**Advance Functions In numpy :**

**1. Truth value testing**

The **numpy.all()** function tests whether all array elements along the mentioned axis evaluate to True.

import numpy as np

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

print("Original array:")

print(x)

print("Test if none of the elements of the said array is zero:")

print(np.all(x)) #true

x = np.array([0, 1, 2, 3])

print("Original array:")

print(x)

print("Test if none of the elements of the said array is zero:")

print(np.all(x)) #false

The **any()** function is used to test whether any array element along a given axis evaluates to True.

import numpy as np

x = np.array([1, 0, 0, 0])

print("Original array:")

print(x)

print("Test whether any of the elements of a given array is non-zero:")

print(np.any(x)) #true

x = np.array([0, 0, 0, 0])

print("Original array:")

print(x)

print("Test whether any of the elements of a given array is non-zero:")

print(np.any(x)) #false

Any VS All

**Any**

Returns true if any of the items is True. It returns False if empty or all are false. Any can be thought of as a sequence of OR operations on the provided iterables.

It short circuit the execution i.e. stop the execution as soon as the result is known.

# Since all are false, false is returned

**print (any([False, False, False, False]))**

# Here the method will short-circuit at the

# second item (True) and will return True.

**print (any([False, True, False, False]))**

# Here the method will short-circuit at the

# first (True) and will return True.

**print (any([True, False, False, False]))**

**Output :**

**False**

**True**

**True**

**All**

Returns true if all of the items are True (or if the iterable is empty). All can be thought of as a sequence of AND operations on the provided iterables. It also short circuit the execution i.e. stop the execution as soon as the result is known.

# Here all the iterables are True so all

# will return True and the same will be printed

**print (all([True, True, True, True]))**

# Here the method will short-circuit at the

# first item (False) and will return False.

**print (all([False, True, True, False]))**

# This statement will return False, as no

# True is found in the iterables

**print (all([False, False, False]))**

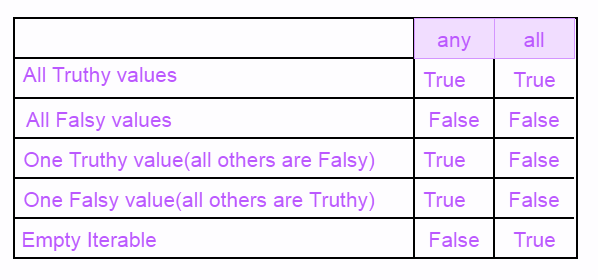
**Output :**

**True**

**False**

**False**

**Truth table :-**

****

**2. Array contents**

## numpy.isfinite() function

The isfinite() function is used to test element-wise for finiteness (not infinity or not Not a Number).

The result is returned as a boolean array.

**Example-1:**

>>> import numpy as np

>>> np.isfinite(5)

Output:

True

**Example-2:**

>>> import numpy as np

>>> np.isfinite(0)

Output:

True

**Example-3:**

>>> import numpy as np

>>> np.isfinite(np.nan)

Output:

False

**Example-4:**

>>> import numpy as np >>> np.isfinite(np.inf)

Output:

False

**Example-5:**

>>> import numpy as np >>> np.isfinite(np.NINF)

Output:

False

## **numpy.isinf() function**

The isinf() function is used to test element-wise for positive or negative infinity.

Returns a boolean

**Example-1:**

>>> import numpy as np

>>> np.isinf(np.inf)

Output:

True

**Example-2:**

>>> import numpy as np

>>> np.isinf(np.nan)

Output:

False

**Example-3:**

>>> import numpy as np

>>> np.isinf(np.NINF)

Output:

True

**Example-4:**

>>> import numpy as np

>>> np.isinf([np.inf, -np.inf, 1.0, np.nan])

Output:

array([ True, True, False, False])

## **numpy.isnan() function**

The isnan() function is used to test element-wise for NaN and return result as a boolean array.

**Example-1:**

>>> import numpy as np

>>> np.isnan(np.nan)

Output:

True

**Example-2:**

>>> import numpy as np

>>> np.isnan(np.inf)

Output:

False

The **isnat()** function is used to test element-wise for NaT (not a time) and return result as a boolean

**Example-1:**

>>> import numpy as np

>>> np.isnat(np.datetime64("NaT"))

Output:

True

**Example-2:**

>>> import numpy as np

>>> np.isnat(np.datetime64("2017-01-01"))

Output:

False

The **isneginf()** function is used to test element-wise for negative infinity, return result as bool

**Example-1:**

>>> import numpy as np

>>> np.isneginf(np.NINF)

Output:

True

**Example-2:**

>>> import numpy as np >>> np.isneginf(np.inf)

Output:

False

**Example-3:**

>>> import numpy as np >>> np.isneginf(np.PINF)

Copy

Output:

False

**Example-4:**

>>> import numpy as np >>> np.isneginf([-np.inf, 0., np.inf])

Copy

Output:

array([ True, False, False])

## **numpy.isposinf() function**

The isposinf() function is used to test element-wise for positive infinity, return result as bool array

**Example-1:**

>>> import numpy as np

>>> np.isposinf(np.PINF)

Output:

True

**Example-2:**

>>> import numpy as np >>> np.isposinf(np.inf)

Output:

True

**Example-3:**

>>> import numpy as np >>> np.isposinf(np.NINF)

Output:

False

**Example-4:**

>>> import numpy as np >>> np.isposinf([-np.inf, 0., np.inf])

Copy

Output:

array([False, False, True])

## **numpy.iscomplex() function**

The iscomplex() function is used to return a bool array, where the True if input element is complex.

What is tested is whether the input has a non-zero imaginary part, not if the input type is complex.

**Example:**

>>> import numpy as np

>>> np.iscomplex([2+2j, 1+0j, 5.5, 4, 2, 2j])

Output:

array([ True, False, False, False, False, True])

he **iscomplexobj()** function is used to check for a complex type or an array of complex numbers.

The type of the input is checked, not the value. Even if the input has an imaginary part equal to zero, iscomplexobj evaluates to True.

**Example-1:**

>>> import numpy as np

>>> np.iscomplexobj(3)

Output:

False

**Example-2:**

>>> import numpy as np

>>> np.iscomplexobj(3+0j)

Output:

True

**Example-3:**

>>> import numpy as np

>>> np.iscomplexobj([5, 3+0j, True])

Output:

True

The **isreal()** function is used to test a bool array, where True if the input element is real.

If an element has a complex type with zero complex part, the return value for that element is True.

**Example:**

>>> import numpy as np

>>> np.isreal([2+1j, 2+0j, 5.5, 4, 2, 3j])

Output:

array([False, True, True, True, True, False])

The **isrealobj()** function is used to return True if x is a not complex type or an array of complex numbers.

The type of the input is checked, not the value. So even if the input has an imaginary part equal to zero,

isrealobj evaluates to False if the data type is complex.

**Example-1:**

>>> import numpy as np

>>> np.isrealobj(2)

Output:

True

**Example-2:**

>>> import numpy as np

>>> np.isrealobj(2+0j)

Output:

False

**Example-3:**

>>> import numpy as np

>>> np.isrealobj([4, 1+0j, True])

Output:

False

The **isscalar()** function is used to returns True if the type of num is a scalar type.

**Example-1:**

>>> import numpy as np

>>> np.isscalar(5.1)

Output:

True

**NumPy.isscalar() method Example-2:**

>>> import numpy as np

>>> np.isscalar([5.1])

Output:

False

**NumPy.isscalar() method Example-3:**

>>> import numpy as np

>>> np.isscalar(False)

Output:

True

**NumPy.isscalar() method Example-4:**

>>> import numpy as np

>>> np.isscalar('numpy')

Output:

True

**3. Logical operations**

The **logical\_and()** function is used to compute the truth value of x1 AND x2 element-wise.

**Example-1:**

>>> import numpy as np

>>> np.logical\_and(True, False)

Output:

False

**Example-2:**

>>> import numpy as np

>>> np.logical\_and([True, False], [False, False])

Output:

array([False, False])

The **logical\_or()** function is used to compute the truth value of x1 OR x2 element-wise.

**Example-1:**

>>> import numpy as np

>>> np.logical\_or(True, False)

Copy

Output:

True

**Example-2:**

>>> import numpy as np

>>> np.logical\_or([True, False], [False, False])

Copy

Output:

array([ True, False])

The **logical\_not()** function is used to returns the truth value of NOT x element-wise.

**Example-1:**

>>> import numpy as np

>>> np.logical\_not(5)

Output:

False

**Example-2:**

>>> import numpy as np

>>> np.logical\_not([True, False, 0, 2])

Output:

array([False, True, True, False])

The **logical\_xor()** function is used to compute the truth value of x1 XOR x2, element-wise.

>>> import numpy as np

>>> np.logical\_xor(True, False)

Output:

True

**Example-2:**

>>> import numpy as np

>>> np.logical\_xor([True, True, False, False], [True, False, True, False])

Output:

array([False, True, True, False])