Universal Functions

- **A universal function, or ufunc, is a function that performs element-wise operations on data in ndarrays.**

> - **These functions operates on ndarray** (N-dimensional array)
> - It **performs fast element-wise array operations.**
> - **Some ufuncs like arithmetic functions are called automatically when the corresponding arithmetic operator is used on arrays.**

> For example: when addition of two array is performed element-wise using '+' operator then np.add() is called internally.

Some Universal Functions are:

> **1. Functions for Arithmetic operations** *(Refer Unit -3 - Lecture Notes - 4)*
>
> **2. Mathematical Functions**
>
> **3. Statistical functions**
>
> **4. Set Functions**
>
> **5. Fast Element-wise Array Functions**
>
> > i) Bit twiddling Functions
> >
> > ii) Comparison Functions
> >
> > iii) Logical Functions

A) Mathematical functions:

The basic mathematical functions operate elementwise on arrays. Some of these are:

Function	Description
floor	Compute the floor value of each element
ceil	Compute the ceiling of each element (i.e., the smallest integer greater than or equal to that number)
log	Compute log(), log2(), log10() of each array element
sqrt	Compute the square root of each element (equivalent to arr ** 0.5)
square	Compute the square of each element (equivalent to arr ** 2)
reciprocal	Compute the reciprocal of each array element
sign	Compute the sign of each element: 1 (positive), 0 (zero), or –1 (negative)

floor()

```
import numpy as np

arr = np.floor([-3.1666, 3.6667])

print(arr)
```

```
[-4.  3.]
```

ceil()

```
import numpy as np

arr = np.ceil([-3.1666, 3.6667])

print(arr)
```

```
[-3.  4.]
```

log()

```
import numpy as np

arr = np.arange(1, 10)

print(np.log(arr))
```

```
[0.         0.69314718 1.09861229 1.38629436 1.60943791 1.79175947
 1.94591015 2.07944154 2.19722458]
```

```
import numpy as np

arr = np.arange(1, 10)

print(np.log2(arr))
```

```
[0.         1.         1.5849625  2.         2.32192809 2.5849625
 2.80735492 3.         3.169925  ]
```

```
import numpy as np

arr = np.arange(1, 10)

print(np.log10(arr))
```

```
[0.         0.30103    0.47712125 0.60205999 0.69897    0.77815125
 0.84509804 0.90308999 0.95424251]
```

Cummulative Sum

Cummulative sum means partially adding the elements in array. Perfom partial sum with the cumsum() function.

> For example: The partial sum of [1, 2, 3, 4] would be [1, 1+2, 1+2+3, 1+2+3+4] = [1, 3, 6, 10].

```
import numpy as np

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

newarr = np.cumsum(arr)

print(newarr)
```

```
[1 3 6]
```

Cummulative Product

Cummulative product means taking the product partially. Perfom partial sum with the cumprod() function.

For Example: The partial product of [1, 2, 3, 4] is [1, 1*2, 1*2*3, 1*2*3*4*] = [1, 2, 6, 24]

```
import numpy as np

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

newarr = np.cumprod(arr)

print(newarr)
```

```
[   5   30  210 1680]
```

lcm()

```
import numpy as np

num1 = 4
num2 = 6

x = np.lcm(num1, num2)

print(x)
```

```
12
```

gcd()

```
import numpy as np

num1 = 6
```

```
num2 = 9

x = np.gcd(num1, num2)

print(x)
```

```
3
```

reciprocal()

```
print(np.reciprocal([1, 2., 3.33]))
```

```
[1.        0.5       0.3003003]
```

B) Statistical functions:

These functions are used to calculate mean, median, variance, minimum of array elements.

Some NumPy Statistical Functions are:

Function	Description
amin, amax	returns minimum or maximum of an array or along an axis
median	compute median of data along specified axis
mean	compute mean of data along specified axis
std	compute standard deviation of data along specified axis
var	compute variance of data along specified axis
average	compute average of data along specified axis
percentile(a, p, axis)	calculate pth percentile of array or along specified axis

```
# Python code demonstrate statistical function
import numpy as np

# construct a weight array
weight = np.array([50.7, 52.5, 50, 58, 55.63, 73.25, 49.5, 45])

# minimum and maximum
print('Minimum and maximum weight of the students: ')
print(np.amin(weight), np.amax(weight))


# percentile
print('Weight below which 70 % student fall: ')
print(np.percentile(weight, 70))

# mean
print('Mean weight of the students: ')
print(np.mean(weight))

# median
```

```
print('Median weight of the students: ')
print(np.median(weight))

# standard deviation
print('Standard deviation of weight of the students: ')
print(np.std(weight))

# variance
print('Variance of weight of the students: ')
print(np.var(weight))

# average
print('Average weight of the students: ')
print(np.average(weight))
```

```
Minimum and maximum weight of the students:
45.0 73.25
Range of the weight of the students:
28.25
Weight below which 70 % student fall:
55.317
Mean weight of the students:
54.3225
Median weight of the students:
51.6
Standard deviation of weight of the students:
8.052773978574091
Variance of weight of the students:
64.84716875
Average weight of the students:
54.3225
```

C) Set Functions

A set in mathematics is a collection of unique elements.

Sets are used for operations involving frequent intersection, union and difference operations.

unique()

> We can use NumPy's unique() method to find unique elements from any array. E.g. create a set array, but remember that the set arrays should only be 1-D arrays.

union1d()

> To find the unique values of two arrays.

intersect1d()

> To find only the values that are present in both arrays.

setdiff1d()

> To find only the values in the first set that is NOT present in the seconds set

setxor1d()

> To find only the values that are NOT present in BOTH sets

Note: the setxor1d() method and the intersect1d() method takes an optional argument assume_unique, which if set to True can speed up computation. It should always be set to True when dealing with sets.

With **assume_unique parameter**, based on the below conditions it would take the decision regarding the duplicate values::

If set to TRUE — the intersect1d() function includes the duplicate values as a part of the output.

If set to FALSE — it does not include the duplicate values as the part of the output.

```
import numpy as np

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

x = np.unique(arr)

print(x)
```

```
[1 2 3 4 5 6 7]
```

```
import numpy as np

arr1 = np.array([[1, 2, 3, 4],[11, 22, 23, 44]])
arr2 = np.array([[11, 22, 23, 4],[1, 2, 3, 44]])

print(arr1)
newarr = np.union1d(arr1, arr2)

print(newarr)
```

```
[[ 1  2  3  4]
 [11 22 23 44]]
[ 1  2  3  4 11 22 23 44]
```

```
import numpy as np

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

newarr = np.intersect1d(arr1, arr2, assume_unique=True)

print(newarr)
```

```
[3 4]
```

```
import numpy as np

arr1 = np.array([[1, 2, 3, 4],[11, 22, 23, 44]])
arr2 = np.array([[11, 22, 23, 4],[1, 2, 3, 44]])

newarr = np.setdiff1d(arr1, arr2, assume_unique=True)

print(newarr)
```

```
[]
```

```
import numpy as np

set1 = np.array([1, 2, 3, 4])
set2 = np.array([3, 4, 5, 6])

newarr = np.setxor1d(set1, set2, assume_unique=True)

print(newarr)
```

```
[1 2 5 6]
```

Double-click (or enter) to edit

D) Fast Element-wise Array Functions

i) Bit-twiddling functions:

These functions accept integer values as input arguments and perform bitwise operations on binary representations of those integers.

Some NumPy Bit-twiddling functions are:

Function	Description
bitwise_and	performs bitwise and operation on two array elements
bitwise_or	performs bitwise or operation on two array elements
bitwise_xor	performs bitwise xor operation on two array elements
invert	performs bitwise inversion of an array elements
left_shift	shift the bits of elements to left
right_shift	shift the bits of elements to left

```
# Python code to demonstrate bitwise-function
import numpy as np

# construct an array of even and odd numbers
even = np.array([0, 2, 4, 6, 8, 16, 32])
odd = np.array([1, 3, 5, 7, 9, 17, 33])


# bitwise_and
print('bitwise_and of two arrays: ')
print(np.bitwise_and(even, odd))

# bitwise_or
```

```
print('bitwise_or of two arrays: ')
print(np.bitwise_or(even, odd))

# bitwise_xor
print('bitwise_xor of two arrays: ')
print(np.bitwise_xor(even, odd))

# invert or not
print('inversion of even no. array: ')
print(np.invert(even))


a = np.array([1])

# left_shift
print('left_shift of array: ')
print(np.left_shift(a, 3))

# right_shift
print('right_shift of array: ')
print(np.right_shift(a, 0))
```

```
bitwise_and of two arrays:
[ 0  2  4  6  8 16 32]
bitwise_or of two arrays:
[ 1  3  5  7  9 17 33]
bitwise_xor of two arrays:
[1 1 1 1 1 1 1]
inversion of even no. array:
[ -1  -3  -5  -7  -9 -17 -33]
left_shift of array:
[8]
right_shift of array:
[1]
```

ii) Comparison Functions

Applies > , >= , < , <= , != , == operators on array elements and returns True or False values.

Function	Description
greater	Return the truth value of (x1 > x2) element-wise.
greater_equal	Return the truth value of (x1 >= x2) element-wise.
less	Return the truth value of (x1 < x2) element-wise.
less_equal	Return the truth value of (x1 =< x2) element-wise.
not_equal	Return (x1 != x2) element-wise.
equal	Return (x1 == x2) element-wise.

Note:

> Do not use the Python keywords and and or to combine logical array expressions. These keywords will test the truth value of the entire array (not element-by-element as you might expect). Use the bitwise operators & and | instead.

```
import numpy as np

a = np.array([1,2,3,4,5,6,7,8,9,10])
b = np.array([12,32,56,4,3,8,23,9,18,1])

print(a>b)
print("\n")

print(a>=b)
print("\n")
print(a<b)
print("\n")

print(a>b)
print("\n")

print(a<=b)
print("\n")

print(a!=b)
print("\n")

print(a==b)
print("\n")
```

```
[False False False False  True False False False False  True]


[False False False  True  True False False False False  True]


[ True  True  True False False  True  True  True  True False]


[False False False False  True False False False False  True]


[ True  True  True  True False  True  True  True  True False]


[ True  True  True False  True  True  True  True  True  True]


[False False False  True False False False False False False]
```

iii) Logical Functions

Function	Description
logical_and	Compute the truth value of x1 AND x2 element-wise.
logical_or	Compute the truth value of x1 OR x2 element-wise.
logical_xor	Compute the truth value of x1 XOR x2, element-wise.
logical_not	Compute the truth value of NOT x element-wise.

```
x = np.arange(15)
print(x)
print("\n")

np.logical_and(x>1, x<4)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]


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

```
x = np.arange(15)

np.logical_or(x>1, x<4)
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True])
```

```
x = np.arange(15)

np.logical_xor(x>1, x<4)
```

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

```
x = np.arange(15)

np.logical_not(x>1, x<4)
```

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