ASSIGNMENT 22

# Q1

The built-in array package in Python provides a basic implementation of arrays, which can offer benefits such as efficient storage and manipulation of homogeneous data elements. It allows for compact memory usage compared to lists, as arrays store data in a contiguous block of memory. Additionally, arrays support various operations like indexing, slicing, and element-wise computations, making them useful for numerical computations and working with large datasets.

# Q2

Some limitations of the array package include the lack of built-in support for high-level mathematical operations and functions commonly used in scientific computing. The array package is also limited in terms of multidimensional array support compared to more advanced packages like NumPy. It primarily provides a low-level interface for basic array functionality and may require additional code for complex operations.

# Q3

Functionality: NumPy offers a more extensive range of mathematical functions, operations, and tools for array manipulation, linear algebra, and numerical computing. It provides a high-level interface for efficient handling of large multidimensional arrays.

Performance: NumPy is designed for optimised performance, utilising efficient algorithms and memory management techniques. It leverages compiled code and allows for vectorized operations, resulting in faster computations compared to the array package.

Multidimensional arrays: NumPy provides robust support for multidimensional arrays and offers functionalities for array reshaping, broadcasting, and advanced indexing.

# Q4

numpy.empty: This function creates a new array without initialising its elements to any particular values. The content of the array is undetermined and may contain arbitrary values depending on the state of the memory.

numpy.ones: This function creates a new array with all elements set to 1. The array has the specified shape and data type.

numpy.zeros: This function creates a new array with all elements set to 0. The array has the specified shape and data type.

# Q5

In the numpy.fromfunction function, the role of the callable argument is to define a function that will be evaluated at each element of the resulting array. The callable should take the coordinates of the element as input and return the desired value for that element. The resulting array will have the shape specified in the shape parameter, and the function will be called for each element to compute its value.

# Q6

When a NumPy array is combined with a single-value operand (a scalar) through addition, such as A + n, the scalar value is broadcasted to match the shape of the array, and element-wise addition is performed. Each element of the array is added to the scalar value, resulting in a new array with the same shape as the original array.

# Q7

Array-to-scalar operations in NumPy do not support combined operation-assign operators like += or \*=. If we attempt to use these operators, a TypeError will be raised.

# Q8

NumPy arrays can contain fixed-length strings by specifying the dtype parameter as str followed by the desired length. For example, np.array(['abc', 'def'], dtype='S3') creates an array of strings with a fixed length of 3 characters. If we attempt to assign a longer string to an array element with a fixed length, the string will be truncated to fit the specified length without raising an error.

# Q9

When we combine two NumPy arrays using operations like addition (+) or multiplication (\*), the arrays must have compatible shapes. The arrays can either have the same shape or be broadcastable to a common shape according to NumPy's broadcasting rules. Broadcasting allows for element-wise operations between arrays of different shapes by automatically extending the smaller array to match the shape of the larger array. The resulting array will have the same shape as the broadcasted arrays, and the operation will be applied element-wise.

# Q10

To use a Boolean array as a mask for another array, we can directly index the masked array with the Boolean array. For example, if mask is a Boolean array and arr is the target array, we can obtain the masked array using masked\_arr = arr[mask]. This operation selects only the elements from arr where the corresponding elements in mask are True, creating a new array with those elements.

# Q11

Using the statistics module: we can use the statistics.stdev function from the standard library's statistics module. This function accepts an iterable of numeric values and returns the standard deviation.

Using the numpy package: we can utilise the numpy.std function to calculate the standard deviation of a NumPy array or a specific axis of a multidimensional array. This function provides efficient computation and supports various options and parameters.

Using the pandas package: If we have a dataset stored in a pandas DataFrame, we can use the DataFrame.std method to compute the standard deviation. This method provides additional functionality for handling missing data and working with labeled data structures.

For large datasets and efficient numerical computations, numpy is generally the preferred option due to its optimised implementation.

# Q12

The dimensionality of a Boolean mask-generated array will be the same as the original array. The Boolean mask is used to select elements from the original array, creating a new array that retains the same dimensionality as the original.