ARRAY
In
python

Single **D**imensional **A**rrays

Single Dimensional Arrays

Creating an Array

Syntax	array_name = array(type_code, [elements])
Example-1	a = array(‘i’, [4, 6, 2, 9])
Example-2	a = array(‘d’, [1.5, -2.2, 3, 5.75])

Single Dimensional Arrays

Creating an Array

Typecode	C Type	Sizes
‘b’	signed integer	1
‘B’	unsigned integer	1
‘i’	signed integer	2
‘I’	unsigned integer	2
‘l’	signed integer	4
‘L’	unsigned integer	4
‘f’	floating point	4
‘d’	double precision floating point	8
‘u’	unicode character	2

Single Dimensional Arrays

Importing an Array Module

```
import array                      a  =  array.array(‘i’, [4, 6, 2, 9])

import array as ar                a  =  ar.array(‘i’, [4, 6, 2, 9])

from array import        *        a  =  array(‘i’, [4, 6, 2, 9])
```

Importing an Array Module

Example-1

```
import array


#Create an array
a = array.array("i", [1, 2, 3, 4])


#print the items of an array  print("Items
are: ")
for i in a:
        print(i)
```

Importing an Array Module

Example-2

```
from array import *


#Create an array
a = array("i", [1, 2, 3, 4])


#print the items of an array  print("Items
are: ")
for i in a:
        print(i)
```

Importing an Array Module

Example-3

```
from array import *


#Create an array
a = array('u', ['a', 'b', 'c', 'd'])  #Here, 'u' stands for unicode
character



#print the items of an array  print("Items
are: ")
for ch in a:
        print(ch)
```

Importing an Array Module

Example-4

```
from array import *


#Create first array

a = array('i', [1, 2, 3, 4])


#From first array create second

b = array(a.typecode, (i for i in a))


#print the second array items  print("Items

are: ")

for i in b:

          print(i)


#From first array create third

c = array(a.typecode, (i * 3 for i in a))


#print the second array items  print("Items

are: ")

for i in c:

          print(i)
```

Indexing & Slicing on Array

Example-1: Indexing

```
#To retrieve the items of an array using array index

from array import *


#Create an array

a = array('i', [1, 2, 3, 4])


#Get the length of the array  n = len(a)


#print the Items  for i in
range(n):
        print(a[i], end=' ')
```

Indexing & Slicing on Array

Example-2: Indexing

```
#To retrieve the items of an array using array index using while loop

from array import *


#Create an array

a = array('i', [1, 2, 3, 4])


#Get the length of the array  n = len(a)


#print the Items  i = 0

while i < n:
        print(a[i], end=' ')  i += 1
```

Indexing & Slicing on Array

Slicing

Syntax	arrayname[start: stop: stride]
Example	arr[1: 4: 1] Prints items from index 1 to 3 with the step size of 1

Indexing & Slicing on Array

Example-3: Slicing

```
#Create an array
x = array('i', [10, 20, 30, 40, 50, 60])
```

```
#Create array y with Items from 1st to 3rd from x  y = x[1: 4]
print(y)
```

```
#Create array y with Items from 0th till the last Item in x  y = x[0: ]
print(y)
```

```
#Create array y with Items from 0th till the 3rd Item in x  y = x[: 4]
print(y)
```

```
#Create array y with last 4 Items in x  y = x[-4: ]
print(y)
```

```
#Stride 2 means, after 0th Item, retrieve every 2nd Item from x  y = x[0: 7: 2]
print(y)
```

```
#To display range of items without storing in an array  for i in x[2: 5]:
        print(i)
```

Indexing & Slicing on Array

Example-4: Slicing

```
#To retrieve the items of an array using array index using for loop

from array import *

#Create an array
a = array('i', [1, 2, 3, 4])

#Display elements from  for i in   2nd  to  4th  only
a[2: 5]:
        print(i)
```

Processing the Array

Method	Description
a.append(x)	Adds an element x at the end of the existing array a
a.count(x)	Returns the numbers of occurrences of x in the array a
a.extend(x)	Appends x at the end of the array a. ‘x’ can be another array or iterable object an
a.index(x)	Returns the position number of the first occurrence of x in the array. Raises ‘ValueError’ if not found
a.insert(i, x)	Inserts x in the position i in the array

Processing the Array

Method	Description
a.pop(x)	Removes the item x from the arry a and returns it
a.pop()	Removes last item from the array a
a.remove(x)	Removes the first occurrence of x in the array a. Raises ‘ValueError’ if not found
a.reverse()	Reverse the order of elements in the array a
a.tolist()	Converts the array ‘a’ into a list

Processing the Array

Examples

```
from array import *
#Create an array
a = array('i', [1, 2, 3, 4, 5])
print(a)

#Append 6 to an array
a.append(6)
print(a)


#Insert 11 at position 1
a.insert(1, 11)
print(a)

#Remove 11 from the array
a.remove(11)
print(a)

#Remove last item using pop()  item =
a.pop()
print(a)
print("Item pop: ", item)
```

Processing the Array

Exercises

1. To store student’s marks into an array and find total marks and percentage of marks

2. Implement Bubble sort

3. To search for the position of an item in an array using sequential search

4. To search for the position of an element in an array using index() method

Single Dimensional Arrays Numpy

Single Dimensional Arrays

Importing an **numpy**

import numpy	a = numpy.array([4, 6, 2, 9])
import numpy as np	a = np.array([4, 6, 2, 9])
from numpy import *	a = array([4, 6, 2, 9])

Single Dimensional Arrays

Creating an Array: **numpy-array()**

Example-1: To create an array of **int** datatype

```
a = array([10, 20, 30, 40, 50], int)
```

Example-2: To create an array of **float** datatype

```
a = array([10.1, 20.2, 30.3, 40.4, 50.5], float)
```

Example-3: To create an array of **float** datatype without specifying the float datatype

```
a = array([10, 20, 30.3, 40, 50])
```

Note: If one item in the array is of float type, then Python interpreter converts remaining items into the float datatype

Example-4: To create an array of **char** datatype

```
a = array([‘a’, ‘b’, ‘c’, ‘d’])
```

Note: No need to specify explicitly the char datatype

Single Dimensional Arrays

Creating an Array: **numpy-array()**

Program-1: To create an array of **char** datatype

```
from numpy import *

a = array(['a', 'b', 'c', 'd'])  print(a)
```

Program-2: To create an array of **str** datatype

```
from numpy import *

a = array(['abc', 'bcd', 'cde', 'def'], dtype=str)  print(a)
```

Single Dimensional Arrays

Creating an Array: **numpy-array()**

Program-3: To create an array from another array using numpy

```
from numpy import *

a = array([1, 2, 3, 4, 5])  print(a)

#Create another array using array() method  b = array(a)
print(a)

#Create another array by just copy  c = a
print(a)
```

Single Dimensional Arrays

Creating an Array: numpy-linspace()

Syntax	linspace(start, stop, n)
Example	a = linspace(0, 10, 5)
Description	Create an array ‘a’ with starting element 0 and ending 10. This range is divide into 5 equal parts Hence, items are 0, 2.5, 5, 7.5, 10

Program-1: To create an array with 5 equal points using linspace

```
from numpy import *


#Divide 0 to 10 into 5 parts and take those points in the array  a = linspace(0, 10, 5)
print(a)
```

Single Dimensional Arrays

Creating an Array: numpy-logspace()

Syntax	logspace(start, stop, n)
Example	a = logspace(1, 4, 5)
Description	Create an array ‘a’ with starting element 10^1 and ending 10^4. This range is divide into 5 equal parts Hence, items are 10. 56.23413252 316.22776602 1778.27941004 10000.

Program-1: To create an array with 5 equal points using logspace

```
from numpy import *


#Divide the range 10^1 to 10^4 into 5 equal parts  a = logspace(1, 4, 5)
print(a)
```

Single Dimensional Arrays

Creating an Array: numpy-arange()

Syntax	arange(start, stop, stepsize)			
Example-1	arange(10)	Produces	items from	0 - 9
Example-2	arange(5, 10)	Produces	items from	5 - 9
Example-3	arange(1, 10, 3)	Produces	items from	1, 4, 7
Example-4	arange(10, 1, -1)	Produces	items from	[10 9 8 7 6 5 4 3 2]
Example-5	arange(0, 10, 1.5)	Produces	[0. 1.5 3.	4.5 6. 7.5 9.]

Program-1: To create an array with even number upto 10

```
from numpy import *

a = arange(2, 11, 2)  print(a)
```

Single Dimensional Arrays

Creating Array: numpy-zeros() & ones()

Syntax	zeros(n, datatype)
	ones(n, datatype)	

Example-1	zeros(5)	Produces items [0. 0. 0. 0. 0.] Default datatype is float
Example-2	zeros(5, int) ones(5,	Produces items [0 0 0 0 0]
Example-3	float)	Produces items [1. 1. 1. 1. 1.]

Program-1: To create an array using zeros() and ones()

```
from numpy import *

a = zeros(5, int)  print(a)

b = ones(5) #Default datatype is float  print(b)
```

Single Dimensional Arrays

Vectorized Operations

Example-1	a = array([10, 20 30.5, -40]) a = a + 5 #Adds 5 to each item of an array

Example-2	a1 = array([10, 20 30.5, -40])
	a2 = array([1, 2, 3, 4])
	a3 = a1 + a2 #Adds each item of a1 and a2

Importance of vectorized operations

1. Operations are faster
   - Adding two arrays in the form a + b is faster than taking corresponding items of both arrays and then adding them.

2. Syntactically clearer
   - Writing a + b is clearer than using the loops

3. Provides compact code

Single Dimensional Arrays

Mathematical Operations

sin(a)	Calculates sine value of each item in the array a
arcsin(a)	Calculates sine inverse value of each item in the array a
log(a)	Calculates natural log value of each item in the array a
abs(a)	Calculates absolute value of each item in the array a
sqrt(a)	Calculates square root value of each item in the array a
power(a, n)	Calculates a ^ n
exp(a)	Calculates exponential value of each item in the array a
sum(a)	Calculates sum of each item in the array a
prod(a)	Calculates product of each item in the array a
min(a)	Returns min value in the array a
max(a)	Returns max value in the array a

Single Dimensional Arrays

Comparing Arrays

- Relational operators are used to compare arrays of same size
- These operators compares corresponding items of the arrays and return another array with Boolean values

Program-1: To compare two arrays and display the resultant Boolean type array

```
from numpy import *

a = array([1, 2, 3])
b = array([3, 2, 3])

c = a == b
print(c)

c = a > b
print(c)

c = a <= b
print(c)
```

Single Dimensional Arrays

Comparing Arrays

- any(): Used to determine if any one item of the array is True
- all(): Used to determine if all items of the array are True

Program-2: To know the effects of any() and all()

```
from numpy import *

a = array([1, 2, 3])
b = array([3, 2, 3])

c = a > b
print(c)

print("any(): ", any(c))
print("all(): ", all(c))

if (any(a > b)):
        print("a contains one item greater than those of b")
```

Single Dimensional Arrays

Comparing Arrays

- logical_and(), logical_or() and logical_not() are useful to get the Boolean array as a
- result of comparing the compound condition

Program-3: To understand the usage of logical functions

```
from numpy import *

a = array([1, 2, 3])
b = array([3, 2, 3])

c = logical_and(a > 0, a < 4)  print(c)
```

Single Dimensional Arrays

Comparing Arrays

- where(): used to create a new array based on whether a given condition is True or False
- Syntax: a = where(condition, exp1, exp2)
    - If condition is True, the exp1 is evaluated, the result is stored in array
    - a, else exp2 will be evaluated

Program-4: To understand the usage of where function

```
from numpy import *

a = array([1, 2, 3], int)

c = where(a % 2 == 0, a, 0)  print(c)
```

Single Dimensional Arrays

Comparing Arrays

- where(): used to create a new array based on whether a given condition is True or False
- Syntax: a = where(condition, exp1, exp2)
    - If condition is True, the exp1 is evaluated, the result is stored in array
    - a, else exp2 will be evaluated

Exercise-1: To retrieve the biggest item after comparing two arrays using where()

Single Dimensional Arrays

Comparing Arrays

- nonzero(): used to know the positions of items which are non-zero
  - Returns an array that contains the indices of the items of the array which are non-zero

- Syntax: a = nonzero(array)

Program-5: To retrieve non zero items from an array

```
from numpy import *

a = array([1, 2, 0, -1, 0, 6], int)  c = nonzero(a)


#Display the indices  for i in c:
        print(i)



#Display the items
print(a[c])
```

Single Dimensional Arrays

Aliasing Arrays

- ‘Aliasing means not copying’. Means another name to the existing object

Program-1: To understand the effect of aliasing

```
from numpy import *

a = arange(1, 6)
b = a
print(a)
print(b)

#Modify 0th Item
b[0] = 99
print(a)
print(b)
```

Single Dimensional Arrays

Viewing & Copying

- view(): To create the duplicate array
- Also called as ‘shallow copying’

Program-1: To understand the view()

```
from numpy import *

a = arange(1, 6)
b = a.view() #Creates new array  print(a)
print(b)

#Modify 0th Item  b[0]
= 99
print(a)
print(b)
```

Single Dimensional Arrays

Viewing & Copying

- copy(): To create the copy the original array
- Also called as ‘deep copying’

Program-1: To understand the view()

```
from numpy import *

a = arange(1, 6)
b = a.copy() #Creates new array  print(a)
print(b)

#Modify 0th Item  b[0]
= 99
print(a)
print(b)
```

Multi **D**imensional **A**rrays
Numpy

Multi Dimensional Arrays

Creating an Array

Example-1: To create an 2D array with 2 rows and 3 cols

```
a  =          array([[1, 2, 3],
                     [4, 5, 6]]
```

Example-2: To create an 3D array with 2-2D arrays with each 2 rows and 3 cols

```
a  =  array([[[1, 2, 3],[4, 5, 6]]
              [[1, 1, 1], [1, 0, 1]]]
```

Multi Dimensional Arrays

Attributes of an Array: *The ndim*

- The 'ndim' attribute represents the number of dimensions or axes of an array
- The number of dimensions are also called as 'rank'

Example-1: To understand the usage of the ndim attribute

```
a = array([1, 2, 3])

print(a.ndim)
```

Example-2: To understand the usage of the ndim attribute

```
a = array([[[1, 2, 3],[4, 5, 6]]
                     [[1, 1, 1], [1, 0, 1]]]

print(a.ndim)
```

Multi Dimensional Arrays

Attributes of an Array: *The shape*

- The ‘shape’ attribute gives the shape of an array
- The shape is a tuple listing the number of elements along each dimensions

Example-1: To understand the usage of the ‘shape’ attribute

```
a = array([1, 2, 3])

print(a.shape)
```

```
Outputs: (5, )
```

Example-2: To understand the usage of the ‘shape’ attribute

```
a = array([[1, 2, 3],[4, 5, 6]])

print(a.shape)
```

```
Outputs: (2, 3)
```

Example-3: To ‘shape’ attribute also changes the rows and cols

```
a = array([[1, 2, 3],[4, 5, 6]])

a.shape = (3, 2)

print(a)
```

```
Outputs:

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

Multi Dimensional Arrays

Attributes of an Array: *The size*

- The ‘size’ attribute gives the total number of items in an array

Example-1: To understand the usage of the ‘size’ attribute

```
a = array([1, 2, 3])

print(a.size)
```

Outputs: 5

Example-2: To understand the usage of the ‘size’ attribute

```
a = array([[1, 2, 3],[4, 5, 6]])

print(a.size)
```

Outputs: 6

Multi Dimensional Arrays

Attributes of an Array: *The itemsize*

- The 'itemsize' attribute gives the memory size of an array element in bytes

Example-1: To understand the usage of the 'itemsize' attribute

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

print(a.itemsize)
```

Outputs: 4

Example-2: To understand the usage of the 'size' attribute

```
a = array([1.1, 2.3])

print(a.itemsize)
```

Outputs: 8

Multi Dimensional Arrays

Attributes of an Array: *The dtype*

- The ‘dtype’ attribute gives the datatype of the elements in the array

Example-1: To understand the usage of the ‘dtype’ attribute

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

print(a.dtype)
```

Outputs: int32

Example-2: To understand the usage of the ‘dtype’ attribute

```
a = array([1.1, 2.3])

print(a.dtype)
```

Outputs: float64

Multi Dimensional Arrays

Attributes of an Array: *The nbytes*

- The ‘nbytes’ attribute gives the total number of bytes occupied by an array

Example-1: To understand the usage of the ‘nbytes’ attribute

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

print(a.nbytes)
```

Outputs: 20

Example-2: To understand the usage of the ‘nbytes’ attribute

```
a = array([1.1, 2.3])

print(a.nbytes)
```

Outputs: 16

Multi Dimensional Arrays

Methods of an Array: *The reshape()*

- The ‘reshape’ method is useful to change the shape of an array

Example-1: To understand the usage of the ‘reshape’ method

```
a = arange(10)

#Change the shape as 2 Rows, 5 Cols  a =
a.reshape(2, 5)

print(a)
```

Outputs:

```
[[0 1 2 3 4]
  [5 6 7 8 9]]
```

Example-2: To understand the usage of the ‘reshape’ method

```
#Change the shape to 5 rows, 2 cols
a = a.reshape(5, 2)

print(a)
```

Outputs:

```
[[0 1]
  [2 3]
  [4 5]
  [6 7]
  [8 9]]
```

Multi Dimensional Arrays

Methods of an Array: *The flatten()*

- The 'flatten' method is useful to return copy of an array collapsed into one dimension

Example-1: To understand the usage of the 'flatten' method

```
#flatten() method
a = array([[1, 2], [3, 4]])
print(a)

#Change to 1D array  a =
a.flatten()  print(a)
```

Outputs:

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

Multi Dimensional Arrays

Methods of creating an 2D-Array

- Using array() function
- Using ones() and zeroes() functions
- Uisng eye() function Using
- reshape() function

Multi Dimensional Arrays

Creation of an 2D-Array: *array()*

Example-1:

```
a = array([[1, 2], [3, 4]])
print(a)
```

Outputs:

```
[[1, 2],
[3, 4]]
```

Multi Dimensional Arrays

Creation of an 2D-Array: *ones() & zeros()*

Syntax	zeros((r, c), dtype)
	ones((r, c), dtype)	

Example-1	a = ones((3, 4), float)	Produces items [[1. 1. 1. 1.] [1. 1. 1. 1.] [1. 1. 1. 1.]]
Example-2	b = zeros((3, 4), int)	Produces items [[0 0 0 0] [0 0 0 0] [0 0 0 0]]

Multi **D**imensional **A**rrays

Creation of an 2D-**A**rray: *The eye()*

- The eye() function creates 2D array and fills the items in the diagonal with 1’s

Syntax	eye(n, dtype=datatype)	
Description	- Creates ‘n’ rows & ‘n’ cols - Default datatype is float	
Example-1	a = eye(3)	- Creates 3 rows and 3 cols [[1. 0. 0.] [0. 1. 0.] [0. 0. 1.]]

Multi Dimensional Arrays

Creation of an 2D-Array: *The reshape()*

- Used to convert 1D into 2D or nD arrays

Syntax	reshape(arrayname, (n, r, c))	
Description	arrayname – Represents the name of the array whose elements converted to be n – Numbers of arrays in the resultant array r, c – Number of rows & cols respectively	

Example-1	a = array([1, 2, 3, 4, 5, 6])
	b = reshape(a, (2, 3))
	print(b)
		Outputs:
		[[1 2 3]
		[4 5 6]]

Multi Dimensional Arrays

Creation of an 2D-Array: *The reshape()*

- Used to convert 1D into 2D or nD arrays

Syntax	reshape(arrayname, (n, r, c))	
Description	arrayname – Represents the name of the array whose elements converted to be n – Numbers of arrays in the resultant array r, c – Number of rows & cols respectively	

Example-2	a = arange(12)
	b = reshape(a, (2, 3, 2))
	print(b)
		Outputs:
		[[0 1]
		[2 3]
		[4 5]]
		
		[[6 7]
		[8 9]
		[10 11]]

Multi Dimensional Arrays

Indexing of an 2D-Array

Program-1: To understand indexing of 2D arrays

```
from numpy import *

#Create an 2D array with 3 rows, 3 cols  a = [[1, 2, 3], [4,
5, 6], [7, 8, 9]]


#Display only rows
for i in range(len(a)):  print(a[i])


#display item by item  for i in
range(len(a)):
        for j in range(len(a[i])):
                print(a[i][j], end=' ')
```

Multi Dimensional Arrays

Slicing of an 2D-Array

#Create an array
a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
a = reshape(a, (3, 3)) print(a)
	Produces:
	
	[[1 2 3]
	[4 5 6]
	[7 8 9]]

a[:, :] a[:] a[: :]	Produces: [[1 2 3] [4 5 6] [7 8 9]]
#Display 0th row a[0, :]	
#Display 0th col a[:, 0]	
#To get 0th row, 0th col item a[0:1, 0:1]	

Matrices **in N**umpy

Matrices in Numpy

Syntax	matrix-name = matrix(2D Array or String)	

Example-1	a = [[1, 2, 3], [4, 5, 6]]
	a = matrix(a)
	print(a)
		Outputs:
		[[1 2 3]
		[4 5 6]]

Example-2	a = matrix([[1, 2, 3], [4, 5, 6]])	Outputs: [[1 2 3] [4 5 6]]

Example-3	a = ‘1 2; 3 4; 5 6’
	b = matrix(a)
		[[1 2]
		[3 4]
		[5 6]]

Matrices in Numpy

Getting Diagonal Items

Function	diagonal(matrix)	

Example-1	#Create 3 x 3 matrix
	a = matrix("1 2 3; 4 5 6; 7 8 9")
	
	#Find the diagonal items d =
	diagonal(a)
	print(d)
		Outputs:
		
		[1 5 9]

Matrices in Numpy

Finding Max and Min Items

Function	max()
	min()	

Example-1	#Create 3 x 3 matrix
	a = matrix("1 2 3; 4 5 6; 7 8 9")
	
	#Print Max + Min Items big =
	a.max()
	small = a.min() print(big,
	small)
		Outputs:
		9 1

Matrices in Numpy

Exercise

1. To find sum, average of elements in 2D array

2. To sort the Matrix row wise and column wise

3. To find the transpose of the matrix

4. To accept two matrices and find thier sum

5. To accept two matrices and find their product

Note: Read the matrices from the user and make the program user friendly

THANK
YOU