PAGE NO. :

Numpy ufuncs

What are ufuncs?
Universal functions
Numpy function that operate on ndarray.

Why ufuncs?
- used to implement vectorization which is faster than iterating over elements.
- Also take additional arguements.

Vectorization
converting iterative statements into vector based operation

1. Add two element of two lists

One way

```
x = [1, 2, 3, 4]
y = [2, 4, 6, 8]
z = []

for i, j in zip(x, y):
    z.append(i+j)
print(z)
```

Note Zip method is used

Teacher's Signature.....

other way

```
x = [1, 2, 3]
y = [4, 5, 6]

z = np.add(x, y)
print(z)
```

only accept two arrays at a time.

features

faster than for loops.
can work on broadcasting type
Memory efficient with out parameter.

Parameters

x1, x2 : input arrays.
out : output array
where : condition
dtype : data type change
casting : controls datatype casting.

Create your own ufunc'
same as creating a normal function.
Add it to the numpy library after.

use 'frompyfunc()' to add to numpy.

It takes three function (parameter)

function - name of the function
inputs - number of input arrays.
output - number of output arrays.

Eg

```
import numpy as np

def myadd(x,y)
    return x+y

myadd = np.frompyfunc(myadd, 2, 1)

x = np.array([1, 2, 3, 4])
y = np.array([5, 6, 7, 9])

print(myadd(x, y))
```

check for normal / ufunc
check type of func.
A ufunc should return → numpy.ufunc
if not recognized then error.

Note Also can use if statement with type

```
if type(myadd) == np.ufunc :
```

PAGE NO.:

Simple arithmetic

We have functions that can take any array like structures like list or tuple etc.. and perform arithmetic conditionally.

means we define condition where operation take place.

Note All function take parameter where().

functions

- add
- substract
- multiply
- divide → answer in float
- power →
- mod() or remainder()
- divmod() → return two arrays.
  i) contain quotient
  ii) contain mod.
- Absolute() and abs() function do same thing but we use absolute.
  return the positive value.

Note Syntax for all the functions above is same as of add function.

Teacher's Signature .....

PAGE NO. :

Rounding Decimals
Their are five ways to do it :-

-1 Truncation
- Remove decimals and return float number closest to zero.
- use trunc() and fix() functions

```
import numpy as np.
arr = np.array([4.667, 3.231, -5.012])

x = np.trunc(arr)
print(x)
=> [4. , 3. , -5. ]
```

-2 Rounding
around() function is used.
increment preceeding digit by 1 if >=5

Takes two parameters.
array and no of decimal places.

```
from numpy import random as rm
import numpy as np.

x = rm.uniform(low=0, high=5, size=4)
arr = np.around(x, 2)
print(arr)
```

Teacher's Signature.....

3. Floor
rounds of decimal to nearest lower integer.
Eg 3.99 → 3.

```
x = np.floor([3.99, -3.167])
print(x)
```

⇒ [3., -4.]

4. ceil
rounds of to nearest upper integer.
Eg 3.1 → 4.

```
x = np.ceil([3.1, -4.77])
print(x)
```

→ [4., -4.]

Numpy logs
Provide function to perform log at base 2, 10 or e.
log function will place -inf or inf in elements if cannot be computed.

log2, log10, log (for e base)

```
x = np.arange(1, 10)
print(np.log(x))
```

⇒ arange creates an integer array from 1-10 excluding 10.
It also take step size.

PAGE NO. :

Numpy Summations

addition is done over two arguements

summation happens over n elements.

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

print(np.add(arr1, arr2))    # [2 4 6]

print(np.sum([arr1, arr2]))  # [12]
```

Note: You can define axis also.

∴ `print(np.sum([arr1, arr2], axis=1)`

⇒ [6 6]

Cumulative Sum

It means ⇒ arr [1, 2, 3]

Its cumulative sum will be ⇒ [1 1+2, 1+2+3]

⇒ [1 3 6]

For this we can use function 'cumsum'.

Numpy Product

Product of all elements in the array.

Teacher's Signature.....

prod() is used.

```
x=np.prod([1, 2, 3, 4])
print(x)
```

=> 24

Or

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

x = np.prod(arr1, arr2)
print(x)
```

=> 36

We can also specify axis

```
x = np.prod([arr1, arr2], axis=1)
print(x)
```

=> [1 4 9]

Cumulative product

```
x = np.prod([1, 2, 3]) => 6
x = np.cumprod([1, 2, 3]) => [1 2 6]
```

PAGE NO.:

Numpy Difference

discrete difference means substracting two successive element.

Eg for [1 2 3 4] => [1 1 1]

We use diff() function.

```
arr = np.array([10, 15, 25, 5])
newarr = np.diff(arr)
print(newarr)
```

=> [15-10 25-15 5-25] => [5 10 -20]

Note: we can do this n number of times using parameter n.

∴ For above.

```
newarr = np.diff(arr, n = 2)
```

=> [5 10 -20] => [5 -30]

Numpy LCM

we can find lcm of numbers by Lcm().

```
num1 = 3
num2 = 4

x = np.lcm(num1, num2)
print(x) => 12
```

Teacher's Signature.....

For numpy array we use reduce().

```
arr = np.array([3, 6, 9])

x = np.lcm.reduce(arr)
print(x)
```

=> 18

finds lcm for all numbers in array.

Numpy Gcd

Also known as hcf → highest common factor.

- To find for numbers

```
num1 = 3
num2 = 6

arr = np.gcd(num1, num2)
print(arr)
```

=> 3

- For arrays.

```
arr = np.array([1, 2, 3, 4, 5])
x = np.gcd.reduce(arr)
print(x)
```

=> 1

Teacher's Signature

PAGE NO.:

Numpy Trigonometric function

Numpy provide ufuncs sin(), cos(), tan()
These take values in radians

```
arr = np.array([np.pi/2, np.pi/3])
x = np.sin(arr)
print(x)
```

=> [1.0, 0.866]

Convert deg to radian

$$rad = \frac{pi}{180} * deg$$

- ```
  arr = np.array([90, 45, 360])
  x = np.deg2rad(arr)
  print(x)
  ```

Note: We can find angles from sin, cos, tan values.
We can use arcsin, arccos, arctan()
These produce radian values.

```
x = np.arcsin(1.0)
print(x)
```

=> 1.570

- hypot() → takes base and perpendicular to calculate hypotenuse.
  give answer in float.

Teacher's Signature.....
```

Numpy set operation

- set is a collection of unique elements.
- used for operations involving frequent intersection union and difference operations.

Creation

we can use unique() to find unique elements.

```
arr = np.array([1, 1, 3, 4, 5, 5, 7, 7, 1])
x = np.unique(arr)
print(x)
```

⇒ [1 3 4 5 7]

Note array should be 1D.

To find unique values of two arrays use 'union1d'

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

x = np.union1d(arr1, arr2)
print(x)
```

⇒ [1 2 3 4 5 6]

intersection()

```
x = np.intersect1d(arr1, arr2, assume_unique=True)
```

This returns elements present in both arrays.

Note assume_unique can speed up computation.

- It should always be set to True when working with sets

To find values present only in set 1 use => setdiff1d()

To find values not present in both set use -> setxor1d()

Teacher's Signature.....