NumPy (Numerical Python) is a fundamental package in the Python scientific computing ecosystem. It provides support for working with arrays and matrices of data, along with a wide range of mathematical functions to operate on these arrays.

Array Initialization:

In [15]:

```
import numpy as np
zero_array = np.zeros((2, 4))  # Creates a 2x3 array filled with zeros
print(zero_array,'\n')

ones_array = np.ones((2, 3))  # Creates a 2x3 array filled with ones
print(ones_array,'\n')

identity_matrix = np.eye(3)  # Creates a 3x3 identity matrix
print(identity_matrix,'\n')

one_to_ten = np.arange(1, 11)  # Creates an array from 1 to 10
print(one_to_ten,'\n')

random_array = np.random.random((2,2 ))
print(random_array)
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]] 

[[1. 1. 1.]
 [1. 1. 1.]] 

[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]] 

[ 1  2  3  4  5  6  7  8  9 10] 

[[0.41949208 0.17810166]
 [0.96046581 0.60224541]]
```

Array Operations:

In [19]:
```
dot = np.dot(one_to_ten, one_to_ten)
summ = np.sum(one_to_ten)
sqrt_array = np.sqrt(one_to_ten)
print(dot,'\n')
print(summ,'\n')
print(sqrt_array)
```

```
385

55

[1.         1.41421356 1.73205081 2.         2.23606798 2.44948974
 2.64575131 2.82842712 3.         3.16227766]
```

Array Manipulation:

In [41]:
```
arr=np.array([1,2,3,4])
arr1=np.array([1,2,3,4])
arr2=np.array([1,2,3])
arr3=np.array([4,5,6])
```

In [59]:
```
reshaped_array = arr.reshape((2, 2))  # Reshapes a 1D array into a 2x3 arra
print(reshaped_array,'\n')

flattened_array = sqrt_array.flatten()  # Converts a 2D array into a 1D arr
print(flattened_array,'\n')

stacked_array = np.vstack((arr2, arr3))  # Stacks arrays vertically
print(stacked_array,'\n')

stacked_array2 = np.hstack((arr2, arr3))  # Stacks arrays vertically
print(stacked_array2,'\n')
```

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

[1.         1.41421356 1.73205081 2.         2.23606798 2.44948974
 2.64575131 2.82842712 3.         3.16227766]

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

[1 2 3 4 5 6]
```

Copy and View

In [53]:
```
arr5=np.array([6,9,6,9])
print(arr5.copy())
# The copy SHOULD NOT be affected by the changes made to the original array
print(arr5.view())
#The view SHOULD be affected by the changes made to the original array.
```

```
[6 9 6 9]
[6 9 6 9]
```

Join, Split, Search and Filter

In [68]:
```
print(np.concatenate((arr2,arr3)),'\n')   #Join

subarrays = np.split(arr, 2)  # Splits a 1D array into two equal parts
print(subarrays,'\n')

print(np.where(arr3==6),'\n')    #Search

arr = np.array([41, 42, 43, 44])

x = [True, False, True, False]

newarr = arr[x]     #Filter

print(newarr)
```

```
[1 2 3 4 5 6]

[array([41, 42]), array([43, 44])]

(array([2], dtype=int64),)

[41 43]
```

Random Module

In [76]:
```
from numpy import random

x=random.randint(100, size=(2,2))  #Random 2D array
print(x,'\n')

y = random.choice([3, 5, 7, 9],size=(2,2))

print(y)
```

```
[[74 16]
 [88  3]]

[[7 9]
 [3 3]]
```

In [4]:
```
np.array([[6, 15.7], [True, False]])
```

Out[4]:
```
array([[ 6. , 15.7],
       [ 1. ,  0. ]])
```

In [5]:
```
np.array([45.67, True], dtype=np.int8)
```

Out[5]:
```
array([45,  1], dtype=int8)
```

In [6]:
```
np.array([34.62, 70.13, 9]).astype(np.int64)
```

Out[6]:
```
array([34, 70,  9], dtype=int64)
```

In [10]:
```
import numpy as np

# Example 1: Finding Indices of Elements Meeting a Condition
arr = np.array([2, 8, 5, 12, 6, 9])
condition = arr > 6
indices = np.where(condition)
print(indices)
print()
# Example 2: Selecting Elements Based on Condition
values_above_six = arr[np.where(arr > 6)]
print(values_above_six)

print()
# Example 3: Replacing Elements Based on Condition
arr[np.where(arr > 6)] = 0
print(arr)
```

```
(array([1, 3, 5], dtype=int64),)

[ 8 12  9]

[2 0 5 0 6 0]
```

keepdims=True: This parameter ensures that the result maintains the same number of dimensions as the original array. By setting keepdims=True, the result will have the same number of rows as the original array but only one column.

In [13]:

```
#axis=0 means rows
#axis=1 means cols

# Example 1: Concatenating Arrays Horizontally (axis=1)
array1 = np.array([[1, 2], [3, 4]])
array2 = np.array([[5, 6]])
result = np.concatenate((array1, array2), axis=0)
print(result)
```

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

In [15]:

```
# Create a 2-dimensional array (3x4)
original_array = np.array([[1, 2, 3, 4],
                           [5, 6, 7, 8],
                           [9, 10, 11, 12]])

# Reshape the array into a 2-dimensional array with 2 rows and 6 columns
reshaped_array = original_array.reshape(2, 6)

print("Original Array:")
print(original_array)
print("Reshaped Array:")
print(reshaped_array)
```

```
Original Array:
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
Reshaped Array:
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]]
```

In [ ]:

```
# Load the mystery_image.npy file
with open("mystery_image.npy", "rb") as f:
    rgb_array = np.load(f)

plt.imshow(rgb_array)
plt.show()
```

In [16]:

```
# Flip rgb_array so that it is the mirror image of the original
mirrored_monet = np.flip(rgb_array, axis=1)
plt.imshow(mirrored_monet)
plt.show()

# Flip rgb_array so that it is upside down
upside_down_monet = np.flip(rgb_array, axis=(0, 1))
plt.imshow(upside_down_monet)
plt.show()

# Transpose rgb_array
transposed_rgb = np.transpose(rgb_array, axes=(1, 0, 2))
plt.imshow(transposed_rgb)
plt.show()
```

```
---------------------------------------------------------------------
--
NameError                                 Traceback (most recent call las
t)
Cell In [16], line 2
      1 # Flip rgb_array so that it is the mirror image of the original
----> 2 mirrored_monet = np.flip(rgb_array, axis=1)
      3 plt.imshow(mirrored_monet)
      4 plt.show()

NameError: name 'rgb_array' is not defined
```