numpy_operators

November 29, 2022

[ ]:
```
import numpy as np
```

1 Numpy Operations

1.1 Arithmetic operations

Arithmetic operations on ndarrays are **element-wise** (point-to-point).

Note, that beside Arithmetic functions, numpy sumply a great variaty of Mathematical functions

Lets have next 2 arrays:

[ ]:
```
arr1 = np.arange(5)
print(f'arr1: {arr1}, {arr1.dtype}')
```

```
arr1: [0 1 2 3 4], int64
```

[ ]:
```
arr2 = np.full(5,2)
print(f'arr2: {arr2}, {arr2.dtype}')
```

```
arr2: [2 2 2 2 2], int64
```

1.1.1 Addition

We can use the '+' operator as a shortcut to np.add()

[ ]:
```
# add array with constant:
print(f'arr1: {arr1}')
arr1+1
```

```
arr1: [0 1 2 3 4]
```

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

[ ]:
```
# add two arrays
print(f'arr1: {arr1}')
print(f'arr2: {arr2}')
arr1 + arr2
```

```
arr1: [0 1 2 3 4]
arr2: [2 2 2 2 2]
```

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

1.1.2 Subtraction

We can use the '-' operator as a shortcut to np.subtract()

[ ]:
```
# subtract array with constant:
print(f'arr1: {arr1}')
arr1-1
```

```
arr1: [0 1 2 3 4]
```

[ ]:
```
array([-1,  0,  1,  2,  3])
```

[ ]:
```
# subtract two arrays
print(f'arr1: {arr1}')
print(f'arr2: {arr2}')
arr1 - arr2
```

```
arr1: [0 1 2 3 4]
arr2: [2 2 2 2 2]
```

[ ]:
```
array([-2, -1,  0,  1,  2])
```

1.1.3 Multiplication

We can use the '*' operator as a shortcut to np.multiply()

[ ]:
```
# multiply array with constant:
print(f'arr1: {arr1}')
arr1*3
```

```
arr1: [0 1 2 3 4]
```

[ ]:
```
array([ 0,  3,  6,  9, 12])
```

[ ]:
```
# multiply two arrays
print(f'arr1: {arr1}')
print(f'arr2: {arr2}')
arr1 * arr2
```

```
arr1: [0 1 2 3 4]
arr2: [2 2 2 2 2]
```

[ ]:
```
array([0, 2, 4, 6, 8])
```

1.1.4 Division

We can use the '/' operator as a shortcut to np.divide()

[ ]:
```
# divide array with constant:
print(f'arr1: {arr1}')
arr1/3
```

```
arr1: [0 1 2 3 4]
```

[ ]:
```
array([0.        , 0.33333333, 0.66666667, 1.        , 1.33333333])
```

[ ]:
```
# divide two arrays
print(f'arr1: {arr1}')
print(f'arr2: {arr2}')
arr1 / arr2
```

```
arr1: [0 1 2 3 4]
arr2: [2 2 2 2 2]
```

[ ]:
```
array([0. , 0.5, 1. , 1.5, 2. ])
```

1.2 Comparison and Logic Operations

1.2.1 Arrays Comparison

Comparison Operations on ndarrays are **element-wise** (point-to-point).

They return an array of same shape with Boolean values.

Reference: Comparison functions

[ ]:
```
### Equal (==) or np.equal() with constant
print(arr1)
print(arr1 == 1)
# print(np.equal(arr1,1))
```

```
[0 1 2 3 4]
[False  True False False False]
```

[ ]:
```
### Equal (==) or np.equal() with another array
print(arr1)
print(arr2)
print(arr1 == arr2)
# print(np.equal(arr1,arr2))
```

```
[0 1 2 3 4]
[2 2 2 2 2]
[False False  True False False]
```

[ ]:
```
### Not equal (!=) or np.not_equal() with constant
print(arr1)
print(arr1 != 1)
# print(np.not_equal(arr1,1))
```

```
[0 1 2 3 4]
[ True False  True  True  True]
```

[ ]:
```
### Not equal (!=) or np.not_equal() with another array
print(arr1)
print(arr2)
print(arr1 != arr2)
# print(np.not_equal(arr1,arr2))
```

```
[0 1 2 3 4]
[2 2 2 2 2]
[ True  True False  True  True]
```

[ ]:
```
### Greater (>) or np.greater() with constant
print(arr1)
print(arr1 > 1)
# print(np.greater(arr1,1))
```

```
[0 1 2 3 4]
[False False  True  True  True]
```

[ ]:
```
### Greater or equal (>=) or np.greater_equal() with another array
print(arr1)
print(arr2)
print(arr1 >= arr2)
# print(np.greater_equal(arr1,arr2))
```

```
[0 1 2 3 4]
[2 2 2 2 2]
[False False  True  True  True]
```

[ ]:
```
### Less (<) or np.less() with constant
print(arr1)
print(arr1 < 1)
# print(np.less(arr1,1))
```

```
[0 1 2 3 4]
[ True False False False False]
```

[ ]:
```
### Less or equal (<=) or np.less_equal() with another array
print(arr1)
print(arr2)
print(arr1 <= arr2)
```

```
# print(np.less_equal(arr1,arr2))
```

```
[0 1 2 3 4]
[2 2 2 2 2]
[ True  True  True False False]
```

1.2.2 Logical operations

Logical operations on ndarrays are **element-wise** (point-to-point).

They return an array of same shape with Boolean values.

Logical operations are commutative

Reference: Logical operations

Logical AND np.logical_and() - Compute the truth value of x1 AND x2 element-wise.

The & operator can be used as a shorthand for np.logical_and() **only on boolean values/ndarrays**.

[ ]:
```
# logical and between 2 booleans:
True & False
# np.logical_and(True, False)
```

[ ]:
```
False
```

[ ]:
```
# logical and between 2 boolean arrays:
print(f'arr1: {arr1}')
print(f'arr2: {arr2}','\n')

x = arr1>2
y = arr2>1
print(f'x: {x}')
print(f'y: {y}','\n')

print(f'x&y: {x&y}')
# print(np.logical_and(x,y))
```

```
arr1: [0 1 2 3 4]
arr2: [2 2 2 2 2]

x: [False False False  True  True]
y: [ True  True  True  True  True]

x&y: [False False False  True  True]
```

[ ]:
```
# logical and between 2 scalars:
# note that a value=0 is interpreted as False,
# and a value != 0 is interpreted as True
```

```
print(np.logical_and(1, -92349))
print(np.logical_and(1, 0))
```

```
True
False
```

Logical OR np.logical_or() - Compute the truth value of x1 OR x2 element-wise.

The | operator can be used as a shorthand for np.logical_or() **only on boolean values/ndarrays**.

[ ]:
```
# logical or between 2 booleans:
True | False
# np.logical_or(True, False)
```

[ ]:
```
True
```

[ ]:
```
# logical or between 2 boolean arrays:
print(f'arr1: {arr1}')
print(f'arr2: {arr2}','\n')

x = arr1>2
y = arr2>1
print(f'x: {x}')
print(f'y: {y}','\n')

print(f'x|y: {x|y}')
# print(np.logical_or(x,y))
```

```
arr1: [0 1 2 3 4]
arr2: [2 2 2 2 2]

x: [False False False  True  True]
y: [ True  True  True  True  True]

x|y: [ True  True  True  True  True]
```

[ ]:
```
# logical or between 2 scalars:
print(np.logical_or(1, -92349))
print(np.logical_or(1, 0))
print(np.logical_or(0, 0))
```

```
True
True
False
```

Logical NOT np.logical_not() - Compute the truth value of NOT x element-wise.

[ ]:
```
# logical not on one boolean:
np.logical_not(True)
```

[ ]:
```
False
```

[ ]:
```
# logical not on boolean array:
print(f'arr1: {arr1}')

x = arr1>2
print(f'x: {x}')

print(np.logical_not(x,y))
```

```
arr1: [0 1 2 3 4]
x: [False False False  True  True]
[ True  True  True False False]
```

[ ]:
```
# logical not on  scalar:
print(np.logical_not(-92349))
print(np.logical_not(1))
print(np.logical_not(0))
```

```
False
False
True
```

[ ]:
```
print(arr1)
print(arr2)
```

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

1.2.3 Truth value testing

np.all() - Test whether all array elements along a given axis evaluate to True.

np.any() - Test whether any array element along a given axis evaluates to True.

[ ]:
```
print(f'arr1>2: {arr1>2}')

print(np.all(arr1>2))
print(np.any(arr1>2))
```

```
arr1>2: [False False False  True  True]
False
True
```