VIT-AP UNIVERSITY, ANDHRA PRADESH

DATA WAREHOUSING AND DATA MINING

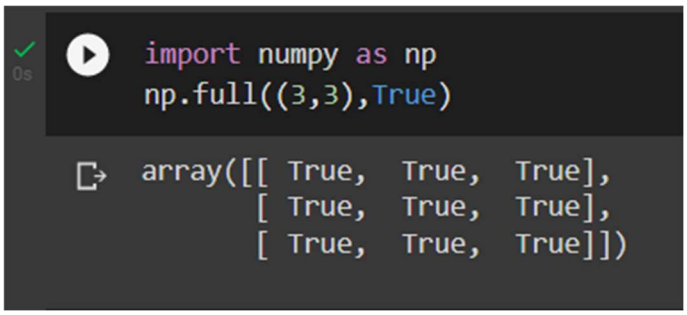
Academic year: 2022-2023 **Branch/ Class:** B.Tech

Semester: Fall **Date:**

Faculty Name: Dr Aravapalli Rama Satish **School:** SCOPE

NAME: Majjiga Jaswanth REGNO:20BCD7171

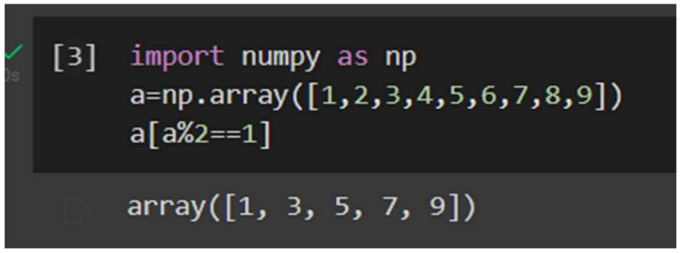
1 . Create a NumPy Boolean array of 3 * 3 with True values.



```
import numpy as np
np.full((3,3),True)
```

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

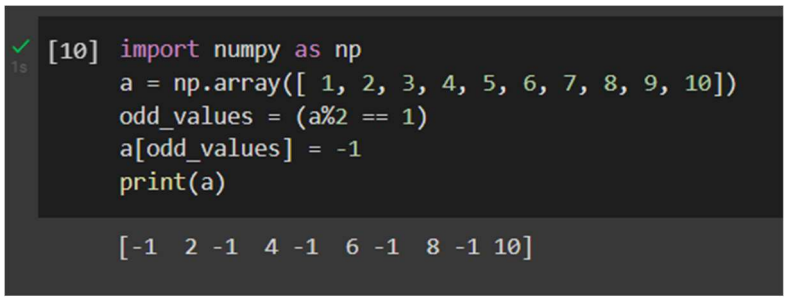
2. Extract Odd numbers from any array.



```
[3] import numpy as np
    a=np.array([1,2,3,4,5,6,7,8,9])
    a[a%2==1]
```

```
array([1, 3, 5, 7, 9])
```

3. Replace all Odd numbers with -1 without affecting the original array.



```
[10] import numpy as np
     a = np.array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
     odd_values = (a%2 == 1)
     a[odd_values] = -1
     print(a)
```

```
[-1  2 -1  4 -1  6 -1  8 -1 10]
```

4. Reshape the Given sequential array in to ‘n’ rows.

```
[12] import numpy as np
     arr=np.array([1,0,0,0,1,0,0,0,1])
     newarr=arr.reshape(3,3)
     print(newarr)

     [[1 0 0]
      [0 1 0]
      [0 0 1]]
```

5. Combine two arrays vertically and horizontally

```
import numpy as np
arr = np.array([5,9,2])
arr1 = np.array([11,22,33])
con = np.concatenate((arr, arr1))
print(con)
con = np.hstack((arr, arr1))
print(con)
con = np.vstack((arr, arr1))
print(con)

[ 5  9  2 11 22 33]
[ 5  9  2 11 22 33]
[[ 5  9  2]
 [11 22 33]]
```

6. Common items between two arrays.

```
import numpy as np
a1=np.array([25,90,26,14,75,36,18,82])
a2=np.array([95,65,25,14,36,18,28,63])
print(np.intersect1d(a1, a2))

[14 18 25 36]
```

7. From array ‘a’ remove all items present in array ‘b’

```
[8] import numpy as np
    a1=np.array([25,90,26,14,75,36,18,82])
    a2=np.array([95,65,25,14,36,18,28,63])
    print(np.setdiff1d(a1, a2))

    [26 75 82 90]
```

8. Print the matching positions of two element arrays.

```
import numpy as np
a=np.array([25,90,26,14,75,36,18,82])
b=np.array([95,65,25,14,36,18,28,63])
np.where(a == b)
```

```
(array([3]),)
```

9. Extract all numbers between range of indices from the given array.

```
[17] import numpy as np
     numpy_array = np.array([25,90,26,14,75,36,18,82])
     indices = np.where((numpy_array >= 7) & (numpy_array < 11))
     print("Indices: ", indices)
     print("Elements satisfied the condition: ",numpy_array[indices])
```

```
Indices:  (array([], dtype=int64),)
Elements satisfied the condition:  []
```

10. Convert a Scalar function to work on NumPy arrays.

```
[20] import numpy as np
     def maxx(x, y):
       """Get the maximum of two items"""
       if x >= y:
         return x
       else:
         return y
     pair_max = np.vectorize(maxx, otypes=[float])
     a = np.array([5, 7, 9, 8, 6, 4, 5])
     b = np.array([6, 3, 4, 8, 9, 7, 1])
     pair_max(a, b)
```

```
array([6., 7., 9., 8., 9., 7., 5.])
```

11. Swap two columns/rows of a 2D NumPy array.

```
import numpy as np

my_array = np.arange(12).reshape(4, 3)
print("Original array:")
print(my_array)
my_array[:, [2, 0]] = my_array[:, [0, 2]]
print("After swapping arrays the last column and first column:")
print(my_array)
import numpy as np
my_array = np.arange(16).reshape(4, 4)
print("Original array:")
print(my_array)
arr = np.arange(16).reshape(4,4)
print("After changing rows:")
arr[[1,0,3,2], :]
```

```
Original array:
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
After swapping arrays the last column and first column:
[[ 2  1  0]
 [ 5  4  3]
 [ 8  7  6]
 [11 10  9]]
Original array:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
After changing rows:
array([[ 4,  5,  6,  7],
       [ 0,  1,  2,  3],
       [12, 13, 14, 15],
       [ 8,  9, 10, 11]])
```

12. Reverse the rows/columns of a 2D NumPy array.

```
arr = np.array(
    [[10, 20, 30],
    [40, 50, 60],
    [70, 80, 90]])
    flipped_arr = np.fliplr(arr)
    print('Array before changing column order:\n', arr)
    print('\nArray after changing column order:\n', flipped_arr)
    import numpy as np
    arr = np.arange(9).reshape(3,3)
    print("Original array: ")
    print(arr)
    print("After reversing rows: ")
    arr[::-1]
```

```
Array before changing column order:
 [[10 20 30]
 [40 50 60]
 [70 80 90]]

Array after changing column order:
 [[30 20 10]
 [60 50 40]
 [90 80 70]]
Original array:
[[0 1 2]
 [3 4 5]
 [6 7 8]]
After reversing rows:
array([[6, 7, 8],
       [3, 4, 5],
       [0, 1, 2]])
```

13. Create a 2D array of shape 5x3 to contain random decimal numbers between 5 and 10. Print only three decimal places for a floating-point number.

```
arr = np.arange(9).reshape(3,3)
rand_arr = np.random.randint(low=5, high=10, size=(5,3)) + np.random.random((5,3))
rand_arr = np.random.uniform(5,10, size=(5,3))
print(rand_arr)
rand_arr = np.random.random((5,3))
rand_arr = np.random.random([5,3])
np.set_printoptions(precision=3)
rand_arr[:4]
```

```
[[8.873 5.333 6.179]
 [6.381 7.649 8.436]
 [7.757 7.28  9.785]
 [8.205 8.586 9.852]
 [7.531 5.821 6.803]]
array([[0.363, 0.171, 0.501],
       [0.798, 0.037, 0.294],
       [0.629, 0.221, 0.095],
       [0.904, 0.282, 0.235]])
```

14. Suppress the scientific notation in printing floating point number.

```
[26] np.set_printoptions(suppress=False)
     np.random.seed(100)
     rand_arr = np.random.random([3,3])/1e3
     rand_arr
```

```
array([[5.434e-04, 2.784e-04, 4.245e-04],
       [8.448e-04, 4.719e-06, 1.216e-04],
       [6.707e-04, 8.259e-04, 1.367e-04]])
```

15. Print the full numpy array ‘a’ without truncating.

```
np.set_printoptions(threshold=6)
sys = np.arange(15)
np.set_printoptions(threshold=sys.size)
sys
```

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

16. Create a 4x2 integer array and Prints its attributes: shape, dimensions, length of each element of the array.

```
[28] firstArray = np.empty([4,2], dtype = np.uint16)
     print("Printing Array")
     print(firstArray)
     print("Printing numpy array Attributes")
     print("1> Array Shape is: ", firstArray.shape)
     print("2>. Array dimensions are ", firstArray.ndim)
     print("3>. Length of each element of array in bytes is ", firstArray.itemsize)
```

```
Printing Array
[[46409 11871]
 [30354 15642]
 [ 8848 49893]
 [47871 27296]]
Printing numpy array Attributes
1> Array Shape is:  (4, 2)
2>. Array dimensions are  2
3>. Length of each element of array in bytes is  2
```

17. Create a 5X2 integer array from a range between 100 to 200 such that the difference between each element is 10.

```
print("Creating 5X2 array using numpy.arange")
sampleArray = np.arange(100, 200, 10)
sampleArray = sampleArray.reshape(5,2)
print (sampleArray)
```

```
Creating 5X2 array using numpy.arange
[[100 110]
 [120 130]
 [140 150]
 [160 170]
 [180 190]]
```

18. Return array of odd rows and even columns from below NumPy array.

```
sampleArray = np.array([[3 ,6, 9, 12], [15 ,18, 21, 24],
[27 ,30, 33, 36], [39 ,42, 45, 48], [51 ,54, 57, 60]])
print("Printing Input Array")
print(sampleArray)
print("\n Printing array of odd rows and even columns")
newArray = sampleArray[::2, 1::2]
print(newArray)
```

```
Printing Input Array
[[ 3  6  9 12]
 [15 18 21 24]
 [27 30 33 36]
 [39 42 45 48]
 [51 54 57 60]]

 Printing array of odd rows and even columns
[[ 6 12]
 [30 36]
 [54 60]]
```

19. Split the array into four equal-sized sub-arrays.

```
print("Creating 8X3 array using numpy.arange")
sampleArray = np.arange(10, 34, 1)
sampleArray = sampleArray.reshape(8,3)
print (sampleArray)
print("\nDividing 8X3 array into 4 sub array\n")
subArrays = np.split(sampleArray, 4)
print(subArrays)
```

```
Creating 8X3 array using numpy.arange
[[10 11 12]
 [13 14 15]
 [16 17 18]
 ...
 [25 26 27]
 [28 29 30]
 [31 32 33]]

Dividing 8X3 array into 4 sub array

[array([[10, 11, 12],
       [13, 14, 15]]), array([[16, 17, 18],
       [19, 20, 21]]), array([[22, 23, 24],
       [25, 26, 27]]), array([[28, 29, 30],
       [31, 32, 33]])]
```

20. Sort following NumPy array: 1: Sort array by the second row 2: Sort the array by the second column

```
print("Printing Original array")
sampleArray = np.array([[34,43,73],[82,22,12],[53,94,66]])
print (sampleArray)
sortArrayByRow = sampleArray[:,sampleArray[1,:].argsort()]
print("Sorting Original array by secoond row")
print(sortArrayByRow)
print("Sorting Original array by secoond column")
sortArrayByColumn = sampleArray[sampleArray[:,1].argsort()]
print(sortArrayByColumn)
```

```
Printing Original array
[[34 43 73]
 [82 22 12]
 [53 94 66]]
Sorting Original array by secoond row
[[73 43 34]
 [12 22 82]
 [66 94 53]]
Sorting Original array by secoond column
[[82 22 12]
 [34 43 73]
 [53 94 66]]
```

21. Print max from axis 0 and min from axis 1 from the 2-D array.

```
[33] print("Printing Original array")
sampleArray = np.array([[34,43,73],[82,22,12],[53,94,66]])
print (sampleArray)
minOfAxisOne = np.amin(sampleArray, 1)
print("Printing amin Of Axis 1")
print(minOfAxisOne)
maxOfAxisOne = np.amax(sampleArray, 0)
print("Printing amax Of Axis 0")
print(maxOfAxisOne)
```

```
Printing Original array
[[34 43 73]
 [82 22 12]
 [53 94 66]]
Printing amin Of Axis 1
[34 12 53]
Printing amax Of Axis 0
[82 94 73]
```

22. Delete the second column from a given array and insert the new column in its place.

```
from numpy.lib import npyio
from numpy.compat.py3k import npy_load_module
print("Printing Original array")
sampleArray = np.array([[34,43,73],[82,22,12],[53,94,66]])
print (sampleArray)
print("Array after deleting column 2 on axis 1")
sampleArray = np.delete(sampleArray , 1, axis = 1)
print (sampleArray)
arr = np.array([[10,10,10]])
print("Array after inserting column 2 on axis 1")
sampleArray = np.insert(sampleArray , 1, arr, axis = 1)
print (sampleArray)
```

```
Printing Original array
[[34 43 73]
 [82 22 12]
 [53 94 66]]
Array after deleting column 2 on axis 1
[[34 73]
 [82 12]
 [53 66]]
Array after inserting column 2 on axis 1
[[34 10 73]
 [82 10 12]
 [53 10 66]]
```

23. Find the positions of:

- elements in x where its value is more than its corresponding element in y, and

- elements in x where its value is equals to its corresponding element in y.

```
import numpy as np
x = np.array([1,2,3,2,3,4,9,4,9,8])
y = np.array([7,2,10,2,7,4,3,4,5,6])
print("elements in x where its value is more than its corresponding element in y,",np.where(x == y))
```

```
elements in x where its value is more than its corresponding element in y, (array([1, 3, 5, 7]),)
```

24. Write a program to multiply two matrices of size (100,100) using for loops and also with NumPy methods. Compare the time of execution of both cases.

```
import time
from timeit import timeit
import numpy as np
a=time.time()
mat1 = np.random.random((100,100))
mat2 = np.random.random((100,100))
final = np.dot(mat1,mat2)
print(final)
time.sleep(1)
end = time.time()
print(end-a)
import time
from timeit import timeit
import numpy as np
a=time.time()
mat1 = np.random.random((100,100))
mat2 = np.random.random((100,100))
def mult(mat1,mat2):
  c=np.zeros((mat1.shape[0],mat2.shape[1]))
  for i in range(mat1.shape[0]):
    for k in range(mat2.shape[1]):
      c[i,k]=0
      for j in range(mat2.shape[0]):
        n=mat1[i,j]*mat2[j,k]
        c[i,k]+=n
        return c
time.sleep(1)
b=time.time()
print(mult(mat1,mat2))
print("time is: ",b-a)
```

```
[[26.771 25.259 26.296 ... 25.473 25.918 26.381]
 [29.598 25.708 28.152 ... 27.078 28.135 27.642]
 [29.351 25.212 26.265 ... 27.017 24.872 26.634]
 ...
 [30.753 26.64  29.517 ... 27.367 29.937 28.292]
 [25.492 23.141 23.668 ... 24.364 22.66  23.351]
 [30.284 23.847 28.    ... 25.105 27.171 27.355]]
1.0077028274536133
[[0.275 0.    0.    ... 0.    0.    0.   ]
 [0.    0.    0.    ... 0.    0.    0.   ]
 [0.    0.    0.    ... 0.    0.    0.   ]
 ...
 [0.    0.    0.    ... 0.    0.    0.   ]
 [0.    0.    0.    ... 0.    0.    0.   ]
 [0.    0.    0.    ... 0.    0.    0.   ]]
time is:  1.0017893314361572
```

25. Write a program to execute the steps below using NumPy:

$$zij = \Sigma wikxkjnk = 1 \; and \; \sigma ij(zij) = 11 + e - zij$$

Where, $w \; and \; x$ are the matrices of random numbers having dimensions (m,n) and (n,k), respectively. $\sigma(z)$ is a function which performs above defined operation on elements of z.

```
import numpy as np
import random as random
m = random.randint(1,100)
n = random.randint(1,100)
k = random.randint(1,100)
mat1 = np.random.random((m, n))
mat2 = np.random.random((n, k))
def oper(x):
  return 1 / (1 + np.exp(-x))
w = np.array(mat1, dtype=float)
x = np.array(mat2, dtype=float)
z = np.dot(w, x)
print(oper(z))
```

```
[[1.    1.    0.999 ... 1.    1.    0.999]
 [1.    0.999 0.998 ... 0.999 0.998 0.997]
 [1.    1.    0.999 ... 1.    0.999 0.998]
 ...
 [1.    1.    1.    ... 1.    1.    1.   ]
 [0.999 0.999 0.998 ... 1.    0.998 0.997]
 [1.    1.    0.998 ... 1.    0.999 0.999]]
```

26. Create two vectors $\boldsymbol{y}$ ***and*** $\hat{\boldsymbol{y}}$ having same dimensions, where $\hat{y}$ should consist of random numbers between [0,1] and y should contain $\mathbf{0}\boldsymbol{s}$ ***and*** $\mathbf{1}\boldsymbol{s}$, for example $\boldsymbol{y}=[\mathbf{0},\mathbf{1},\mathbf{1},\mathbf{0},\mathbf{10},\mathbf{0},\mathbf{1},\ldots,\mathbf{1}]$. Compute the given expression: $\boldsymbol{O} = -\frac{1}{n}\sum_{i=1}^{n}[y_i\log_2(\hat{y_i})+(1-y_i)\log_2(1-\hat{y_i})]$

where n is the total number of elements in y *and* $\hat{y}$.

```
import random as random
import math
y=[]
yy=[]
n=100
for i in range(n):
  y.append(random.randint(0,1))
  yy.append(random.random())
O=[]
for i in range(n):
  a=-(1/n)*((y[i]*math.log2(yy[i]))+((1-y[i])*math.log2(1-yy[i])))
  O.append(a)
  print(y)
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
```
