Exp.no: 3.1.a

AIM: a . To get help on the add function

DESCRIPTION:

- This program uses the **NumPy library**, which is mainly used for performing **mathematical and array operations** efficiently in Python.
- The function np.add() is a **universal function (ufunc)** that performs **element-wise addition** on arrays or numbers.
- The function np.info() displays **detailed information or documentation** about any NumPy object.
- By writing np.info(np.add), the program prints information about how np.add works, including its syntax, usage, and supported data types.
- The concept behind this program is **introspection**, which means getting details about functions or objects at runtime.

PROGRAM:

```
import numpy as np

print(np.info(np.add))
```

OUTPUT:

```
add(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[, signature])

Add arguments element-wise.

Parameters
----------
x1, x2 : array_like
    The arrays to be added.
    If ``x1.shape != x2.shape``, they must be broadcastable to a common
    shape (which becomes the shape of the output).
out : ndarray, None, or tuple of ndarray and None, optional
    A location into which the result is stored. If provided, it must have
    a shape that the inputs broadcast to. If not provided or None,
    a freshly-allocated array is returned. A tuple (possible only as a
    keyword argument) must have length equal to the number of outputs.
where : array_like, optional
    This condition is broadcast over the input. At locations where the
    condition is True, the `out` array will be set to the ufunc result.
    Elsewhere, the `out` array will retain its original value.
    Note that if an uninitialized `out` array is created via the default
    ``out=None``, locations within it where the condition is False will
    remain uninitialized.
**kwargs
    For other keyword-only arguments, see the
    :ref:`ufunc docs <ufuncs.kwargs>`.

Returns
-------
add : ndarray or scalar
    The sum of `x1` and `x2`, element-wise.
    This is a scalar if both `x1` and `x2` are scalars.

Notes
-----
Equivalent to `x1` + `x2` in terms of array broadcasting.
```

RESULT:

Hence the program To get help on the add function is executed and it's output is verified successfully

Exp.no: 3.1.b

AIM: b. To test whether none of the elements of a given array is zero.

DESCRIPTION:

1. This program uses the NumPy library to demonstrate the use of the np.all() function.
2. The np.all() function checks whether all elements in a given array are non-zero (True).
3. In NumPy, zero is treated as False and any non-zero value is treated as True.
4. Since all the elements in the array [1, 2, 3, 4] are non-zero, the function returns True.
5. The statement np.info(np.all) displays the built-in documentation of the np.all() function, explaining its syntax and purpose.
6. The concept shown here is **logical array evaluation**, where NumPy performs condition checks on entire arrays at once.
7. It also demonstrates **introspection** — getting information about functions directly from the program using np.info().

PROGRAM:

```
import numpy as np

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

print("All elements are non-zero", np.all(arr))

print(np.info(np.all))
```

OUTPUT:

```
all elements are non-zero: True
Test whether all array elements along a given axis evaluate to True.

Parameters
----------
a : array_like
    Input array or object that can be converted to an array.
axis : None or int or tuple of ints, optional
    Axis or axes along which a logical AND reduction is performed.
    The default (``axis=None``) is to perform a logical AND over all
    the dimensions of the input array. `axis` may be negative, in
    which case it counts from the last to the first axis.

    .. versionadded:: 1.7.0

    If this is a tuple of ints, a reduction is performed on multiple
    axes, instead of a single axis or all the axes as before.
out : ndarray, optional
    Alternate output array in which to place the result.
    It must have the same shape as the expected output and its
```

RESULT:

Hence the program To test whether none of the elements of a given array is zero.

is executed and it's output is verified successfully

Exp.no: 3.1.c

AIM: c. To create an element-wise comparison (greater, greater_equal, less and less_equal, equal, equal within a tolerance) of two given arrays.

DESCRIPTION:

1. This program demonstrates various **comparison operations** performed on NumPy arrays.
2. NumPy provides element-wise comparison functions such as np.greater(), np.less(), np.equal(), and others.
3. Each of these functions compares corresponding elements of two arrays and returns a **Boolean array** showing the result for each element.
4. For example, np.greater(a, b) returns True wherever an element in a is greater than the corresponding element in b.
5. The functions np.greater_equal(), np.less(), and np.less_equal() perform similar comparisons for other relational operators.
6. The function np.equal() checks whether corresponding elements are exactly the same.
7. The function np.allclose() compares two arrays of floating-point numbers to check if they are approximately equal, allowing for tiny rounding differences.
8. The main concept shown here is **element-wise comparison and numerical tolerance** in NumPy, which allows large data sets to be compared efficiently without using loops.
9. Such operations are important in **data analysis, scientific computing, and machine learning**, where comparing large arrays or matrices is common.

PROGRAM:

```
import numpy as np

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

b=np.array([15,20,25])

print("Greater : ",np.greater(a,b))

print("Greater equal : ",np.greater_equal(a,b))

print("Less : ",np.less(a,b))

print("Less equal : ",np.less_equal(a,b))
```

```
print("Equal :",np.equal(a,b))

c=np.array([1.00001,2.00001])

d=np.array([1.00002,2.00002])

print("All close :",np.allclose(c,d))
```

OUTPUT:

```
Greater: [False False False]
Greater Equal: [False  True False]
Less: [ True False  True]
Less equal: [ True  True  True]
Equal: [False  True False]
Allclose: True
```

RESULT:

Hence the program To create an element-wise comparison (greater, greater_equal, less and less_equal, equal, equal within a tolerance) of two given arrays. is executed and it’s output is verified successfully

Exp.no: 3.2.a

AIM: a.To extract all numbers from a given array which are less and greater than a specified number.

DESCRIPTION:

1. This program demonstrates how to perform **conditional data extraction** from a NumPy array.
2. The array arr contains numeric elements, and conditions like arr > 30 or arr < 30 are used to filter elements based on their values.
3. NumPy allows **Boolean indexing**, where a condition applied to an array returns a Boolean array indicating True or False for each element.
4. When this Boolean array is used as an index (for example, arr[arr > 30]), NumPy automatically selects only the elements that satisfy the condition.
5. This approach eliminates the need for loops and makes array filtering **faster and more efficient**.
6. The concept shown here is **vectorized conditional selection**, which is widely used in **data analysis, preprocessing, and scientific computing**.

PROGRAM:

```
import numpy as np

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

print("Greater than 30:",arr[arr>30])

print(arr)

print("Less than 30:",arr[arr<30])

print(arr)
```

OUTPUT:

```
Greater than 30: [40 50]
[10 20 30 40 50]
Less than 30: [10 20]
[10 20 30 40 50]
```

RESULT:

Hence the program To extract all numbers from a given array which are less and greater than a specified number. is executed and it's output is verified successful

Exp.no: 3.2.b

AIM: b. To find the indices of the maximum and minimum numbers along the given axis of an array

DESCRIPTION:

1. This program demonstrates how to find the **indices (positions)** of the maximum and minimum values in a NumPy array using the functions np.argmax() and np.argmin().
2. The function np.argmax() returns the **index of the largest element** along a specified axis, while np.argmin() returns the **index of the smallest element**.
3. The parameter axis defines the direction of operation:
    - axis=0 means the function operates **column-wise** (down each column).
    - axis=1 means it operates **row-wise** (across each row).
4. In this example, np.argmax(arr2d, axis=0) returns the row indices of the largest elements in each column, while np.argmin(arr2d, axis=1) returns the column indices of the smallest elements in each row.
5. The concept here is **axis-based computation**, which allows functions to work efficiently on multi-dimensional data.
6. These operations are useful in **data analysis, statistics, and image processing**, where identifying extreme values along specific directions is important.
7. NumPy performs these computations internally using optimized C code, which makes it much faster than traditional Python loops.

PROGRAM:

```
# 3.2.b Find the indices of the max and min numbers along the given axis

import numpy as np

arr2d=np.array([[10,20,30],[40,5,25],[7,50,60]])

# index of max along axix 0 (column wise)

print("ArgMax (axis=0): ",np.argmax(arr2d,axis=0))

# index of min along axix 1 (row wise)

print("ArgMin (axis=1): ",np.argmin(arr2d,axis=1))
```

OUTPUT:

```
ArgMax (axis=0):  [1 2 2]
ArgMin (axis=1):  [0 1 0]
```

RESULT:

Hence the program To find the indices of the maximum and minimum numbers along the given axis of an array is executed and it's output is verified successfully