Home Task:

- **CODE:**

```python
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity

# Sample dataset: ratings by users for books
ratings_dict = {
    "user": ['U1', 'U1', 'U2', 'U2', 'U3', 'U3', 'U4', 'U4', 'U5'],
    "book": ['B1', 'B2', 'B3', 'B4', 'B1', 'B3', 'B2', 'B4', 'B1'],
    "rating": [5, 3, 4, 4, 4, 5, 2, 3, 5]
}

# Convert the dictionary to a DataFrame
df = pd.DataFrame(ratings_dict)

# Create a user-item matrix with ratings
user_item_matrix = df.pivot(index='user', columns='book', values='rating').fillna(0)

# Transpose the user-item matrix to work with items (books) as rows
item_user_matrix = user_item_matrix.T

# Compute cosine similarity between items (books)
cosine_sim = cosine_similarity(item_user_matrix)

# Convert to DataFrame for easier handling
cosine_sim_df = pd.DataFrame(cosine_sim, index=item_user_matrix.index, columns=item_user_matrix.index)

# Function to recommend books for a given user
def recommend_books_for_user(user_id, n_recommendations=5):
    # Get the ratings for the selected user
    user_ratings = user_item_matrix.loc[user_id]

    # Get the list of books rated by the user
    rated_books = user_ratings[user_ratings > 0].index.tolist()

    # Dictionary to store predicted ratings for unrated books
    predicted_ratings = {}

    # Iterate over unrated books to predict ratings
    for book in user_item_matrix.columns:
        if user_ratings[book] == 0:  # Book is unrated by the user
            # Get the books most similar to the current unrated book from the cosine similarity matrix
            similar_books = cosine_sim_df[book]

            # Predict rating for this unrated book
            weighted_sum = 0
            similarity_sum = 0
            for rated_book in rated_books:
                if rated_book in similar_books.index:  # Ensure that we have a valid similarity score
                    weighted_sum += user_ratings[rated_book] * similar_books[rated_book]
                    similarity_sum += similar_books[rated_book]

            # Predicted rating (normalized by similarity sum)
            if similarity_sum != 0:
                predicted_ratings[book] = weighted_sum / similarity_sum
            else:
                predicted_ratings[book] = 0  # No similar books, give a rating of 0

    # Sort books by predicted rating and return top N recommendations
    recommended_books = sorted(predicted_ratings.items(), key=lambda x: x[1], reverse=True)[:n_recommendations]

    # Return book IDs and their predicted ratings
    return recommended_books

# Example usage
user_id = 'U1'
recommended_books = recommend_books_for_user(user_id)
print(f"Recommended books for user {user_id}:")
for book, rating in recommended_books:
    print(f"Book ID: {book}, Predicted Rating: {rating}")
```

- **OUTPUT:**

```
Recommended books for user U1:
Book ID: B3, Predicted Rating: 5.0
Book ID: B4, Predicted Rating: 3.0
```