Q1. What are the benefits of the built-in array package, if any?

ANSWER.

The built-in `array` module in Python provides a way to create arrays, which are data structures that store homogeneous elements of a single data type in contiguous memory locations.

Q2. What are some of the array package's limitations?

ANSWER.

Q3. Describe the main differences between the array and numpy packages.

While the built-in `array` module in Python offers benefits such as memory efficiency and performance, it also has some limitations compared to other data structures like lists:

1. Fixed Size: Arrays in Python have a fixed size, meaning that once created, the size cannot be changed dynamically. Unlike lists, arrays cannot grow or shrink in size by appending or removing elements. This limitation can be restrictive in situations where dynamic resizing is required.

2. Homogeneous Data Type: Arrays store elements of a single data type, which is specified when creating the array. While this ensures data integrity and efficient memory usage, it restricts arrays to homogeneous data types. Lists, on the other hand, can store elements of different data types within the same list.

3. Lack of Built-in Operations: Arrays in Python lack many built-in operations and methods that are available for lists. For example, arrays do not support operations like `append()`, `extend()`, `insert()`, or `remove()`, which are commonly used with lists for dynamic element manipulation.

4. Limited Data Types: Although arrays support various numeric data types such as integers, floats, and doubles, they have limited support for other data types such as strings, tuples, or custom objects. Lists, in contrast, can store elements of any data type.

5. Memory Management: Arrays in Python do not offer automatic memory management features like those provided by some high-level programming languages. Developers must manually manage memory allocation and deallocation when working with arrays, which can lead to memory leaks or memory fragmentation if not handled properly.

6. Index-Based Access Only: Similar to lists, arrays support index-based access for retrieving and modifying elements. However, arrays do not offer other data manipulation features such as slicing, comprehensions, or built-in functions like `map()` or `filter()`.

Q4. Explain the distinctions between the empty, ones, and zeros functions.

ANSWER.

`empty` creates an uninitialized array, `ones` creates an array filled with ones, and `zeros` creates an array filled with zeros. The choice of which function to use depends on whether you need to initialize the array with specific values (ones or zeros) or if you are fine with uninitialized values and want to avoid unnecessary initialization overhead (empty).

Q5. In the fromfunction function, which is used to construct new arrays, what is the role of the callable argument?

ANSWER.

In the `numpy.fromfunction` function, the `callable` argument refers to a Python function or other callable object that defines the relationship between the indices of the array being constructed and the values assigned to those indices. This callable function is responsible for generating the elements of the new array based on their indices.

Q6. What happens when a numpy array is combined with a single-value operand (a scalar, such as an int or a floating-point value) through addition, as in the expression A + n?

ANSWER.

When a NumPy array is combined with a single-value operand (a scalar) through addition, each element of the array is added to the scalar value individually, resulting in a new array where each element is the sum of the corresponding element in the original array and the scalar value.

If the array and the scalar have different data types, NumPy will perform type coercion to ensure that the data types are compatible for addition. The resulting array will typically have the data type of the more general operand (e.g., if one operand is an integer array and the other is a floating-point scalar, the resulting array will have a floating-point data type).

Q7. Can array-to-scalar operations use combined operation-assign operators (such as += or \*=)? What is the outcome?

ANSWER.

No, array-to-scalar operations cannot use combined operation-assign operators (such as `+=` or `\*=`) directly in NumPy. These operators are designed for in-place modification of arrays, where the modification is applied to every element of the array.

When you try to use combined operation-assign operators with a scalar operand, NumPy will raise a TypeError, as these operators are not defined for scalar operands.

Q8. Does a numpy array contain fixed-length strings? What happens if you allocate a longer string to one of these arrays?

ANSWER.

In NumPy, arrays can contain fixed-length strings using the `numpy.string\_` data type. When you create an array with a fixed-length string data type, each element of the array is expected to have a string with the specified length.

If you allocate a longer string to one of these arrays, NumPy will truncate the string to fit within the specified length. No error will be raised, but the excess characters beyond the specified length will be discarded.

Q10. What is the best way to use a Boolean array to mask another array?

ANSWER.

The best way to use a boolean array to mask another array in NumPy is by using boolean indexing. Boolean indexing allows you to use a boolean array (also known as a boolean mask) to select elements from another array based on the corresponding boolean values.

Q11. What are three different ways to get the standard deviation of a wide collection of data using both standard Python and its packages? Sort the three of them by how quickly they execute.

ANSWER.

The NumPy approach (`np.std()`) is typically the fastest for calculating the standard deviation, followed by Pandas' `DataFrame.std()` method. Standard Python with the `statistics` module is the slowest among the three options, especially for large datasets.

12. What is the dimensionality of a Boolean mask-generated array?

ANSWER.

The dimensionality of the Boolean mask-generated array matches the dimensionality of the original array being masked, while the shape of the resulting array may change based on the elements selected by the boolean mask.