ASSIGNMENT 2:

CREATING A SIMPLE BOOK MANAGEMENT SYSTEM USING DJANGO

Create the Book Model

```
from django.db import models

class Book(models.Model):

    title = models.CharField(max_length=200)

    author = models.CharField(max_length=100)

    publication_date = models.DateField()


    def __str__(self):

        return self.title
```

Create Views for Adding and Listing Books

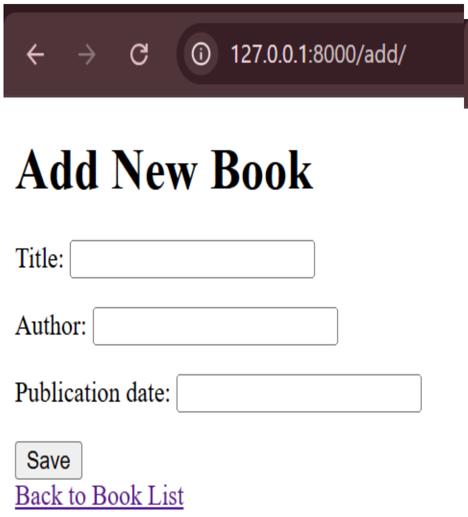
```
from django.shortcuts import render, redirect

from .models import Book

from .forms import BookForm

# View to list all books

def book_list(request):

    books = Book.objects.all()

    return render(request, 'library/book_list.html', {'books': books})

# View to add a new book

def add_book(request):

    if request.method == 'POST':

        form = BookForm(request.POST)

        if form.is_valid():

            form.save()

            return redirect('book_list')

    else:

        form = BookForm()
    return render(request, 'library/add_book.html', {'form': form})
```

Create Forms for Book Input

```
from django import forms

from .models import Book

class BookForm(forms.ModelForm):

    class Meta:

        model = Book

        fields = ['title', 'author', 'publication_date']
```

Create Templates

```
<!DOCTYPE html>

<html>

<head>

    <title>Book List</title>

</head>

<body>

    <h1>Book List</h1>

    <table border="1">

        <tr>

            <th>Title</th>

            <th>Author</th>

            <th>Publication Date</th>

        </tr>

        {% for book in books %}

        <tr>

            <td>{{ book.title }}</td>

            <td>{{ book.author }}</td>

            <td>{{ book.publication_date }}</td>

        </tr>

        {% endfor %}

    </table>
    <a href="{% url 'add_book' %}">Add a New Book</a>
</body>
</html>

<!DOCTYPE html>
<html>
<head>
    <title>Add Book</title>
</head>
<body>
    <h1>Add New Book</h1>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Save</button>
    </form>
    <a href="{% url 'book_list' %}">Back to Book List</a>
</body>
</html>
```



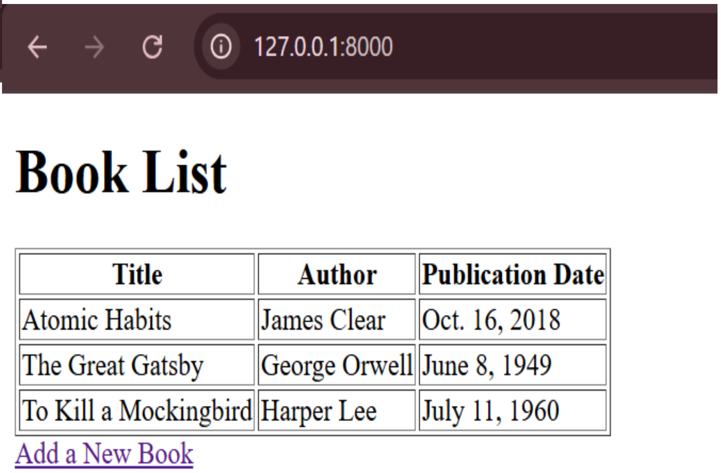
Add New Book

Title:

Author:

Publication date:

Back to Book List



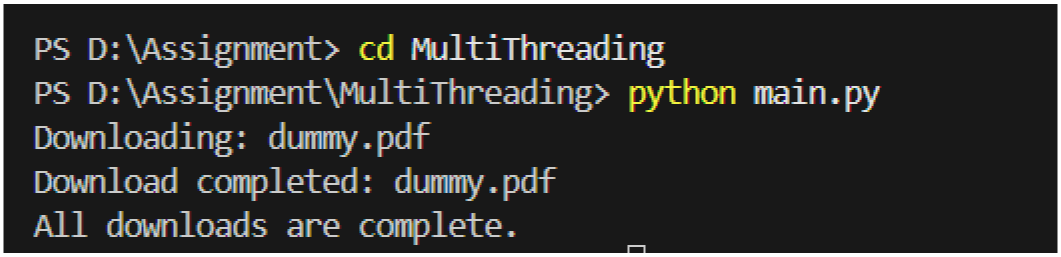
Book List

Title	Author	Publication Date
Atomic Habits	James Clear	Oct. 16, 2018
The Great Gatsby	George Orwell	June 8, 1949
To Kill a Mockingbird	Harper Lee	July 11, 1960

Add a New Book

Multi Threading File Download

```
import threading

import requests

from urllib.parse import urlparse

import os

# Function to download a file from a URL

def download_file(url):

    try:

        # Send a GET request to the URL

        response = requests.get(url, stream=True)


        # Extract the filename from the URL

        filename = os.path.basename(urlparse(url).path)


        # Check if the request was successful

        if response.status_code == 200:

            print(f"Downloading: {filename}")

            with open(filename, 'wb') as file:

                for chunk in response.iter_content(chunk_size=1024):

                    if chunk:

                        file.write(chunk)

            print(f"Download completed: {filename}")

        else:

            print(f"Failed to download {url}, status code: {response.status_code}")


    except Exception as e:

        print(f"Error downloading {url}: {str(e)}")


# List of URLs to download files from

urls = [

    'https://www.w3.org/WAI/ER/tests/xhtml/testfiles/resources/pdf/dummy.pdf',
]

# List to hold the thread objects
threads = []
# Create and start a new thread for each URL
for url in urls:
    thread = threading.Thread(target=download_file, args=(url,))
    threads.append(thread)
    thread.start()
# Wait for all threads to complete
for thread in threads:
    thread.join()
print("All downloads are complete.")
```

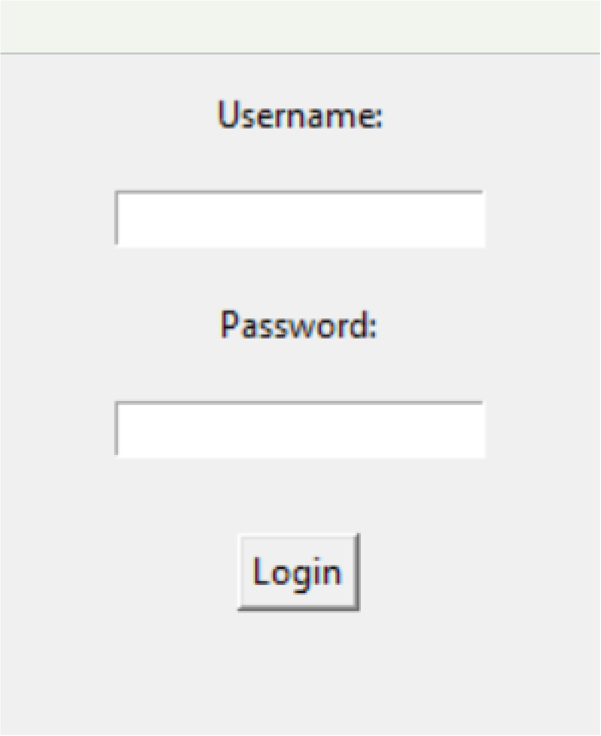


```
PS D:\Assignment> cd MultiThreading
PS D:\Assignment\MultiThreading> python main.py
Downloading: dummy.pdf
Download completed: dummy.pdf
All downloads are complete.
```

LOGIN WINDOW GUI USING TKINTER

```
import tkinter as tk
from tkinter import messagebox

# Function to check login credentials
def check_login():
    username = entry_username.get()
    password = entry_password.get()

    # Dummy credentials (you can modify or connect to a real database for validation)
    correct_username = "admin"
    correct_password = "password123"

    if username == correct_username and password == correct_password:
        messagebox.showinfo("Login Success", "Welcome, you have logged in successfully!")
    else:
        messagebox.showerror("Login Failed", "Incorrect username or password.")

# Create the main window
window = tk.Tk()
window.title("Login Window")
window.geometry("300x200")  # Width x Height

# Create a label for username
label_username = tk.Label(window, text="Username:")
label_username.pack(pady=10)

# Create a text entry field for username
entry_username = tk.Entry(window)
entry_username.pack(pady=5)
```

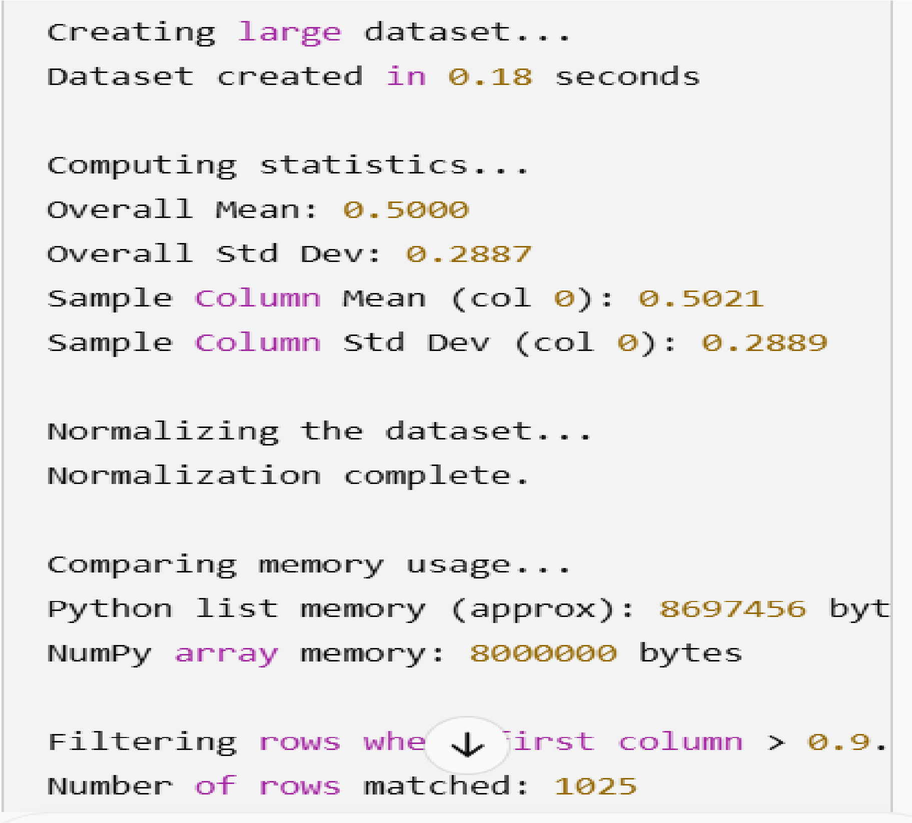
```
# Create a label for password
label_password = tk.Label(window, text="Password:")
label_password.pack(pady=10)

# Create a text entry field for password (with masking)
entry_password = tk.Entry(window, show="*")
entry_password.pack(pady=5)

# Create a login button
login_button = tk.Button(window, text="Login", command=check_login)
login_button.pack(pady=20)

# Start the GUI event loop
window.mainloop()
```



"Handling and Analyzing Large Datasets Efficiently with NumPy"

```
import numpy as np
import time
import sys

# Step 1: Create large dataset
print("Creating large dataset...")
start_time = time.time()
data = np.random.rand(10000, 1000)  # Random values between 0 and 1
print("Dataset created in {:.2f} seconds\n".format(time.time() - start_time))

# Step 2: Compute statistics
print("Computing statistics...")
mean_per_column = np.mean(data, axis=0)
std_per_column = np.std(data, axis=0)
overall_mean = np.mean(data)
overall_std = np.std(data)

# Displaying the results
print("Overall Mean: {:.4f}".format(overall_mean))
print("Overall Std Dev: {:.4f}".format(overall_std))
print("Sample Column Mean (col 0): {:.4f}".format(mean_per_column[0]))
print("Sample Column Std Dev (col 0): {:.4f}".format(std_per_column[0]))
print()

# Step 3: Normalize the dataset
print("Normalizing the dataset...")
normalized_data = (data - overall_mean) / overall_std
print("Normalization complete.\n")
```

```
# Step 4: Compare memory usage between Python list and NumPy array
print("Comparing memory usage...")
py_list = [float(i) for i in range(1000000)]
np_array = np.array(py_list)

# Displaying memory usage
print("Python list memory (approx):", sys.getsizeof(py_list), "bytes")
print("NumPy array memory:", np_array.nbytes, "bytes\n")

# Step 5: Filter rows where the first column > 0.9
print("Filtering rows where first column > 0.9...")
filtered_data = data[data[:, 0] > 0.9]
print("Number of rows matched:", filtered_data.shape[0])
```

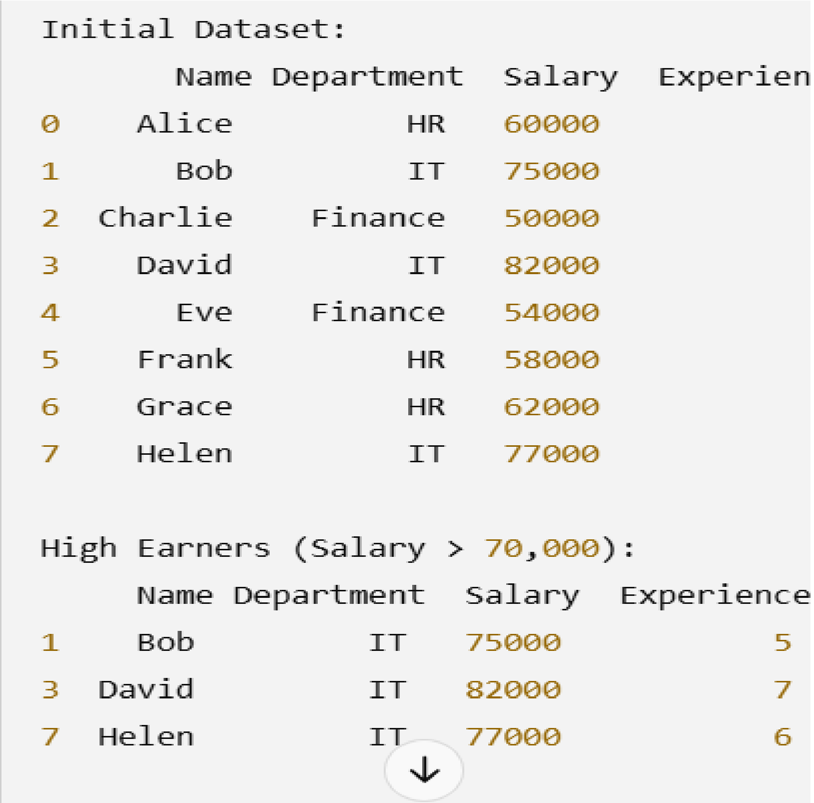


```
Creating large dataset...
Dataset created in 0.18 seconds

Computing statistics...
Overall Mean: 0.5000
Overall Std Dev: 0.2887
Sample Column Mean (col 0): 0.5021
Sample Column Std Dev (col 0): 0.2889

Normalizing the dataset...
Normalization complete.

Comparing memory usage...
Python list memory (approx): 8697456 byt
NumPy array memory: 8000000 bytes

Filtering rows whe  irst column > 0.9.
Number of rows matched: 1025
```

"Efficient Data Analysis and Manipulation with Pandas"

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Create the initial dataset
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank', 'Grace', 'Helen'],
    'Department': ['HR', 'IT', 'Finance', 'IT', 'Finance', 'HR', 'HR', 'IT'],
    'Salary': [60000, 75000, 50000, 82000, 54000, 58000, 62000, 77000],
    'Experience': [2, 5, 1, 7, 3, 4, 2, 6]
}
df = pd.DataFrame(data)

# Display initial dataset
print("Initial Dataset:\n", df, "\n")

# Step 2: Filter high earners (Salary > 70,000)
high_earners = df[df['Salary'] > 70000]
print("High Earners (Salary > 70,000):\n", high_earners, "\n")

# Step 3: Calculate average salary by department
avg_salary_by_dept = df.groupby('Department')['Salary'].mean()
print("Average Salary by Department:\n", avg_salary_by_dept, "\n")

# Step 4: Rename 'Experience' column and calculate 'SalaryPerYear'
df.rename(columns={'Experience': 'YearsExperience'}, inplace=True)
df['SalaryPerYear'] = df['Salary'] / df['YearsExperience']
print("Data with 'SalaryPerYear':\n", df, "\n")

# Step 5: Create bonus dataframe and merge with the original dataframe
```

```
bonus_df = pd.DataFrame({
    'Department': ['HR', 'IT', 'Finance'],
    'BonusPercent': [10, 15, 12]
})
merged_df = pd.merge(df, bonus_df, on='Department')
print("Merged Dataset with Bonuses:\n", merged_df, "\n")

# Step 6: Create a pivot table for salary by department and experience
pivot = pd.pivot_table(df, values='Salary', index='Department', columns='YearsExperience', aggfunc='mean')
print("Pivot Table (Salary by Dept & Experience):\n", pivot, "\n")

# Step 7: Plot Average Salary by Department
plt.figure(figsize=(8, 5))
df.groupby('Department')['Salary'].mean().plot(kind='bar', color='skyblue')
plt.title("Average Salary by Department")
plt.xlabel("Department")
plt.ylabel("Average Salary")
plt.tight_layout()
plt.show()
```



```
Initial Dataset:
       Name Department  Salary  Experien
0    Alice          HR  60000
1      Bob          IT  75000
2  Charlie     Finance  50000
3    David          IT  82000
4      Eve     Finance  54000
5    Frank          HR  58000
6    Grace          HR  62000
7    Helen          IT  77000

High Earners (Salary > 70,000):
     Name Department  Salary  Experience
1    Bob          IT  75000            5
3  David          IT  82000            7
7  Helen          IT  77000            6
```