Practical Task 1: Basic Array Creation and Manipulation with NumPy

Objective:

Introduce basic operations in NumPy by creating and manipulating simple arrays.

Requirements:

1. **Array Creation:**
    - Create a one-dimensional NumPy array with values ranging from 1 to 10.
    - Create a two-dimensional NumPy array (matrix) with shape (3, 3) containing values from 1 to 9.
2. **Basic Operations:**
    - Indexing and Slicing:
        1. Access and print the third element of the one-dimensional array.
        2. Slice and print the first two rows and columns of the two-dimensional array.
    - Basic Arithmetic:
        1. Add 5 to each element of the one-dimensional array and print the result.
        2. Multiply each element of the two-dimensional array by 2 and print the result.
3. **Output Function:**
    - Implement a separate function named **print_array** that takes an array and an optional message as input, and prints them to the console. This function will be used to display the state of the array, maintaining separation from the manipulation logic.
4. **Manipulation Workflow:**
    - Create the initial arrays.
    - Perform each basic operation sequentially. Using the **print_array** function to output the arrays and output results to the console, when needed.
5. **Execution and Verification:** Test the script to ensure that code executes as expected and that the console outputs correctly display the changes to the array.

Deliverables: A Python script (.py file) containing all the functions along with the code to create the initial arrays and execute all manipulations.

Practical Task 2: Analyzing and Visualizing E-Commerce Transactions with NumPy

Objective:

Develop a set of Python functions using NumPy to manipulate and analyze a simulated e-commerce dataset.

Requirements:

1. **Array Creation:**
    - Generate a multi-dimensional NumPy array with predefined values to simulate e-commerce transactions. The array should include **transaction_id**, **user_id**, **product_id**, **quantity**, **price**, and **timestamp**.
2. **Data Analysis Functions:**
    - **Total Revenue Function:** Create a function to calculate the total revenue generated by multiplying **quantity** and **price**, and summing the result.
    - **Unique Users Function:** Develop a function to determine the number of unique users who made transactions.
    - **Most Purchased Product Function:** Implement a function to identify the most purchased product based on the quantity sold.
    - **Type Casting and Checking Functions:**
        - Create a function to convert the price array from float to integer.
        - Develop a function to check the data types of each column in the array.
3. **Array Manipulation Functions:**
    - **Product Quantity Array Function:** Create a function that returns a new array with only the **product_id** and **quantity** columns.
    - **User Transaction Count Function:** Implement a function to generate an array of transaction counts per user.
    - **Masked Array Function:** Create a function to generate a masked array that hides transactions where the quantity is zero.
4. **Arithmetic and Comparison Functions:**
    - **Price Increase Function:** Develop a function to increase all prices by a certain percentage (e.g., 5% increase).
    - **Filter Transactions Function:** Implement a function to filter transactions to only include those with a quantity greater than 1.
    - **Revenue Comparison Function:** Create a function to compare the revenue from two different time periods.
5. **Indexing and Slicing Functions:**

- **User Transactions Function:** Create a function to extract all transactions for a specific user.
- **Date Range Slicing Function:** Develop a function to slice the dataset to include only transactions within a specific date range.
- **Top Products Function:** Implement a function using advanced indexing to retrieve transactions of the top 5 products by revenue.

6. **Output Function:**
   - Implement a separate function named **print_array** that takes an array and an optional message as input, and prints them to the console. This function will be used to display the state of the array, maintaining separation from the manipulation logic.
7. **Manipulation Workflow:**
   - Call each analysis and manipulation function sequentially on the array. Use the **print_array** function to output arrays to the console with appropriate messages. Also output results to the console, when needed.
8. **Execution and Verification:**
   - Test the script to ensure that all functions execute as expected and that the console outputs correctly display the changes to the array.
   - Use assertions to verify that the dimensions and integrity of the array are maintained or appropriately altered after each manipulation.

Deliverables:

- A Python script (.py file) containing all the functions, along with the code to create the initial array and execute all manipulations.

Practical Task 3: Array Manipulation with Separate Output Function in NumPy

Objective: Develop a set of Python functions using NumPy to manipulate arrays through operations such as transposing, reshaping, splitting, and combining.

Requirements:

1. **Array Creation:**
   - Generate a multi-dimensional **NumPy** array with random values. The array should have a complex structure (e.g., a 6x6 matrix of integers) to clearly demonstrate changes through each manipulation.
2. **Array Manipulation Functions:**
   - **Transpose Function:** Create a function to transpose the array and return the result.

- **Reshape Function:** Develop a function to reshape the array into a new configuration (e.g., from a 6x6 matrix to a 3x12 matrix) and return the reshaped array.
- **Split Function:** Implement a function that splits the array into multiple sub-arrays along a specified axis and returns the sub-arrays.
- **Combine Function:** Construct a function that combines several arrays into one and returns the combined array.

3. **Output Function:**
    - Implement a separate function named **print_array** that takes an array and an optional message as input, and prints them to the console. This function will be used to display the state of the array before and after each manipulation, maintaining separation from the manipulation logic.
4. **Manipulation Workflow:**
    - Call each manipulation function sequentially on the initial array, passing the output of one function as input to the next where necessary. After each manipulation, use the **print_array** function to output the results to the console with appropriate messages.
5. **Execution and Verification:**
    - Test the script to ensure that all functions execute as expected and that the console outputs correctly display the changes to the array.
    - Use assertions to verify that the dimensions and integrity of the array are maintained or appropriately altered after each manipulation.

Deliverables:

- A Python script (.py file) containing all the functions, along with the code to create the initial array and execute all manipulations.

Practical Task 4: Comprehensive Data Handling and Analysis with NumPy

Objective: Develop a set of Python functions using NumPy to handle reading/writing data and performing aggregate analyses on arrays.

Requirements:

1. **Array Creation:**
    - Generate a multi-dimensional NumPy array with random values. The array should have a complex structure (e.g., a 10x10 matrix of integers) to clearly demonstrate changes through each manipulation.

2. **Data I/O Functions:**
    - **Save Function:** Create a function to save the array to a text file, a CSV file, and a binary format (**.npy** or **.npz**).
    - **Load Function:** Develop a function to load the array from each of the saved formats back into NumPy arrays.
3. **Aggregate Functions:**
    - **Summation Function:** Create a function to compute the summation of all elements.
    - **Mean Function:** Develop a function to calculate the mean of the array.
    - **Median Function:** Implement a function to find the median of the array.
    - **Standard Deviation Function:** Construct a function to calculate the standard deviation of the array.
    - **Axis-Based Aggregate Functions:** Create functions to apply these aggregate operations along different axes (row-wise and column-wise).
4. **Output Function:**
    - Implement a separate function named **print_array** that takes any array and an optional message as input, and prints them to the console. This function will be used to display the state of the array before and after each manipulation, maintaining separation from the manipulation logic.
5. **Manipulation Workflow:**
    - **Array Creation and Saving:** Create the initial array and save it in different formats using the save function.
    - **Loading and Verification:** Load the arrays back and verify their integrity using the load function and the **print_array** function to display the original and loaded arrays.
    - **Aggregate Computation and Reporting:** Compute aggregate functions on the original array and output the results of aggregate computations with appropriate messages.
6. **Execution and Verification:**
    - Test the script to ensure that all functions execute as expected and that the console outputs correctly display the changes to the array.
    - Use assertions to verify that the dimensions and integrity of the array are maintained or appropriately altered after each manipulation. Also verify the results of aggregate functions.

Deliverables:

- A Python script (.py file) containing all the functions, along with the code to create the initial array, save/load it in different formats, perform aggregate calculations, and execute all manipulations.