Seatwork#2

April 18, 2024

Author: Willie M. Bonavente

[2]:
```
import numpy as np
```

Program 1: Creating array object

[4]:
```
arr = np.array( [[ 1, 2, 3],
                 [ 4, 2, 5]] )
# Printing type of arr object
print("Array is of type: ", type(arr))
# Printing array dimensions (axes)
print("No. of dimensions: ", arr.ndim)
# Printing shape of array
print("Shape of array: ", arr.shape)
# Printing size (total number of elements) of array
print("Size of array: ", arr.size)
# Printing type of elements in array
print("Array stores elements of type: ", arr.dtype)
```

```
Array is of type:  <class 'numpy.ndarray'>
No. of dimensions:  2
Shape of array:  (2, 3)
Size of array:  6
Array stores elements of type:  int32
```

Program 2: Array creation techniques

[5]:
```
import numpy as np

# Creating array from list with type float
a = np.array([[1, 2, 4], [5, 8, 7]], dtype = 'float')
print("Array created using passed list:\n", a)

# Creating array from tuple
b = np.array((1 , 3, 2))
print("\nArray created using passed tuple:\n", b)

# Creating a 3X4 array with all zeros
c = np.zeros((3, 4))
```

```
print("\nAn array initialized with all zeros:\n", c)

# Create a constant value array of complex type
d = np.full((3, 3), 6, dtype = 'complex')
print("\nAn array initialized with all 6s. Array type is complex:\n", d)

# Create an array with random values
e = np.random.random((2, 2))
print("\nA random array:\n", e)

# Create a sequence of integers from 0 to 30 with steps of 5
f = np.arange(0, 30, 5)
print("\nA sequential array with steps of 5:\n", f)

# Create a sequence of 10 values in range 0 to 5
g = np.linspace(0, 5, 10)
print("\nA sequential array with 10 values between 0 and 5:\n", g)

# Reshaping 3X4 array to 2X2X3 array
arr = np.array([[1, 2, 3, 4], [5, 2, 4, 2], [1, 2, 0, 1]])
newarr = arr.reshape(2, 2, 3)
print("\nOriginal array:\n", arr)
print("Reshaped array:\n", newarr)

# Flatten array
arr = np.array([[1, 2, 3], [4, 5, 6]])
flarr = arr.flatten()
print("\nOriginal array:\n", arr)
print("Flattened array:\n", flarr)
```

```
Array created using passed list:
 [[1. 2. 4.]
 [5. 8. 7.]]

Array created using passed tuple:
 [1 3 2]

An array initialized with all zeros:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

An array initialized with all 6s. Array type is complex:
 [[6.+0.j 6.+0.j 6.+0.j]
 [6.+0.j 6.+0.j 6.+0.j]
 [6.+0.j 6.+0.j 6.+0.j]]

A random array:
```

```
 [[0.3849498  0.8791349 ]
 [0.17037919 0.48800669]]

A sequential array with steps of 5:
 [ 0  5 10 15 20 25]

A sequential array with 10 values between 0 and 5:
 [0.         0.55555556 1.11111111 1.66666667 2.22222222 2.77777778
 3.33333333 3.88888889 4.44444444 5.        ]

Original array:
 [[1 2 3 4]
 [5 2 4 2]
 [1 2 0 1]]
Reshaped array:
 [[[1 2 3]
  [4 5 2]]

 [[4 2 1]
  [2 0 1]]]

Original array:
 [[1 2 3]
 [4 5 6]]
Flattened array:
 [1 2 3 4 5 6]
```

Program 3: Array Indexing

[6]:
```python
import numpy as np

# An exemplar array
arr = np.array([[-1, 2, 0, 4],
                [4, -0.5, 6, 0],
                [2.6, 0, 7, 8],
                [3, -7, 4, 2.0]])

# Slicing array
temp = arr[:2, ::2]
print("Array with first 2 rows and alternate columns(0 and 2):\n", temp)

# Integer array indexing example
temp = arr[[0, 1, 2, 3], [3, 2, 1, 0]]
print("\nElements at indices (0, 3), (1, 2), (2, 1), (3, 0):\n", temp)

# boolean array indexing example
cond = arr > 0  # cond is a boolean array
temp = arr[cond]
```

```
print("\nElements greater than 0:\n", temp)
```

```
Array with first 2 rows and alternate columns(0 and 2):
 [[-1.  0.]
 [ 4.  6.]]

Elements at indices (0, 3), (1, 2), (2, 1), (3, 0):
 [4. 6. 0. 3.]

Elements greater than 0:
 [2.  4.  4.  6.  2.6 7.  8.  3.  4.  2. ]
```

Program 4: Basic Operations

[8]:
```
# basic operations on single array
a = np.array([1, 2, 5, 3])

# add 1 to every element
print("Adding 1 to every element:", a + 1)

# subtract 3 from each element
print("Subtracting 3 from each element:", a - 3)

# multiply each element by 10
print("Multiplying each element by 10:", a * 10)

# square each element
print("Squaring each element:", a ** 2)

# modify existing array
a *= 2
print("Doubled each element of original array:", a)

# transpose of array
a = np.array([[1, 2, 3], [3, 4, 5], [9, 6, 0]])

print("\nOriginal array:\n", a)
print("Transpose of array:\n", a.T)
```

```
Adding 1 to every element: [2 3 6 4]
Subtracting 3 from each element: [-2 -1  2  0]
Multiplying each element by 10: [10 20 50 30]
Squaring each element: [ 1  4 25  9]
Doubled each element of original array: [ 2  4 10  6]

Original array:
 [[1 2 3]
 [3 4 5]
 [9 6 0]]
```

```
Transpose of array:
 [[1 3 9]
 [2 4 6]
 [3 5 0]]
```

[10]:
```
# unary operators in numpy
arr = np.array([[1, 5, 6],
                [4, 7, 2],
                [3, 1, 9]])

# maximum element of array
print("Largest element is:", arr.max())
print("Row-wise maximum elements:", arr.max(axis = 1))

# minimum element of array
print("Column-wise minimum elements:", arr.min(axis = 0))

# sum of array elements
print("Sum of all array elements:", arr.sum())

# cumulative sum along each row
print("Cumulative sum along each row:\n", arr.cumsum(axis = 1))
```

```
Largest element is: 9
Row-wise maximum elements: [6 7 9]
Column-wise minimum elements: [1 1 2]
Sum of all array elements: 38
Cumulative sum along each row:
 [[ 1  6 12]
 [ 4 11 13]
 [ 3  4 13]]
```

[11]:
```
# binary operators in Numpy
a = np.array([[1, 2],
              [3, 4]])
b = np.array([[4, 3],
              [2, 1]])

# add arrays
print("Array sum:\n", a + b)

# multiply arrays (elementwise multiplication)
print("Array multiplication:\n", a * b)

# matrix multiplication
print("Matrix multiplication:\n", a.dot(b))
```

```
Array sum:
 [[5 5]
```

```
 [5 5]]
Array multiplication:
 [[4 6]
 [6 4]]
Matrix multiplication:
 [[ 8  5]
 [20 13]]
```

[12]:
```
# universal functions in numpy

# create an array of sine values
a = np.array([0, np.pi/2, np.pi])
print("Sine values of array elements:", np.sin(a))

# exponential values
a = np.array([0, 1, 2, 3])
print("Exponent of array elements:", np.exp(a))

# square root of array values
print("Square root of array elements:", np.sqrt(a))
```

```
Sine values of array elements: [0.0000000e+00 1.0000000e+00 1.2246468e-16]
Exponent of array elements: [ 1.          2.71828183  7.3890561  20.08553692]
Square root of array elements: [0.          1.          1.41421356 1.73205081]
```

Program 5: Sorting an Array

[16]:
```
a = np.array([[1, 4, 2],
              [3, 4, 6],
              [0, -1, 5]])

# sorted array
print("Array elements in sorted order:\n", np.sort(a, axis=None))

# sort array row-wise
print("Row-wise sorted array:\n", np.sort(a, axis=1))

# specify sort algorithm
print("Column wise sort by applying merge-sort:\n", np.sort(a, axis=0,␣
 ↪kind='mergesort'))

# Example to show sorting of structured array
# set alias names for dtypes
dtypes = [('name', 'S10'), ('grad_year', int), ('cgpa', float)]

# Values to be put in array
values = [('Hrithik', 2022, 8.5), ('Ajay', 2020, 8.7),
          ('Pankaj', 2013, 7.9), ('Aakash', 2019, 9.0)]
```

```
# Creating array
arr = np.array(values, dtype=dtypes)
print("\nArray sorted by names:\n", np.sort(arr, order='name'))

print("Array sorted by graduation year and then cgpa:\n", np.sort(arr,␣
 ↪order=['grad_year', 'cgpa']))
```

```
Array elements in sorted order:
 [-1  0  1  2  3  4  4  5  6]
Row-wise sorted array:
 [[ 1  2  4]
 [ 3  4  6]
 [-1  0  5]]
Column wise sort by applying merge-sort:
 [[ 0 -1  2]
 [ 1  4  5]
 [ 3  4  6]]

Array sorted by names:
 [(b'Aakash', 2019, 9. ) (b'Ajay', 2020, 8.7) (b'Hrithik', 2022, 8.5)
 (b'Pankaj', 2013, 7.9)]
Array sorted by graduation year and then cgpa:
 [(b'Pankaj', 2013, 7.9) (b'Aakash', 2019, 9. ) (b'Ajay', 2020, 8.7)
 (b'Hrithik', 2022, 8.5)]
```

[14]:
```
# import matplotlib.pyplot as plt
```

[ ]:
```
# # Extract names and cgpa from the sorted array
# names = [name.decode('utf-8') for name in np.sort(arr, order='name')['name']]
# cgpa = [cg for cg in np.sort(arr, order='name')['cgpa']]

# # Create bar graph
# plt.bar(names, cgpa)

# # Add labels and title
# plt.xlabel('Names')
# plt.ylabel('CGPA')
# plt.title('CGPA of Students')

# # Display the graph
# plt.show()
```