NumPy (Numerical Python)

NumPy is a free, open-source Python library for scientific computing. It's used for complex mathematical operations, such as linear algebra, statistical operations.

1. Array Creation Functions

- np.array(data) → Creates an array.
- np.zeros(shape) → Creates an array of zeros.
- np.ones(shape) → Creates an array of ones.
- np.empty(shape) → Creates an uninitialized array.
- np.arange(start, stop, step) → Creates an array with evenly spaced values.
- np.linspace(start, stop, num) → Creates an array with a specified number of values between start and stop.
- np.random.rand(shape) → Generates an array of random numbers (uniform distribution).
- np.random.randn(shape) → Generates random numbers (normal distribution).

---

2. Array Manipulation Functions

- np.reshape(a, newshape) → Reshapes an array.
- np.transpose(a) → Transposes the array.
- np.flatten() → Flattens a multi-dimensional array into a 1D array.
- np.concatenate((a1, a2), axis) → Concatenates arrays along an axis.
- np.vstack((a1, a2)) → Stacks arrays vertically.
- np.hstack((a1, a2)) → Stacks arrays horizontally.
- np.split(a, indices) → Splits an array into sub-arrays.

---

3. Mathematical Functions

- **Basic Math**:
  np.add(a, b), np.subtract(a, b), np.multiply(a, b), np.divide(a, b).
- **Exponents and Logarithms**:
  np.exp(a), np.log(a), np.log10(a), np.sqrt(a).
- **Trigonometric Functions**:
  np.sin(a), np.cos(a), np.tan(a), np.arcsin(a).
- **Statistical Functions**:
  np.mean(a), np.median(a), np.std(a), np.var(a), np.sum(a), np.prod(a).

---

4. Indexing and Slicing Functions

- np.where(condition) → Returns indices where the condition is True.
- np.take(a, indices) → Selects elements from an array.
- np.nonzero(a) → Returns indices of non-zero elements.

---

5. Linear Algebra Functions

- np.dot(a, b) → Dot product of two arrays.
- np.matmul(a, b) → Matrix multiplication.
- np.linalg.inv(a) → Inverse of a matrix.
- np.linalg.det(a) → Determinant of a matrix.
- np.linalg.eig(a) → Eigenvalues and eigenvectors of a matrix.

---

6. Sorting and Searching Functions

- np.sort(a, axis) → Sorts an array.
- np.argsort(a) → Returns indices that would sort an array.
- np.argmin(a) → Index of the minimum value.
- np.argmax(a) → Index of the maximum value.
- np.unique(a) → Returns the sorted unique elements of an array.

---

7. Broadcasting and Vectorized Operations

- np.tile(a, reps) → Repeats an array.
- np.repeat(a, repeats) → Repeats elements of an array.

---

8. Logical Operations

- np.logical_and(a, b) → Element-wise logical AND.
- np.logical_or(a, b) → Element-wise logical OR.
- np.logical_not(a) → Element-wise NOT.
- np.all(a) → Checks if all elements are True.
- np.any(a) → Checks if any element is True.

---

9. Random Number Generation

- np.random.seed(seed) → Sets the seed for random number generation.
- np.random.randint(low, high, size) → Generates random integers.
- np.random.choice(a, size) → Randomly selects elements.

---

10. Utility Functions

- np.clip(a, a_min, a_max) → Limits the values in an array.
- np.cumsum(a) → Cumulative sum of elements.
- np.cumprod(a) → Cumulative product of elements.
- np.isclose(a, b) → Checks if values are close.
- np.isnan(a) → Checks for NaN values.

1. Array Creation Functions

1. numpy.array()

- **Definition**: Creates an array from a list or tuple.
- **Syntax**: numpy.array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)

- **Examples**:

```
import numpy as np
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
print(arr1)
print(arr2)
```

Output:

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

---

2. numpy.zeros()

- **Definition**: Creates an array filled with zeros.
- **Syntax**: numpy.zeros(shape, dtype=float, order='C')

- **Examples**:

```
arr1 = np.zeros(5)
arr2 = np.zeros((2, 3), dtype=int)
print(arr1)
print(arr2)
```

Output:

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

---

3. numpy.ones()

- **Definition**: Creates an array filled with ones.
- **Syntax**: numpy.ones(shape, dtype=float, order='C')

- **Examples**:

```
arr1 = np.ones(4)
arr2 = np.ones((2, 2), dtype=int)
print(arr1)
print(arr2)
```

Output:

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

---

4. numpy.full()

- **Definition**: Creates an array filled with a specified value.
- **Syntax**: numpy.full(shape, fill_value, dtype=None, order='C')

- **Examples**:

```
arr1 = np.full(3, 7)
arr2 = np.full((2, 3), 5.5)
print(arr1)
print(arr2)
```

Output:

```
[7 7 7]
[[5.5 5.5 5.5]
 [5.5 5.5 5.5]]
```

---

5. numpy.arange()

- **Definition**: Creates an array with evenly spaced values within a given range.
- **Syntax**: numpy.arange(start, stop, step, dtype=None)

- **Examples**:

```
arr1 = np.arange(5)
arr2 = np.arange(2, 10, 2)
print(arr1)
print(arr2)
```

Output:

```
[0 1 2 3 4]
[2 4 6 8]
```

---

6. numpy.linspace()

- **Definition**: Creates an array of evenly spaced numbers over a specified range.
- **Syntax**: numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)

- **Examples**:

```
arr1 = np.linspace(1, 10, 5)
arr2 = np.linspace(0, 1, 4, endpoint=False)
print(arr1)
print(arr2)
```

Output:

```
[ 1.   3.25  5.5   7.75 10.  ]
[0.   0.25 0.5  0.75]
```

---

7. numpy.eye()

- **Definition**: Creates an identity matrix (diagonal elements are 1, rest are 0).
- **Syntax**: numpy.eye(N, M=None, k=0, dtype=float, order='C')
- **Examples**:

```
arr1 = np.eye(3)
arr2 = np.eye(3, 4, k=1)
print(arr1)
print(arr2)
```

Output:

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

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

---

8. numpy.random.rand()

- **Definition**: Generates an array with random values between 0 and 1.
- **Syntax**: numpy.random.rand(d0, d1, ..., dn)
- **Examples**:

```
arr1 = np.random.rand(3)
arr2 = np.random.rand(2, 2)
print(arr1)
print(arr2)
```

Output:

```
[0.4321 0.8723 0.2145]
[[0.5421 0.6789]
 [0.1234 0.9987]]
```

---

9. numpy.random.randint()

- **Definition**: Generates an array with random integers within a given range.
- **Syntax**: numpy.random.randint(low, high=None, size=None, dtype=int)
- **Examples**:

```
arr1 = np.random.randint(1, 10, 5)
arr2 = np.random.randint(0, 20, (2, 3))
print(arr1)
print(arr2)
```

Output:

```
[3 7 1 9 5]
[[14 2 18]
 [ 5 11 9]]
```

---

10. numpy.empty()

- **Definition**: Creates an empty array (values are uninitialized).
- **Syntax**: numpy.empty(shape, dtype=float, order='C')
- **Examples**:

```
arr1 = np.empty(4)
arr2 = np.empty((2, 2), dtype=int)
print(arr1)
print(arr2)
```

Output: *(values may vary due to uninitialized memory)*

```
[4.5678e-307 1.2345e-305 2.3456e-308 3.4567e-307]
[[123456789 987654321]
 [ 456789123 789123456]]
```

2. Array Manipulation Functions

1. numpy.reshape()

- **Definition**: Changes the shape of an array without changing its data.
- **Syntax**: numpy.reshape(a, newshape, order='C')
- **Examples**:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])
reshaped_arr = arr.reshape(2, 3)
print(reshaped_arr)
```

Output:

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

---

2. numpy.ravel()

- **Definition**: Flattens a multi-dimensional array into a 1D array.
- **Syntax**: numpy.ravel(a, order='C')
- **Examples**:

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

Output: [1 2 3 4 5 6]

3. numpy.transpose()

- **Definition**: Swaps the rows and columns of an array.
- **Syntax**: numpy.transpose(a, axes=None)
- **Examples**:

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

Output:

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

---

4. numpy.hstack()

- **Definition**: Stacks arrays horizontally (column-wise).
- **Syntax**: numpy.hstack(tup)
- **Examples**:

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
result = np.hstack((arr1, arr2))
print(result)
```

Output: [1 2 3 4 5 6]

---

5. numpy.vstack()

- **Definition**: Stacks arrays vertically (row-wise).
- **Syntax**: numpy.vstack(tup)
- **Examples**:

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
result = np.vstack((arr1, arr2))
print(result)
```

Output:

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

---

6. numpy.hsplit()

- **Definition**: Splits an array into multiple sub-arrays horizontally.
- **Syntax**: numpy.hsplit(ary, indices_or_sections)
- **Examples**:

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

Output: [array([1, 2]), array([3, 4]), array([5, 6])]

7. numpy.vsplit()

- **Definition**: Splits an array into multiple sub-arrays vertically.
- **Syntax**: numpy.vsplit(ary, indices_or_sections)
- **Examples**:

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

Output:
```
[array([[1, 2, 3]]), array([[4, 5, 6]])]
```

---

8. numpy.concatenate()

- **Definition**: Joins multiple arrays along an existing axis.
- **Syntax**: numpy.concatenate((a1, a2, ...), axis=0, out=None, dtype=None)
- **Examples**:

```
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6]])
result = np.concatenate((arr1, arr2), axis=0)
print(result)
```

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

---

9. numpy.expand_dims()

- **Definition**: Expands the dimensions of an array.
- **Syntax**: numpy.expand_dims(a, axis)
- **Examples**:

```
arr = np.array([1, 2, 3])
expanded_arr = np.expand_dims(arr, axis=0)
print(expanded_arr)
```

Output:
```
[[1 2 3]]
```

---

10. numpy.squeeze()

- **Definition**: Removes single-dimensional entries from an array.
- **Syntax**: numpy.squeeze(a, axis=None)
- **Examples**:

```
arr = np.array([[[1, 2, 3]]])
squeezed_arr = np.squeeze(arr)
print(squeezed_arr)
```

Output:
```
[1 2 3]
```

3. Mathematical Functions

1. numpy.add()

- **Definition**: Adds two arrays element-wise.
- **Syntax**: numpy.add(x1, x2, out=None, where=True, dtype=None)
- **Examples**:

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
result = np.add(arr1, arr2)
print(result)
```

Output: [5 7 9]

---

2. numpy.subtract()

- **Definition**: Subtracts elements of the second array from the first array element-wise.
- **Syntax**: numpy.subtract(x1, x2, out=None, where=True, dtype=None)
- **Examples**:

```
arr1 = np.array([7, 8, 9])
arr2 = np.array([1, 2, 3])
result = np.subtract(arr1, arr2)
print(result)
```

Output: [6 6 6]

---

3. numpy.multiply()

- **Definition**: Multiplies two arrays element-wise.
- **Syntax**: numpy.multiply(x1, x2, out=None, where=True, dtype=None)
- **Examples**:

```
arr1 = np.array([2, 3, 4])
arr2 = np.array([5, 6, 7])
result = np.multiply(arr1, arr2)
print(result)
```

Output: [10 18 28]

---

4. numpy.divide()

- **Definition**: Divides elements of the first array by the second array element-wise.
- **Syntax**: numpy.divide(x1, x2, out=None, where=True, dtype=None)

- **Examples**:

```
arr1 = np.array([10, 20, 30])
arr2 = np.array([2, 5, 6])
result = np.divide(arr1, arr2)
print(result)
```

Output: [5. 4. 5.]

---

5. numpy.power()

- **Definition**: Raises elements of the first array to the power of the corresponding elements in the second array.
- **Syntax**: numpy.power(x1, x2, out=None, where=True, dtype=None)

- **Examples**:

```
arr1 = np.array([2, 3, 4])
arr2 = np.array([3, 2, 1])
result = np.power(arr1, arr2)
print(result)
```

Output: [8 9 4]

---

6. numpy.mod()

- **Definition**: Computes the remainder of division element-wise.
- **Syntax**: numpy.mod(x1, x2, out=None, where=True, dtype=None)

- **Examples**:

```
arr1 = np.array([10, 20, 30])
arr2 = np.array([3, 7, 4])
result = np.mod(arr1, arr2)
print(result)
```

Output: [1 6 2]

---

7. numpy.absolute()

- **Definition**: Returns the absolute values of an array element-wise.
- **Syntax**: numpy.absolute(x, out=None, where=True, dtype=None)

- **Examples**:

```
arr = np.array([-5, -3, 0, 4, -8])
result = np.absolute(arr)
print(result)
```

Output:

```
[5 3 0 4 8]
```

---

8. numpy.exp()

- **Definition**: Computes the exponential (e^x) of all elements in the array.
- **Syntax**: numpy.exp(x, out=None, where=True, dtype=None)

- **Examples**:

```
arr = np.array([0, 1, 2])
result = np.exp(arr)
print(result)
```

Output:

```
[1.        2.71828183 7.3890561 ]
```

---

9. numpy.log()

- **Definition**: Computes the natural logarithm (log base e) element-wise.
- **Syntax**: numpy.log(x, out=None, where=True, dtype=None)
- **Examples**:

```
arr = np.array([1, np.e, np.e**2])
result = np.log(arr)
print(result)
```

Output: [0. 1. 2.]

---

10. numpy.log10()

- **Definition**: Computes the logarithm (base 10) of each element in the array.
- **Syntax**: numpy.log10(x, out=None, where=True, dtype=None)
- **Examples**:

```
arr = np.array([1, 10, 100])
result = np.log10(arr)
print(result)
```

Output: [0. 1. 2.]

---

11. numpy.sqrt()

- **Definition**: Computes the square root of each element in the array.
- **Syntax**: numpy.sqrt(x, out=None, where=True, dtype=None)
- **Examples**:

```
arr = np.array([4, 9, 16])
result = np.sqrt(arr)
print(result)
```

Output:

```
[2. 3. 4.]
```

---

12. numpy.sin()

- **Definition**: Computes the sine of each element in the array (in radians).
- **Syntax**: numpy.sin(x, out=None, where=True, dtype=None)
- **Examples**:

```
arr = np.array([0, np.pi/2, np.pi])
result = np.sin(arr)
print(result)
```

Output:

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

---

13. numpy.cos()

- **Definition**: Computes the cosine of each element in the array (in radians).
- **Syntax**: numpy.cos(x, out=None, where=True, dtype=None)
- **Examples**:

```
arr = np.array([0, np.pi/2, np.pi])
result = np.cos(arr)
print(result)
```

Output:

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

---

14. numpy.tan()

- **Definition**: Computes the tangent of each element in the array (in radians).
- **Syntax**: numpy.tan(x, out=None, where=True, dtype=None)
- **Examples**:

```
arr = np.array([0, np.pi/4, np.pi/2])
result = np.tan(arr)
print(result)
```

Output:

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

15. numpy.floor()

- **Definition**: Rounds each element in the array down to the nearest integer.
- **Syntax**: numpy.floor(x, out=None, where=True, dtype=None)
- **Examples**:

```
arr = np.array([1.7, 2.5, -3.9])
result = np.floor(arr)
print(result)
```

Output:

```
[ 1.  2. -4.]
```

---

16. numpy.ceil()

- **Definition**: Rounds each element in the array up to the nearest integer.
- **Syntax**: numpy.ceil(x, out=None, where=True, dtype=None)
- **Examples**:

```
arr = np.array([1.7, 2.5, -3.9])
result = np.ceil(arr)
print(result)
```

Output:

```
[ 2.  3. -3.]
```

---

17. numpy.round()

- **Definition**: Rounds each element in the array to the nearest integer.
- **Syntax**: numpy.round(x, decimals=0, out=None)
- **Examples**:

```
arr = np.array([1.49, 2.5, -3.9])
result = np.round(arr)
print(result)
```

Output:

```
[ 1.  2. -4.]
```

---

18. numpy.clip()

- **Definition**: Limits the values in an array within a given range.
- **Syntax**: numpy.clip(a, a_min, a_max, out=None)
- **Examples**:

```
arr = np.array([2, 5, 8, 12])
result = np.clip(arr, 3, 10)
print(result)
```

Output:

```
[ 3  5  8 10]
```

19. numpy.lcm()

- **Definition**: Computes the least common multiple (LCM) element-wise.
- **Syntax**: numpy.lcm(x1, x2, out=None, where=True, dtype=None)

- **Examples**:

```
arr1 = np.array([12, 15, 21])
arr2 = np.array([8, 10, 14])
result = np.lcm(arr1, arr2)
print(result)
```

Output: [24 30 42]

❖ Statistical Functions:

1. numpy.mean()

- **Definition**: Computes the arithmetic mean (average) of the elements in an array.
- **Syntax**: numpy.mean(a, axis=None, dtype=None, out=None, keepdims=False)
- **Examples**:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
result = np.mean(arr)
print(result)
```

Output: 3.0

---

2. numpy.median()

- **Definition**: Computes the median (middle value) of the elements in an array.
- **Syntax**: numpy.median(a, axis=None, out=None, overwrite_input=False, keepdims=False)
- **Examples**:'

```
arr = np.array([1, 3, 5, 7, 9])
result = np.median(arr)
print(result)
```

Output: 5.0

---

3. numpy.std()

- **Definition**: Computes the standard deviation of the elements in an array.
- **Syntax**: numpy.std(a, axis=None, dtype=None, out=None, ddof=0, keepdims=False)
- **Examples**:

```
arr = np.array([1, 2, 3, 4, 5])
result = np.std(arr)
print(result)
```

Output: 1.4142135623730951

4. numpy.var()

- **Definition**: Computes the variance of the elements in an array.
- **Syntax**: numpy.var(a, axis=None, dtype=None, out=None, ddof=0, keepdims=False)

- **Examples**:

```
arr = np.array([1, 2, 3, 4, 5])
result = np.var(arr)
print(result)
```

Output: 2.0

5. numpy.min()

- **Definition**: Returns the minimum value in an array.
- **Syntax**: numpy.min(a, axis=None, out=None, keepdims=False)

- **Examples**:

```
arr = np.array([4, 7, 1, 9, 3])
result = np.min(arr)
print(result)
```

Output: 1

6. numpy.max()

- **Definition**: Returns the maximum value in an array.
- **Syntax**: numpy.max(a, axis=None, out=None, keepdims=False)

- **Examples**:

```
arr = np.array([4, 7, 1, 9, 3])
result = np.max(arr)
print(result)
```

Output: 9

7. numpy.percentile()

- **Definition**: Computes the nth percentile of the elements in an array.
- **Syntax**: numpy.percentile(a, q, axis=None, out=None, overwrite_input=False, interpolation='linear', keepdims=False)

- **Examples**:

```
arr = np.array([1, 2, 3, 4, 5])
result = np.percentile(arr, 50)
print(result)
```

Output: 3.0

8. numpy.quantile()

- **Definition**: Computes the nth quantile of the elements in an array.
- **Syntax**: numpy.quantile(a, q, axis=None, out=None, overwrite_input=False, interpolation='linear', keepdims=False)

- **Examples**:

```
arr = np.array([1, 2, 3, 4, 5])
result = np.quantile(arr, 0.5)
print(result)
```

Output: 3.0

---

9. numpy.ptp()

- **Definition**: Returns the range (max - min) of values in an array.
- **Syntax**: numpy.ptp(a, axis=None, out=None, keepdims=False)

- **Examples**:

```
arr = np.array([10, 2, 8, 4])
result = np.ptp(arr)
print(result)
```

Output: 8

---

10. numpy.corrcoef()

- **Definition**: Computes the Pearson correlation coefficient between arrays.
- **Syntax**: numpy.corrcoef(x, y=None, rowvar=True, bias=<no value>, ddof=<no value>)

- **Examples**:

```
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])
result = np.corrcoef(x, y)
print(result)
```

Output:

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

---

11. numpy.histogram()

- **Definition**: Computes the histogram of an array.
- **Syntax**: numpy.histogram(a, bins=10, range=None, normed=None, weights=None, density=None)

- **Examples**:

```
arr = np.array([1, 2, 1, 3, 2, 3, 4, 5, 4, 5, 6])
hist, bins = np.histogram(arr, bins=3)
print(hist)
print(bins)
```

Output:

```
[4 4 3]
[1. 2.66666667 4.33333333 6.]
```

12. numpy.mode() (Using SciPy for Mode Calculation)

- **Definition**: Computes the mode (most frequent value) in an array.
- **Syntax**: scipy.stats.mode(a, axis=0, nan_policy='propagate')

- **Examples**:

```
from scipy import stats
arr = np.array([1, 2, 3, 2, 2, 4, 5, 6])
mode_result = stats.mode(arr)
print(mode_result.mode)
```

Output: [2]

➢Indexing and Slicing Functions

1. numpy.take()

- **Definition**: Selects elements from an array using indices.
- **Syntax**: numpy.take(a, indices, axis=None, out=None, mode='raise')
- **Examples**:

```
import numpy as np
arr = np.array([10, 20, 30, 40, 50])
result = np.take(arr, [0, 2, 4])
print(result)
```

Output: [10 30 50]

2. numpy.put()

- **Definition**: Replaces specified elements in an array with given values.
- **Syntax**: numpy.put(a, indices, values, mode='raise')
- **Examples**:

```
arr = np.array([1, 2, 3, 4, 5])
np.put(arr, [0, 3], [10, 40])
print(arr)
```

Output: [10 2 3 40 5]

3. numpy.choose()

- **Definition**: Constructs an array by selecting elements from multiple arrays.
- **Syntax**: numpy.choose(a, choices, out=None, mode='raise')

- **Examples**:

```
x = np.array([0, 1, 2])
choices = [np.array([10, 20, 30]), np.array([40, 50, 60]), np.array([70, 80, 90])]
result = np.choose(x, choices)
print(result)
```

Output: [10 50 90]

---

4. numpy.compress()

- **Definition**: Filters elements from an array using a condition.
- **Syntax**: numpy.compress(condition, a, axis=None, out=None)
- **Examples**:

```
arr = np.array([1, 2, 3, 4, 5])
condition = arr > 2
result = np.compress(condition, arr)
print(result)
```

Output: [3 4 5]

---

5. numpy.extract()

- **Definition**: Returns elements of an array that satisfy a given condition.
- **Syntax**: numpy.extract(condition, arr)
- **Examples**:

```
arr = np.array([1, 2, 3, 4, 5])
condition = arr % 2 == 0
result = np.extract(condition, arr)
print(result)
```

Output: [2 4]

---

6. numpy.flat[]

- **Definition**: Returns a flat iterator over an array.
- **Syntax**: array.flat[index]
- **Examples**:

```
arr = np.array([[1, 2], [3, 4]])
result = arr.flat[2]
print(result)
```

Output: 3

---

7. numpy.diagonal()

- **Definition**: Extracts the diagonal elements of a 2D array.
- **Syntax**: numpy.diagonal(a, offset=0, axis1=0, axis2=1)
- **Examples**:

```
arr = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])
result = arr.diagonal()
print(result)
```

Output: [10 50 90]

---

8. numpy.flip()

- **Definition**: Reverses the order of elements in an array along a specified axis.
- **Syntax**: numpy.flip(a, axis=None)
- **Examples**:

```
arr = np.array([1, 2, 3, 4])
result = np.flip(arr)
print(result)
```

Output: [4 3 2 1]

9. numpy.resize()

- **Definition**: Reshapes an array to a new shape, filling with repeated copies if necessary.
- **Syntax**: numpy.resize(a, new_shape)
- **Examples**:

```
arr = np.array([1, 2, 3])
result = np.resize(arr, (2, 3))
print(result)
```

Output:

```
[[1 2 3]
 [1 2 3]]
```

---

10. numpy.reshape()

- **Definition**: Changes the shape of an array without changing data.
- **Syntax**: numpy.reshape(a, newshape, order='C')
- **Examples**:

```
arr = np.array([1, 2, 3, 4])
result = np.reshape(arr, (2, 2))
print(result)
```

Output:

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

11. numpy.ravel()

- **Definition**: Flattens an array into a 1D array.
- **Syntax**: numpy.ravel(a, order='C')
- **Examples**:

```
arr = np.array([[1, 2], [3, 4]])
result = np.ravel(arr)
print(result)
```

Output: [1 2 3 4]

---

12. numpy.shares_memory()

- **Definition**: Checks if two arrays share memory.
- **Syntax**: numpy.shares_memory(a, b)
- **Examples**:

```
arr1 = np.array([1, 2, 3])
arr2 = arr1.view()
print(np.shares_memory(arr1, arr2))
```

Output: True

13. numpy.nonzero()

- **Definition**: Returns the indices of non-zero elements in an array.
- **Syntax**: numpy.nonzero(a)
- **Examples**:

```
arr = np.array([0, 1, 0, 3, 4])
result = np.nonzero(arr)
print(result)
```

Output: (array([1, 3, 4]),)

---

14. numpy.where()

- **Definition**: Returns indices where a condition is met.
- **Syntax**: numpy.where(condition, x, y)
- **Examples**:

```
arr = np.array([10, 20, 30, 40, 50])
result = np.where(arr > 25, arr, -1)
print(result)
```

Output:

```
[-1 -1 30 40 50]
```

9. Random Number Generation

1. numpy.random.rand()

- **Definition**: Generates random numbers from a uniform distribution over [0, 1).
- **Syntax**: numpy.random.rand(d0, d1, ..., dn)
- **Examples**:

```
arr = np.random.rand(2, 3)
print(arr)
```

Output:

```
[[0.3799573  0.21177711 0.08730328]
 [0.36634686 0.57382249 0.30249379]]
```

---

2. numpy.random.randn()

- **Definition**: Generates random numbers from a standard normal distribution (mean 0, variance 1).
- **Syntax**: numpy.random.randn(d0, d1, ..., dn)
- **Examples**:

```
arr = np.random.randn(2, 3)
print(arr)
```

Output:

```
[[ 0.26113726  1.34620184 -0.7529059 ]
 [-0.75856359  0.24897796 -0.35871314]]
```

---

3. numpy.random.randint()

- **Definition**: Generates random integers from a specified range.
- **Syntax**: numpy.random.randint(low, high=None, size=None, dtype=int)
- **Examples**:

```
arr = np.random.randint(0, 10, size=(2, 3))
print(arr)
```

Output:

```
[[1 6 2]
 [9 7 8]]
```

---

4. numpy.random.choice()

- **Definition**: Generates a random sample from a given 1D array. It can sample with or without replacement.
- **Syntax**: numpy.random.choice(a, size=None, replace=True, p=None)
- **Examples**:

```
arr = np.random.choice([1, 2, 3, 4, 5], size=3, replace=False)
print(arr)
```

Output: [4 2 5]

5. numpy.random.random()

- **Definition**: Generates random floating-point numbers between [0, 1).
- **Syntax**: numpy.random.random(size)
- **Examples**:

```
arr = np.random.random(3)
print(arr)
```

Output:
```
[0.6971904  0.05841519 0.42885265]
```

---

6. numpy.random.uniform()

- **Definition**: Generates random floating-point numbers from a uniform distribution within a specified range.
- **Syntax**: numpy.random.uniform(low=0.0, high=1.0, size=None)
- **Examples**:

```
arr = np.random.uniform(5, 10, size=(2, 3))
print(arr)
```

Output:
```
[[7.23628586 6.41171372 7.98802126]
 [6.59630724 9.45535643 9.05439756]]
```

---

7. numpy.random.normal()

- **Definition**: Generates random numbers from a normal (Gaussian) distribution with a specified mean and standard deviation.
- **Syntax**: numpy.random.normal(loc=0.0, scale=1.0, size=None)
- **Examples**:

```
arr = np.random.normal(0, 1, 5)
print(arr)
```

Output:
```
[-0.21460435  0.35090495 -1.0259114  -1.50753555  0.93816872]
```

---

8. numpy.random.seed()

- **Definition**: Sets the seed for the random number generator to make the output reproducible.
- **Syntax**: numpy.random.seed(seed)
- **Examples**:

```
np.random.seed(42)
arr = np.random.rand(2, 3)
print(arr)
```

Output:
```
[[0.37454012 0.95071431 0.73199394]
 [0.59865848 0.15601864 0.15599452]]
```

9. numpy.random.permutation()

- **Definition**: Returns a random permutation of a sequence or an array.
- **Syntax**: numpy.random.permutation(x)

- **Examples**:

```
arr = np.array([1, 2, 3, 4, 5])
result = np.random.permutation(arr)
print(result)
```

Output: [4 2 5 1 3]

---

10. numpy.random.shuffle()

- **Definition**: Shuffles the elements of an array in place.
- **Syntax**: numpy.random.shuffle(x)

- **Examples**:

```
arr = np.array([1, 2, 3, 4, 5])
np.random.shuffle(arr)
print(arr)
```

Output: [4 2 3 5 1]

---

11. numpy.random.binomial()

- **Definition**: Generates random numbers from a binomial distribution.
- **Syntax**: numpy.random.binomial(n, p, size=None)

- **Examples**:

```
arr = np.random.binomial(10, 0.5, size=5)
print(arr)
```

Output: [5 4 6 5 3]

---

12. numpy.random.poisson()

- **Definition**: Generates random numbers from a Poisson distribution.
- **Syntax**: numpy.random.poisson(lam=1.0, size=None)

- **Examples**:

```
arr = np.random.poisson(5, size=3)
print(arr)
```

Output: [4 5 7]

---

13. numpy.random.exponential()

- **Definition**: Generates random numbers from an exponential distribution with a specified scale.
- **Syntax**: numpy.random.exponential(scale=1.0, size=None)
- **Examples**:

```
arr = np.random.exponential(1, size=5)
print(arr)
```

Output:

```
[0.18831022 0.25710176 0.37850516 0.21641669 0.83946703]
```

---

14. numpy.random.chisquare()

- **Definition**: Generates random numbers from a chi-square distribution.
- **Syntax**: numpy.random.chisquare(df, size=None)
- **Examples**:

```
arr = np.random.chisquare(2, size=5)
print(arr)
```

Output:

```
[2.42461824 2.62422495 4.24337991 0.73102028 1.36856961]
```