**Wednesday, March 27, 2024**

**NumPy, short for "Numerical Python,". it is open source library. it have many features to do scientific calculations. Travis oliphant founded in 2005. He is American computer scientist.**

**Book: Guide To NumPy and associated manuals**

**1.Differentiate between NumPy arrays and Python lists.**

**NumPy arrays and Python lists are both used for storing collections of elements, but they differ in terms of performance and functionality.**

**NumPy arrays are specialized for numerical computations, offering better efficiency and support for vectorized operations due to their homogeneous nature.**

**Python lists can accommodate elements of different data types, providing greater flexibility but sacrificing some performance. NumPy is favored in scientific computing and data manipulation tasks where numerical efficiency is crucial, while Python lists are more versatile for general-purpose scenarios that require mixed data types and diverse operations.**

**2.How to create NumPy arrays? Explain with an example.**

**import numpy as np**

**arr\_1d=np.array([2,1,4,5])**

**print(arr\_1d)**

**3.How can you reverse a NumPy array?**

**using flip:**

**import numpy as np**

**arr\_1d=np.array([2,1,4,5])**

**arr\_reverse = np.flip(arr\_1d)**

**print(arr\_reverse)**

**using slicing:**

**import numpy as np**

**arr\_1d=np.array([2,1,4,5])**

**arr\_reverse = arr\_1d[::-1]**

**print(arr\_reverse)**

**6.What is the difference between np.dot() and np.matmul() functions.**

**np.dot() function computes the dot product of two arrays.**

**import numpy as np**

**# Creating two arrays**

**array1 = np.array([1, 2, 3])**

**array2 = np.array([4, 5, 6])**

**# Computing dot product using np.dot()**

**dot\_product = np.dot(array1, array2)**

**print("Array 1:", array1)**

**print("Array 2:", array2)**

**print("Dot product:", dot\_product)**

**np.dot(array1, array2) calculates the dot product of array1 and array2, which is 1\*4 + 2\*5 + 3\*6 = 32.**

**import numpy as np**

**# Creating two matrices**

**matrix1 = np.array([[1, 2], [3, 4]])**

**matrix2 = np.array([[5, 6], [7, 8]])**

**# Computing matrix product using np.matmul()**

**matrix\_product = np.matmul(matrix1, matrix2)**

**print("Matrix 1:\n", matrix1)**

**print("Matrix 2:\n", matrix2)**

**print("Matrix product:\n", matrix\_product)**

**output:**

**Matrix 1:**

**[[1 2]**

**[3 4]]**

**Matrix 2:**

**[[5 6]**

**[7 8]]**

**Matrix product:**

**[[19 22]**

**[43 50]]**

**for 2-D arrays and 1-D arrays, np.dot() and np.matmul() produce the same results. However, for N-dimensional arrays, np.dot() and np.matmul() may produce different results due to their handling of higher-dimensional arrays.**

**7.What distinguishes np.mean() from np.average() in NumPy?**

**np.mean() and np.average() without specifying weights give the same result, which is the arithmetic mean of the elements in the array.**

**np.average() with weights calculates the weighted average of the elements based on the provided weights.**

**The np.mean() function calculates the arithmetic mean of elements along the specified axis or of the entire array if no axis is specified.**

**import numpy as np**

**arr = np.array([1, 2, 3, 4, 5])**

**mean = np.mean(arr)**

**print("Mean (np.mean()):", mean)**

**# Using np.average() without weights (equivalent to np.mean())**

**average\_no\_weights = np.average(arr)**

**print("Average without weights (np.average()):", average\_no\_weights)**

**# Using np.average() with weights**

**weights = np.array([0.1, 0.2, 0.3, 0.2, 0.2]) # Example weights**

**weighted\_average = np.average(arr, weights=weights)**

**print("Weighted average (np.average() with weights):", weighted\_average)**

**Mean (np.mean()): 3.0**

**Average without weights (np.average()): 3.0**

**Weighted average (np.average() with weights): 3.1**

**10.How do you check whether an array is empty or contains zero elements?**

**In NumPy, you can use the size attribute to check whether an array is empty or contains zero elements. The size attribute returns the total number of elements in the array. If the size is zero, it indicates that the array is empty or contains zero elements.**

**import numpy as np**

**arr\_1d=np.zeros((1,0))**

**print(arr\_1d.size)**

**np.zeros() is a NumPy function used to create an array filled with zeros.**

**import numpy as np**

**# Creating a 1D array filled with zeros**

**zeros\_1d = np.zeros(5)**

**print("1D array:\n", zeros\_1d)**

**# Creating a 2D array filled with zeros**

**zeros\_2d = np.zeros((3, 4)) # 3 rows, 4 columns**

**print("\n2D array:\n", zeros\_2d)**

**# Creating a 3D array filled with zeros**

**zeros\_3d = np.zeros((2, 3, 4)) # 2 slices, 3 rows, 4 columns**

**print("\n3D array:\n", zeros\_3d)**

**1D array:**

**[0. 0. 0. 0. 0.]**

**2D array:**

**[[0. 0. 0. 0.]**

**[0. 0. 0. 0.]**

**[0. 0. 0. 0.]]**

**3D array:**

**[[[0. 0. 0. 0.]**

**[0. 0. 0. 0.]**

**[0. 0. 0. 0.]]**

**[[0. 0. 0. 0.]**

**[0. 0. 0. 0.]**

**[0. 0. 0. 0.]]]**

**11.How can you determine the data type of the elements stored in a given NumPy array?**

**You can determine the data type of the elements stored in an array using the dtype attribute.**

**import numpy as np**

**arr\_1d=np.array([2,3,4])**

**print(arr\_1d.size)**

**print(arr\_1d.dtype)**

**12.How can you find peak or local maxima in a 1D array?**

**In NumPy, identifying the peaks in an array where the values are greater than their neighboring elements.**

**You can use the scipy.signal.find\_peaks() function from SciPy to find local maxima in an array.**

**import numpy as np**

**from scipy.signal import find\_peaks**

**# Example array**

**arr = np.array([1, 3, 2, 4, 1, 6, 5, 2])**

**# Find local maxima**

**peaks, \_ = find\_peaks(arr)**

**print("Indices of local maxima:", peaks)**

**print("Values of local maxima:", arr[peaks])**

**Indices of local maxima: [1 3 5]**

**Values of local maxima: [3 4 6]**

**13.How to convert a Pandas DataFrame into a NumPy array?**

**You can convert a Pandas DataFrame into a NumPy array using the values attribute of the DataFrame. The values attribute returns the underlying data as a NumPy array.**

**import pandas as pd**

**import numpy as np**

**# Example DataFrame**

**data = {'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]}**

**df = pd.DataFrame(data)**

**# Convert DataFrame to NumPy array**

**array\_from\_df = df.values**

**print("DataFrame:")**

**print(df)**

**print("\nNumPy array:")**

**print(array\_from\_df)**

**DataFrame:**

**A B C**

**0 1 4 7**

**1 2 5 8**

**2 3 6 9**

**NumPy array:**

**[[1 4 7]**

**[2 5 8]**

**[3 6 9]]**

**14.How can you randomly shuffle the elements of a NumPy array?**

**You can randomly shuffle the elements of a NumPy array using the numpy.random.shuffle() function.**

**import numpy as np**

**# Example array**

**arr = np.array([1, 2, 3, 4, 5])**

**# Randomly shuffle the array**

**np.random.shuffle(arr)**

**print("Shuffled array:", arr)**

**Shuffled array: [5 3 2 4 1]**

**15.np.copy() usage numpy**

**The np.copy() function in NumPy is used to create a copy of an array. This function ensures that modifications made to the copied array do not affect the original array and vice versa.**

**import numpy as np**

**# Create an array**

**original\_array = np.array([1, 2, 3, 4, 5])**

**# Create a copy of the array**

**copied\_array = np.copy(original\_array)**

**# Modify the copied array**

**copied\_array[0] = 100**

**# Print both arrays**

**print("Original array:", original\_array)**

**print("Copied array:", copied\_array)**

**Original array: [1 2 3 4 5]**

**Copied array: [100 2 3 4 5]**

**16.difference between shallow copy and deep copy in the context of NumPy arrays.**

**the terms "shallow copy" and "deep copy" refer to different ways of creating copies of arrays, and they have different implications for how the data is stored and referenced.**

**Shallow Copy:A shallow copy creates a new array object but does not copy the data itself. Instead, it references the same data as the original array.Any modifications made to the elements of the shallow copy will affect the original array, and vice versa.**

**Deep Copy:A deep copy creates a completely new array object and copies the data from the original array into the new array.Modifications made to the elements of the deep copy do not affect the original array, and vice versa.**

**import numpy as np**

**# Original array**

**original\_array = np.array([1, 2, 3, 4, 5])**

**# Shallow copy**

**shallow\_copy = original\_array.copy() # or shallow\_copy = original\_array[:]**

**# Deep copy**

**deep\_copy = np.copy(original\_array)**

**# Modify the copies**

**shallow\_copy[0] = 100**

**deep\_copy[0] = 1000**

**# Print original array and copies**

**print("Original array:", original\_array)**

**print("Shallow copy:", shallow\_copy)**

**print("Deep copy:", deep\_copy)**

**Original array: [1 2 3 4 5]**

**Shallow copy: [100 2 3 4 5]**

**Deep copy: [1000 2 3 4 5]**

**17.How can you compute the cross-product of two vectors in NumPy?**

**In NumPy, you can compute the cross-product of two vectors using the numpy.cross() function.**

**import numpy as np**

**# Define two 3D vectors**

**vector1 = np.array([1, 2, 3])**

**vector2 = np.array([4, 5, 6])**

**# Compute the cross-product**

**cross\_product = np.cross(vector1, vector2)**

**print("Cross-product:", cross\_product)**

**Cross-product: [-3 6 -3]**

**18.Explain the concept of memory layout in NumPy arrays and the difference between C-order and F-order.**

**In NumPy, the memory layout refers to how the elements of a multidimensional array are stored in memory. NumPy allows you to specify the memory layout when creating arrays, and the two main options are C-order (row-major order) and F-order (column-major order).**

**C-order (row-major order):In C-order, also known as row-major order, elements of the array are stored row by row in memory.This memory layout is suitable for accessing elements row-wise, which is typical in many programming languages and applications.defaults order is C-order.**

**F-order (column-major order):In F-order, also known as column-major order, elements of the array are stored column by column in memory.This memory layout is suitable for accessing elements column-wise, which is typical in some numerical computing tasks.**

**import numpy as np**

**# Creating a C-order array (row-major order)**

**c\_order\_array = np.arange(6).reshape(2, 3)**

**print("C-order array:\n", c\_order\_array)**

**# Creating an F-order array (column-major order)**

**f\_order\_array = np.arange(6).reshape(2, 3, order='F')**

**print("\nF-order array:\n", f\_order\_array)**

**C-order array:**

**[[0 1 2]**

**[3 4 5]]**

**F-order array:**

**[[0 2 4]**

**[1 3 5]]**

**19.How do you calculate the mean and standard deviation of a NumPy array?**

**You can calculate the mean and standard deviation of a NumPy array using the numpy.mean() and numpy.std() functions,**

**import numpy as np**

**# Example array**

**arr = np.array([1, 2, 3, 4, 5])**

**# Calculate mean**

**mean\_value = np.mean(arr)**

**# Calculate standard deviation**

**std\_deviation = np.std(arr)**

**print("Mean:", mean\_value)**

**print("Standard deviation:", std\_deviation)**

**Mean: 3.0**

**Standard deviation: 1.4142135623730951**

**20.Discuss the purpose of the np.random module in NumPy.**

**The np.random module in NumPy is designed for generating random numbers and random data for various applications, such as simulations, statistical analysis, and machine learning.**

**numpy.random.rand():This function generates random numbers from a uniform distribution over the interval [0, 1).**

**Example usage:**

**import numpy as np**

**# Generate a 2x3 array of random numbers between 0 and 1**

**random\_array = np.random.rand(2, 3)**

**print(random\_array)**

**numpy.random.randint():This function generates random integers from a discrete uniform distribution.It returns random integers within a specified range [low, high) (including low but excluding high).**

**import numpy as np**

**# Generate a random integer between 0 and 9**

**random\_int = np.random.randint(10)**

**print(random\_int)**

**# Generate a 1D array of 5 random integers between 1 and 100**

**random\_array = np.random.randint(1, 101, size=5)**

**print(random\_array)**

**21.Discuss the importance of vectorization in NumPy.**

**Vectorization is a fundamental concept in NumPy that plays a crucial role in improving the performance and efficiency of numerical computations.**

**It refers to the process of applying operations on entire arrays rather than looping over individual elements, thus taking advantage of the highly optimized, parallelized implementations of these operations provided by NumPy.**

**Performance**

**Concise and Readable Code**

**Simplified Parallelism**

**Broadcasting**

**Interoperability and Integration**

**22.What is the difference between hstack() and vstack() in NumPy.**

**In NumPy, both numpy.hstack() and numpy.vstack() are functions used to stack arrays horizontally and vertically.**

**numpy.hstack():**

**This function stacks arrays horizontally, meaning it concatenates arrays along the second axis (axis 1).It takes a sequence of arrays as input and stacks them horizontally to form a single array.**

**import numpy as np**

**# Example arrays**

**arr1 = np.array([[1, 2], [3, 4]])**

**arr2 = np.array([[5, 6], [7, 8]])**

**# Horizontal stack**

**hstack\_result = np.hstack((arr1, arr2))**

**print("Horizontal stack:")**

**print(hstack\_result)**

**Horizontal stack:**

**[[1 2 5 6]**

**[3 4 7 8]]**

**numpy.vstack():This function stacks arrays vertically, meaning it concatenates arrays along the first axis (axis 0).It takes a sequence of arrays as input and stacks them vertically to form a single array.The arrays to be stacked must have the same shape along all dimensions except the first (axis 0).**

**import numpy as np**

**arr1 = np.array([[1, 2], [3, 4]])**

**arr2 = np.array([[5, 6], [7, 8]])**

**vstack\_result = np.vstack((arr1, arr2))**

**print("Vertical stack:")**

**print(vstack\_result)**

**Vertical stack:**

**[[1 2]**

**[3 4]**

**[5 6]**

**[7 8]]**

**23.How does NumPy integrate with popular machine learning libraries?**

**many machine learning algorithms and models in scikit-learn also leverage NumPy arrays for data manipulation and mathematical operations.**

**24.Discuss the role of NumPy in feature engineering for machine learning.**

**NumPy plays a critical role in feature engineering for machine learning by providing powerful tools for data manipulation, preprocessing, and feature extraction. Feature engineering involves transforming raw data into a format that is suitable for input into machine learning algorithms, and NumPy's capabilities are instrumental in this process.**

**25.Describe the implementation of moving average for a 1D array in NumPy.**

**Implementing moving average for a 1D array in NumPy with below steps.**

**Define the Input Array**

**Specify the Window Size**

**Use NumPy's Convolution Function: NumPy's numpy.convolve() function**

**Normalize the Result**

**import numpy as np**

**def moving\_average(arr, window\_size):**

**# Define the kernel for the moving average**

**kernel = np.ones(window\_size) / window\_size**

**# Use NumPy's convolution function to compute the moving average**

**moving\_avg = np.convolve(arr, kernel, mode='valid')**

**return moving\_avg**

**# Example input array**

**input\_array = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])**

**# Specify the window size**

**window\_size = 3**

**# Compute the moving average**

**result = moving\_average(input\_array, window\_size)**

**print("Moving average:", result)**

**Moving average: [ 2. 3. 4. 5. 6. 7. 8.]**

**Q.2 How do I create a NumPy array?**

**Using np. array()**

**np.zeros()**

**np.ones()**

**np.full()**

**np.arange()**

**np.linspace()**

**Q.3 What are the main features of Numpy?**

**Here are some main features of the NumPy.**

**Arrays**

**Efficiency**

**Mathematical Functions**

**Broadcasting**

**Integration with other libraries**

**Multi-dimensional arrays**

**Indexing and Slicing**

**Memory Management**

**Q.4 How do you calculate the dot product of two NumPy arrays?**

**numpy.dot(a, b)**

**Q.5 How do I access elements in a NumPy array?**

**access elements in an array by indexing and slicing**

**1d array:**

array1 = np.array([1, 3, 5, 7, 9])

* array1[0] - to access the first element, i.e. **1**
* array1[2] - to access the third element, i.e. **5**

**2d array;**

array1 = np.array([[1, 3, 5],

[7, 9, 2],

[4, 6, 8]])

Now, say we want to access the element in the third row and second column we specify the index as:

array1[2, 1] # returns 6

**Q.7 How do you reshape a NumPy array?**

**np.reshape() function**

**Q.8 How to perform element-wise operations on NumPy arrays?**

**To perform element-wise operations on NumPy arrays, you can use standard arithmetic operators.**

**result\_addition= array1 + array2**

**result\_subtract = array1 - array2**

**result\_multiply = array1 \* array2**

**result\_divide = array1 / array2**

**result\_power = np.power(array1, 2)**

**Q.9 Define the var function in NumPy.**

**In NumPy, the var function is used to compute the variance of elements in an array .**

**Variance is a measure of the spread of data points.**

**np.var(a, axis=None, dtype=None)**

**Q.10 Define the min and max function in NumPy.**

**np.min()**

**np.max()**

**Q.11 How to generate random numbers with NumPy?**

**Generating a Random Float between 0 and 1 using np.random.rand()**

**Generating a Random Integer within a Range using np.random.randint().**

**Q.12 What is the purpose of NumPy in Python?**

**NumPy (Numerical Python) is a fundamental library in Python for scientific computing and data analysis. it is the main purpose for providing support for large and multi-dimensional arrays and matrices.**

**Q.13 How can you create a NumPy array from a Python list?**

**We can create a NumPy array from a Python list using the np.array() constructor provided by NumPy.**

**python\_list = [1, 2, 3, 4, 5]**

**# Convert the Python list to a NumPy array**

**numpy\_array = np.array(python\_list)**

**Q.15 What are some common data types supported by NumPy?**

**int**

**float**

**complex**

**bool**

**object**

**datetime**

**Q.16 How can you concatenate two NumPy arrays vertically?**

**Using np.vstack() : array= np.vstack((array1, array2))**

**Using np.concatenate() with axis: array= np.concatenate((array1, array2), axis=0)**

**Q.17 What is the significance of the random module in NumPy?**

**Random Number Generation**

**Random Sequences**

**Probability Distributions**

**Random Choices**

**Q.18 How can you generate random numbers following a normal distribution using NumPy?**

**numpy.random.normal.**

**Q.19 What is Matrix Inversion in NumPy?**

**NumPy provides a convenient function called numpy.linalg.inv() to compute the inverse of a square matrix.**

**Q.20 Define the mean function in NumPy.**

**The arithmetic mean (average) in NumPy can be calculated using numpy.mean().**

**The summation of all elements is then divided by the overall number of elements, which provides the average.**

**import numpy as np**

**arr =np.array([3,4,5])**

**print(arr.mean())**

**Q.20 Convert a multidimensional array to 1D array.**

**Using flatten():**

**# Create a multidimensional array**

**multidimensional\_array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])**

**# Use the flatten() method to convert it to a 1D array**

**one\_dimensional\_array = multidimensional\_array.flatten()**

**Q.21 Write a NumPy code snippet to create an array of zeros.**

**zeros\_1d = np.zeros(5)**

**Q.22 How can you identify outliers in a NumPy array?**

**Outliers are data points that significantly deviate from the majority of the data and can adversely affect the results of data analysis.**

**Identifying Outliers:**

**Calculate Descriptive Statistics: Compute basic statistics like the mean and standard deviation of the array to understand the central tendency and spread of the data.**

**Define a Threshold: Decide on a threshold where data points are considered outliers.**

**Q.23 How do you remove missing or null values from a NumPy array?**

**mask = np.isnan(my\_array)**

**Q.24 What is the difference between slicing and indexing in NumPy?**

**In NumPy, both slicing and indexing are fundamental operations for accessing and manipulating elements in arrays.**

**Slicing is the process of extracting a portion or a subset of elements from a NumPy array by specifying a range or slice of indices.**

**Indexing refers to the process of accessing individual elements or subsets of elements from a NumPy array using specific indices or index values.**

**Q.26 How can you create array with same values.**

**arr = np.full(5, 7)**

**This function takes two arguments: the shape of the array and the fill value.**

**Using Broadcasting:**

**# Create a 1D array with 5 elements, all set to 7**

**arr = 7 \* np.ones(5)**

**Q.27 How can you modify the data type of a NumPy array?**

**Using the astype() method:**

**new\_array = original\_array.astype(float)**

**Q.28 What is a masked array in NumPy.**

**This allows you to work with data that has missing or invalid values without having to modify the original data.**

**Q.29 What are some of the limitations of NumPy.**

**Homogeneous Data Types**

**Memory Usage**

**Limited Support for Missing Data**

**Limited Support for Labeling Data**

**Limited Support for Advanced Statistics**

**Limited Support for Distributed Computing**

**Q.30 How do you sort a NumPy array in ascending or descending order?**

**# Create a NumPy array**

**my\_array = np.array([3, 1, 2, 4, 5])**

**# Sort the array in ascending order**

**sorted\_array = np.sort(my\_array)**

**Q.39 How to Get the eigen values of a matrix.**

**np.eigvals(matrix)**

**Q.42 How to compare two NumPy arrays?**

**Method 2: Using array\_equal()**

**numpy.array\_equal(arr1, arr2)**

**Q.45 Define a polynomial function.**