**NumPy**

**Top 100 NumPy interview questions, covering basic to advanced topics.**

**Basic NumPy Questions**

**1. What is NumPy?**

NumPy (Numerical Python) is a powerful library for numerical computing in Python, providing support for large, multi-dimensional arrays and mathematical operations.

**2. How do you install NumPy?**

pip install numpy

**3. How do you import NumPy?**

import numpy as np

**4. How to create a 1D NumPy array?**

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

print(arr) # Output: [1 2 3 4 5]

**5. How to create a 2D NumPy array?**

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

print(arr)

**6. How to check the version of NumPy?**

print(np.\_\_version\_\_)

**7. How to create an array of zeros?**

zeros = np.zeros((3, 3))

print(zeros)

**8. How to create an array of ones?**

ones = np.ones((2, 3))

print(ones)

**9. How to create an identity matrix?**

identity = np.eye(4)

print(identity)

**10. How to create an array with a range of numbers?**

arr = np.arange(1, 10, 2)

print(arr) # Output: [1 3 5 7 9]

**Array Manipulation Questions**

**11. How to reshape a NumPy array?**

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

reshaped\_arr = arr.reshape((2, 3))

print(reshaped\_arr)

**12. How to flatten a NumPy array?**

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

flat\_arr = arr.flatten()

print(flat\_arr) # Output: [1 2 3 4 5 6]

**13. How to concatenate NumPy arrays?**

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

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

concat\_arr = np.concatenate((a, b))

print(concat\_arr)

**14. How to stack arrays vertically?**

vstacked = np.vstack((a, b))

print(vstacked)

**15. How to stack arrays horizontally?**

hstacked = np.hstack((a, b))

print(hstacked)

**Indexing and Slicing Questions**

**16. How to access elements in a NumPy array?**

arr = np.array([10, 20, 30, 40])

print(arr[2]) # Output: 30

**17. How to slice a NumPy array?**

arr = np.array([10, 20, 30, 40, 50])

print(arr[1:4]) # Output: [20 30 40]

**18. How to get elements at specific indices?**

arr = np.array([10, 20, 30, 40, 50])

indices = [0, 2, 4]

print(arr[indices]) # Output: [10 30 50]

**19. How to filter elements based on a condition?**

arr = np.array([10, 20, 30, 40, 50])

filtered = arr[arr > 25]

print(filtered) # Output: [30 40 50]

**Mathematical Operations**

**20. How to perform element-wise addition in NumPy?**

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

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

result = a + b

print(result) # Output: [5 7 9]

**21. How to calculate the mean of an array?**

arr = np.array([10, 20, 30])

mean\_value = np.mean(arr)

print(mean\_value) # Output: 20.0

**22. How to calculate the median of an array?**

median\_value = np.median(arr)

print(median\_value) # Output: 20.0

**23. How to calculate the standard deviation?**

std\_value = np.std(arr)

print(std\_value)

**24. How to compute the dot product of two arrays?**

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

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

dot\_product = np.dot(a, b)

print(dot\_product) # Output: 32

**Advanced NumPy Questions**

**25. How to find unique elements in an array?**

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

unique\_values = np.unique(arr)

print(unique\_values) # Output: [1 2 3 4]

**26. How to compute the cumulative sum of an array?**

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

cumsum = np.cumsum(arr)

print(cumsum) # Output: [ 1 3 6 10]

**27. How to generate a random integer matrix?**

random\_matrix = np.random.randint(1, 100, (3, 3))

print(random\_matrix)

**28. How to sort a NumPy array?**

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

sorted\_arr = np.sort(arr)

print(sorted\_arr) # Output: [1 2 3 4]

**29. What is broadcasting in NumPy?**

NumPy allows arrays with different shapes to be combined mathematically without creating extra copies.

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

result = arr + 10

print(result) # Output: [11 12 13]

**1. How to get the shape of an array?**

You can use the .shape attribute to get the shape of a NumPy array.

import numpy as np

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

print(arr.shape) # Output: (2, 3)

This means the array has **2 rows** and **3 columns**.

**2. How to get the datatype of an array?**

Use the .dtype attribute to find the datatype of the elements in the array.

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

print(arr.dtype) # Output: int32 or int64 (depends on the system)

arr\_float = np.array([1.2, 3.4, 5.6])

print(arr\_float.dtype) # Output: float64

**3. How to create a boolean mask?**

A boolean mask is used to filter elements based on conditions.

arr = np.array([10, 20, 30, 40, 50])

mask = arr > 25 # Creates a boolean mask

print(mask) # Output: [False False True True True]

filtered\_arr = arr[mask] # Use mask to filter values

print(filtered\_arr) # Output: [30 40 50]

**4. How to generate a normal distribution?**

The np.random.normal() function generates values from a normal (Gaussian) distribution.

mean = 0 # Mean of the distribution

std\_dev = 1 # Standard deviation

size = (3, 3) # Shape of the array

normal\_dist = np.random.normal(mean, std\_dev, size)

print(normal\_dist)

This generates a **3x3 matrix** of random numbers from a normal distribution.

**5. How to apply a function to all elements?**

You can use np.vectorize() or NumPy’s built-in functions.

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

# Using a lambda function to square each element

squared = np.vectorize(lambda x: x \*\* 2)(arr)

print(squared) # Output: [ 1 4 9 16 25]

# Using a built-in function (e.g., np.sqrt)

sqrt\_arr = np.sqrt(arr)

print(sqrt\_arr) # Output: [1. 1.41 1.73 2. 2.23]

**6. How to convert a NumPy array to a list?**

Use the .tolist() method to convert a NumPy array into a Python list.

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

list\_arr = arr.tolist()

print(list\_arr) # Output: [1, 2, 3, 4]

**7. How to save and load NumPy arrays?**

Use np.save() to save and np.load() to load NumPy arrays.

**Saving an array**

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

np.save('my\_array.npy', arr)

**Loading the saved array**

loaded\_arr = np.load('my\_array.npy')

print(loaded\_arr) # Output: [1 2 3 4 5]

**8. What is np.linalg.inv()?**

np.linalg.inv() computes the **inverse of a matrix**.

matrix = np.array([[4, 7], [2, 6]])

inverse\_matrix = np.linalg.inv(matrix)

print(inverse\_matrix)

The inverse of a matrix **A** is a matrix **A⁻¹** such that **A × A⁻¹ = I (Identity matrix)**.

**9. How to find the rank of a matrix?**

Use np.linalg.matrix\_rank() to find the rank of a matrix.

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

rank = np.linalg.matrix\_rank(matrix)

print(rank) # Output: 2 (for this singular matrix)

The **rank** is the number of linearly independent rows or columns.

**10. How to calculate eigenvalues?**

Eigenvalues are calculated using np.linalg.eig().

matrix = np.array([[4, -2], [1, 1]])

eigenvalues, eigenvectors = np.linalg.eig(matrix)

print("Eigenvalues:", eigenvalues)

print("Eigenvectors:\n", eigenvectors)

Eigenvalues represent **scaling factors** in transformation.

**11. What is np.meshgrid()?**

np.meshgrid() creates coordinate grids for vectorized evaluations.

**Example:**

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

y = np.array([4, 5])

X, Y = np.meshgrid(x, y)

print("X:\n", X)

print("Y:\n", Y)

**Output:**

X:

[[1 2 3]

[1 2 3]]

Y:

[[4 4 4]

[5 5 5]]

It helps in **3D plotting** and **function evaluations** over a grid.

**Summary of Answers**

| **Question** | **Method/Function** |
| --- | --- |
| Get shape of an array | .shape |
| Get datatype of an array | .dtype |
| Create a boolean mask | Conditional indexing (arr[arr > x]) |
| Generate normal distribution | np.random.normal(mean, std, size) |
| Apply function to all elements | np.vectorize() or NumPy functions |
| Convert NumPy array to list | .tolist() |
| Save and load NumPy arrays | np.save() / np.load() |
| Compute inverse of a matrix | np.linalg.inv() |
| Find rank of a matrix | np.linalg.matrix\_rank() |
| Compute eigenvalues | np.linalg.eig() |
| Create coordinate grids | np.meshgrid() |

**Basic NumPy Theory Questions**

**1. What is NumPy? Why is it used?**

**Answer:**

NumPy (Numerical Python) is a Python library used for numerical computing. It provides support for large, multi-dimensional arrays, matrices, and mathematical functions to operate on these arrays efficiently. NumPy is widely used in **data science, machine learning, and scientific computing** due to its speed and efficiency.

**2. What are the key features of NumPy?**

**Answer:**

* **Multidimensional Arrays:** Supports n-dimensional arrays (ndarray).
* **Mathematical Operations:** Provides fast element-wise operations.
* **Broadcasting:** Allows operations on arrays of different shapes.
* **Linear Algebra:** Functions like inverse, determinant, and eigenvalues.
* **Random Sampling:** Tools for generating random numbers.
* **Integration with Other Libraries:** Used in Pandas, SciPy, TensorFlow, etc.

**3. How is NumPy different from Python lists?**

**Answer:**

| **Feature** | **NumPy Arrays** | **Python Lists** |
| --- | --- | --- |
| Memory Usage | Less (efficient memory storage) | More (due to pointers & references) |
| Speed | Faster (written in C) | Slower |
| Operations | Supports vectorized operations | Requires loops for element-wise |
| Fixed Data Type | Yes | Operations.  No (stores mixed data types) |

Example: NumPy is **much faster** than lists for large computations.

**4. What are NumPy arrays (ndarray)?**

**Answer:**

NumPy’s main object is the ndarray (n-dimensional array), which is a collection of elements of the **same data type**.

Example:

import numpy as np

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

print(arr.shape) # Output: (2, 3)

**5. What are the different ways to create a NumPy array?**

**Answer:**

* **From a list or tuple:** np.array([1, 2, 3])
* **Using arange():** np.arange(1, 10, 2)
* **Using zeros():** np.zeros((3, 3))
* **Using ones():** np.ones((2, 2))
* **Using linspace():** np.linspace(0, 10, 5)

**6. What is the difference between arange() and linspace()?**

**Answer:**

| **Function** | **Purpose** | **Example** |
| --- | --- | --- |
| np.arange(start, stop, step) | Generates numbers with a fixed step size | np.arange(1, 10, 2) → [1 3 5 7 9] |
| np.linspace(start, stop, num) | Generates num evenly spaced numbers between start and stop | np.linspace(0, 10, 5) → [0. 2.5 5. 7.5 10.] |

**7. What is NumPy broadcasting?**

**Answer:**

Broadcasting allows NumPy to perform operations on arrays of **different shapes** without explicitly replicating the smaller array.

Example:

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

print(arr + 10) # Output: [11 12 13]

Here, 10 is **broadcasted** to match the shape of arr.

**8. What is the difference between reshape() and ravel()?**

**Answer:**

| **Function** | **Purpose** | **Example** |
| --- | --- | --- |
| reshape(shape) | Changes the shape of the array | arr.reshape((2, 3)) |
| ravel() | without modifying its data  Flattens the array into a 1D array | arr.ravel() |

Example:

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

print(arr.reshape((3, 2))) # Reshaped to 3x2

print(arr.ravel()) # Flattened to 1D

**9. What are NumPy’s aggregate functions?**

**Answer:**

NumPy provides functions like:

* np.sum(arr): Sum of elements
* np.mean(arr): Mean value
* np.median(arr): Median value
* np.std(arr): Standard deviation
* np.max(arr), np.min(arr): Max and min values

Example:

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

print(np.sum(arr)) # Output: 10

print(np.mean(arr)) # Output: 2.5

**10. What is np.linalg.inv() used for?**

**Answer:**

It computes the **inverse of a square matrix**.

matrix = np.array([[4, 7], [2, 6]])

inverse\_matrix = np.linalg.inv(matrix)

print(inverse\_matrix)

**11. What is np.meshgrid()?**

**Answer:**

It creates a **grid of coordinates** for 2D or 3D plotting.

Example:

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

y = np.array([4, 5])

X, Y = np.meshgrid(x, y)

print(X)

print(Y)

This is useful in **contour plotting** and **surface plotting**.

**12. How does NumPy handle missing values?**

**Answer:**

NumPy does **not have built-in support for missing values**, but it uses np.nan (Not a Number) to represent them.

Example:

arr = np.array([1, 2, np.nan, 4])

print(np.isnan(arr)) # Output: [False False True False]

**13. What is the difference between np.vstack() and np.hstack()?**

**Answer:**

| **Function** | **Purpose** | **Example** |
| --- | --- | --- |
| np.vstack((a, b)) | Stacks arrays vertically (row-wise) | [[1 2 3] [4 5 6]] |
| np.hstack((a, b)) | Stacks arrays horizontally (column-wise) | [1 2 3 4 5 6] |

Example:

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

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

print(np.vstack((a, b))) # Vertical stacking

print(np.hstack((a, b))) # Horizontal stacking

**Here are the top 30 NumPy interview questions with answers for an AI Engineer role:**

**Basic NumPy Questions**

**1. What is NumPy, and why is it used in AI and Data Science?**

**Answer:**  
NumPy (Numerical Python) is a **library for numerical computing** in Python. It provides a powerful **ndarray** object for handling large datasets efficiently. It is widely used in AI and Data Science for:

* Fast numerical computations
* Handling large multi-dimensional arrays
* Linear algebra, Fourier transforms, and random number generation

**2. How does NumPy differ from Python lists in terms of performance and memory efficiency?**

**Answer:**

* **Performance:** NumPy is faster because it uses **contiguous memory allocation** and performs operations using **compiled C code**.
* **Memory Efficiency:** NumPy arrays require **less memory** than Python lists due to fixed data types and better memory management.

**3. What is the primary data structure in NumPy, and how is it different from Python lists?**

**Answer:**  
The primary data structure in NumPy is **ndarray (n-dimensional array)**.

* It is **homogeneous** (stores elements of the same type).
* More memory efficient than Python lists.
* Supports **vectorized operations**, making computations faster.

**4. Explain the significance of ndarray in NumPy.**

**Answer:**  
ndarray is the core data structure in NumPy that allows operations on large datasets efficiently. It provides:

* Multi-dimensional array handling
* Broadcasting and vectorized computations
* Fast mathematical operations

**5. What are the different ways to create a NumPy array?**

**Answer:**  
NumPy arrays can be created using:

* np.array([1, 2, 3]) – From a list or tuple
* np.zeros((3,3)) – Array of zeros
* np.ones((2,2)) – Array of ones
* np.full((3,3), 7) – Array of a constant value
* np.linspace(0, 10, 5) – Evenly spaced numbers
* np.arange(1, 10, 2) – Sequence of numbers

**NumPy Array Operations**

**6. What is the difference between np.array(), np.asarray(), and np.copy()?**

**Answer:**

* np.array() – Creates a new array from a list/tuple.
* np.asarray() – Converts an existing array-like object (without copying if it’s already an array).
* np.copy() – Creates a deep copy of an array.

**7. How do you create an array of all zeros, ones, or a constant value in NumPy?**

**Answer:**

np.zeros((3,3)) # 3x3 array of zeros

np.ones((2,2)) # 2x2 array of ones

np.full((3,3), 5) # 3x3 array filled with 5

**8. How do you generate a NumPy array with a sequence of numbers?**

**Answer:**

np.arange(1, 10, 2) # [1, 3, 5, 7, 9]

np.linspace(0, 10, 5) # [0, 2.5, 5, 7.5, 10]

**9. What is the difference between np.linspace() and np.arange()?**

**Answer:**

* np.arange(start, stop, step): Generates values with a fixed **step size**.
* np.linspace(start, stop, num): Generates values with a fixed **number of elements**.

**10. How do you generate random numbers using NumPy?**

**Answer:**

np.random.rand(3,3) # Uniform distribution

np.random.randn(3,3) # Standard normal distribution

np.random.randint(1,10, (2,2)) # Random integers

**Indexing & Slicing**

**11. How does slicing work in NumPy arrays compared to Python lists?**

**Answer:**  
Slicing in NumPy returns a **view**, not a copy (unlike lists).

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

print(arr[1:4]) # Output: [2 3 4]

**12. What is the difference between shallow copy and deep copy in NumPy?**

**Answer:**

* **Shallow Copy (view)**: Shares the same memory location.
* **Deep Copy (copy)**: Creates a new memory location.

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

view\_arr = arr.view() # Shallow copy

copy\_arr = arr.copy() # Deep copy

**13. How can you extract specific rows and columns from a NumPy array?**

**Answer:**

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

print(arr[1, :]) # Second row

print(arr[:, 2]) # Third column

**14. What is Boolean indexing in NumPy?**

**Answer:**  
Filtering elements using conditions.

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

print(arr[arr > 2]) # Output: [3 4 5]

**15. How can you use fancy indexing in NumPy?**

**Answer:**  
Using integer arrays as indices.

arr = np.array([10, 20, 30, 40, 50])

print(arr[[0, 3, 4]]) # Output: [10 40 50]

**Mathematical & Statistical Functions**

**16. What are the commonly used mathematical functions in NumPy?**

**Answer:**

* np.sum(), np.mean(), np.std(), np.min(), np.max(), np.exp(), np.log(), np.sqrt()

**17. How do you compute mean, median, variance, and standard deviation?**

**Answer:**

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

print(np.mean(arr)) # 3.0

print(np.median(arr)) # 3.0

print(np.var(arr)) # 2.0

print(np.std(arr)) # 1.41

**18. What is the purpose of np.dot()?**

**Answer:**  
Computes dot product of two arrays.

a = np.array([1,2])

b = np.array([3,4])

print(np.dot(a, b)) # Output: 11

**19. How can you normalize a NumPy array?**

**Answer:**

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

normalized = (arr - np.min(arr)) / (np.max(arr) - np.min(arr))

**Linear Algebra & AI Applications**

**20. How does NumPy handle matrix inversion?**

**Answer:**  
Using np.linalg.inv() for square matrices.

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

inv\_A = np.linalg.inv(A)

**Reshaping & Broadcasting**

**21. What is broadcasting in NumPy, and why is it useful?**

**Answer:**  
Broadcasting allows NumPy to perform operations on arrays of **different shapes** without explicitly replicating data.  
Example:

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

b = 2

print(a \* b) # Output: [2 4 6]

NumPy **automatically expands** b to match the shape of a.

**22. How does the .reshape() function work, and how is it different from .ravel()?**

**Answer:**

* .reshape() changes the shape of an array **without modifying data**.
* .ravel() returns a **flattened (1D) view** of the array.

Example:

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

print(arr.reshape(4, 1))

print(arr.ravel()) # Output: [1 2 3 4]

**23. Explain the difference between .flatten() and .ravel() in NumPy.**

**Answer:**

* .flatten() **returns a copy** of the array.
* .ravel() **returns a view** (if possible, otherwise a copy).

Example:

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

flat\_copy = arr.flatten()

flat\_view = arr.ravel()

**24. What is the purpose of np.newaxis in reshaping arrays?**

**Answer:**  
np.newaxis increases the **dimension of an array**.  
Example:

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

print(arr[:, np.newaxis]) # Converts (3,) to (3,1)

**25. How do you merge or split arrays using NumPy?**

**Answer:**

* **Concatenation**: np.concatenate((a, b), axis=0)
* **Stacking**: np.vstack((a, b)), np.hstack((a, b))
* **Splitting**: np.split(arr, sections), np.hsplit(arr, sections)

Example:

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

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

print(np.concatenate((a, b), axis=0)) # Stack rows

print(np.hstack((a, b))) # Stack horizontally

**Linear Algebra & AI Applications**

**26. How does NumPy handle matrix operations like inversion and determinant calculation?**

**Answer:**  
NumPy provides **linear algebra functions**:

* np.linalg.inv(A) → Matrix Inversion
* np.linalg.det(A) → Determinant

Example:

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

print(np.linalg.inv(A)) # Inverse matrix

print(np.linalg.det(A)) # Determinant

**27. How is NumPy used in deep learning and AI applications?**

**Answer:**

* **Tensor Operations**: NumPy handles multi-dimensional tensors for deep learning.
* **Preprocessing**: Used for image & text preprocessing.
* **Optimized Computation**: Efficient matrix multiplications (like in neural networks).
* **Feature Engineering**: Feature scaling, normalization, etc.

Example: Converting images to NumPy arrays in AI:

from PIL import Image

img = Image.open("image.jpg")

arr = np.array(img) # Convert image to NumPy array

**28. What is the use of np.linalg.eig() in AI and ML?**

**Answer:**  
Computes **eigenvalues and eigenvectors** of a matrix, essential in **Principal Component Analysis (PCA)**.

Example:

A = np.array([[2, -1], [-1, 2]])

eigenvalues, eigenvectors = np.linalg.eig(A)

print(eigenvalues)

print(eigenvectors)

**29. How can NumPy help in performing Singular Value Decomposition (SVD)?**

**Answer:**  
SVD is widely used in **dimensionality reduction, image compression, and ML models**.

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

U, S, V = np.linalg.svd(A)

print(U) # Left singular vectors

print(S) # Singular values

print(V) # Right singular vectors

**30. What is the difference between np.linalg.inv() and np.linalg.pinv()?**

**Answer:**

* np.linalg.inv(A): Computes **exact inverse**, only for **square, non-singular matrices**.
* np.linalg.pinv(A): Computes **Moore-Penrose pseudo-inverse**, used when **exact inversion is not possible** (useful for AI & ML).

Example:

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

print(np.linalg.inv(A)) # Exact inverse

print(np.linalg.pinv(A)) # Pseudo-inverse