ANSWERS

SECTION A

1A)

The Python package NumPy, which stands for “Numerical Python,” is the basis for numerical and scientific computing. It offers support for arrays, matrices, and a variety of mathematical operations that can effectively operate on these arrays. In the Python environment, NumPy is a fundamental library for manipulating and analyzing data. It also used in many additional libraries that are used in data science, machine learning areas.

2A)

We can create NumPy arrays using various methods. Here are some common ways to create NumPy arrays:

- Using `np.array()`
- `np.zeros()`
- `np.ones()`
- `np.full()`
- `np.arange()`
- `np.linspace()`

3.A)

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

4A)

Calculating the dot product of two NumPy arrays we used `numpy.dot()` function and we also used the `@` operator:

- Using `numpy.dot()` function:

  ```python
  numpy.dot(a, b)
  ```

  `a`: The first input array (NumPy array).
  `b`: The second input array (NumPy array).
- Using the `@` operator:

K. PRAKASH SENAPATI

```python
a @ b
```

Both methods will return the dot product of the two arrays as a scalar value.

5A)

In NumPy, we can access elements in an array by indexing and slicing. Here's how we can do it:

- Slicing: You can also access a range of elements using slicing. Slicing allows you to extract a subset of the array based on the indices or ranges you provide.
- Boolean indexing: Boolean indexing is used to access elements based on a condition. It helps us when we want to select elements that meet the criteria.

6A)

In NumPy, we have two ways to copy an array: shallow copy and deep copy. Here is the main difference between both of them:

- Shallow copy:
  - A shallow copy is a new array that is a view of the original array's data.
  - The new array is referencing the same data as the original array, so there's actually no duplication happening.
  - Changes made in the original array will also be reflected in the shallow copy, and vice versa.
- Deep copy:
  - A deep copy is a completely new and independent array that does not share any data with the original array.
  - The data is duplicated, creating a separate copy in memory.
  - Changes made in the original array do not affect the deep copy, and vice versa.

7A)

We can reshape a NumPy array by using the `reshape()` method or the `np.reshape()` function. It helps us to change the dimensions of the array and keep all the elements constant.

- Using the `reshape()` method:

```python
array1 = original_array.reshape(new_shape)
```

- Using the `np.reshape()` function:

```python
array1 = np.reshape(original_array, new_shape)
```

In both cases, `original_array` is the existing NumPy array you want to reshape, and `new_shape` is a tuple specifying the desired shape of the new array.

K. PRAKASH SENAPATI

8A)

. To perform element-wise operations on NumPy arrays, you can use standard arithmetic operators. NumPy automatically applies these operations element-wise when you use them with arrays of the same shape.

```python
import numpy as np

# Create two NumPy arrays
array1 = np.array([1, 2, 3, 4, 5])
array2 = np.array([6, 7, 8, 9, 10])

# Perform element-wise operations
result_addition = array1 + array2
result_subtract = array1 - array2
result_multiply = array1 * array2
result_divide = array1 / array2
result_power = np.power(array1, 2)
```

9A)

In NumPy, the `var` function is used to compute the variance of elements in an array or along a specified axis. Variance is a measure of the spread or dispersion of data points.

```python
np.var(a, axis=None, dtype=None)
```

- `a`: The input array for which you want to calculate the variance.
- `axis`: Axis or axes along which the variance is computed. If not specified, the variance is calculated for the whole array. It can be an integer or a tuple of integers to specify multiple axes.
- `dtype`: The data type for the returned variance. If not specified, the data type is inferred from the input array.

10.A)

In NumPy, you can use the `min()` and `max()` functions to find the minimum and maximum values in a NumPy array, respectively.

- `np.min()` Function:

  The `np.min()` function calculates the minimum value in a NumPy array along a specified axis or for the entire array.

  ```python
  np.min(a, axis=None)
  ```

  - `a`: The input array in which you want to find the minimum value.
  - `axis`: Axis or axes along which the minimum value is computed. If not specified, the minimum is calculated for the whole array. It can be an integer or a tuple of integers to

K. PRAKASH SENAPATI

specify multiple axes.

- `np.max()` Function:

  The `np.max()` function calculates the maximum value in a NumPy array along a specified axis or for the entire array.

  ```python
  np.max(a, axis=None)
  ```

  - `a`: The input array in which you want to find the maximum value.
  - `axis` (optional): Axis or axes along which the maximum value is computed. If not specified, the maximum is calculated for the whole array

.

11A)

NumPy provides a wide range of functions for generating random numbers. You can generate random numbers from various probability distributions, set seeds for reproducibility, and more. Here are some common ways to generate random numbers with NumPy:

- Using `np.random.rand()`

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

  ```python
  random_float = np.random.rand()
  ```

- Using `np.random.randint()`

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

  ```python
  random_integer = np.random.randint()
  ```

- Using `np.random.randn()`

  ```python
  random_float = np.random.rand()
  ```

- Using `np.random.seed()`

  We can set a seed using `np.random.seed()` to ensure that the generated random numbers are reproducible.

  ```python
  np.random.seed(seed_value)
  ```

12A)

NumPy (Numerical Python) is a fundamental library in Python for scientific computing and data analysis. It provides support for large and multi-dimensional arrays and matrices.

13A)

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

```python

K. PRAKASH SENAPATI
```

```
python_list = [1, 2, 3, 4, 5]

# Convert the Python list to a NumPy

array
numpy_array = np.array(python_list)
```

14A)

We can access elements in a NumPy array based on specific conditions using boolean indexing. Boolean indexing allows you to select elements that meet the specified criteria.

15A)

Here are some common data types (dtypes) supported by NumPy:

- `np.int64`: 64-bit integer
- `np.float64`: 64-bit floating-point number
- `np.bool_`: Boolean (True or False)
- `np.complex128`: Complex number (128-bit)
- `np.str_`: String
- `np.object_`: Generic object

16A)

. In NumPy, we can concatenate two arrays vertically (along rows) using the `np.vstack()` function or the `np.concatenate()` function with the `axis` parameter set to `0`.

- Using `np.vstack()`

```python
array1 = np.array([1, 2, 3])
array2 = np.array([4, 5, 6])

concatenated_array = np.vstack((array1, array2))
```

- Using `np.concatenate()`

```python
array1 = np.array([1, 2, 3])
array2 = np.array([4, 5, 6])

concatenated_array = np.concatenate((array1, array2), axis=0)
```

17A)

The random module in NumPy, accessed through `np.random`, is used to generate random numbers and perform random sampling. It provides a wide range of functions for generating random numbers from various probability distributions.

K. PRAKASH SENAPATI

18.A)

NumPy provides the `np.random.normal()` function to generate random numbers following a normal (Gaussian) distribution. The function allows us to specify the mean, standard deviation, and size of the generated random numbers.

```python
import numpy as np

# Generate random numbers following a normal distribution
mean = 0
std_dev = 1
size = 10

random_numbers = np.random.normal(loc=mean, scale=std_dev, size=size)
```

- `loc`: The mean (average) of the normal distribution.
- `scale`: The standard deviation of the normal distribution.
- `size`: The number of random numbers to generate.

19.

Matrix inversion in NumPy refers to the process of finding the inverse of a square matrix. The inverse of a matrix A is another matrix, typically denoted as A^(-1), such that the product of A and A^(-1) is the identity matrix.

```python
import numpy as np

# Create a square matrix
matrix = np.array([[1, 2],
                   [3, 4]])

# Compute the inverse of the matrix
inverse_matrix = np.linalg.inv(matrix)
```

20.

In NumPy, the `mean` function is used to compute the arithmetic mean (average) of elements in an array or along a specified axis.

```python
import numpy as np

# Create a NumPy array
array = np.array([1, 2, 3, 4, 5])

# Compute the mean of the entire array
```

K. PRAKASH SENAPATI

```
mean_value = np.mean(array)

# Compute the mean along a specific axis
matrix = np.array([[1, 2, 3],
```

K. PRAKASH SENAPATI

```
                [4, 5, 6]])

mean_row = np.mean(matrix, axis=0)
mean_column = np.mean(matrix, axis=1)
```

21.

In NumPy, we can convert a multidimensional array to a 1D array using the `flatten()` method or the `ravel()` function.

- Using the `flatten()` method:

```python
import numpy as np

# Create a multidimensional array
multi_array = np.array([[1, 2, 3], [4, 5, 6]])

# Convert to a 1D array using flatten()
flat_array = multi_array.flatten()
```

- Using the `ravel()` function:

```python
import numpy as np

# Create a multidimensional array
multi_array = np.array([[1, 2, 3], [4, 5, 6]])

# Convert to a 1D array using ravel()
flat_array = np.ravel(multi_array)
```

22

. We can create an array of zeros using the `np.zeros()` function. The `np.zeros()` function allows us to specify the shape and data type of the array.

```python
import numpy as np

# Create an array of zeros with a specified shape
array_of_zeros = np.zeros(shape=(3, 4), dtype=float)
```

23.

In NumPy, we can identify outliers in an array using statistical methods such as the Z-score or the Interquartile Range (IQR).

- Using the Z-score:

  The Z-score measures how many standard deviations a data point is from the mean. Data

K.PRAKASH SENAPATI

points with a Z-score greater than a certain threshold (e.g., 3) can be considered outliers.

```python
import numpy as np
from scipy import stats

# Create a NumPy array
data = np.array([1, 2, 3, 4, 5, 6, 7, 100])

# Calculate Z-scores
z_scores = stats.zscore(data)

# Identify outliers (e.g., Z-score greater than 3)
outliers = data[np.abs(z_scores) > 3]
```

- Using the Interquartile Range (IQR):

The IQR is the range between the first quartile (25th percentile) and the third quartile (75th percentile). Data points that fall below the lower bound (Q1 - 1.5*IQR) or above the upper bound (Q3 + 1.5*IQR) can be considered outliers.

```python
import numpy as np

# Create a NumPy array
data = np.array([1, 2, 3, 4, 5, 6, 7, 100])

# Calculate the first and third quartiles
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)

# Calculate the IQR
IQR = Q3 - Q1

# Identify outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = data[(data < lower_bound) | (data > upper_bound)]
```

24.

In NumPy, we can remove missing or null values from an array using the `np.isnan()` function to identify the NaN (Not a Number) values and then filter them out.

```python
import numpy as np

# Create a NumPy array with missing values (NaN)
array = np.array([1, 2, np.nan, 4, 5, np.nan, 7])
```

K. PRAKASH SENAPATI

```
# Remove missing values (NaN) from the array
cleaned_array = array[~np.isnan(array)]
```

25.

In NumPy, slicing and indexing are two different methods for accessing elements in an array.

- Indexing:

  Indexing allows you to access individual elements in an array by specifying their indices. NumPy arrays are zero-based, so the first element is at index 0.

- Slicing:

  Slicing allows you to access a range or subset of elements in an array by specifying a start, stop, and step value. The syntax for slicing is `array[start:stop:step]`.

26

. In NumPy, we can compute the Fourier transform of a signal using the `np.fft.fft()` function. The Fourier transform is used to convert a signal from the time domain to the frequency domain.

```python
import numpy as np

# Create a sample signal (e.g., a sine wave)
t = np.linspace(0, 1, 1000)
signal = np.sin(2 * np.pi * 50 * t)

# Compute the Fourier transform of the signal
fourier_transform = np.fft.fft(signal)

# Compute the corresponding frequencies
frequencies = np.fft.fftfreq(signal.size, d=t[1] - t[0])
```

27.

We can create an array with the same values in NumPy using the `np.full()` function. This function allows you to create an array of a specified shape and fill it with a constant value.

```python
import numpy as np

# Create an array with the same values
array_with_same_values = np.full(shape=(3, 4), fill_value=5, dtype=int)
```

28

. In NumPy, we can modify the data type of an array using the `astype()` method. This method allows you to create a new array with the desired data type.

```python
```

K. PRAKASH SENAPATI

```
import numpy as np

# Create a NumPy array
array = np.array([1, 2, 3, 4, 5])

# Modify the data type to float
modified_array = array.astype(float)
```

29

. In NumPy, a masked array is an array that allows you to mask certain elements so that they are excluded from computations. Masked arrays are useful for handling missing or invalid data.

```python
import numpy as np
import numpy.ma as ma

# Create a NumPy array with missing values (NaN)
array = np.array([1, 2, np.nan, 4, 5, np.nan, 7])

# Create a masked array
masked_array = ma.masked_invalid(array)

# Compute the mean of the masked array (excluding masked elements)
mean_value = masked_array.mean()
```

30.

In NumPy, the `where()` function is used to create an array of indices based on a specified condition. It returns the indices of elements that satisfy the condition.

```python
import numpy as np

# Create a NumPy array
array = np.array([1, 2, 3, 4, 5, 6, 7])

# Use the where() function to find the indices of elements greater than 4
indices = np.where(array > 4)

# Extract the elements that satisfy the condition
filtered_elements = array[indices]
```

K. PRAKASH SENAPATI

31.

NumPy provides powerful array manipulation capabilities, and Matplotlib is a popular library for creating various types of plots and charts. Here's how you can use NumPy with Matplotlib:

```
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)
# Create a basic line plot
plt.plot(x, y)
```

32.

The diag() function in NumPy is used to extract or manipulate the diagonal elements of a square matrix (a matrix with the same number of rows and columns). It can serve various purposes in matrix operations and linear algebra. Here are some common uses of the diag() function:

33.

NumPy arrays offer several advantages over Python lists when it comes to numerical and scientific computing. Here are some key reasons why NumPy arrays are often preferred:

1. Performance
2. Vectorization
3. Broadcasting
4. Multidimensional Arrays
5. Memory Management
6. Standardization

34.

Negative indexing in NumPy arrays allows individuals to access elements from the end of an array by using negative integers as indices. This feature offers a convenient way to retrieve elements relative to the array's end, without the need for precise knowledge of its length. In NumPy, -1 corresponds to the last element, -2 refers to the second-to-last element, and so forth.

35.

Using NumPy and Matplotlib together, you can create a simple plot. NumPy is primarily a library for numerical computations with arrays, while Matplotlib is a popular Python library for creating various plots and charts. To create a plot, first import NumPy and Matplotlib,

K. PRAKASH SENAPATI

then use the functions from both libraries.

```
# Create any sample data using NumPy
x = np.linspace(0, 2 * np.pi, 100) #generate 100 points between 0 and 2*pi
y = np.sin(x) #compute the sine of each point
```

K. PRAKASH SENAPATI

```
plt.plot(x,y,label = 'Sine Wave') #plotting the sine  wave
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
plt.grid(True)
#show the plot
plt.show()
```

36.

The vstack() and hstack() functions in NumPy are used to stack or concatenate arrays vertically and horizontally, respectively. These functions are essential for combining arrays in different dimensions and are widely used in various data processing and manipulation tasks. Here's a discussion of their uses:

vstack() (Vertical Stack):

np.vstack() is used to vertically stack or concatenate arrays along the vertical axis (axis 0). This means that it stacks arrays on top of each other.

It is commonly used when you want to combine arrays with the same number of columns (i.e., the same number of features or variables) but different rows (i.e., different data points).

hstack() (Horizontal Stack):

np.hstack() is used to horizontally stack or concatenate arrays along the horizontal axis (axis 1). This means that it stacks arrays side by side.

It is commonly used when we want to combine arrays with the same number of rows.

37.

NumPy handles numerical exceptions, such as overflow, underflow, and invalid mathematical operations, in a way that ensures stability and well-defined behavior in numerical computations.

NumPy's handling of numerical exceptions is designed to provide predictable behavior in numerical computations while adhering to standard conventions and IEEE 754 rules. It allows users to work with exceptional cases and maintain control over error handling when needed.

38

. The random module in NumPy is a powerful and essential component for generating random numbers and performing various random processes in scientific computing, simulations, statistics, and machine learning. Here are some significant aspects of the random module in NumPy.

K. PRAKASH SENAPATI

39.

With the help of np.eigvals() method, we can get the eigen values of a matrix by using np.eigvals() method.

np.eigvals(matrix)

40.

The Determinant of a square matrix is a unique number that can be derived from a square matrix. Using the numpy.linalg.det() method, NumPy gives us the ability to determine the determinant of a square matrix.

numpy.linalg.det(array)

41.

We employ the numpy.linalg.norm() method of the NumPy Python library to determine a matrix or vector norm. Depending on the value of its parameters, this function returns either one of the seven matrix norms or one of the infinite vector norms.

numpy.linalg.norm(x, ord=None, axis=None)

42.

Here, we'll concentrate on the array comparison performed with NumPy. When two NumPy arrays are compared, the presence of the same element at each corresponding index indicates whether the arrays are comparable.

Method 1: We generally use the == operator to compare two NumPy arrays to generate a new array object. Call ndarray.all() with the new array object as ndarray to return True if the two NumPy arrays are equivalent.

Method 2: Using array_equal()

This array_equal() function checks if two arrays have the same elements and same shape.

numpy.array_equal(arr1, arr2)

43.

A matrix's decomposition into the form "A=QR," where Q is an orthogonal matrix and R is an upper-triangular matrix, is known as QR factorization. With the aid of numpy, we can determine the QR decomposition of a given matrix.linalg.qr().

numpy.linalg.qr(a, mode='reduced')

a : matrix(M,N) which needs to be factored.

K. PRAKASH SENAPATI

mode : it is optional.

44.

K. PRAKASH SENAPATI

The aim is to remove integers from an array that contains float and integers using a numpy array. Let's look at a couple approaches to completing a task.

- Using astype(int)
- Using np.equal() and np.mod()
- Using np.isclose()
- Using round()

45

. Using NumPy's numpy.poly1d class, you can define polynomial functions. Using the coefficients of a polynomial, we can create a new function using this class. In NumPy, define a polynomial function as follows:

numpy.poly1d(arr, root, var)

46.

An ndarray (short for "N-dimensional array") is a fundamental data structure used in NumPy (Numerical Python) for effectively storing and manipulating data, particularly numerical data. It offers a multidimensional, homogeneous array with a variety of capabilities for mathematical and array-oriented operations and is the fundamental building block of the NumPy library.

47.

Due to a number of distinguishing characteristics, NumPy (Numerical Python) is a distinctive and crucial library in the Python ecosystem for numerical and scientific computing.

The ndarray (N-dimensional array), a very effective, homogeneous, and multi-dimensional data structure, is introduced by NumPy. An ndarray's same data types enable for memory-efficient storage and mathematical operations that are optimised.

48.

Shape and size attributes of an array give information about the dimension of the array but there are some differences.

shape	size
The array's dimensions are represented by a tuple that the shape attribute returns. The size of the array along each axis or	The attribute 'size' gives the amount of elements in the array, regardless of its

K. PRAKASH SENAPATI

shape	size
dimension is represented by an individual member of the tuple.	shape. It provides a single integer value representing the size of the flattened array.
It gives details about the array's dimensions, including the number of rows, columns, and other dimensions.	It is the result of all the shape tuple's components.
A single element serving as the array's length makes up the form tuple for a 1D array.	It provides you with the array's overall element count.

49.

The key distinctions between NumPy arrays and common Python sequences (such as lists and tuples) make NumPy arrays the best option for numerical and scientific computing. Here are some significant variations:

While NumPy arrays are homogeneous and require that every member have the same data type, Python sequences can hold items of various data types. In NumPy, this homogeneity enables effective storage and optimised operations.

50

. The core building block of NumPy (Numerical Python), universal functions, or "ufuncs," allow for effective element-wise operations on ndarrays (N-dimensional arrays). Ufuncs in NumPy are functions that work on ndarray items, allowing you to efficiently and vectorizely carry out operations like exponentiation, trigonometry, and more.

51

. The terms "ndarray" and "array" in NumPy essentially refer to the same object: a multi-dimensional array object that makes it possible to effectively conduct mathematical operations on data. While "array" is a more casual term that is frequently used in everyday speech and documentation to refer to NumPy arrays, "ndarray" is the official term used within NumPy. Both terms refer to the same data structure that NumPy uses for effective numerical computations.

ndarray

- The official NumPy word for a multi-dimensional array is "ndarray".

K. PRAKASH SENAPATI

- In particular, it alludes to the NumPy array object.
- You generate “ndarray” objects when you use NumPy functions or constructors to create arrays, such as np.array(), np.zeros(), or np.ones().

array

- While it is not the official nomenclature, the word “array” is frequently used informally to refer to NumPy arrays.
- In reality, when people use the word “array” in relation to NumPy, they often mean “ndarray.”

52

. You can use the DataFrame’s.values attribute to convert a Pandas DataFrame into a NumPy array.

```
import pandas as pd
import numpy as np

# Create a Pandas DataFrame (replace this with your actual DataFrame)
data = {'A': [1, 2, 3], 'B': [4, 5, 6]}
df = pd.DataFrame(data)

# Convert the DataFrame to a NumPy array
numpy_array = df.values
```

53.

In NumPy, the term “vectorization” refers to the capacity to carry out element-wise operations on whole arrays or sizable sections of arrays without the use of explicit Python loops. You can perform a single action on the entire array or a sizable section of it all at once rather than iterating through each element in an array one at a time. This idea is comparable to how operations on vectors or matrices operate in linear algebra.

54.

We can reverse a NumPy array using the [::-1] slicing technique.

```
import numpy as np

# Create a NumPy array (replace this with your array)
original_array = np.array([1, 2, 3, 4, 5])

# Reverse the array
reversed_array = original_array[::-1]

print(reversed_array)
```

K. PRAKASH SENAPATI

55.

Unlike Pandas DataFrames, NumPy arrays do not by default support missing or null values. However, you may achieve a similar feature by filtering out missing or null data using a masked array or a boolean mask.

Using Masked array:

```
import numpy as np

# Create a NumPy array with missing values (NaN)
arr = np.array([1.0, 2.0, np.nan, 4.0, 5.0])

# Create a masked array where missing values are masked
masked_arr = np.ma.masked_invalid(arr)

# Access only non-missing values
clean_data = masked_arr.compressed()

print(clean_data)
```

56.

Indexing

- In NumPy, the term "indexing" refers to the process of choosing particular elements or groups of elements from an array according to their precise places, generally by using integers or boolean masks.
- You can use it to access and change individual items or particular elements at predetermined locations in the array.
- When you wish to locate and change or remove particular values in the array based on their positions, indexing might be handy for focused data cleaning operations.
- For instance, indexing can be used to set specific items to NaN (not-a-number) for missing data or to replace all negative values with zeros.

Slicing

- Using a range of indices or a boolean mask, slicing entails extracting a continuous portion of an array.
- It enables you to build an array from a portion of the original data, which is helpful for extracting particular rows or columns.
- When you want to filter and extract rows or columns that fit specific criteria or when you only want to work with a section of the data, slicing is useful for cleaning the data.

K. PRAKASH SENAPATI

- Slicing, for instance, can be used to choose rows that meet particular criteria, such as all rows where values in a particular column are greater than a predetermined threshold.

57

. The np.ma.masked_array() function in NumPy is used to build a masked array. It is useful when dealing with data when some numbers are regarded as "masked" or "invalid" and ought to be left out of calculations. When dealing with missing data or poor data quality, masked arrays are very helpful since they let you conduct operations while disregarding the masked values.

```
import numpy as np

# Create a NumPy array with some missing or invalid values (e.g., -999)
data = np.array([1, 2, -999, 4, -999, 6])

# Create a masked array where -999 is considered masked (invalid)
masked_data = np.ma.masked_array(data, mask=(data == -999))
```

In conclusion, a masked array in NumPy is a useful tool for working with data that contains blank or incorrect values. It enables you to perform operations on legitimate data while excluding masked numbers, guaranteeing that the data used in your calculations and analysis is accurate.

58.

Data analysis and machine learning frequently utilize normalization, a preprocessing method, to scale data to a standard range, usually between 0 and 1, or with a mean of 0 and a standard deviation of 1. Many machine learning algorithms can perform better with the aid of normalized data, which also ensures that features with various scales contribute equally to the study.

Some common technique are:

- Min-Max Scaling
- Z-Score Normalization (Standardization)
- Log Transformation
- Box-Cox Transformation
- Robust Scaling

59.

Unlike Pandas DataFrames, NumPy arrays do not by default support missing or null values. However, you may deal with missing values in NumPy arrays by either filtering them out or by replacing them with suitable values.

K. PRAKASH SENAPATI

Replacing Missing values:

```
import numpy as np

# Create a NumPy array with missing values
arr = np.array([1, 2, -999, 4, -999, 6])

# Replace -999 with a specific value
arr[arr == -999] = 0
```

60.

```
import NumPy
print("Printing Original array")
sampleArray = NumPy.array([[34,43,73],[82,22,12],[53,94,66]])
print (sampleArray)
print("Array after deleting column 2 on axis 1")
sampleArray = NumPy.delete(sampleArray , 1, axis = 1)
print (sampleArray)
arr = NumPy.array([[10,10,10]])
print("Array after inserting column 2 on axis 1")
sampleArray = NumPy.insert(sampleArray , 1, arr, axis = 1)
print (sampleArray)
```

61

. The main goal of NumPy is to enable huge, multi-dimensional arrays and matrices while also offering a range of mathematical operations that may be used effectively to these arrays. It serves as the basis for Python's numerical computations.

On the other hand, Pandas is based on NumPy and intended for data analysis and manipulation. Series and DataFrame, two fundamental data structures that are effective for managing structured data types like spreadsheets and SQL tables, are introduced. With its high-level features, Pandas makes data cleaning, transformation, and analysis simple. It is also frequently used for data exploration and manipulation jobs.

In conclusion, NumPy and Pandas are both crucial Python tools for data analysis, but NumPy concentrates on numerical computing using arrays, while Pandas is an expert in data manipulation and analysis with

62

. The following justifies why Python Lists are slower than NumPy Arrays:

Due to its uniform data type storage, which enables best memory consumption and effective vectorized operations, NumPy is quicker than lists. Compared to Python's interpreted code, NumPy offers faster execution because it is implemented in C. It makes use of parallel processing capabilities and well optimized numerical algorithms to speed up numerical

K. PRAKASH SENAPATI

computations. Its memory management guarantees improved cache locality and utilization. The combination of NumPy with compiled languages also improves performance for computationally demanding jobs.

63.

If the variable is an array, you can use the size attribute to determine whether the array is empty. The variable might, however, be of the list or sequence type in which case len() can be used. The size attribute is the best technique to look for a zero element.

```
>>> a = NumPy.zeros((1,0))
>>> a.size
0
```

whereas

```
>>> len(a)
1
```

64.

The bincount() function can be used to count the instances of a given value. It should be noted that the bincount() function takes boolean expressions or positive integers as arguments. Integers that are negative cannot be used.

Implement NumPy.bincount().

```
arr = NumPy.array([0, 5, 4, 0, 4, 4, 3, 0, 0, 5, 2, 1, 1, 9])
NumPy.bincount(arr)
```

65.

The sort function in NumPy can be used to sort an array. This function arranges the members of an array in ascending order while altering the array already in use. As a result, it sorts the array itself rather than returning a new sorted array.

```
import numpy as np
# Create an unsorted array
arr = np.array([3, 2, 1])
# Sort the array
arr.sort()
# Print the sorted array
print(arr)
```

66.

Using the max and min functions, you may determine an array's maximum or minimum value in NumPy. These operations accept an array as an input and output the array's maximum or minimum value.

```
import numpy as np
# Create an array
```

K. PRAKASH SENAPATI

```
arr = np.array([3, 2, 1])
# Find the maximum value of the array
max_value = np.max(arr)
# Find the minimum value of the array
min_value = np.min(arr)
# Print the maximum and minimum values
print(max_value)
print(min_value)
```

67.

In this example, negative values are located and replaced with zeros using indexing, and a new array with more than two members is created using slicing. Both indexing and slicing are useful methods for cleaning data because they let you modify or filter data based on particular criteria or target particular data points for modification.

```
import numpy as np

# Sample NumPy array
data = np.array([1, 2, -1, 4, 5, -2, 7])

# Indexing: Replace negative values with zeros
data[data < 0] = 0

# Slicing: Extract elements greater than 2
subset = data[data > 2]
```

68.

A tuple that describes the size of the array along each dimension is the shape attribute of a NumPy array. For instance, an array of the shape (3, 4) has 3 rows and 4 columns. The array's dimensions can be found by using the form attribute, or the array can be reshaped by altering the sizes of the individual dimensions.

```
import numpy as np
# Create a NumPy array
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
# Get the shape of the array
shape = arr.shape
print(shape) # Output: (3, 4)
# Access the individual dimensions
num_rows = shape[0]
num_cols = shape[1]
print(num_rows)
print(num_cols)
```

K. PRAKASH SENAPATI

A NumPy array's size attribute gives the total number of elements in the array. This is just the sum of the array's various dimensions' sizes. For instance, an array with the shape (3, 4) has 3 * 4 = 12 elements overall.

```
import numpy as np
# Create a NumPy array
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
# Get the size of the array
size = arr.size
print(size) # Output: 12
# Calculate the size manually
num_rows = arr.shape[0]
num_cols = arr.shape[1]
size = num_rows * num_cols
print(size)
```

69.

In a contiguous block of memory, NumPy arrays are data structures that hold values of the same data type. Similar to Python lists, they may hold values of any data type and are more effective for some tasks. Shape, size, and dtype are a few useful properties of umPy arrays. The array's size along each dimension is specified by a tuple that the shape attribute returns. The array's total number of elements is returned by the size attribute. The data type of each element in the array is returned by the dtype property.

- A additional data type called a matrix, a subclass of the array data type, is also included in NumPy. Similar to a NumPy array, a NumPy matrix includes a few more properties that make it more practical for linear algebra operations.
- Arrays employ the element-wise * operator, whereas matrices contain a distinct * operator for matrix multiplication.
- Additionally, matrices provide attributes for inverse and transpose, respectively.

70.

Apply the unique function from the NumPy module to identify the unique elements in an array in NumPy. This function returns the array's unique elements in sorted order, along with a count of how many times each element occurred.

```
import numpy as np
array = np.array([1, 2, 3, 1, 2, 3, 3, 4, 5, 6, 7, 5])
unique, counts = np.unique(array, return_counts=True)
print(unique)
print(counts)
```

The unique items of the input array array are contained in this example's output arrays unique and counts, along with a count of each element.

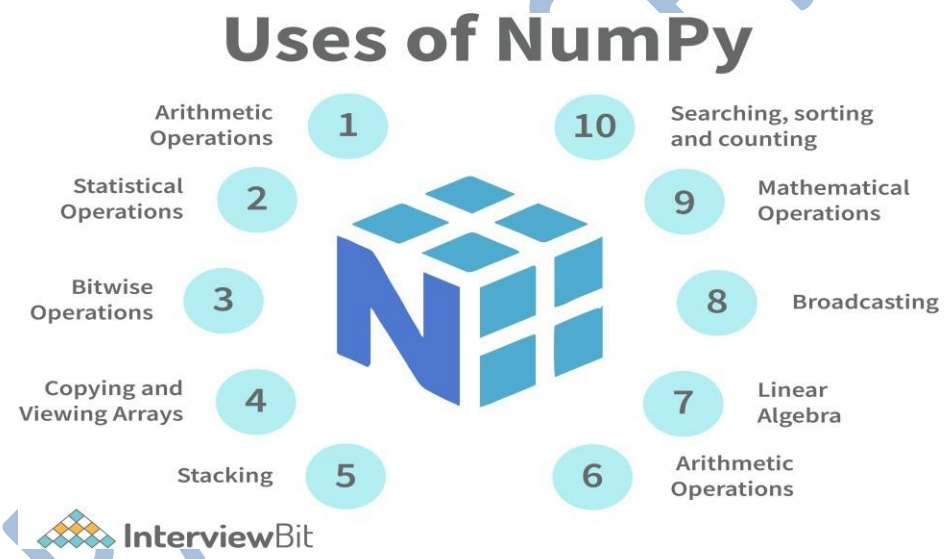
K. PRAKASH SENAPATI

Section B

Answers

B.1

NumPy (also called Numerical Python) is a highly flexible, optimized, open-source package meant for array processing. It provides tools for delivering high-end performance while dealing with N-dimensional powerful array objects. It is also beneficial for performing scientific computations, mathematical, and logical operations, sorting operations, I/O functions, basic statistical and linear algebra-based operations along with random simulation and broadcasting functionalities. Due to the vast range of capabilities, NumPy has become very popular and is the most preferred package. The following image represents the uses of NumPy.



B.2

The to_numpy() method of the NumPy package can be used to convert Pandas DataFrame, Index and Series objects.Consider we have a DataFrame df, we can either convert the whole Pandas DataFrame df to NumPy array or even select a subset of Pandas DataFrame to NumPy array by using the to_numpy() method as shown in the example below:

CODE:

```
import pandas as pd
```

K.PRAKASH SENAPATI

```
import numpy as np
# Pandas DataFrame
df = pd.DataFrame(data={'A': [3, 2, 1], 'B': [6,5,4], 'C': [9, 8, 7]},
            index=['i', 'j', 'k'])
print("Pandas DataFrame: ")
print(df)
# Convert Pandas DataFrame to NumPy Array
np_arr = df.to_numpy()
print("Pandas DataFrame to NumPy array: ")
print(np_arr)
# Convert specific columns of Pandas DataFrame to NumPy array
arr = df[['B', 'C']].to_numpy()
print("Convert B and C columns of Pandas DataFrame to NumPy Array: ")
print (arr)
```

The output of the above code is:

```
Pandas DataFrame:
  A B C
i 3 6 9
j 2 5 8
k 1 4 7
Pandas DataFrame to NumPy array:
[[3 6 9]
 [2 5 8]
 [1 4 7]]
Convert B and C columns of Pandas DataFrame to NumPy Array:
[[6 9]
 [5 8]
 [4 7]]
```

K. PRAKASH SENAPATI

B:3

Concatenating 2 arrays by adding elements to the end can be achieved by making use of the concatenate() method of the NumPy package. Syntax:

np.concatenate((a1, a2, ...), axis=0, out=None)

where,

- a1,a2: arrays of the same shape
- axis:Represents the axis along which the arrays are joined. The default value is 0.
- out: If mentioned, it specifies the destination for placing the result.

For example:

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6]])

# Concatenate with axis 0
c = np.concatenate((a,b), axis=0)
print("With axis 0: \n",c )

# Concatenate with axis 1 (b.T represents transpose matrix)
d = np.concatenate((a,b.T), axis=1)
print("With axis 1: \n",d )
```

The output would be:

```
With axis 0:
 [[1 2]
 [3 4]
 [5 6]]
With axis 1:
 [[1 2 5]
 [3 4 6]]
```

Notice how the arrays are concatenated with different values of the axis.

B.4

We can make use of the dot() for multiplying matrices represented as NumPy arrays. This is represented in the code snippet below:

```
import numpy as np

# NumPy matrices
A = np.arange(15,24).reshape(3,3)
B = np.arange(20,29).reshape(3,3)
print("A: ",A)
print("B: ",B)

# Multiply A and B
result = A.dot(B)
print("Result: ", result)
```

Output

```
A:  [[15 16 17]
 [18 19 20]
 [21 22 23]]
B:  [[20 21 22]
 [23 24 25]
 [26 27 28]]
Result:  [[1110 1158 1206]
 [1317 1374 1431]
 [1524 1590 1656]]
```

B.5:

arr[:,0] - Returns 0th index elements of all rows. In other words, return the first column elements.

CODE

```
import numpy as np
arr  = np.array([[1,2,3,4],[5,6,7,8]])
new_arr =arr[:,0]
print(new_arr)
```

Output:

```
[1 5]
```

arr[:,[0]] - This returns the elements of the first column by adding extra dimension to it.

K. PRAKASH SENAPATI

CODE

```
import numpy as np
arr  =  np.array([[1,2,3,4],[5,6,7,8]])
new_arr =arr[:,[0]]
print(new_arr)
```

Output:

```
[[1]
[5]]
```

B.6:-

We can check for the emptiness of a NumPy array by making use of the size attribute. Let us consider the below example. We have NumPy array arr filled with zeros. If the size element returns zero, that means the array is empty or it only consists of zeros.

CODE

```
import numpy as np
arr = np.zeros((1,0))  #returns empty array
print(arr.size)       #returns 0
```

This return 0

B.7:-

We can make use of the bincount() function to compute the number of times a given value is there in the array. This function accepts only positive integers and boolean expressions as the arguments.

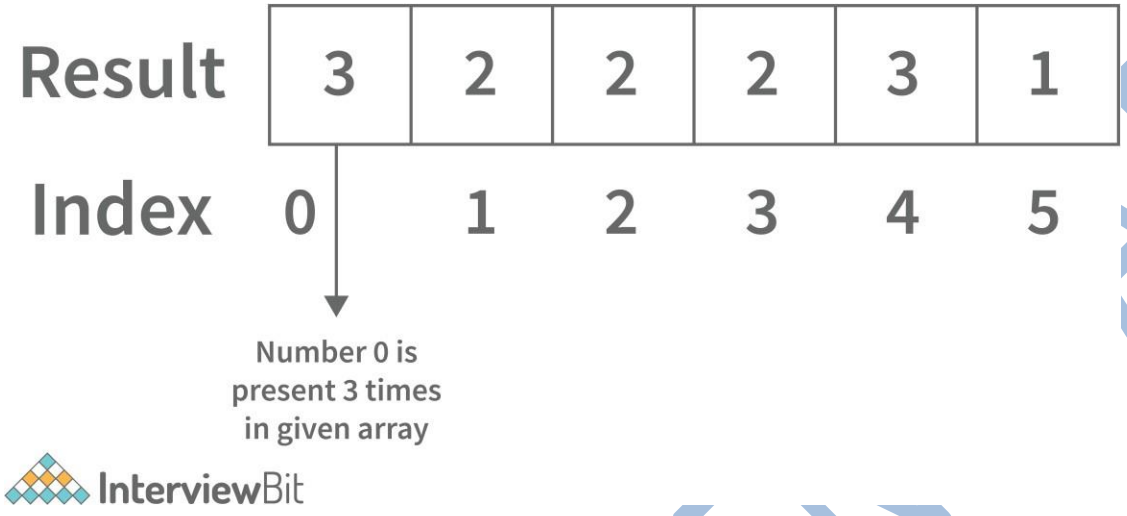
Explain

```
import numpy as np
arr = np.array([1, 2, 1, 3, 5, 0, 0, 0, 2, 3])
result = np.bincount(arr)
print(result)
```

The result is: [3 2 2 2 0 1]

K. PRAKASH SENAPATI

It has to be noted here that each element represents the count of the corresponding index value present in the original array. This is demonstrated in the below image:



B.8-

- np.mean() method calculates the arithmetic mean and provides additional options for input and results. For example, it has the option to specify what data types have to be taken, where the result has to be placed etc.
- np.average() computes the weighted average if the weights parameter is specified. In the case of weighted average, instead of considering that each data point is contributing equally to the final average, it considers that some data points have more weightage than the others (unequal contribution).

B.G-

There are two ways of reversing a NumPy array.

- **Method 1:** Using the slicing method: We can make use of [::-1] for reversing the array. The following example demonstrates this:

CODE:

```
import numpy as np
# create numpy array
arr = np.array([1, 2, 4, 6])
# To reverse array
```

```
reverse_arr = arr[::-1]
print(reverse_arr)
```

Output:

[6 4 2 1]

- **Method 2:** flipud function: This function is provided by NumPy to reverse the NumPy array. Let us see the below example about its usage.

Explain

```
import numpy as np
# create numpy array
arr = np.array([1, 2, 4, 5, 6])
#flipud method for reversing
reverse_arr = np.flipud(arr)
print(reverse_arr)
```

Output:

[6 5 4 2 1]

B.10-

NumPy supports the following datatypes:

- i - integer
- S - string
- b - boolean
- f - float
- u - unsigned integer
- c - complex float
- m - timedelta
- M - datetime
- O - object
- U - unicode string

K. PRAKASH SENAPATI

- V - fixed memory chunk for types such as void
  We can make use of the dtype property that returns the type of the elements stored in the NumPy array. Let us consider the below code snippet. We create some sample arrays and we see what the data types of these arrays are.

CODE:

```
import numpy as np
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array(['I', 'love', 'Interviewbit'])  # Stored as Unicode characters with length of characters ranging from 1 to 12
arr3 = np.array([1, 2, 3, 4], dtype='S') # Creating numpy array of defined type string
print(arr1.dtype)
print(arr2.dtype)
print(arr3.dtype)
```

The output will be:

int64

<U12

|S1

B.11

Consider you have a normal python list. From this, we can create NumPy arrays by making use of the array function as follows:

- One-Dimensional array

```
import numpy as np
arr = [1,2,3,4]        #python list
numpy_arr = np.array(arr)   #numpy array
```

- Two-Dimensional array

```
import numpy as np
arr = [[1,2,3,4],[4,5,6,7]]
```

K. PRAKASH SENAPATI

```
numpy_arr = np.array(arr)
```

- Three-Dimensional array

```
import numpy as np
arr = [[[1,2,3,4],[4,5,6,7],[7,8,9,10]]]
numpy_arr = np.array(arr)
```

Using the np.array() function, we can create NumPy arrays of any dimensions.

B.12-

ndarray object is the core of the NumPy package. It consists of n-dimensional arrays storing elements of the same data types and also has many operations that are done in compiled code for optimised performance. These arrays have fixed sizes defined at the time of creation. Following are some of the properties of ndarrays:

- When the size of ndarrays is changed, it results in a new array and the original array is deleted.
- The ndarrays are bound to store homogeneous data.
- They provide functions to perform advanced mathematical operations in an efficient manner.

B.13-

- Python lists support storing heterogeneous data types whereas NumPy arrays can store datatypes of one nature itself. NumPy provides extra functional capabilities that make operating on its arrays easier which makes NumPy array advantageous in comparison to Python lists as those functions cannot be operated on heterogeneous data.
- NumPy arrays are treated as objects which results in minimal memory usage. Since Python keeps track of objects by creating or deleting them based on the requirements, NumPy objects are also treated the same way. This results in lesser memory wastage.
- NumPy arrays support multi-dimensional arrays.
- NumPy provides various powerful and efficient functions for complex computations on the arrays.

K. PRAKASH SENAPATI

- NumPy also provides various range of functions for BitWise Operations, String Operations, Linear Algebraic operations, Arithmetic operations etc. These are not provided on Python's default lists.

B.14

NumPy is an open-source, high-performing library that allows complex mathematical and scientific computational capabilities. It makes use of Python language which is a high-level, easy-to-learn, general-purpose programming language. It supports the following:

- Powerful functions for performing complex mathematical operations on multi-dimensional matrices and arrays. The operations on ndarrays of NumPy are approximately up to 50% faster when compared to operations on native lists using loops. This efficiency is very much useful when the arrays have millions of elements.
- Provides indexing syntax to access portions of data easily in a large array.
- Provides built-in functions which help to easily perform operations related to linear algebra and statistics.
- It takes only a few lines of code to achieve complex computations using NumPy.

K. PRAKASH SENAPATI

NumPY Interview Questions for Experienced

B.15-

Both methods are used for combining the NumPy arrays. The main difference is that the hstack method combines arrays horizontally whereas the vstack method combines arrays vertically.
For example, consider the below code.

Explain

```
import numpy as np
a = np.array([1,2,3])
b = np.array([4,5,6])
# vstack arrays
c = np.vstack((a,b))
print("After vstack: \n",c)
# hstack arrays
d = np.hstack((a,b))
print("After hstack: \n",d)
```

The output of this code would be:

After vstack:

```
[[1 2 3]
[4 5 6]]
```

After hstack:

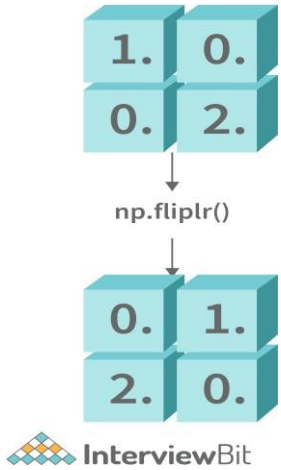
```
[1 2 3 4 5 6]
```

Notice how after the vstack method, the arrays were combined vertically along the column and how after the hstack method, the arrays were combined horizontally along the row.

K. PRAKASH SENAPATI

B.16

The fliplr() method is used for flipping an array in the left or right direction. The columns are preserved but the order of elements in the columns would be different than before. This has been represented in the image below:



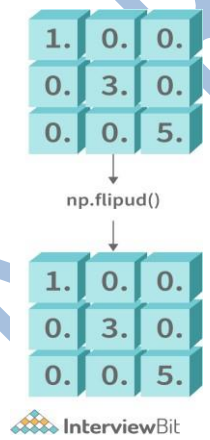
We see that the positions of the elements are flipped left or right than their original position in the result.

Syntax of fliplr:

np.fliplr(arr)

where arr is the array that has to be flipped.

The flipud function also flips the array but in the up or down direction. The rows are preserved in this case but they can appear in a different order in the result. This is represented in the image below:



Here, we see that the numbers 1, 3 and 5 elements are flipped in the up/down direction in the result.

K. PRAKASH SENAPATI

The syntax for flipud:

np.flipud(arr)

where arr is the array that has to be flipped.

B.17-

We can make use of the convolve() method. Here, it leverages the way discrete convolution is computed and uses it to find the rolling mean (moving average). Here, the sequence of ones of length equal to the length of the sliding window is convolved with the array.

We first define a calculate_moving_average function which performs the convolution of an array with the sequence of ones of sliding window length w. The mode of the convolve method will be 'valid' to generate the points only where the overlapping of the sequence is complete.

```
import numpy as np

def calculate_moving_average(arr, w):

  return np.convolve(arr, np.ones(w),'valid')/w
```

The above-defined function can be then used for finding the moving average as shown in the examples below:

CODE:

```
arr1  =  np.array([4,5,8,9,3,2,4,2,0,2])
print("Moving average of window length 2: ")
av1 = calculate_moving_average(arr1, 2)
print(av1)
print("Moving average of window length 4: ")
av2 = calculate_moving_average(arr1, 4)
print(av2)
```

Output:

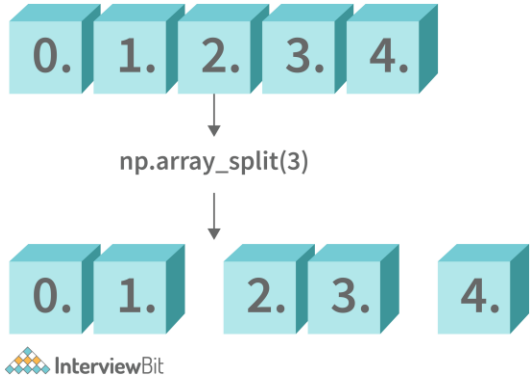
Moving average of window length 2: [4.5 6.5 8.5 6. 2.5 3. 3. 1. 1.]

Moving average of window length 4: [6.5 6.25 5.5 4.5 2.75 2. 2.]

K. PRAKASH SENAPATI

B.18-

The array_split() method is similar to the split() method as it helps in splitting a given array to multiple subarrays. The main difference is that the array_split() allows sections to be an integer which does not result in equal array division. For an array of length L, if we want it to split to N subarrays, then L % N subarrays of size (L//N + 1) and remaining subarrays are of size L//N.



In the above figure, we see there are 5 elements in the array, we want to split the array to 3 subarrays. So L % N = 5%3 = 2 subarrays of size (L//N +1) = (5//3 +1) = 2 are returned and remaining 1 subarray of size L//N = 1 is returned.

Syntax:

np.array_split(array, sections, axis=0)

where,

- array - Given Input array.
- sections - List of indices or Number of subarrays to be returned.
- axis - Axis along which values have to be appended.

The code for the example illustrated above is:

CODE:

```
import numpy as np
arr = np.arange(5.0)
split_arrs = np.array_split(arr, 3)
split_arrs
```

Output:

```
[array([0., 1.]), array([2., 3.]), array([4.])]
```

K. PRAKASH SENAPATI

B.19

1. np.split() : Equally splits arrays into multiple sub-arrays. It raises Value Error when the split cannot be equal.

- Syntax:

np.split(array, sections, axis=0)

where,

- array - array that needs to be split
- sections -
    - If we give an integer X, X equal sub-arrays are obtained after dividing the array. If the split is not possible, ValueError is raised.
        - For example:

CODE;

```
import numpy as np
a = np.arange(8)
split_arr = np.split(a, 2)
split_arr
```

Output

```
[array([0, 1, 2, 3]), array([4, 5, 6, 7])]
```

- If we give a 1-D sorted array then the entries would represent where the array would be split along the axis. For instance if we provide [2:3] and axis as 0, then the result would be
  [arr[0:2], arr[2:3], arr[3:]]
    - If the provided index exceeds the array dimension along the given axis, then an empty subarray will be returned.
    - For example:

```
[array([0., 1., 2.]),
 array([3.]),
 array([4.]),
 array([5.]),
 array([],  dtype=float64),
 array([],  dtype=float64),
 array([], dtype=float64)]
```

K. PRAKASH SENAPATI

The output would be:

CODE:

```
import numpy as np
a = np.arange(6.0)
split_arr = np.split(a, [3, 4, 5, 6, 7,8])
split_arr
```

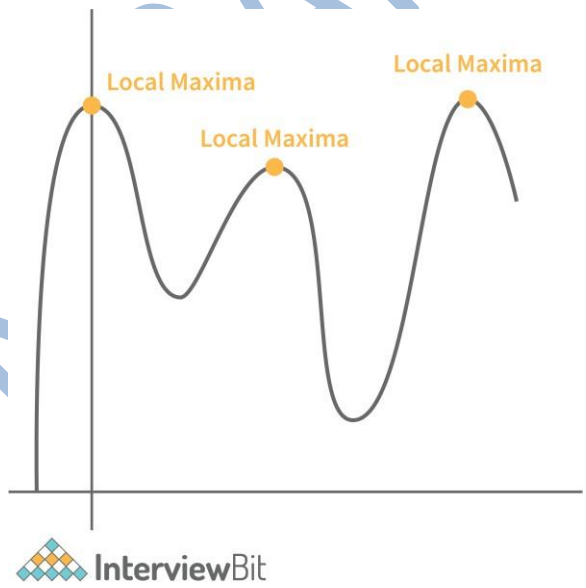
- axis - Along what axis the array has to be split. By default, the value is 0

B.20-

Vectorization involves delegating NumPy operations internally to optimized C language functions to result in faster Python code. Whereas Broadcasting refers to the methods that allow NumPy to perform array-related arithmetic operations. The size or shape of the arrays does not matter in this case. Broadcasting solves the problem of mismatched shaped arrays by replicating the smaller array along the larger array to ensure both arrays are having compatible shapes for NumPy operations. Performing Broadcasting before Vectorization helps to vectorize operations which support arrays of different dimensions.

B.21-

Peaks are the points that are surrounded by smaller value points on either side as shown in the image below:



K. PRAKASH SENAPATI

There are two ways of finding local maxima:

Using .where() method: This method lists all positions/indices where the element value at position i is greater than the element on either side of it. This method does not check for the points that have only one neighbour. This is demonstrated in the example below:

CODE:

```
import numpy as np
# define NumPy array
arr = np.array([1, 4, 8, 1, 3, 5, 1, 6, 1, -5, -1, 19, 2])
maxima_peaks_positions = np.where((arr[1:-1] > arr[0:-2]) * (arr[1:-1] > arr[2:]))[0] + 1
print(maxima_peaks_positions)
```

Output:

```
[ 2 5 7 11]]
```

- The +1 at the end of the expression is required as it finds the indexes within the slice arr[1:-1] and not the entire array arr.
- The where() method returns a tuple of arrays where the first element is our required array. Hence we add [0] after the where method.

Using combination of .diff(), .sign() and .where() method:

- In this method, we calculate the difference between each element using the diff() method of NumPy.
- Then we use the sign() method on the array to get the sign of difference.
- The value can be either -1 or +1. This result is then passed on to another diff() method which returns 0, -2 or +2 value. The value 0 indicates that the points are continuously increasing or decreasing, +2 indicates minimum peak and -2 indicates maximum peak (local maxima).
- We then identify the position or indexes of the local maxima using the where() method. The reason for using +1 at the end of where and [0] after where is the same as the explanation described in Method 1 for finding local maxima.

The following code example demonstrates this:

CODE:

```
import numpy as np
# define NumPy array
arr = np.array([1, 4, 8, 1, 3, 5, 1, 6, 1, -5, -1, 19, 2])
all_peaks = np.diff(np.sign(np.diff(arr)))
```

K. PRAKASH SENAPATI

```
maxima_peaks_positions = np.where(all_peaks == -2)[0] + 1
print(maxima_peaks_positions)
```

Output:

```
[ 2 5 7 11]]
```

B.22-

Function Vectorization technically means that the function is applied to all elements in the array. Typically, certain python functionalities on arrays (such as loops) are slower in nature because python arrays can contain elements of different data types. Since the C program expects a specific datatype, there are chances of compiler optimisation which makes C code run faster. Since NumPy arrays support storing elements of a single datatype, most of the implementations of the functions written in NumPy meant for arithmetic, logical operations etc have optimised C program code under their hood. Additionally, NumPy also helps developers create their own vectorised functions by following the below steps:

- Write your required function that takes array elements as parameters.
- Vectorize the function by making use of the vectorize() method of the NumPy package.
- Give array inputs to the vectorized function.

The below example demonstrates the process of vectorization.

CODE:

```
import numpy as np
# Define your function
def add(arr1, arr2):
  return (arr1 + arr2)
arr1 = np.array([1,2,3])
arr2 = np.array([4,5,6])
#vectorize add method
vectorized_add = np.vectorize(add)
#call vectorized method
result = vectorized_add(arr1, arr2)
print(result)
```

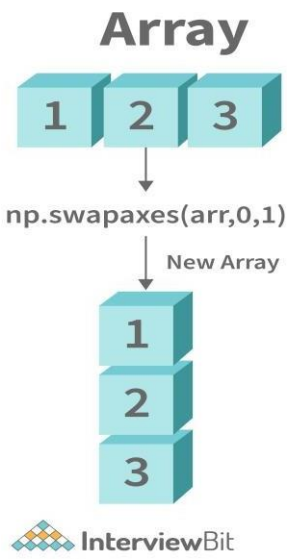
K. PRAKASH SENAPATI

The output of above code:

[5 7 9]

B.23-

This can be achieved by using the swapaxes method of NumPy. The below image illustrates the meaning of swapping axes.



CODE:

```
import numpy as np
arr = np.array([[1,2,3]])
print("Original array: ")
print(arr)
#Swap axes
axis_swapped_arr = np.swapaxes(arr,0,1)
print("Transformed array: ")
print(axis_swapped_arr)
```

Output:

Original array:

[[1 2 3]]

Transformed array:

[[1]

K. PRAKASH SENAPATI

[2]

[3]]

Conclusion

The popularity of the NumPy package has grown immensely among the data science and python developers community ever since it was first introduced in 2005 due to the wide range of high-performing functionalities it offers. This is why it becomes essential to learn and be prepared for the interview questions about this package. In this article, we have seen the most commonly asked NumPy interview questions for freshers and experienced people, along with some questions on writing python programs which make use of NumPy functions.

B.24-

We can achieve this by overriding the shape attribute of the NumPy array.

Sample Solution:

```
import numpy as np
#Create NumPy array
arr = np.array([1,2,3,4,5,6,7,8,9])
print("Original Shape: ", arr.shape)
# Change the shape/dimension of the array
arr.shape = (3, 3)
print("Transformed Matrix :")
print(arr)
print("Transformed Shape: ",arr.shape)
```

Output:

```
Original Shape:  (9,)
Transformed Matrix :
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Transformed Shape: (3, 3)
```

K. PRAKASH SENAPATI

In this approach, care has to be taken w.r.t the number of elements present in the original array before changing the dimensions. Otherwise, it will result in the ValueError as shown below:

Explain

```
import numpy as np
# We have array of 8 elements
arr = np.array([1,2,3,4,5,6,7,8])
# We are trying to convert the 1D array to a 3D array which expects 9 elements
arr.shape = (3, 3)
print(arr)
```

Running this code would result in:

Explain

```
   1 import numpy as np
   2 arr = np.array([1,2,3,4,5,6,7,8])
----> 3 arr.shape = (3, 3)
   4 print(arr)
```

ValueError: cannot reshape array of size 8 into shape (3,3)

B.25-

For example,
If you have the below array:

```
[[ 1. 1. 1. 1.]
 [ 1. 1. 1. 1.]
 [ 1. 1. 1. 1.]]
```

The resultant array should be: (zeros on the border and 1s within it)

Explain

```
[[ 0. 0. 0. 0. 0. 0.]
 [ 0. 1. 1. 1. 1. 0.]
```

K. PRAKASH SENAPATI

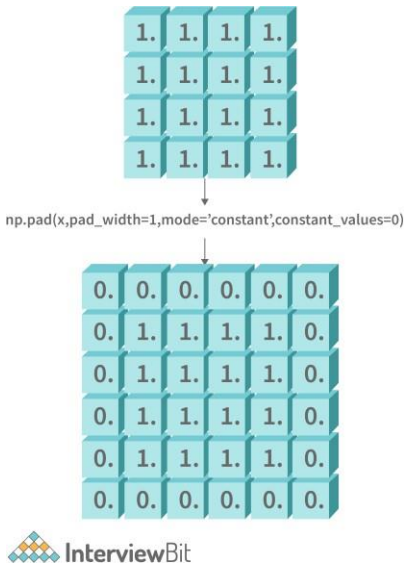
[0. 1. 1. 1. 1. 0.]

[0. 1. 1. 1. 1. 0.]

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

Solution:-

This can be achieved by using the pad method of the NumPy library.



CODE:

```
import numpy as np

# Create NumPy arrays filled with ones

ones_arr = np.ones((4,4))

print("Transformed array:")

transformed_array = np.pad(ones_arr, pad_width=1, mode='constant',
constant_values=0)

print(transformed_array)
```

Output:

Transformed array:

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

[0. 1. 1. 1. 1. 0.]

[0. 1. 1. 1. 1. 0.]

[0. 1. 1. 1. 1. 0.]

K. PRAKASH SENAPATI

```
[0. 1. 1. 1. 1. 0.]
[0. 0. 0. 0. 0. 0.]]
```

B.26-

```
import numpy as np
arr = np.arange(10, 60)
print(arr)
```

Output:

```
[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
 58 59]
```

B.27-

Sample Solution:-

```
import numpy as np
# Create Sample NumPy Array
arr = np.array(['i', 'love', 'NumPy', 'AND', 'interviewbit'], dtype=str)
transformed_array = np.char.multiply(arr, 5)
print("Transformed array:")
print(transformed_array)
```

Output:

```
Transformed array:
['iiiii' 'lovelovelovelovelove' 'NumPyNumPyNumPyNumPyNumPy'
 'ANDANDANDANDAND'
 'interviewbitinterviewbitinterviewbitinterviewbitinterviewbit']
```

K. PRAKASH SENAPATI

B.28-

Sample Solution:-

```
import numpy as np
# Create Sample NumPy Array
arr = np.array(['i', 'love', 'NumPy', 'AND', 'interviewbit'], dtype=str)
transformed_arr = np.char.join(" ", arr)
print("Transformed Array: ")
print(transformed_arr)
```

Output:

Transformed Array:

['i' 'l o v e' 'N u m P y' 'A N D' 'i n t e r v i e w b i t']

B.2G-

Sample Solution:-

```
import numpy as np
# Create Sample NumPy array
arr = np.array(['22', '9', '1234', '567', '89102'], dtype=str)
zeroes_filled_arr = np.char.zfill(arr, 8)
print("Transformed array: ")
print(zeroes_filled_arr)
```

Output:

Transformed array:

['00000022' '00000009' '00001234' '00000567' '00089102']

K. PRAKASH SENAPATI

B.30-

Sample Solution:-

```
import numpy as np
# Create Sample NumPy array
arr = np.array(['i', 'love', 'NumPy', 'AND', 'interviewbit'], dtype=str)
upper_case_arr = np.char.upper(arr)
lower_case_arr = np.char.lower(arr)
capitalize_case_arr = np.char.capitalize(arr)
titlecase_arr = np.char.title(arr)
swapcase_arr = np.char.swapcase(arr)
print("Upper Conversion: ", upper_case_arr)
print("Lower Conversion: ", lower_case_arr)
print("Capitalize First Letter Conversion: ", capitalize_case_arr)
print("Titlecase Conversion: ", titlecase_arr)
print("Swapcase Conversion: ", swapcase_arr)
```

Output:

```
Upper Conversion:  ['I' 'LOVE' 'NUMPY' 'AND' 'INTERVIEWBIT']
Lower Conversion:  ['i' 'love' 'numpy' 'and' 'interviewbit']
Capitalize First Letter Conversion:  ['I' 'Love' 'Numpy' 'And' 'Interviewbit']
Titlecase Conversion:  ['I' 'Love' 'Numpy' 'And' 'Interviewbit']
Swapcase Conversion:  ['I' 'LOVE' 'nUMpY' 'and' 'INTERVIEWBIT']
```

K. PRAKASH SENAPATI

SECTION C:

Commonly Asked Interview Answers on NumPy

C.1. Answer:

- **NumPy**: NumPy is a powerful Python library for numerical and matrix operations. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays efficiently.
- **Popularity**: NumPy is popular due to its ease of use, high performance, and integration with other scientific computing libraries. It simplifies complex mathematical operations and enhances the performance of numerical computations in Python.

C.2. Answer:

Differences:

- NumPy arrays are homogeneous and contain elements of the same data type, whereas Python lists can contain elements of different data types.
- NumPy arrays provide more functionality for mathematical operations and array manipulations compared to Python lists.

Advantages:

- NumPy arrays are more memory-efficient and faster than Python lists for numerical operations.
- NumPy provides a wide range of functions for array manipulation, linear algebra, statistical operations, and random number generation.

C.3. Answer:

- **Broadcasting** is a feature in NumPy that allows operations on arrays of different shapes and sizes. In cases where dimensions are compatible, NumPy automatically stretches or "broadcasts" the smaller array to match the shape of the larger one.

Python Example:

```
Import numpy as np
A = np.array([[1,2,3,], [4,5,6]])
B = np.array([10,20,30])
 Result = A +B # Broadcasting the smaller array B to match the shape of A
```

C.4. Answer:

- The **np.arange()** function creates an array with regularly spaced values within a specified range. It is similar to Python's built-in range() function but returns a NumPy array.

Python Example:

```
import numpy as np
arr = np.arange(1, 10, 2)  # Creates an array from 1 to 10 (exclusive) with a step of 2
```

C.5. Answer:

- To find the dimensions, use the ndim attribute: **array.ndim**.
- To find the shape, use the shape attribute: **array.shape**.

Python Example:

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
dimensions = arr.ndim  # Returns 2
shape = arr.shape  # Returns (2, 3)
```

K. PRAKASH SENAPATI

C.6. Answer:

- **np.zeros(shape)**: Creates an array filled with zeros of the specified shape.
- **np.ones(shape)**: Creates an array filled with ones of the specified shape.

Python Example:

```
import numpy as np
zeros_array = np.zeros((3, 4))  # Creates a 3×4 array filled with zeros
ones_array = np.ones((2, 2))    # Creates a 2×2 array filled with ones
```

C.7. Answer:

Element-wise multiplication can be performed using the * operator or the np.multiply() function.

Python Example:

```
import numpy as np
A = np.array([1, 2, 3])
B = np.array([4, 5, 6])
result = A * B  # Element-wise multiplication using the * operator
# Alternatively:
result = np.multiply(A, B)  # Using np.multiply() function
```

C.8. Answer:

- **Universal functions (ufuncs)** in NumPy are functions that operate element-wise on arrays, performing fast vectorized operations. These functions are optimized for performance and can significantly improve the efficiency of numerical computations.

Python Example:

```
import numpy as np
arr = np.array([1, 2, 3])
# Using a ufunc to calculate the square root element-wise
result = np.sqrt(arr).
```

C.9. Answer:

- Horizontal Concatenation: Use **np.hstack((arr1, arr2))**.
- Vertical Concatenation: Use **np.vstack((arr1, arr2))**.

Python Example:

```
import numpy as np
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6]])
# Horizontal Concatenation
result_horizontal = np.hstack((A, B))
# Vertical Concatenation
result_vertical = np.vstack((A, B))
```

C.10. Answer:

- The **np.linalg.inv()** function in NumPy is used to compute the (multiplicative) inverse of a square matrix. It is commonly employed in linear algebra for solving systems of linear equations.

Python Example:

```
import numpy as np
A = np.array([[1, 2], [3, 4]])
```

K. PRAKASH SENAPATI

```
# Compute the inverse of matrix A
A_inv = np.linalg.inv(A)
```

These questions cover various aspects of NumPy, including array operations, broadcasting, universal functions, and linear algebra functions.

Technical Questions Asked in NumPy Interviews:

C.11. Answer:

- **array:** Represents a general n-dimensional array and is more versatile.
- **matrix:** Represents a specialized 2-dimensional matrix, but its use is discouraged in favor of np.array.

Preference:

- Prefer np.array for general-purpose numerical operations as it is more commonly used and supported.

C.12. Answer:

- The np.reshape() function is used to change the shape of an array without changing its data. It returns a new array with the specified shape.

Python Example:

```
import numpy as np
arr = np.arange(1, 10)
reshaped_arr = np.reshape(arr, (3, 3))
```

C.13. Answer:

- np.newaxis is used to increase the dimension of an existing array by one more dimension. It is often used to convert a 1-dimensional array into a 2-dimensional column or row vector.

Python Example:

```
import numpy as np
arr = np.array([1, 2, 3])
column_vector = arr[:, np.newaxis]  # Convert to a column vector
```

C.14. Answer:

- NumPy broadcasting rules apply when performing operations on arrays of different shapes. Broadcasting allows NumPy to operate on these arrays without the need for explicit expansion.

Python Example:

```
import numpy as np
A = np.array([[1, 2, 3], [4, 5, 6]])
B = np.array([10, 20, 30])
result = A + B  # Broadcasting the smaller array B to match the shape of A
```

C.15. Answer:

- **np.linspace(start, stop, num)** returns evenly spaced numbers over a specified range. It is often used to generate sequences for plotting.

Python Example:

```
import numpy as np
sequence = np.linspace(1, 10, 5)  # Creates an array with 5 evenly spaced values from 1 to 10
```

K. PRAKASH SENAPATI

C.16. Answer:

- The np.random module provides functions for generating random numbers and distributions.

Python Example:

```
import numpy as np
random_numbers = np.random.rand(3, 3)  # Generates a 3×3 array of random numbers
between 0 and 1
```

C.17. Answer:

- The dot product of two arrays can be calculated using np.dot() or the @ operator. The dot product is significant in linear algebra for finding the projection of one vector onto another.

Python Example:

```
import numpy as np
A = np.array([1, 2, 3])
B = np.array([4, 5, 6])
dot_product = np.dot(A, B)
```

C.18. Answer:

- **np.save(file, arr):** Saves an array to a binary file in NumPy .npy format.
- **np.load(file):** Loads an array from a NumPy .npy file.

Python Example:

```
import numpy as np
arr = np.array([1, 2, 3])
np.save('saved_array.npy', arr)
loaded_array = np.load('saved_array.npy')
```

C.19. Answer:

- **np.vectorize()** is used to create a vectorized function from a non-vectorized function, allowing it to operate element-wise on arrays.

Python Example:

```
import numpy as np
def square(x):
return x ** 2
vectorized_square = np.vectorize(square)
arr = np.array([1, 2, 3])
result = vectorized_square(arr)
"
```

Conceptual Interview Questions:

C.20. Answer:

- **Concept:** A universal function (ufunc) in NumPy is a function that operates element-wise on NumPy arrays, allowing for fast and efficient vectorized operations.

Python Example:

```
import numpy as np
arr = np.array([1, 2, 3])
```

K. PRAKASH SENAPATI

```
# Ufunc example: Square root
result = np.sqrt(arr)
```

Significance: Ufuncs enhance the performance of numerical operations by applying functions to entire arrays without the need for explicit looping.

C.21. Answer:

Advantages:

- NumPy arrays are homogeneous and can efficiently store large datasets.
- NumPy provides optimized, vectorized operations, resulting in faster numerical computations.
- Broadcasting allows for seamless operations on arrays of different shapes.

C.22. Answer:

- Purpose: The dtype attribute specifies the data type of elements in a NumPy array.
- Influence: The dtype determines the size of each element in memory, influencing the overall memory consumption of the array.

C.23. Answer:

- Shallow Copy: A shallow copy creates a new array, but the data is still shared with the original array. Changes in one array affect the other.
- Deep Copy: A deep copy creates a new array with a copy of the data, ensuring that changes in one array do not affect the other.

C.24. Answer:

- **Purpose:** The axis parameter specifies the axis along which an operation is performed.

Python Example:

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
# Sum along axis 0 (columns)
column_sum = np.sum(arr, axis=0)
```

C.25. Answer:

- **Role:** The ufunc.reduce() method repeatedly applies a binary operation to the elements of an array, reducing it to a single result.

Python Example:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
# Example of ufunc.reduce(): Multiplication
result = np.multiply.reduce(arr)
```

C.26. Answer:

- **Concept:** Slicing involves extracting a portion of an array by specifying a range of indices along each axis.
- **Difference:** Slicing returns a view of the original array, while indexing returns a copy. Slicing allows for efficient manipulation of array subsets without copying data.

C.27. Answer:

K. PRAKASH SENAPATI

Purpose: np.meshgrid() is used to create coordinate matrices from coordinate vectors, commonly used in plotting.

Python Example:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-5, 5, 1)
y = np.arange(-5, 5, 1)
X, Y = np.meshgrid(x, y)
Z = X**2 + Y**2
plt.contour(X, Y, Z)
plt.show()
```

C.28. Answer:

Concept: Broadcasting allows operations on arrays of different shapes and sizes by automatically aligning dimensions.

Python Example:

```
import numpy as np
A = np.array([[1, 2, 3], [4, 5, 6]])
B = np.array([10, 20, 30])
result = A + B  # Broadcasting the smaller array B to match the shape of A
```

C.29. Answer:

Purpose:

- **np.concatenate()** is used to concatenate arrays along a specified axis.
- **np.concatenate()** allows concatenation along any specified axis.

Differences:

- **np.vstack():** Concatenates arrays vertically (along axis 0).
- **np.hstack():** Concatenates arrays horizontally (along axis 1).

These core concept-based questions cover key aspects of NumPy, including universal functions, data types, array operations, and broadcasting.

In-depth Interview Questions on NumPy

C.30. Answer:

- Memory Layout: Refers to how elements are stored in the computer's memory.
- Order Parameter: Determines whether the array should be stored in row-major order ('C') or column-major order ('F').

Python Example:

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]], order='F')  # Column-major order
```

C.31. Answer:

- Broadcasting Rules: Specify how NumPy handles element-wise operations on arrays with different shapes.
- Automatic Alignment: NumPy compares dimensions element-wise, starting from the trailing dimensions. It automatically adds dimensions to the smaller array to match the larger one.

Python Example:

K. PRAKASH SENAPATI

```
Import numpy as np
A = np.array([[1, 2, 3], [4, 5, 6]])
B = np.array([10, 20, 30])
result = A + B  # Broadcasting the smaller array B to match the shape of A
```

C.32. Answer:

- **np.vectorize():** Converts a non-vectorized function into a vectorized function, allowing it to operate element-wise on arrays.
- **Difference:** While standard Python loops can be used, np.vectorize() provides a more concise and efficient way to apply functions across array elements.

C.33. Answer:

- **np.meshgrid():** Creates coordinate matrices from coordinate vectors.

Python Example:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-5, 5, 1)
y = np.arange(-5, 5, 1)
X, Y = np.meshgrid(x, y)
Z = X**2 + Y**2
plt.contour(X, Y, Z)
plt.show()
```

C.34. Answer:

- **np.linalg:** Contains functions for linear algebra operations such as matrix inversion, determinant calculation, eigenvalue decomposition, and singular value decomposition.

Python Example:

```
import numpy as np
A = np.array([[1, 2], [3, 4]])
inverse_A = np.linalg.inv(A)
```

C.35. Answer:

- **Structured Arrays:** Allow the creation of arrays with multiple fields, each with its data type.
- **Difference:** In regular arrays, all elements have the same data type. In structured arrays, each field can have a different data type.
- **Usefulness:** Useful when dealing with heterogeneous data, such as data tables with named columns.

C.36. Answer:

- **np.ma:** Stands for "masked arrays" and is used to handle arrays with masked elements (elements to be ignored).
- **Benefit:** Useful when dealing with data that contains missing or invalid values, allowing for operations to be performed while ignoring masked elements.

C.37. Answer:

- **np.fromiter():** Creates a new one-dimensional array from an iterable object.
- **Difference:** Unlike np.array(), which requires an existing array-like object, np.fromiter() can be used with any iterable, making it more memory-efficient for large datasets.

C.38. Answer:

K. PRAKASH SENAPATI

- **Broadcasting in Complex Scenarios: NumPy compares dimensions element-wise, starting from the trailing dimensions. It automatically adds dimensions to the smaller array to match the larger one.**

Python Example:

```
import numpy as np
A = np.array([[1, 2, 3], [4, 5, 6]])  # Shape: (2, 3)
B = np.array([[10], [20]])          # Shape: (2, 1)
result = A + B  # Broadcasting in more complex scenarios
```

C.39. Answer:

- **np.ufunc.at():** Applies a ufunc (universal function) to selected elements with specified indices.

Python Example:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
indices = np.array([0, 2])
np.add.at(arr, indices, 10)  # Adds 10 to elements at specified indices
```

These in-depth questions cover advanced aspects of NumPy, including memory layout, broadcasting in complex scenarios, vectorization, and specialized functions.

Situational Interview Questions

C.40. Answer:

In this scenario, use NumPy's masked arrays, which can be created using the np.ma.masked_array() function. This allows you to represent missing or invalid values as masked elements, and perform operations on the data while ignoring these masked elements. Additionally, you can use functions like np.ma.masked_invalid() to automatically identify and mask invalid values in the dataset.

C.41. Answer:

To optimize memory usage, you would consider the following strategies:

- Choose the appropriate data type using the dtype parameter when creating the array to minimize memory consumption.
- Utilize structured arrays if the data has multiple fields with different data types.
- Explore the use of sparse matrices using scipy.sparse if the data has a significant number of zero values.
- Use views or slices of arrays instead of creating unnecessary copies.
- Consider breaking down large arrays into smaller chunks and processing them iteratively to avoid loading the entire dataset into memory at once.

C.42. Answer:

Feature scaling is crucial in machine learning to ensure that all features contribute equally. You would typically use one of the following methods:

- Standardization: Scaling features to have a mean of 0 and a standard deviation of 1 using np.mean() and np.std().

K. PRAKASH SENAPATI

- Min-Max Scaling: Scaling features to a specified range (e.g., [0, 1]) using np.min() and np.max().
- Robust Scaling: Scaling features using the median and interquartile range to handle outliers.

C.43. Answer:

In scenarios requiring numerical stability, implement the following strategies:

- Choose appropriate data types with higher precision (e.g., np.float64) to reduce the impact of floating-point errors.
- Use specialized NumPy functions designed for numerical stability, such as np.linalg.lstsq() for solving linear least squares problems.
- Regularly check for and handle potential numerical instability, such as using functions like np.isfinite() to identify problematic values.

C.44. Answer:

To efficiently apply a custom function to each element of a NumPy array and maximize performance, consider the following approaches:

- Utilize vectorized operations whenever possible by expressing the function in a way that NumPy can operate on entire arrays.
- Leverage NumPy's np.vectorize() function to create a vectorized version of the custom function.
- Explore the use of the np.frompyfunc() function to create a universal function, which can be more efficient for more complex operations.
- Consider using the np.apply_along_axis() function if the custom function needs to be applied along a specific axis of a multi-dimensional array.

K. PRAKASH SENAPATI

SECTION D

1 Ans.

⟶ It is hands down one of the most asked numpy interview questions. NumPy, aka Numerical Python, is a free and open-source Python library. It's made to work with matrices and multidimensional arrays to conduct complicated mathematical, image processing, quantum computing, and statistical operations, among other things. Indexing, slicing, splitting, joining, reshaping, and other complicated manipulations are all possible with NumPy N-dimensional arrays.

NumPy is an excellent choice for data science applications because of its powerful built-in features. Because the demand for data scientists and machine learning professions is increasing, studying NumPy and carving a golden career path with it is a sensible choice.

2 Ans.

⟶ NumPy is a high-performance library for scientific calculations written in the Python programming language. Because it is open-source and free, it is chosen over IDL, Matlab, Octave, or Yorick. It also outperforms a generic programming language when it comes to linking Python's interpreter to C/C++ and Fortran code because it employs Python, which is a general-purpose programming language.

NumPy works with multi-dimensional arrays and matrices, allowing for complicated mathematical operations to be performed on them.

3 Ans.

⟶ The bincount() function can be used to count the number of times a given value appears. It's worth noting that the bincount() function can take either positive integers or boolean expressions as an argument. Negative integers aren't allowed to be utilized.

Use the NumPy.bincount function (). The resulting array is as follows:

```
import numpy as np

arr = np.array([0, 5, 4, 0, 4, 4, 3, 0, 0, 5, 2, 1, 1, 9])

print(np.bincount(arr))
```

Output:

K. PRAKASH SENAPATI

```
[4 2 1 1 3 2 0 0 0 1]
```

4 Ans.

```
import numpy as np
sa = np.array([[1,2,3],[4,5,6],[7,8,9]])
newColumn = NumPy.array([[0,0,0]])
```

Expected Output:

```
Printing Original Array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Printing Array after deleting col 2 on axis 1
[[1 3]
 [4 6]
 [7 9]]
Printing Array after inserting col 2 on axis 1
[[1 0 3]
 [4 0 6]
 [7 0 9]]
```

Solution:

```
import numpy as np
print("Printing Original Array:")
sa = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(sa)
```

K. PRAKASH SENAPATI

```
print("Printing Array after deleting col 2 on axis 1")
sa = np.delete(sa , 1, axis = 1)
print(sa)
arr = np.array([[0,0,0]])
print("Printing Array after inserting col 2 on axis 1")
sa = np.insert(sa , 1, arr, axis = 1)
print(sa)
```

Output:

```
Printing Original Array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Printing Array after deleting col 2 on axis 1
[[1 3]
 [4 6]
 [7 9]]
Printing Array after inserting col 2 on axis 1
[[1 0 3]
 [4 0 6]
 [7 0 9]]
```

5 Ans.

⟶ NumPy would be perfect if it only had the array data type and the most basic operations: indexing, sorting, reshaping, basic elementwise functions, and so on. SciPy would house all numerical codes. However, compatibility is one of NumPy's main goals, therefore it tries to keep all features supported by any of its predecessors. As a result, NumPy includes several linear algebra routines that are better appropriately found in SciPy. In any case, SciPy has more advanced linear algebra modules, as well as a variety of additional numerical techniques. You should probably install both NumPy and SciPy if you're undertaking scientific computing with Python. The majority of new features belong in SciPy, not NumPy.

6 Ans.

```
import numpy as np

arr = np.eye(4)

print("\ndiaglonal values : \n",arr)
```

Output:

```
diaglonal values :
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

7 Ans.

⟶ Python 2.x , as well as Python 3.2 and newer, are supported by NumPy and SciPy. NumPy 1.5.0 was the first version of NumPy to support Python 3. SciPy version 0.9.0 introduces Python 3 support.

8 Ans.

```
import numpy as np

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

print("\n square matrix: \n",arr)
```

Output:

```
square matrix:
[[1 0 0 0 0 0]
 [0 2 0 0 0 0]
 [0 0 3 0 0 0]
 [0 0 0 4 0 0]
 [0 0 0 0 5 0]
 [0 0 0 0 0 6]]
```

K. PRAKASH SENAPATI

9 Ans.

⟶ When compared to NumPy arrays, Python's lists, despite being extremely efficient containers capable of a wide range of functions, have many drawbacks. Vectorized operations, such as element-by-element addition and multiplication, are not conceivable.

Because they support objects of various types, they also require Python to retain the type information of each element. This means that every time an operation on an element is performed, a type dispatching code must be run. In addition, each iteration would have to go through type checks and require Python API accounting, resulting in C loops carrying very few actions.

10 Ans.

⟶ NumPy has a flexible working mechanism that allows it to take advantage of CPU SIMD features to give quicker and more stable performance across all platforms.

11 Ans.

```
import numpy as np

# Float to Integer

ar = np.array([3.3, 4.2, 5.1])

newar = ar.astype('i')

print(newar)

print(newar.dtype)
```

Output:

```
[3 4 5]
int32
```

12 Ans.

K. PRAKASH SENAPATI

Copy :

A replica of the original array is returned.

Do not use the original array's data or memory location.

Any changes made to the copy won't appear in the original.

```
import numpy as np
ar = np.array([1,2,3,4])
a= ar.copy()
a[0] = 5
print(ar)
print(a)
```

Output:

```
[1 2 3 4]
[5 2 3 4]
```

View:

Returns a visual representation of the original array.

The data and memory location of the original array is used.

Any changes made to the copy will also be reflected in the original.

```
import numpy as np
arr = np.array([1,2,3,4])
a= arr.view()
#changing a value in the original array
arr[0] = 100
print(arr)
```

print(a)

Output:

```
[100   2   3   4]
[100   2   3   4]
```

13 Ans.

⟶ The @ symbol will be defined as a matrix multiplication operator in Python 3.5, and Numpy and Scipy will take advantage of it. PEP 465 was dedicated to this addition. To compensate for the unavailability of this operator in earlier versions of Python, separate matrix and array types were created.

14 Ans.

⟶ The ndarray object lies at the heart of the NumPy package. Many operations are performed in compiled code for performance, and this encapsulates n-dimensional arrays of homogeneous data types. Between NumPy arrays and normal Python sequences, there are a few key differences:

Unlike Python lists, NumPy arrays have a fixed size when created (which can grow dynamically). When the size of a ndarray is changed, a new array is created and the old one is deleted.

In a NumPy array, all of the elements must be of the same data type. The exception is that arrays of (Python, including NumPy) objects can be used, allowing for arrays of various sizes.

NumPy arrays make it easier to do advanced mathematical and other operations on massive amounts of data. Such actions are typically performed more quickly and with less code than utilizing Python's built-in sequences.

NumPy arrays are used by a growing number of scientific and mathematical Python-based packages; while they normally support Python-sequence input, they convert it to NumPy arrays before processing, and they frequently output NumPy arrays. To put it another way, knowing how to utilize Python's built-in sequence types isn't enough to efficiently use much (if not all) of today's scientific/mathematical Python-based software; you also need to know how to use NumPy arrays.

15 Ans.

K. PRAKASH SENAPATI

```
import numpy as np

arr = np.arange(1,10).reshape(3,3)

print(arr)
```

Output:

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

16 Ans.

⟶ Numpy.array is only a helper function for making a ndarray; it isn't a class in and of itself. You can alternatively use numpy. ndarray to create an array; however, this is not advised.

You can generate an array from the ndarray class in two ways, as shown below:

Using the methods array(), zeros(), and empty(): Arrays should be built with array, zeros, or empty values. The parameters relate to a low-level mechanism for instantiating an array (ndarray(...)).

Directly from the ndarray class: When using __new__to create an array, there are two options: Only shape, dtype, and order are used in the buffer is None. All keywords are interpreted if the buffer is an object that exposes the buffer interface.

17 Ans.

⟶ No. Simply explained, Jython runs on top of the Java Virtual Machine and has no way of interacting with C extensions for the regular Python interpreter (CPython).

18 Ans.

⟶ Vectorization applies a single elemental operation to all of the elements in an array. The implementations below are written in C, resulting in significant speed improvements. NumPy already vectorizes a huge number of operations, such as all arithmetic operators, logical operators, and so on. Numpy also gives you the option of vectorizing your function. All you have to do now is:

Create a function that performs the desired operation and takes the members of the array as arguments.

The function should be vectorized.

This vectorized function requires the arrays as inputs.

19 Ans.

K. PRAKASH SENAPATI

→ NumPy and SciPy, which are focused on numerical objects and methods, do not provide plotting. Several programs, like the widely used Matplotlib and the versatile, modular toolkit Chaco, work closely with NumPy to produce high-quality charts.

20 Ans.

```
import numpy as np

rand = np.random.randint(1,20,4)

print("\n Random number ranges from 1-20 ",rand)
```

Output:

```
Random number ranges from 1-20 [ 4  6  1 17]
```

21 Ans.

```
import numpy as np

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

reversed_arr = arr[::-1]

print(reversed_arr)
```

Output:

```
[5 4 3 2 1]
```

22 Ans.

→ On 32-bit Windows, some users have reported success with NumPy and Ironclad. The present state of Ironclad support for SciPy is uncertain, but it is less possible than NumPy due to various complicating circumstances (most notably the Fortran compiler situation on Windows).

23 Ans.

```
import numpy as np
```

```
num2=[[3,2,1],[4,5,6]]
num2 = np.array(num2)
print("\n2d array : \n",num2)
```

Output:

```
2d array :
[[3 2 1]
 [4 5 6]]
```

24 Ans.

```
import numpy as np
num2=[[3,2,1],[4,5,6]]
num2 = np.array(num2)
print("\n2d array : \n",num2)
```

Output:

```
2d array :
[[3 2 1]
 [4 5 6]]
```

25 Ans.

→ Add: '+' represents the addition of elements from two arrays.

Multiply: Multiplies the items by a certain number, as indicated by the asterisk (*).

Power: Array items can be squared using the '**' operator.

Utilize the syntax below to use a higher power-like cube.

Conditional Expressions: A conditional expression compares the items of an array.

K. PRAKASH SENAPATI

```
[6 6 6 6 6]
[ 5 16 27 16  1]
[125  64  27   8   1]
[ True  True  True False False]
```

K. PRAKASH SENAPATI

SECTION E

Answers

Answer 1 :

NumPy is an open-source, versatile general-purpose package used for array-processing. It is short on Numerical Python. It is known for its high-end performance with powerful N-dimensional array objects and the tools it is loaded with to work with arrays. The package is an extension of Python and is used to perform scientific computations and other broadcasting functions.

NumPy is easy to use, well-optimized, and highly flexible. It is compared with MATLAB on the basis of their functionalities as both of them facilitate writing fast programs as long as most of the functions work on the arrays. NumPy is closely integrated with Python and makes it a much more sophisticated programming language.

Answer 2 :

The open-source numerical library on Python supports multi-dimensional arrays and contains matrix data structures. Different types of mathematical operations can be performed on arrays using NumPy. This includes trigonometric operations as well as statistical and algebraic computations. Numeric and Numarray are extensions of NumPy.

Another answer for NumPy data science interview questions could be – "NumPy is used for scientific computing, deep learning, and financial analysis. Various functions can be performed with the aid of NumPy such as the arithmetic operations, stacking, matrix operations, broadcasting, linear algebra, etc."

Answer 3 :

NumPy is a high-performance library in the Python programming language that allows scientific calculations. It is preferred to Idl, Matlab, Octave, Or Yorick because it is open-source and free. Also, since it uses Python which is a general-purpose programming language, it scores over a generic programming language when it comes to connecting Python's interpreter to C/C++ and Fortran code.

NumPy supports multi-dimensional arrays and matrices and helps to perform complex mathematical operations on them.

Answer 4 :

As a powerful open-source package used for array-processing, NumPy has various useful features. They are:

1. Contains an N-dimensional array object

K. PRAKASH SENAPATI

2. It is interoperable; compatible with many hardware and computing platforms
3. Works extremely well with array libraries; sparse, distributed or GPU
4. Ability to perform complicated (broadcasting) functions
5. Tools that enable integration with C or C++ and Fortran code
6. Ability to perform high-level mathematical functions like statistics, Fourier transform, sorting, searching, linear algebra, etc
7. It can also behave as a multi-dimensional container for generic data
8. Supports scientific and financial calculations.
9. Can work with various types of databases
10. Provides multi-dimensional arrays
11. Indexing, Slicing, or Masking with other arrays facilitate sin accessing the specific pixels of an image.

Answer 5 :

To install NumPy on Windows, you must first download and install Python on your computer.

Follow the steps given below to install Python:

Step 1: Visit the official page of Python and download Python and Python executable binaries on your Windows 10/8/7

Step 2: Open Python executable installer and press Run

Step 3: Install pip on your Windows system

Using pip, you can install NumPy in Python. Below is the Installation Process of NumPy:

Step 1: Start the terminal

Step 2: Type pip

Step 3: install NumPy

Answer 6 :

Python's lists, even though hugely efficient containers capable of a number of functions, have several limitations when compared to NumPy arrays. It is not possible to perform vectorised operations which includes element-wise addition and multiplication.

They also require that Python store the type information of every element since they support objects of different types. This means a type dispatching code must be executed each time an operation on an element is done. Also, each iteration would have to undergo type checks and require Python API bookkeeping resulting in very few operations being carried by C loops.

This makes for one of the commonly asked numpy questions, where the advantages are required to enlist. Another advantage could be the less memory space that is utilized to store the data which

K. PRAKASH SENAPATI

helps in further optimization of the code. Scientific computing and array-oriented computing are more aligned advantages of NumPy.

Answer 7:

A one-dimensional array is created as follows:

num=[1,2,3]

num = np.array(num)

print("1d array : ",num)

A two-dimensional array is created as follows:

num2=[[1,2,3],[4,5,6]]

num2 = np.array(num2)

print("\n2d array : ",num2)

A 1-D array stands for a one-dimensional array that creates the array in one dimension. Whereas the 2D arrays have a collection of rows and columns.

Answer 8:

A three-dimensional array is created as follows:

num3=[[[1,2,3],[4,5,6],[7,8,9]]]

num3 = np.array(num3)

print("\n3d array : ",num3)

Answer 9:

1D Array:

num=[1,2,3] if not added

print('\nshpae of 1d ',num.shape)

2D Array:

num2=[[1,2,3],[4,5,6]] if not added

print('\nshpae of 2d ',num2.shape)

3D or ND Array:

num3=[[[1,2,3],[4,5,6],[7,8,9]]] if not added

print('\nshpae of 3d ',num3.shape)

K. PRAKASH SENAPATI

Answer 10:

Use the following sequence of codes to identify the datatype of a NumPy array.

print('\n data type num 1 ',num.dtype)

print('\n data type num 2 ',num2.dtype)

print('\n data type num 3 ',num3.dtype)

Answer 11:

You can count the number of times a given value appears using the bincount() function. It should be noted that the bincount() function accepts positive integers or boolean expressions as its argument. Negative integers cannot be used.

Use NumPy.bincount(). The resulting array is

>>> arr = NumPy.array([0, 5, 4, 0, 4, 4, 3, 0, 0, 5, 2, 1, 1, 9])

>> NumPy.bincount(arr)

Answer 12:

If the variable is an array, you can check for an empty array by using the size attribute. However, it is possible that the variable is a list or a sequence type, in that case, you can use len().

The preferable way to check for a zero element is the size attribute. This is because:

>>> a = NumPy.zeros((1,0))

>>> a.size

0

whereas

>>> len(a)

1

Answer 13:

You may use the function numpy.nonzero() to find the indices or an array. You can also use the nonzero() method to do so.

In the following program, we will take an array a, where the condition is a > 3. It returns a boolean array. We know False on Python and NumPy is denoted as 0. Therefore, np.nonzero(a > 3) will return the indices of the array a where the condition is True.

>>> import numpy as np

K. PRAKASH SENAPATI

```
>>> a = np.array([[1,2,3],[4,5,6],[7,8,9]])
>>> a > 3
array([[False, False, False],
       [ True,  True,  True],
       [ True,  True,  True]], dtype=bool)
>>> np.nonzero(a > 3)
(array([1, 1, 1, 2, 2, 2]), array([0, 1, 2, 0, 1, 2]))
```

You can also call the nonzero() method of the boolean array.

```
>>> (a > 3).nonzero()
(array([1, 1, 1, 2, 2, 2]), array([0, 1, 2, 0, 1, 2]))
```

Answer 14:

```
import NumPy
sampleArray = NumPy.array([[34,43,73],[82,22,12],[53,94,66]])
newColumn = NumPy.array([[10,10,10]])
```

upGrad's Exclusive Data Science Webinar for you –

The Future of Consumer Data in an Open Data Economy

Expected Output:

```
Printing Original array
[[34 43 73]
 [82 22 12]
 [53 94 66]]
Array after deleting column 2 on axis 1
[[34 73]
 [82 12]

 [53 66]]
Array after inserting column 2 on axis 1
[[34 10 73]
 [82 10 12]
 [53 10 66]]
```

Solution:

K. PRAKASH SENAPATI

```
import NumPy
print("Printing Original array")
sampleArray = NumPy.array([[34,43,73],[82,22,12],[53,94,66]])
print (sampleArray)
print("Array after deleting column 2 on axis 1")
sampleArray = NumPy.delete(sampleArray , 1, axis = 1)
print (sampleArray)
arr = NumPy.array([[10,10,10]])
print("Array after inserting column 2 on axis 1")
sampleArray = NumPy.insert(sampleArray , 1, arr, axis = 1)
print (sampleArray)
```

Answer 15:

```
import NumPy
print("Printing Original array")
sampleArray = NumPy.array([[34,43,73],[82,22,12],[53,94,66]])
print (sampleArray)
print("Array after deleting column 2 on axis 1")
sampleArray = NumPy.delete(sampleArray , 1, axis = 1)
print (sampleArray)
arr = NumPy.array([[10,10,10]])
print("Array after inserting column 2 on axis 1")
sampleArray = NumPy.insert(sampleArray , 1, arr, axis = 1)
print (sampleArray)
```

K. PRAKASH SENAPATI

SECTION-F (Answers)

Python NumPy Interview Answers for Beginners

K. PRAKASH SENAPATI

Here is a compilation of fundamental NumPy interview questions and answers designed for beginners to help strengthen their understanding of the Python NumPy library.

F.1

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. In contrast, 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.

F.2

NumPy is a powerful numerical computing library in Python that provides support for creating and manipulating arrays. Here's a brief explanation of how to create NumPy arrays along with an example:
First, start by importing the NumPy library using import NumPy as np
Then, you can create a one dimensional and two dimensional array as follows-

```
# Creating a 1-dimensional array
arr_1d = np.array([1, 2, 3, 4, 5])
print("1D Array:")
print(arr_1d)

# Creating a 2-dimensional array (matrix)
arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("\n2D Array:")
print(arr_2d)
```

```
1D Array:
[1 2 3 4 5]

2D Array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

F.3

There are two ways to reverse a NumPy array:

Method 1: You can reverse a NumPy array using the numpy.flip() function. Here's an example:

```
import numpy as np

# Create a NumPy array
original_array = np.array([1, 2, 3, 4, 5])

# Reverse the array
reversed_array = np.flip(original_array)

# Print the result
print("Original array:", original_array)
print("Reversed array:", reversed_array)
```

```
Original array: [1 2 3 4 5]
Reversed array: [5 4 3 2 1]
```

Method 2: You can also use slicing to achieve the same result.

```
import numpy as np

# Create a NumPy array
original_array = np.array([1, 2, 3, 4, 5])

# Reverse the array using slicing
reversed_array = original_array[::-1]

# Print the result
print("Original array:", original_array)
print("Reversed array:", reversed_array)
```

```
Original array: [1 2 3 4 5]
Reversed array: [5 4 3 2 1]
```

F.4

Broadcasting in NumPy allows for efficient array operations with different shapes, treating smaller arrays during arithmetic operations. This feature, subject to constraints, enables vectorized operations in optimized C code, enhancing computational efficiency without unnecessary data copying. While typically requiring identical shapes, NumPy's broadcasting rule relaxes this constraint, particularly when combining arrays with scalars. This intelligent stretching of scalars

K. PRAKASH SENAPATI

during operations optimizes memory usage, but it's essential to be cautious, as inefficient memory usage can occur in certain cases.

K. PRAKASH SENAPATI

F.5

A ufunc is a vectorized function that operates on NumPy arrays element-wise, allowing for efficient and concise array calculations. Here are some key aspects of ufuncs:

- Ufuncs perform operations on each element of one or more input NumPy arrays, producing an array of the same shape as the input array. This enables concise and efficient array expressions without the need for explicit loops.
- Ufuncs are designed to take advantage of vectorized hardware operations, such as SIMD (Single Instruction, Multiple Data) on modern CPUs and GPUs. This results in faster and more efficient computations compared to traditional scalar operations.
- Ufuncs support NumPy's broadcasting mechanism, allowing arrays with different shapes to be used in arithmetic and logical operations. Broadcasting automatically extends smaller arrays to match the shape of larger arrays, making it easier to perform operations on arrays of different shapes.
- Ufuncs handle type casting automatically. For example, if you perform an operation between arrays with different data types, NumPy will internally cast the data types to a common type according to a set of rules.
- Ufuncs can accept scalars, arrays, or a mix of both as input. When operating on arrays with different shapes, broadcasting rules are applied to make the shapes compatible.
- Each ufunc has a fixed number of specific inputs and produces a fixed number of specific outputs. The function signature defines the number of input and output arguments, as well as their data types.
- NumPy provides a wide range of ufuncs covering basic arithmetic operations, trigonometric functions, exponential and logarithmic functions, bitwise operations, comparison operators, and more. Users can also create custom ufuncs using the numpy.frompyfunc or numpy.vectorize functions.

K. PRAKASH SENAPATI

F.6

The np.dot() function in NumPy is a versatile tool for performing dot products and matrix multiplications. It adapts its behavior based on the dimensions of the input arrays. For 2-D arrays, it acts as a matrix multiplication operator, while for 1-D arrays, it computes the inner product. This flexibility makes np.dot()suitable for a wide range of array operations.

On the other hand, np.matmul() is specifically designed for matrix multiplication. While it behaves similarly to np.dot() when dealing with 2-D arrays, its explicit name suggests a focused purpose: matrix multiplication. Using np.matmul() is recommended when clarity and readability are crucial in your code, especially when working extensively with matrices. This function serves as a more specialized and self-explanatory option for matrix multiplication operations in NumPy.

F.7

np.mean() and np.average() in NumPy both calculate the mean of an array, but np.average() allows for the use of weights. While np.mean() treats all elements equally, np.average() considers weights assigned to each element, providing a weighted average when specified. This makes np.average() more flexible for situations where certain elements contribute more or less to the overall average based on their assigned weights.

Here are examples illustrating the use of np.mean() and np.average() in NumPy:

K. PRAKASH SENAPATI

```python
import numpy as np

# Example for np.mean()
arr_mean = np.array([1, 2, 3, 4, 5])
mean_result = np.mean(arr_mean)
print("Mean:", mean_result)  # Output: Mean: 3.0

# Example for np.average() with weights
arr_average = np.array([1, 2, 3, 4, 5])
weights = np.array([0.1, 0.2, 0.3, 0.2, 0.2])
weighted_average_result = np.average(arr_average, weights=weights)
print("Weighted Average:", weighted_average_result)  # Output: Weighted Average: 3.2
```

```
Mean: 3.0
Weighted Average: 3.2
```

F.8

You can perform matrix multiplication using the np.dot() function or the @ operator. Here's an example of both methods:-

```python
import numpy as np

# Create two matrices
matrix_A = np.array([[5, 7], [3, 4]])
matrix_B = np.array([[2, 6], [7, 8]])

# Method 1: Using np.dot()
result_dot = np.dot(matrix_A, matrix_B)

# Method 2: Using @ operator
result_at = matrix_A @ matrix_B

# Display the results
print("Result using np.dot():")
print(result_dot)

print("\nResult using @ operator:")
print(result_at)
```

```
Result using np.dot():
[[59 86]
 [34 50]]

Result using @ operator:
[[59 86]
 [34 50]]
```

Both methods will give you the same result. Note that for matrix multiplication, the number of columns in the first matrix must be equal to the number of rows in the second matrix. In the example above, both matrices are 2x2, and the multiplication is valid.

F.9

You can count the frequency of a specified positive value within a NumPy array using the np.count_nonzero() function along with boolean indexing. Here's an example:

```
import numpy as np

# Create a NumPy array
arr = np.array([1, 2, 3, 4, 5, 2, 1, 3, 4, 1])

# Specify the positive value you want to count
specified_value = 1

# Use boolean indexing to create a mask for elements equal to the specified value
mask = (arr == specified_value)

# Count the occurrences of the specified value using np.count_nonzero()
frequency = np.count_nonzero(mask)

# Print the result
print(f"The frequency of {specified_value} in the array is: {frequency}")
```

```
The frequency of 1 in the array is: 3
```

This code first creates a NumPy array (arr). It then creates a boolean mask (mask) by checking which elements are equal to the specified positive value (specified_value). Finally, it uses np.count_nonzero() to count the occurrences of True values in the mask, giving you the frequency of the specified positive value in the array.

F.10

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 = np.zeros((1, 0))    # returns empty array
print(arr.size)           # returns 0
```

```
0
```

The arr.size returns 0, confirming that the array is empty or has zero elements. This is a convenient way to check for emptiness in NumPy arrays.

Python NumPy Interview Answers for Intermediates

Here are a few intermediate NumPy interview questions and answers that cover concepts like checking data types, random functions, and statistical functions:-

F.11

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

K. PRAKASH SENAPATI

```python
import numpy as np

# Create a NumPy array
arr = np.array([1, 2, 3, 4, 5])

# Get the data type of the elements in the array
data_type = arr.dtype

# Print the data type
print("Data Type:", data_type)
```

```
Data Type: int64
```

Here, the dtype attribute is used to retrieve the data type of the elements in the NumPy array arr. The result will be something like int64, indicating that the array contains 64-bit integer values. NumPy supports a variety of data types such as integers, floats, complex numbers, etc. You can explicitly set the data type when creating an array using the dtype parameter in the np.array() function.

F.12

Peaks or local maxima in a 1D array are the points where the value is higher than the surrounding values. There are 3 major approaches to achieve this.

Method 1: Using .where() method

```python
import numpy as np
# Define NumPy array
arr = np.array([1, 7, 3, -1, 2, 6, 8, 1, 0, 6, 1, -3, -1, 10, 4])

# Find local maxima using .where() method
maxima_peaks_positions = np.where((arr[:-2] < arr[1:-1]) & (arr[1:-1] > arr[2:]))[0] + 1
print(maxima_peaks_positions)
```

```
[ 1  6  9 13]
```

Here, the .where() method identifies positions where the element value at a particular index is greater than both the element on its left and right. The condition (arr[:-2] < arr[1:-1]) & (arr[1:-1] > arr[2:]) checks this criterion. The result is an array of positions where local maxima occur, and we add 1 to adjust for the fact that indexing starts from 0.

Method 2: Using a combination of .diff(), .sign(), and .where() Method

K. PRAKASH SENAPATI

```python
import numpy as np
# Define NumPy array
arr = np.array([1, 7, 3, -1, 2, 6, 8, 1, 0, 6, 1, -3, -1, 10, 4])

# Calculate differences and signs
diff_arr = np.sign(np.diff(arr))
all_peaks = np.diff(diff_arr)

# Find local maxima using .where() method
maxima_peaks_positions = np.where(all_peaks == -2)[0] + 1
print(maxima_peaks_positions)
```

```
[ 1  6  9 13]
```

Here, we calculate the differences between consecutive elements using np.diff(). Then, we determine the sign of each difference using np.sign(). The second differences are calculated with np.diff(diff_arr). Peaks (local maxima) are identified where the second differences equal -2 using np.where().

F.13

You can convert a Pandas DataFrame into a NumPy array using the values attribute of the DataFrame.

```python
import pandas as pd
import numpy as np

# Create a sample DataFrame
data = {'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]}
df = pd.DataFrame(data)

# Convert DataFrame to NumPy array
numpy_array = df.values

print("Pandas DataFrame:")
print(df)

print("\nNumPy Array:")
print(numpy_array)
```

```
Pandas 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]]
```

K SENAPATI

This example uses the values attribute of the DataFrame df to obtain the underlying NumPy array. The resulting numpy_array will have the same data as the DataFrame, and you can perform NumPy array operations on it.

F.14

You can use the numpy.random.shuffle function to randomly shuffle the elements of a NumPy array.

```
import numpy as np

# Create a NumPy array
my_array = np.array([5, 2, 1, 4, 5])

# Shuffle the array in-place
np.random.shuffle(my_array)

# Print the shuffled array
print(my_array)
```

```
[1 5 4 2 5]
```

F.15

The np.copy() function explicitly generates a deep copy of the array, ensuring that changes made to the copied array do not affect the original array, and vice versa. This method is particularly useful when you want to create an independent duplicate of an array. On the other hand, using the assignment operator (=) for array assignment creates a reference to the original array rather than a new copy. Consequently, modifications made to the copied array will also reflect in the original array, and vice versa.

F.16

A shallow copy and a deep copy refer to different ways of duplicating an array. A shallow copy creates a new array object, but it still references the same underlying data as the original array. Changes made to the data in the shallow copy will be reflected in the original array and vice versa. On the other hand, a deep copy creates a completely independent copy with its own separate data. Modifications to the data in a deep copy do not affect the original array, providing a more isolated duplication. In NumPy, the copy module provides functions like numpy.copy() for shallow copies and numpy.deepcopy() for deep copies to facilitate these operations on arrays.

F.17

K. PRAKASH SENAPATI

You can compute the cross-product of two vectors in NumPy using the numpy.cross() function. The cross() function takes two arrays as arguments and returns their cross product.

K. PRAKASH SENAPATI

```python
import numpy as np

# Define two vectors
vector1 = np.array([1, 2, 3])
vector2 = np.array([4, 5, 6])

# Compute the cross product
cross_product = np.cross(vector1, vector2)

# Print the result
print("Cross Product:", cross_product)
```

```
Cross Product: [-3  6 -3]
```

Also Check: [How to calculate dot product of two vectors using numpy in python?]

F.18

NumPy provides two main memory layouts: C-order (row-major) and F-order (column-major). In C-order, elements are stored in a contiguous block row by row, meaning that the elements of a row are stored together in memory. This is the default memory layout in NumPy. On the other hand, F-order stores elements column by column, making it more suitable for Fortran-style array indexing.

F.19

You can use the numpy.mean() and numpy.std() functions to calculate the mean and standard deviation of a NumPy array.

```python
import numpy as np

# Create a NumPy array
data = np.array([3, 2, 4, 7, 5])

# Calculate the mean
mean_value = np.mean(data)

# Calculate the standard deviation
std_deviation = np.std(data)

# Print the results
print("Mean:", mean_value)
print("Standard Deviation:", std_deviation)
```

```
Mean: 4.2
Standard Deviation: 1.7204650534085253
```

K. PRAKASH SENAPATI

F.20

The np.random module in NumPy is designed to provide functionalities for generating random numbers and random data for various applications, such as simulations, statistical analysis, and machine learning. It offers a range of random number generation methods and distributions.

```
import numpy as np

# Generate a random float between 0 and 1
random_float = np.random.rand()
print("Random Float:", random_float)

# Generate an array of 5 random integers between 1 and 10
random_integers = np.random.randint(1, 11, 5)
print("Random Integers:", random_integers)

# Generate a 3x3 matrix with random values from a standard normal distribution
random_matrix = np.random.randn(3, 3)
print("Random Matrix:\n", random_matrix)
```

```
Random Float: 0.24764614221368897
Random Integers: [2 6 4 6 9]
Random Matrix:
 [[ 0.8175449  -0.6799794   0.59380013]
 [-1.15853259 -1.22873957 -0.18061066]
 [ 1.02128195 -0.05162602 -0.58507392]]
```

np.random.rand() generates a random float, np.random.randint() creates an array of random integers, and np.random.randn() generates a matrix with random values from a standard normal distribution.

Advanced NumPy Interview Answers

Check out the following advanced NumPy interview questions and answers to deepen your understanding of complex topics such as vectorization, broadcasting, and advanced functionality.

F.21

Vectorization in NumPy is crucial for efficient numerical operations in Python. It involves applying operations to entire arrays rather than individual elements, leveraging the underlying C implementation of NumPy. This leads to faster execution times as it takes advantage of optimized, parallelized routines. Vectorized code is concise, readable, and more idiomatic, promoting better code quality and ease of maintenance

Also Check: How to Create a Vector or Matrix in Python?

F.22

hstack() and vstack() are NumPy functions used for stacking arrays horizontally and vertically, respectively.

hstack(): Stacks arrays horizontally, i.e., along the second axis.

vstack(): Stacks arrays vertically, i.e., along the first axis.

K. PRAKASH SENAPATI

Example -

K. PRAKASH SENAPATI

```python
import numpy as np

array1 = np.array([1, 2, 3])
array2 = np.array([4, 5, 6])

hstack_result = np.hstack((array1, array2))
print("hstack_result:", hstack_result)
vstack_result = np.vstack((array1, array2))
print("vstack_result:")
print(vstack_result)
```

```
hstack_result: [1 2 3 4 5 6]
vstack_result:
[[1 2 3]
 [4 5 6]]
```

F.23

NumPy seamlessly integrates with popular machine learning libraries, serving as a foundational numerical computing library in the Python ecosystem. It is a fundamental building block for libraries like TensorFlow and PyTorch, providing efficient array operations. NumPy arrays are often used as the standard data representation in these libraries, enabling smooth data exchange and computation between different components of machine learning workflows. Additionally, many machine learning algorithms and models in scikit-learn also leverage NumPy arrays for data manipulation and mathematical operations.

F.24

NumPy plays a crucial role in feature engineering for machine learning by providing a powerful and efficient array manipulation library in Python. It enables easy handling of numerical data, essential for creating and transforming features in machine learning models. NumPy's functions facilitate tasks like data manipulation, mathematical operations, and statistical analysis, allowing practitioners to preprocess and engineer features effectively. Its array structures enhance performance and compatibility with other machine learning libraries, making NumPy a fundamental tool in the feature engineering pipeline.

K. PRAKASH SENAPATI

F.25

You can use the np.convolve function to implement a moving average for a 1D array in NumPy.

```
import numpy as np

# Example data
data = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Window size for moving average
window_size = 3

# Create a 1D array of ones to act as the weights for the moving average
weights = np.ones(window_size) / window_size

# Use np.convolve to calculate the moving average
# mode='valid' ensures that only valid elements are included in the result
moving_avg_result = np.convolve(data, weights, mode='valid')

# Print the results
print("Original data:", data)
print("Moving average with window size", window_size, ":", moving_avg_result)
```

```
Original data: [ 1  2  3  4  5  6  7  8  9 10]
Moving average with window size 3 : [2. 3. 4. 5. 6. 7. 8. 9.]
```

First, define your data array and specify a window size. Create an array of weights, typically a set of ones normalized by the window size. Then, use np.convolve to calculate the moving average by convolving the data array with the weights, specifying 'valid' mode to include only valid elements.

F.26

NumPy allows you to perform broadcasting, which is a powerful feature that simplifies operations between arrays of different shapes, as long as they meet certain compatibility conditions. Broadcasting allows you to perform operations on arrays of different shapes, including combining a 2D array with a scalar value.

K. PRAKASH SENAPATI

```python
import numpy as np

# Create a 2D array
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

# Scalar value
scalar_value = 10

# Perform broadcasting operation: add the scalar to the entire matrix
result = matrix + scalar_value

# Print the result
print("Original Matrix:")
print(matrix)
print("\nScalar Value:")
print(scalar_value)
print("\nResult after broadcasting:")
print(result)
```

```
Original Matrix:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Scalar Value:
10

Result after broadcasting:
[[11 12 13]
 [14 15 16]
 [17 18 19]]
```

The scalar value 10 is broadcasted to each element of the 2D array matrix. The operation is performed element-wise, and the result is a new matrix with the same shape as the original matrix, where each element has been increased by the scalar value.

Also check: [How to ADD numerical value to each element of a matrix using numpy in python?]

NumPy Coding Interview Answers

The following NumPy coding interview questions and answers will enhance your understanding of NumPy's application in data manipulation and analysis.

F.27

You can filter elements from a NumPy array based on a boolean condition using boolean indexing.

```python
import numpy as np
# Create a NumPy array
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Define a boolean condition
condition = (arr % 2 == 0)  # Keep even numbers

# Use boolean indexing to filter elements
filtered_arr = arr[condition]

# Print the result
print(filtered_arr)
```

```
[ 2  4  6  8 10]
```

The condition array is a boolean array where each element is True if the corresponding element in arr satisfies the condition (arr % 2 == 0) (even numbers), and False otherwise. The arr[condition] expression selects only the elements from arr where the corresponding element in the condition array is True.

F.28

You can use the reshape function to reshape a 1D NumPy array into a 2D array with a specified number of rows.

```python
import numpy as np

# Create a 1D NumPy array
original_array = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])

# Specify the number of rows for the 2D array
num_rows = 3

# Reshape the array into a 2D array with the specified number of rows
reshaped_array = original_array.reshape(num_rows, -1)

print("Original 1D array:")
print(original_array)

print("\nReshaped 2D array:")
print(reshaped_array)
```

```
Original 1D array:
[1 2 3 4 5 6 7 8 9]

Reshaped 2D array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

The reshape function is used with the argument num_rows to specify the desired number of rows for the 2D array. The -1 in the second dimension is a placeholder that automatically calculates the appropriate number of columns based on the total number of elements in the original array.
[Note - The total number of elements in the original array is divisible evenly by the specified number of rows to avoid any errors during reshaping.]

F.29

You can use array slicing to extract a subarray from a larger array based on given slicing parameters. The syntax for array slicing is array[start:stop:step], where start is the index to start the slicing, stop is the index to end the slicing (exclusive), and step is the step size for slicing. If any of these parameters is omitted, it uses the default values (0 for start, the length of the array for stop, and 1 for step).

```
# Example array
larger_array = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Extract a subarray from index 2 to 6 (exclusive)
subarray = larger_array[2:6]

print(subarray)
```

```
[2, 3, 4, 5]
```

F.30

NumPy provides a convenient way to find the eigenvalues and eigenvectors of a matrix using the numpy.linalg.eig function.

```
import numpy as np
# Define your matrix
matrix = np.array([[5, 7], [3, 4]])

# Use numpy.linalg.eig to compute eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(matrix)

# Print the results
print("Eigenvalues:")
print(eigenvalues)
print("\nEigenvectors:")
print(eigenvectors)
```

```
Eigenvalues:
[ 9.10977223 -0.10977223]

Eigenvectors:
[[ 0.86235788 -0.80769951]
 [ 0.50629921  0.58959436]]
```

F.31

```
import numpy as np
# Generate a 2D array F of shape (5, 5) with random integer values between 1 and 100
F = np.random.randint(1, 101, size=(5, 5))
# Compute the mean and standard deviation for each row
row_means = np.mean(F, axis=1)
row_std_devs = np.std(F, axis=1)
# Compute the mean and standard deviation for each column
col_means = np.mean(F, axis=0)
col_std_devs = np.std(F, axis=0)
# Print the generated array, row means, row standard deviations, column means, and column standard deviations
print("Generated 2D array:")
print(F)
print("\nRow Means:")
print(row_means)
print("Row Standard Deviations:")
print(row_std_devs)
print("\nColumn Means:")
print(col_means)
print("Column Standard Deviations:")
print(col_std_devs)
```

```
Generated 2D array:
[[60 84 18 97  2]
 [87 58 72  3 31]
 [43 64 72 79  7]
 [85 83 45 93 18]
 [ 8 30 41 19 93]]

Row Means:
[52.2 50.2 53.  64.8 38.2]
Row Standard Deviations:
[36.79347768 29.95596769 25.97691283 28.6942503  29.52558213]

Column Means:
[56.6 63.8 49.6 58.2 30.2]
Column Standard Deviations:
[29.28890575 19.76259092 20.48023437 39.32632706 32.94480232]
```

F.32

You can use the diagonal method to extract the diagonal elements of a square 2D NumPy array matrix.

```
import numpy as np
# Create a square 2D NumPy array
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

# Extract diagonal elements using np.diagonal()
diagonal_elements = np.diagonal(matrix)
print("Diagonal elements:", diagonal_elements)
```

```
Diagonal elements: [1 5 9]
```

Also check: [How to find the Rank of a Matrix using numpy in python]

PYTHON ANSWERS

1A)

Deepcopy creates a different object and populates it with the child objects of the original object. Therefore, changes in the original object are not reflected in the copy.

copy.deepcopy() creates a Deep Copy.

Shallow copy creates a different object and populates it with the references of the child objects within the original object. Therefore, changes in the original object are reflected in the copy.

copy.copy creates a Shallow Copy.

2A)

Multithreading usually implies that multiple threads are executed concurrently. The Python Global Interpreter Lock doesn't allow more than one thread to hold the Python interpreter at that particular point of time. So multithreading in python is achieved through context switching. It is quite different from multiprocessing which actually opens up multiple processes across multiple threads.

3A)

Here you can also find a comprehensive guide on Python Django Tutorial that is very easy to understand.

Django is a web service used to build your web pages. Its architecture is as shown:

- Template: the front end of the web page
- Model: the back end where the data is stored
- View: It interacts with the model and template and maps it to the URL
- Django: serves the page to the user

4A)

Numpy is written in C so that all its complexities are backed into a simple to use a module. Lists, on the other hand, are dynamically typed. Therefore, Python

K. PRAKASH SENAPATI

must check the data type of each element every time it uses it. This makes Numpy arrays much faster than lists.

Numpy has a lot of additional functionality that list doesn't offer; for instance, a lot of things can be automated in Numpy.

5A)

Pickling	Unpickling
• Converting a Python object hierarchy to a byte stream is called pickling • Pickling is also referred to as serialization	• Converting a byte stream to a Python object hierarchy is called unpickling • Unpickling is also referred to as deserialization

If you just created a neural network model, you can save that model to your hard drive, pickle it, and then unpickle to bring it back into another software program or to use it at a later time.

6A)

Python has a private heap space that stores all the objects. The Python memory manager regulates various aspects of this heap, such as sharing, caching, segmentation, and allocation. The user has no control over the heap; only the Python interpreter has access.

7A)

Arguments are passed in python by a reference. This means that any changes made within a function are reflected in the original object.

Consider two sets of code shown below:

K. PRAKASH SENAPATI

```
def fun(l):
    l[0]=3
l=[1,2,3,4]
fun(l)
print(l)
```

Lists are mutable

A new object is created

```
def fun(l):
    l=[3,2,3,4]
l=[1,2,3,4]
fun(l)
print(l)
```

Output: [3,2,3,4]

Output: [1,2,3,4]

In the first example, we only assigned a value to one element of 'l', so the output is [3, 2, 3, 4].

In the second example, we have created a whole new object for 'l'. But, the values [3, 2, 3, 4] doesn't show up in the output as it is outside the definition of the function.

8A)

To generate random numbers in Python, you must first import the random module.

The random() function generates a random float value between 0 & 1.

> random.random()

The randrange() function generates a random number within a given range.

Syntax: randrange(beginning, end, step)

Example - > random.randrange(1,10,2)

9A)

In Python, the / operator performs division and returns the quotient in the float.

For example: 5 / 2 returns 2.5

The // operator, on the other hand, returns the quotient in integer.

For example: 5 // 2 returns 2

10A)

The 'is' operator compares the id of the two objects.

K. PRAKASH SENAPATI

list1=[1,2,3]

list2=[1,2,3]

list3=list1

list1 == list2 □ True

list1 is list2 □ False

list1 is list3 □ True

11A)

The pass statement is used when there's a syntactic but not an operational requirement. For example - The program below prints a string ignoring the spaces.

```
var="Si mplilea rn"

for i in var:

  if i==" ":

    pass

  else:

    print(i,end="")
```

12)

Python has an inbuilt method isalnum() which returns true if all characters in the string are alphanumeric.

Example -

>> "abcd123".isalnum()

Output: True

>>”abcd@123#”.isalnum()

Output: False

K. PRAKASH SENAPATI

Another way is to use regex as shown.

```
>>import re
>>bool(re.match('[A-Za-z0-9]+$','abcd123'))
```

Output: True

```
>> bool(re.match('[A-Za-z0-9]+$','abcd@123'))
```

Output: False

13A)

There are three types of sequences in Python:

- Lists
- Tuples
- Strings

Example of Lists -

```
>>l1=[1,2,3]
>>l2=[4,5,6]
>>l1+l2
```

Output: [1,2,3,4,5,6]

Example of Tuples -

```
>>t1=(1,2,3)
>>t2=(4,5,6)
>>t1+t2
```

Output: (1,2,3,4,5,6)

Example of String -

```
>>s1="Simpli"
>>s2="learn"
```

K. PRAKASH SENAPATI

>>s1+s2

Output: ‘Simplilearn’

14A)

Python provides the inbuilt function lstrip() to remove all leading spaces from a string.

>>“ Python”.lstrip

Output: Python

15A)

The replace() function can be used with strings for replacing a substring with a given string. Syntax:

str.replace(old, new, count)

replace() returns a new string without modifying the original string.

Example -

>>"Hey John. How are you, John?".replace(“john",“John",1)

Output: “Hey John. How are you, John?

16A)

del	remove()

K. PRAKASH SENAPATI

• del removes all elements of a list within a given range • Syntax: del list[start:end]	• remove() removes the first occurrence of a particular character • Syntax: list.remove(element)

Here is an example to understand the two statements -

```
>>lis=['a', 'b', 'c', 'd']
>>del lis[1:3]
>>lis
Output: ["a","d"]
>>lis=['a', 'b', 'b', 'd']
>>lis.remove('b')
>>lis
Output: ['a', 'b', 'd']
```

17A)

You can display the contents of a text file in reverse order using the following steps:

- Open the file using the open() function
- Store the contents of the file into a list
- Reverse the contents of the list
- Run a for loop to iterate through the list

18A)

append()	extend()
• append() adds an element to the end of the list • Example -	• extend() adds elements from an iterable to the end of the list • Example -

K.PRAKASH SENAPATI

>>lst=[1,2,3] >>lst.append(4) >>lst Output:[1,2,3,4]	>>lst=[1,2,3] >>lst.extend([4,5,6]) >>lst Output:[1,2,3,4,5,6]

19A)

```
>def addToList(val, list=[]):
>> list.append(val)
>> return list
>>list1 = addToList(1)
>>list2 = addToList(123,[])
>>list3 = addToList('a')
>>print ("list1 = %s" % list1)
>>print ("list2 = %s" % list2)
>>print ("list3 = %s" % list3)
```

Output:

list1 = [1,'a']

list2 = [123]

lilst3 = [1,'a']

Note that list1 and list3 are equal. When we passed the information to the addToList, we did it without a second value. If we don't have an empty list as the second value, it will start off with an empty list, which we then append. For list2, we appended the value to an empty list, so its value becomes [123].

For list3, we're adding 'a' to the list. Because we didn't designate the list, it is a shared value. It means the list doesn't reset and we get its value as [1, 'a'].

20A)

K.PRAKASH SENAPATI

Lists are mutable while tuples are immutable.

Example:

List

>>lst = [1,2,3]

>>lst[2] = 4

>>lst

Output:[1,2,4]

Tuple

>>tpl = (1,2,3)

>>tpl[2] = 4

>>tpl

Output:TypeError: 'tuple'

the object does not support item

assignment

There is an error because you can't change the tuple 1 2 3 into 1 2 4. You have to completely reassign tuple to a new value.

21A)

Lists are mutable while tuples are immutable.

Example:

List

>>lst = [1,2,3]

>>lst[2] = 4

>>lst

Output:[1,2,4]

Tuple

>>tpl = (1,2,3)

>>tpl[2] = 4

K. PRAKASH SENAPATI

>>tpl

Output:TypeError: 'tuple'

the object does not support item

assignment

There is an error because you can't change the tuple 1 2 3 into 1 2 4. You have to completely reassign tuple to a new value.

22A)

The solution to this depends on the Python version you are using.

Python v2

>>print(“Hi. ”),

>>print(“How are you?”)

Output: Hi. How are you?

Python v3

>>print(“Hi”,end=“ ”)

>>print(“How are you?”)

Output: Hi. How are you?

23A)

The solution to this depends on the Python version you are using.

Python v2

>>print(“Hi. ”),

>>print(“How are you?”)

Output: Hi. How are you?

Python v3

>>print(“Hi”,end=“ ”)

>>print(“How are you?”)

Output: Hi. How are you?

K. PRAKASH SENAPATI

24A)

Python is considered a multi-paradigm language.

Python follows the object-oriented paradigm

- Python allows the creation of objects and their manipulation through specific methods
- It supports most of the features of OOPS such as inheritance and polymorphism

Python follows the functional programming paradigm

- Functions may be used as the first-class object
- Python supports Lambda functions which are characteristic of the functional paradigm

25A)

The function prototype is as follows:

```
def function_name(*list)
>>def fun(*var):
>> for i in var:
print(i)
>>fun(1)
>>fun(1,25,6)
```

In the above code, * indicates that there are multiple arguments of a variable.

26A)

*args

- It is used in a function prototype to accept a varying number of arguments.
- It's an iterable object.
- Usage - def fun(*args)

K. PRAKASH SENAPATI

*kwargs

- It is used in a function prototype to accept the varying number of keyworded arguments.
- It's an iterable object
- Usage - def fun(**kwargs):

fun(colour="red".units=2)

27A)

It means that a function can be treated just like an object. You can assign them to variables, or pass them as arguments to other functions. You can even return them from other functions.

28A)

__name__is a special variable that holds the name of the current module. Program execution starts from main or code with 0 indentations. Thus, __name__has a value __main__in the above case. If the file is imported from another module, __name__holds the name of this module.

29A)

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of non-negative integers. The number of dimensions determines the rank of the array. The shape of an array is a tuple of integers giving the size of the array along each dimension.

30A)

Matrices	Arrays

K. PRAKASH SENAPATI

• A matrix comes from linear algebra and is a two-dimensional representation of data • It comes with a powerful set of mathematical operations that allow you to manipulate the data in interesting ways	• An array is a sequence of objects of similar data type • An array within another array forms a matrix

31A)

```
>>import numpy as np
>>arr=np.array([1, 3, 2, 4, 5])
>>print(arr.argsort( ) [ -N: ][: : -1])
```

32A)

```
>>train_set=np.array([1, 2, 3])
>>test_set=np.array([[0, 1, 2], [1, 2, 3])
```

Res_set □ [[1, 2, 3], [0, 1, 2], [1, 2, 3]]

Choose the correct option:

1. res_set = train_set.append(test_set)
2. res_set = np.concatenate([train_set, test_set]))
3. resulting_set = np.vstack([train_set, test_set])
4. None of these

Here, options a and b would both do horizontal stacking, but we want vertical stacking. So, option c is the right statement.

resulting_set = np.vstack([train_set, test_set])

33A)

1. from sklearn.decision_tree import DecisionTreeClassifier
2. from sklearn.ensemble import DecisionTreeClassifier
3. from sklearn.tree import DecisionTreeClassifier
4. None of these

Answer - 3. from sklearn.tree import DecisionTreeClassifier

K. PRAKASH SENAPATI

34A)

We can use the following code:

```
>>link = https://docs.google.com/spreadsheets/d/...
>>source = StringIO.StringIO(requests.get(link).content))
>>data = pd.read_csv(source)
```

35A)

df[‘Name’] and df.loc[:, ‘Name’], where:

df = pd.DataFrame(['aa', 'bb', 'xx', 'uu'], [21, 16, 50, 33], columns = ['Name', 'Age'])

Choose the correct option:

1. 1 is the view of original dataframe and 2 is a copy of original dataframe
2. 2 is the view of original dataframe and 1 is a copy of original dataframe
3. Both are copies of original dataframe
4. Both are views of original dataframe

Answer - 3. Both are copies of the original dataframe.

36A)

Error:

Traceback (most recent call last): File "<input>", line 1, in<module> UnicodeEncodeError:

'ascii' codec can't encode character.

Choose the correct option:

1. pd.read_csv(“temp.csv”, compression=’gzip’)

K. PRAKASH SENAPATI

2. pd.read_csv(“temp.csv”, dialect=’str’)
3. pd.read_csv(“temp.csv”, encoding=’utf-8′)
4. None of these

The error relates to the difference between utf-8 coding and a Unicode.

So option 3. pd.read_csv(“temp.csv”, encoding=’utf-8′) can correct it.

37A)

```
>>import matplotlib.pyplot as plt
>>plt.plot([1,2,3,4])
>>plt.show()
```

Choose the correct option:

1. In line two, write plt.plot([1,2,3,4], width=3)
2. In line two, write plt.plot([1,2,3,4], line_width=3
3. In line two, write plt.plot([1,2,3,4], lw=3)
4. None of these

Answer - 3. In line two, write plt.plot([1,2,3,4], lw=3)

38A)

1. df.reset_index(new_index,)
2. df.reindex(new_index,)
3. df.reindex_like(new_index,)
4. None of these

Answer - 3. df.reindex_like(new_index,)

39A)

The function used to copy objects in Python are:

K. PRAKASH SENAPATI

copy.copy for shallow copy and

copy.deepcopy() for deep copy

40A)

range()	xrange()
• range returns a Python list object	• xrange returns an xrange object

41A)

The attribute df.empty is used to check whether a pandas data frame is empty or not.

```
>>import pandas as pd
>>df=pd.DataFrame({A:[]})
>>df.empty
```

Output: True

42A)

This can be achieved by using argsort() function. Let us take an array X; the code to sort the (n-1)th column will be x[x [: n-2].argsoft()]

The code is as shown below:

```
>>import numpy as np
>>X=np.array([[1,2,3],[0,5,2],[2,3,4]])
>>X[X[:,1].argsort()]
```

Output:array([[1,2,3],[0,5,2],[2,3,4]])

43A)

```
>> #Input
```

K. PRAKASH SENAPATI

```
>>import numpy as np
>>import pandas as pd
>>mylist = list('abcedfghijklmnopqrstuvwxyz’)
>>myarr = np.arange(26)
>>mydict = dict(zip(mylist, myarr))
>> #Solution
>>ser1 = pd.Series(mylist)
>>ser2 = pd.Series(myarr)
>>ser3 = pd.Series(mydict)
>>print(ser3.head())
```

44A)

```
>> #Input
>>import pandas as pd
>>ser1 = pd.Series([1, 2, 3, 4, 5])
>>ser2 = pd.Series([4, 5, 6, 7, 8])
>> #Solution
>>ser_u = pd.Series(np.union1d(ser1, ser2)) # union
>>ser_i = pd.Series(np.intersect1d(ser1, ser2)) # intersect
>>ser_u[~ser_u.isin(ser_i)]
```

45A)

```
>> #Input
>>import pandas as pd
>>np.random.RandomState(100)
>>ser = pd.Series(np.random.randint(1, 5, [12]))
>> #Solution
>>print("Top 2 Freq:", ser.value_counts())
```

K.PRAKASH SENAPATI

```
>>ser[~ser.isin(ser.value_counts().index[:2])] = 'Other'
>>ser
```

46A)

```
>> #Input
>>import pandas as pd
>>ser = pd.Series(np.random.randint(1, 10, 7))
>>ser
>> #Solution
>>print(ser)
>>np.argwhere(ser % 3==0)
```

47A)

```
>> #Input
>>p = pd.Series([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
>>q = pd.Series([10, 9, 8, 7, 6, 5, 4, 3, 2, 1])
>> #Solution
>>sum((p - q)**2)**.5
>> #Solution using func
>>np.linalg.norm(p-q)
```

You can see that the Euclidean distance can be calculated using two ways.

48A)

```
>> #Input
>>df = pd.DataFrame(np.arange(25).reshape(5, -1))
>> #Solution
>>df.iloc[::-1, :]
```

49A)

Yes. One common beginner mistake is re-tuning a model or training new models with different parameters after seeing its performance on the test set.

K. PRAKASH SENAPATI

50A)

Seaborn is a Python library built on top of matplotlib and pandas to ease data plotting. It is a data visualization library in Python that provides a high-level interface for drawing statistical informative graphs.

Did you know the answers to these Python interview questions? If not, here is what you can do.

51A)

- Python is a scripting language. Python, unlike other programming languages like C and its derivatives, does not require compilation prior to execution.
- Python is dynamically typed, which means you don't have to specify the kinds of variables when declaring them or anything.
- Python is well suited to object-oriented programming since it supports class definition, composition, and inheritance.

52A)

Although Python can be used to write scripts, it is primarily used as a general-purpose programming language.

53A)

Any programming language that is not in machine-level code before runtime is called an interpreted language. Python is thus an interpreted language.

54A)

PEP denotes Python Enhancement Proposal. It's a collection of guidelines for formatting Python code for maximum readability.

K. PRAKASH SENAPATI

55A)

In Python, a namespace refers to the name that is assigned to each object.

56A)

Decorators are used for changing the appearance of a function without changing its structure. Decorators are typically defined prior to the function they are enhancing.

57A)

Decorators are typically defined prior to the function they are enhancing. To use a decorator, we must first specify its function. Then we write the function to which it is applied, simply placing the decorator function above the function to which it must be applied**.**

58A)

The .py files are the source code files for Python. The bytecode of the python files are stored in .pyc files, which are created when code is imported from another source. The interpreter saves time by converting the source .py files to .pyc files.

59A)

Slicing is a technique for gaining access to specific bits of sequences such as strings, tuples, and lists.

60A)

Slicing is a technique for gaining access to specific bits of sequences such as lists, tuples, and strings. The slicing syntax is [start:end:step]. This step can also be skipped. [start:end] returns all sequence items from the start (inclusive) to the end-1 element. It means the ith element from the end of the start or end element

K. PRAKASH SENAPATI

is negative i. The step represents the jump or the number of components that must be skipped.

61A)

In Python, keywords are reserved words with a specific meaning. They are commonly used to specify the type of variables. Variable and function names cannot contain keywords. Following are the 33 keywords of Python:

- Yield
- For
- Else
- Elif
- If
- Not
- Or
- And
- Raise
- Nonlocal
- None
- Is
- In
- Import
- Global
- From
- Finally
- Except
- Del
- Continue
- Class
- Assert

K. PRAKASH SENAPATI

- With
- Try
- False
- True
- Return
- Pass
- Lambda
- Def
- As
- Break
- While

62A)

This is one of the most commonly asked Python interview questions

The following are the ways through which the data frames in Pandas can be combined:

- Concatenating them by vertically stacking the two dataframes.
- Concatenating them by horizontally stacking the two dataframes.
- Putting them together in a single column.

63A)

- Zoneinfo and graphlib are two new modules.
- Improved modules such as asyncio and ast.
- Optimizations include improved idiom for assignment, signal handling, and Python built-ins.
- Removal of erroneous methods and functions.
- Instead of LL1, a new parser is based on PEG.
- Remove Prefixes and Suffixes with New String Methods.
- Generics with type hinting in standard collections.

K. PRAKASH SENAPATI

64A)

- Python's private heap space is in charge of memory management. A private heap holds all Python objects and data structures. This secret heap is not accessible to the programmer. Instead, the Python interpreter takes care of it.
- Python also includes a built-in garbage collector, which recycles all unused memory and makes it available to the heap space.
- Python's memory management is in charge of allocating heap space for Python objects. The core API allows programmers access to some programming tools.

65A)

it's an environment variable that is used when you import a module. When a module is imported, PYTHONPATH is checked to see if the imported modules are present in various folders. It is used by the interpreter to determine which module to load.

66A)

Local Variables:

A local variable is any variable declared within a function. This variable exists only in local space, not in global space.

Global Variables:

Global variables are variables declared outside of a function or in a global space. Any function in the program can access these variables.

67A)

Yes, Python is case sensitive.

68A)

K. PRAKASH SENAPATI

- Download Python from https://www.python.org/downloads/
- Install it on your computer. Using your command prompt, look for the location where PYTHON is installed on your computer by typing cmd python.
- Then, in advanced system settings, create a new variable called PYTHON_NAME and paste the copied path into it.
- Search the path variable, choose its value and select 'edit'.
- If the value doesn't have a semicolon at the end, add one, and then type %PYTHON HOME%

69A)

Indentation is required in Python. It designates a coding block. An indented block contains all of the code for loops, classes, functions, and so on. Typically, four space characters are used. Your code will not execute correctly if it is not indented, and it will also generate errors.

70A)

Script file should start with #!/usr/bin/env python.

71A)

Self is used to represent the class instance. In Python, you can access the class's attributes and methods with this keyword. It connects the attributes to the arguments. Self appears in a variety of contexts and is frequently mistaken for a term. Self is not a keyword in Python, unlike in C++.

72A)

For primitive data types, a literal in Python source code indicates a fixed value

73A)

K.PRAKASH SENAPATI

For primitive data types, a literal in Python source code indicates a fixed value. Following are the 5 types of literal in Python:

- String Literal: A string literal is formed by assigning some text to a variable that is contained in single or double-quotes. Assign the multiline text encased in triple quotes to produce multiline literals.
- Numeric Literal: They may contain numeric values that are floating-point values, integers, or complex numbers.
- Character Literal: It is made by putting a single character in double-quotes.
- Boolean Literal: True or False
- Literal Collections: There are four types of literals such as list collections, tuple literals, set literals, dictionary literals, and set literals.

74A)

Python modules aíe files that contain Python code. Functions, classes, oí vaíiables can be used in this code. A Python module is a .py file that contains code that may be executed. I'he following aíe the commonly used built-in modules:

- JSON
- data time
- íandom
- math
- sys
- OS

75A)

init is a constíuctoí oí method in Python. I'his method is used to allocate memoíy when a new object is cíeated.

76A)

A lambda function is a type of anonymous function. I'his function can take as many paíameteís as you want, but just one statement.

77A)

K. PRAKASH SENAPATI

Lambda is typically utilized in instances where an anonymous function is required for a short period of time. Lambda functions can be applied in two different ways:

- Assigning Lambda functions to a variable
- Wrapping Lambda function into another function

78A)

Continue	When a specified condition is met, the control is moved to the beginning of th loop, allowing some parts of the loop to be transferred.
Break	When a condition is met, the loop is terminated and control is passed to the next statement.
Pass	When you need a piece of code syntactically but don't want to execute it, use this. This is a null operation.

79A)

Iterators are things that can be iterated through or traversed.

80A)

In terms of functionality, xrange and range are essentially the same. They both provide you the option of generating a list of integers to use whatever you want. The sole difference between range and xrange is that range produces a Python list object whereas x range returns an xrange object. This is especially true if you are working with a machine that requires a lot of memory, such as a phone because range will utilize as much memory as it can to generate your array of numbers, which can cause a memory error and crash your program. It is a beast with a memory problem.

81A)

The Pickle module takes any Python object and converts it to a string representation, which it then dumps into a file using the dump method. This

K. PRAKASH SENAPATI

is known as pickling. Unpickling is the process of recovering original Python objects from a stored text representation.

82A)

Functions which return an iterable set of items are known as generators.

83A)

The assignment statement (= operator) in Python does not copy objects. Instead, it establishes a connection between the existing object and the name of the target variable. The copy module is used to make copies of an object in Python. Furthermore, the copy module provides two options for producing copies of a given object –

Deep Copy: Deep Copy recursively replicates all values from source to destination object, including the objects referenced by the source object.

```
from copy import copy, deepcopy

list_1 = [1, 2, [3, 5], 4]

## shallow copy

list_2 = copy(list_1)

list_2[3] = 7

list_2[2].append(6)

list_2    # output => [1, 2, [3, 5, 6], 7]

list_1    # output => [1, 2, [3, 5, 6], 4]

## deep copy

list_3 = deepcopy(list_1)

list_3[3] = 8

list_3[2].append(7)

list_3    # output => [1, 2, [3, 5, 6, 7], 8]

list_1    # output => [1, 2, [3, 5, 6], 4]
```

Shallow Copy: A bit-wise copy of an object is called a shallow copy. The values in the copied object are identical to those in the original object. If one

K. PRAKASH SENAPATI

of the values is a reference to another object, only its reference addresses are copied.

84A)

Pass by value: The actual item's copy is passed. Changing the value of the object's copy has no effect on the original object's value.

Pass by reference: The actual object is passed as a reference. The value of the old object will change if the value of the new object is changed.

Arguments are passed by reference in Python.

```
def appendNumber(arr):
  arr.append(4)
arr = [1, 2, 3]
print(arr)  #Output: => [1, 2, 3]
appendNumber(arr)
print(arr)  #Output: => [1, 2, 3, 4]
```

85A

Use command os.remove(file_name) to delete a file in Python.

86A)

The join() function can be used to combine a list of strings based on a delimiter into a single string.

The split() function can be used to split a string into a list of strings based on a delimiter.

```
string = "This is a string."
string_list = string.split(' ') #delimiter is ‘space’ character or ‘ ‘
print(string_list) #output: ['This', 'is', 'a', 'string.']
print(' '.join(string_list)) #output: This is a string.
```

K. PRAKASH SENAPATI

87A)

*args

- The function definition uses the *args syntax to pass variable-length parameters.
- "*" denotes variable length, while "args" is the standard name. Any other will suffice.

**kwargs

- **kwargs is a special syntax for passing variable-length keyworded arguments to functions.
- When a variable is passed to a function, it is called a keyworded argument.
- "Kwargs" is also used by convention here. You are free to use any other name.

88A)

- The indexes from the end of the list, tuple, or string are called negative indexes.
- Arr[-1] denotes the array's last element. Arr[]

89A)

The capitalize() function in Python capitalizes a string's initial letter. It returns the original text if the string already contains a capital letter at the beginning.

90A)

The lower() function can be used to convert a string to lowercase.

91A)

Comments that involve multiple lines are known as multi-line comments. A # must prefix all lines that will be commented. You can also use a convenient shortcut to remark several lines. All you have to do is hold down

K. PRAKASH SENAPATI

the ctrl key and left-click anywhere you want a # character to appear, then input a # once. This will add a comment to every line where you put your cursor.

92A)

Docstrings are documentation strings. Within triple quotations are these docstrings. They are not allocated to any variable and, as a result, they can also be used as comments.

93A)

Special functions are known as operators. They take one or more input values and output a result.

not- returns the boolean value's inverse

is- returns true when both operands are true

in- determines whether a certain element is present in a series

94A)

Both help() and dir() are available from the Python interpreter and are used to provide a condensed list of built-in functions.

dir() function: The defined symbols are displayed using the dir() function.

help() function: The help() function displays the documentation string and also allows you to access help for modules, keywords, attributes, and other items.

95A)

- When Python quits, some Python modules, especially those with circular references to other objects or objects referenced from global namespaces, are not necessarily freed or deallocated.

K. PRAKASH SENAPATI

- Python would try to de-allocate/destroy all other objects on exit because it has its own efficient cleanup mechanism.
- It is difficult to de-allocate memory that has been reserved by the C library.
-

96A)

Dictionary is one of Python's built-in datatypes. It establishes a one-to-one correspondence between keys and values. Dictionary keys and values are stored in pairs in dictionaries. Keys are used to index dictionaries.

97A)

The Ternary operator is the operator for displaying conditional statements. This is made of true or false values and a statement that must be evaluated.

98A)

Python's "re" module provides three ways for modifying strings. They are:

split (): a regex pattern is used to "separate" a string into a list

subn(): It works similarly to sub(), returning the new string as well as the number of replacements.

sub(): identifies all substrings that match the regex pattern and replaces them with a new string

99A)

Python sequences are indexed, and they include both positive and negative values. Positive numbers are indexed with '0' as the first index and '1' as the second index, and so on.

The index for a negative number begins with '-1,' which is the last index in the sequence, and ends with '-2,' which is the penultimate index, and the sequence continues like a positive number. The negative index is used to

K. PRAKASH SENAPATI

eliminate all new-line spaces from the string and allow it to accept the last character S[:-1]. The negative index can also be used to represent the correct order of the string.

100A)

Packages in Python are namespaces that contain numerous modules.

K. PRAKASH SENAPATI