**NumPy Interview Questions**

1. **Array Creation & Initialization**
   * How do you create a NumPy array of shape (3, 3) filled with random integers between 1 and 10?
2. **Array Properties**
   * Given an array arr = np.array([[1,2,3],[4,5,6]]), how would you find its **shape**, **number of dimensions**, and **data type**?
3. **Reshaping & Flattening**
   * What is the difference between reshape() and ravel()? When would you prefer ravel() over flatten()?
4. **Indexing & Slicing**
   * Given arr = np.arange(10), how would you extract all even numbers?
5. **Stacking & Splitting**
   * How do np.vstack, np.hstack, and np.concatenate differ? Can you give an example where concatenate is more flexible?
6. **Mathematical Operations**
   * How would you compute the **row-wise mean** and **column-wise standard deviation** of a 2D NumPy array?
7. **Aggregation Functions**
   * Given arr = np.array([1, 2, np.nan, 4]), how do you compute the **sum** while ignoring NaN values?
8. **Sorting & Searching**
   * What is the difference between np.sort() and np.argsort()? Can you show how to get indices of the top 3 largest values in an array?
9. **Linear Algebra**
   * How do you compute the **dot product** and **matrix inverse** using NumPy? What will happen if the matrix is singular?
10. **Performance & Memory**

* Why is NumPy faster than native Python lists for numerical computations? Can you explain the role of **contiguous memory** and **vectorization**?

ANSWERS

Here’s a **NumPy interview Q&A cheat sheet** with short, to-the-point answers:

**NumPy Interview Questions & Short Answers**

**Array Creation & Initialization**

np.random.randint(1, 11, size=(3, 3))

Creates a 3×3 array with random integers between 1–10.

**Array Properties**

arr.shape # (2, 3)

arr.ndim # 2

arr.dtype # int64 (depends on system)

**Reshaping & Flattening**

reshape() → returns new view (if possible) with different shape.

ravel() → returns flattened **view** (no copy if possible).

flatten() → always returns a **copy**.

Prefer ravel() for performance when copy not needed.

**Indexing & Slicing**

arr = np.arange(10)

arr[arr % 2 == 0] # array([0, 2, 4, 6, 8])

**Stacking & Splitting**

np.vstack([a,b]) # stack vertically (rows)

np.hstack([a,b]) # stack horizontally (cols)

np.concatenate([a,b], axis=0/1) # more flexible (choose axis)

**Mathematical Operations**

arr.mean(axis=1) # row-wise mean

arr.std(axis=0) # column-wise std

**Aggregation Functions**

np.nansum(arr) # ignores NaN

**Sorting & Searching**

np.sort(arr) # sorted array

np.argsort(arr) # indices of sorted elements

arr[np.argsort(arr)[-3:]] # top 3 largest values

**Linear Algebra**

np.dot(A, B) # dot product

np.linalg.inv(A) # inverse (error if singular)

**Performance & Memory**

NumPy arrays are stored in **contiguous memory blocks**.

Uses **vectorized C/Fortran routines** under the hood.

Avoids Python loops → much faster than lists.