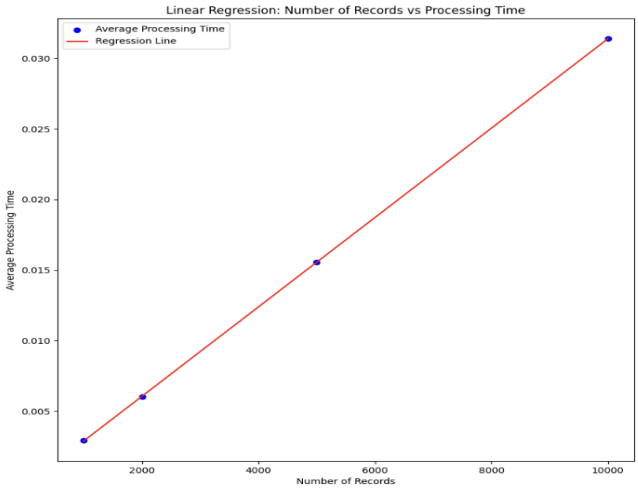
Script

ex3.2.py > ...

```python
import json
import matplotlib.pyplot as plt
import numpy as np
import timeit

def size_42(n):
    if type(n) == dict:
        for key, value in n.items():
            if key == "size":
                n[key] = 42
            else:
                size_42(value)
    elif type(n) == list:
        for i in n:
            size_42(i)

with open('large-file.json', 'r') as lrg_data:
    data = json.load(lrg_data)

records = [1000, 2000, 5000, 10000]
avrg_times = []

for count in records:
    subset = data[:count]
    time_taken = timeit.timeit(lambda: [size_42(i) for i in subset], number=100)
    average_time = time_taken / 100
    avrg_times.append(average_time)
    print(f"Average time for {count} records: {average_time:.6f} seconds")

slope, intercept = np.polyfit(records, avrg_times, 1)

plt.figure(figsize=(10, 10))
plt.scatter(records, avrg_times, color='blue', label='Average Processing Time')

linevalues = [slope * x + intercept for x in records]
plt.plot(records, linevalues, 'red', label=f'Regression Line')

plt.xlabel('Number of Records')
plt.ylabel('Average Processing Time')
plt.title('Linear Regression: Number of Records vs Processing Time')
plt.legend()

plt.savefig('output.3.2.png')
```

Output

```
TERMINAL                                                    Python
/usr/local/bin/python3 /Users/manmohitsingh/Desktop/ENSF_338/Lab_1/Group_5_Lab_1/ex3.2.py
(base) manmohitsingh@Manmohits-MacBook-Pro Group_5_Lab_1 % /usr/local/bin/python3 /Users/manmohitsingh/Desktop/ENSF_338/Lab_1/Group_5_Lab_1/ex3.2.py
Average time for 1000 records: 0.002918 seconds
Average time for 2000 records: 0.006038 seconds
Average time for 5000 records: 0.015535 seconds
Average time for 10000 records: 0.031394 seconds
(base) manmohitsingh@Manmohits-MacBook-Pro Group_5_Lab_1 %
```



Part 3.3:

Script

ex3.3.py > ...

```python
import json
import matplotlib.pyplot as plt
import timeit

def size_42(n):
    if type(n) == dict:
        for key, value in n.items():
            if key == "size":
                n[key] = 42
            else:
                size_42(value)
    elif type(n) == list:
        for i in n:
            size_42(i)

with open('large-file.json', 'r') as lrg_data:
    data = json.load(lrg_data)

data_1000 = data[:1000]

times = []
for i in range(1000):
    time_taken = timeit.repeat(lambda: [size_42(j) for j in data_1000], repeat=1000, number=1)
    times.append(time_taken)

plt.figure(figsize=(10, 10))
plt.hist(times, color='blue')
plt.xlabel('Processing Time (seconds)')
plt.ylabel('Frequency')
plt.title('Distribution of Processing Times for 1000 Records (1000 repetitions)')

plt.savefig('output.3.3.png')
```

Output