IE - 416 Robot Programming

Assignment 1: Basics of **Python**

Team Name: GENWIN

Team Members:

- Tirth Modi (202201513)
- Natansh Shah (202201445)

Prof. : Tapas Kumar Maiti

January 29, 2025

Python Operations and Data Visualizations

Q1: Function to Determine the Number of Days in a Given Year

Ans:

```python
def days_in_year(year):
    return 366 if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0) else 365

print(days_in_year(1990))
print(days_in_year(2044))
```

✓ 0.0s Python

```
365
366
```

This function determines if a given year is a **leap year** or not.

It follows the standard leap year rule:

- A year is a leap year if it is **divisible by 4** but **not divisible by 100**.
- However, if it is divisible by **400**, it is still a leap year.

Example:

- **Input:** `year = 2000` → **Output:** `366` (Leap Year)
- **Input:** `year = 1999` → **Output:** `365` (Non-Leap Year)

Q2: Count Character Frequency in a String

Ans:

```python
def char_frequency(s):
    freq = {}
    for char in s:
        freq[char] = freq.get(char, 0) + 1
    return freq

print(char_frequency('adcbbdaacd'))
```

✓ 0.0s Python

```
{'a': 3, 'd': 3, 'c': 2, 'b': 2}
```

This function takes a string as input and counts the frequency of each character.

It uses a **dictionary (Counter from the collections module)** to store character occurrences efficiently.

Example:

- **Input:** `"hello"`
- **Output:** `{'h': 1, 'e': 1, 'l': 2, 'o': 1}`

Q3: Remove Duplicates from a List while Keeping the First Occurrence

Ans:

```python
def remove_duplicates(lst):
    seen = set()
    return [x for x in lst if not (x in seen or seen.add(x))]

print(remove_duplicates([1, 2, 3, 4, 2, 3, 5, 6, 1, 4]))
```

✓ 0.0s Python

```
[1, 2, 3, 4, 5, 6]
```

This function removes duplicate elements while **preserving the order** of appearance.
A **set** is used to track elements that have already been added.
Example:

- **Input:** `[1, 2, 3, 2, 1, 4, 5, 4]`
- **Output:** `[1, 2, 3, 4, 5]`

Q4: Sorting a Stack Using Another Stack

Ans:

```python
def sort_stack(stack):
    sorted_stack = []
    while stack:
        temp = stack.pop()
        while sorted_stack and sorted_stack[-1] > temp:
            stack.append(sorted_stack.pop())
        sorted_stack.append(temp)
    return sorted_stack[::-1]

print(sort_stack([9, 5, 1, 3]))
```

✓ 0.0s Python

```
[9, 5, 3, 1]
```

This function sorts a stack using **only another stack**, without additional data structures. The method works by **popping elements from the original stack and inserting them in sorted order into a new stack**.

Example:

- **Input Stack:** `[3, 1, 4, 2]`
- **Sorted Stack Output:** `[1, 2, 3, 4]`

Q5: Pascal's Triangle

Ans:

```python
def pascalTriangle(numOfRows):
    result = [[1]]
    for i in range(1, numOfRows):
        row = [1] + [result[i-1][j] + result[i-1][j+1] for j in range(len(result[i-1])-1
        result.append(row)
    for row in result:
        print(row)

pascalTriangle(4)
```

✓ 0.0s Python

```
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
```

- This function generates Pascal's Triangle using **binomial coefficients**.
- Each row is generated based on the sum of two elements from the previous row.

Example for n=5:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

Q6: 6×6 Matrix Operations

Ans:

```python
import numpy as np

matrix = np.random.rand(6, 6)
matrix_binary = np.where(matrix > 0.5, 1, 0)
submatrix = matrix[2:5, 2:5]
mean_value = np.mean(submatrix)

print("Binary Matrix:\n", matrix_binary)
print("3x3 Submatrix:\n", submatrix)
print("Mean of submatrix:", mean_value)
```

✓ 0.0s Python

```
Binary Matrix:
 [[1 1 1 1 1 0]
 [1 0 1 1 0 0]
 [1 0 1 0 1 0]
 [1 0 0 0 0 1]
 [0 1 1 1 1 1]
 [0 1 0 1 0 0]]
3x3 Submatrix:
 [[0.92793312 0.09468982 0.91321507]
 [0.1916823  0.14938037 0.40075137]
 [0.81162256 0.77430576 0.90840313]]
Mean of submatrix: 0.5746648345477117
```

A **random 6×6 matrix** is generated using NumPy.

Operations performed:

- **Thresholding:** Replacing values **greater than 0.5 with 1** and others with 0.
- **Submatrix Extraction:** Extracting a **3×3 submatrix** starting at index `(2,2)`.

- **Mean Calculation:** Computing the average of the submatrix elements.

Q7: Array Reshaping

Ans:

```python
array1d = np.arange(16)
matrix4x4 = array1d.reshape(4, 4)

array3d = np.random.rand(3, 3, 3)
flattened = array3d.flatten()

reshaped_matrix = np.reshape(array1d, (2, 8))

print("4x4 Matrix:\n", matrix4x4)
print("Flattened 3x3x3 array:\n", flattened)
print("Reshaped matrix:\n", reshaped_matrix)
```

✓ 0.0s Python

```
4x4 Matrix:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
Flattened 3x3x3 array:
 [0.30034695 0.54620273 0.40770318 0.41239851 0.00180273 0.01164083
 0.48773035 0.89127574 0.42742171 0.672746   0.6030842  0.26566399
 0.00765487 0.4289555  0.85359924 0.46194949 0.3490498  0.46477486
 0.31638614 0.16963968 0.80992154 0.36430746 0.30150397 0.19100454
 0.46313264 0.47690762 0.62602457]
Reshaped matrix:
 [[ 0  1  2  3  4  5  6  7]
 [ 8  9 10 11 12 13 14 15]]
```

Created a **1D NumPy array of 16 elements** and reshaped it into a **4×4 matrix**.
Performed the following transformations:

- **Flattening a 3×3×3 array into a 1D array using flatten function.**
- **Reshaping a 4×4 matrix into another shape without modifying data using reshape function.**

Q8: Recursive Function for Fibonacci Sum

Ans:

```python
def Fibonacci_sum(n, a=0, b=1, total=0):
    if n == 1:
        return total
    return Fibonacci_sum(n - 1, b, a + b, total + a)

print(Fibonacci_sum(1))
print(Fibonacci_sum(4))
```

✓ 0.0s Python

```
0
2
```

This function calculates the sum of the first `n` Fibonacci numbers recursively.
Example:

- **Input:** `n = 5`
- **Fibonacci Sequence:** `0, 1, 1, 2, 3`
- **Sum:** `0 + 1 + 1 + 2 + 3 = 7`

Q9: Dictionary Lookup with Exception Handling

Ans:

```python
def get_value_from_dict(d, key):
    if key not in d:
        raise KeyError(f"Key '{key}' not found in dictionary.")
    return d[key]

try:
    sample_dict = {'a': 10, 'b': 20, 'c': 30}
    key = input("Enter key: ")
    print(get_value_from_dict(sample_dict, key))
except KeyError as e:
    print("Error:", e)
```

✓ 1.7s Python

```
10
```

- This function retrieves a value from a dictionary using a given key.
- If the key does not exist, it raises a **KeyError** with a custom error message.
- **Exception Handling** is implemented using `try-except`.

Q10: Data Visualization with Multiple Plots

Ans:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go

# Load dataset
df = pd.read_csv('Loan_train.csv')

# Basic Info and Statistics
print(df.info())
print(df.describe())

# Bar Chart
plt.figure(figsize=(12, 6))
sns.countplot(x='loan_intent', data=df, palette='coolwarm')
plt.title("Loan Intent Distribution")
plt.xlabel("Loan Intent")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()

# Pie Chart
plt.figure(figsize=(6, 6))
df['person_home_ownership'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, cmap='coolwarm')
plt.title("Home Ownership Distribution")
plt.ylabel('')
plt.show()

# Line Graph
plt.figure(figsize=(12, 6))
sns.lineplot(x=df.index, y=df['loan_int_rate'])
plt.title("Interest Rate Variation")
plt.xlabel("Index")
plt.ylabel("Loan Interest Rate (%)")
plt.show()

# Scatter Plot
plt.figure(figsize=(12, 6))
sns.scatterplot(x='person_income', y='loan_amnt', hue='loan_status', data=df, palette='viridis')
plt.title("Loan Amount vs Income")
plt.xlabel("Person Income")
plt.ylabel("Loan Amount")
plt.show()

# Histogram
plt.figure(figsize=(12, 6))
sns.histplot(df['person_age'], bins=30, kde=True)
plt.title("Age Distribution")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
```

```python
# Box Plot
plt.figure(figsize=(12, 6))
sns.boxplot(x='loan_grade', y='loan_amnt', data=df, palette='Set2')
plt.title("Loan Amount Distribution by Grade")
plt.show()

# Pair Plot
sns.pairplot(df[['person_age', 'person_income', 'loan_amnt', 'loan_int_rate']], diag_kind='kde')
plt.show()

# KDE Plot
plt.figure(figsize=(12, 6))
sns.kdeplot(df['loan_int_rate'], shade=True)
plt.title("KDE Plot of Interest Rates")
plt.show()

# Violin Plot
plt.figure(figsize=(12, 6))
sns.violinplot(x='loan_grade', y='loan_int_rate', data=df, palette='muted')
plt.title("Interest Rate Distribution by Loan Grade")
plt.show()

# Strip Plot
plt.figure(figsize=(12, 6))
sns.stripplot(x='loan_grade', y='loan_amnt', data=df, jitter=True, palette='pastel')
plt.title("Loan Amount Distribution by Grade")
plt.show()

# Treemap
fig = px.treemap(df, path=['loan_intent', 'loan_grade'], values='loan_amnt', title="Loan Distribution Treemap")
fig.show()

# Sunburst Chart
fig = px.sunburst(df, path=['person_home_ownership', 'loan_intent'], values='loan_amnt', title="Sunburst of Loan Data")
fig.show()

# Density Plot
plt.figure(figsize=(12, 6))
sns.kdeplot(df['person_income'], fill=True)
plt.title("Income Density Distribution")
plt.show()

# Radial Bar Chart
fig = px.bar_polar(df, r='loan_amnt', theta='loan_grade', color='loan_grade', title="Radial Bar Chart")
fig.show()
```

```python
# Step Plot
plt.figure(figsize=(12, 6))
plt.step(df.index, df['loan_int_rate'], where='mid', label='Interest Rate')
plt.title("Step Plot of Interest Rate")
plt.xlabel("Index")
plt.ylabel("Loan Interest Rate")
plt.legend()
plt.show()

# Bubble Chart
fig = px.scatter(df, x='person_income', y='loan_amnt', size='loan_int_rate', color='loan_grade', title="Bubble Chart of Loans")
fig.show()

# Parallel Coordinates Plot
fig = px.parallel_coordinates(df, dimensions=['person_age', 'person_income', 'loan_amnt', 'loan_int_rate'], color='loan_status', title="Parallel Coordinates Plot")
fig.show()

# Interactive Plotly Visualizations
fig = px.scatter(df, x='person_income', y='loan_amnt', color='loan_status', title="Interactive Scatter Plot")
fig.show()

fig = px.bar(df, x='loan_intent', y='loan_amnt', title="Interactive Loan Amount by Intent")
fig.show()

fig = px.box(df, x='loan_grade', y='loan_int_rate', title="Interactive Box Plot of Interest Rates by Grade")
fig.show()

fig = px.violin(df, x='loan_grade', y='loan_amnt', title="Interactive Violin Plot of Loan Amounts by Grade")
fig.show()

fig = go.Figure(go.Pie(labels=df['person_home_ownership'].unique(), values=df['person_home_ownership'].value_counts(), hole=0.3))
fig.update_layout(title_text="Interactive Home Ownership Distribution")
fig.show()
```

- **Dataset Loaded:** `Loan_train.csv` using pandas.
- **Categorical Data Encoding:** Converted categorical variables to numeric values for analysis.
- **Plots Implemented:**
  - **Bar Chart:** Count distribution of loan intents.
  - **Pie Chart:** Home ownership type distribution.
  - **Line Graph:** Trend of interest rates.
  - **Scatter Plot:** Relationship between income and loan amount.
  - **Histogram:** Age distribution.
  - **Box Plot:** Loan amounts categorized by grade.
  - **Violin Plot:** Spread of interest rates across grades.
  - **KDE Plot:** Density of interest rates.
  - **Strip Plot:** Loan amounts for each grade.
  - **Treemap:** Loan amount distribution by intent and grade.
  - **Sunburst Chart:** Breakdown of home ownership and loan intent.
  - **Radial Bar Chart:** Circular representation of loan amounts.
  - **Step Plot:** Interest rate variations over time.
  - **Bubble Chart:** Loan amounts visualized with bubble sizes.
  - **Parallel Coordinates Plot:** Comparison of multiple loan-related attributes.

Each visualization includes **labels, titles, and legends** for clear interpretation.

Important Links :

Ipynb Notebook: Lab1.ipynb

Github Link: https://github.com/202201513/IE416_Robot_Programming_Labs/tree/main/Lab1