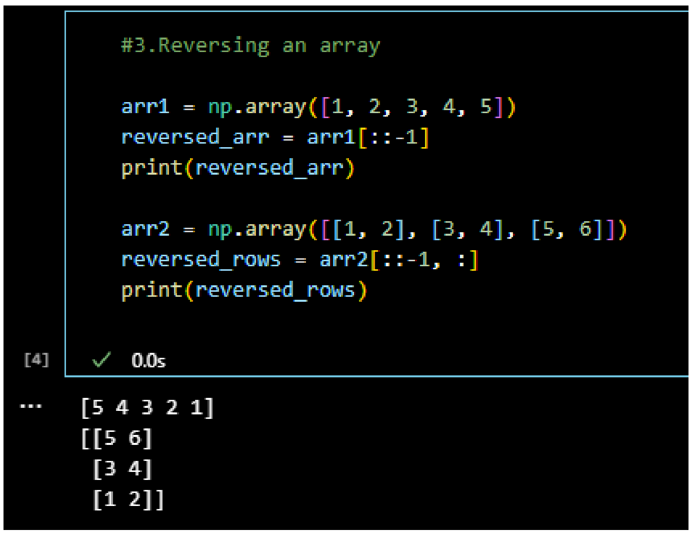
PYTHON ASSIGNMENT 5
NUMPY

GITHUB: Bha411/Github

1. The purpose and advantages of NumPy lie in the fact that it can be used to perform mathematical functions on a array which follows homogeneity and the efficient memory handling done for the same.
   It can handle very large arrays very easily by consuming even less time.
   Thus it can enhance the python capabilities and make it more versatile library.

2. -np.mean(): Computes the arithmetic mean (simple average) of an array.
   -np.average(): Calculates the average, but it allows you to provide weights for each value, so some values can contribute more to the average.
   We use np.mean() for basic averaging, and np.average() when you need a weighted average, i.e., when some values are more important than others.

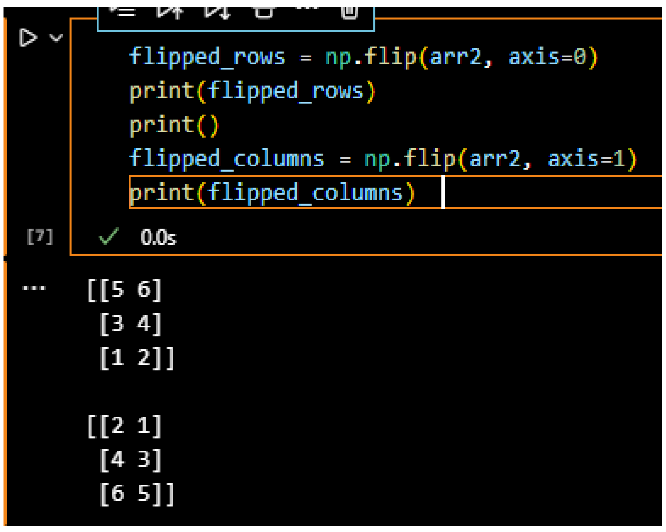
3. Arrays can be reversed using slicing, built-in functions and flipping it vertically too.



```
#3.Reversing an array

arr1 = np.array([1, 2, 3, 4, 5])
reversed_arr = arr1[::-1]
print(reversed_arr)

arr2 = np.array([[1, 2], [3, 4], [5, 6]])
reversed_rows = arr2[::-1, :]
print(reversed_rows)
```

```
[4]  ✓ 0.0s
```

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

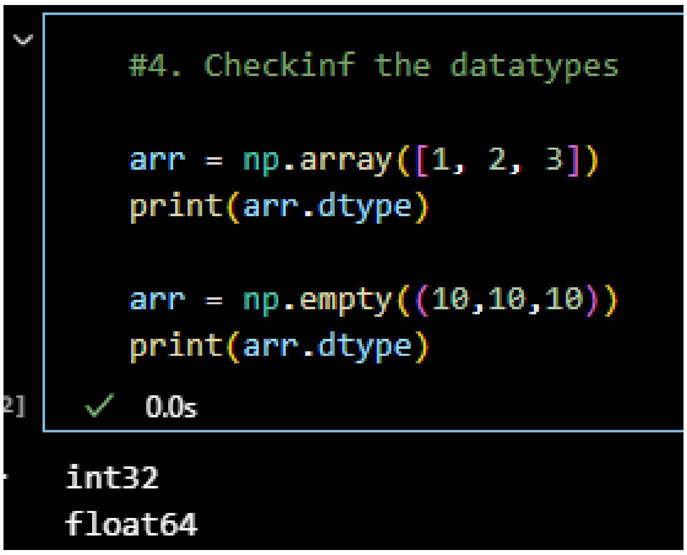


```
flipped_rows = np.flip(arr2, axis=0)
print(flipped_rows)
print()
flipped_columns = np.flip(arr2, axis=1)
print(flipped_columns)
```

```
[7]  ✓ 0.0s
```

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

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

4. We can use the built in attribute called “dtype”, which returns the subsequent datatype of the variable. Choosing the correct dtype is important as it can save a lot of memory and improve the performance. For example int32 type takes less space and is used for smaller arrays while the int64 takes more space.



```
#4. Checkinf the datatypes

arr = np.array([1, 2, 3])
print(arr.dtype)

arr = np.empty((10,10,10))
print(arr.dtype)
✓ 0.0s

int32
float64
```

5. Ndarrays in NumPy are the n dimensional arrays that store the elements of same datatype, meaning they are homogenous and are highly efficient for mathematical operations.

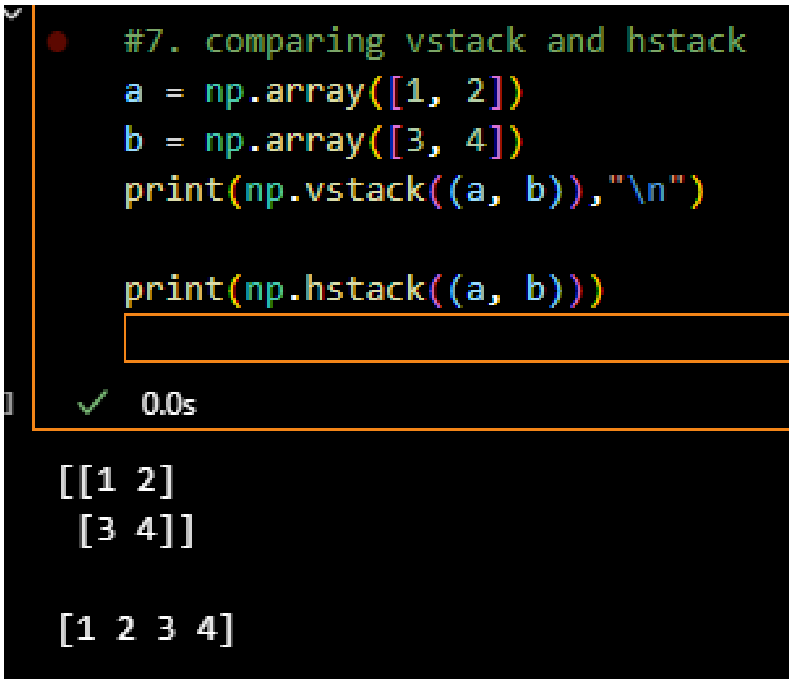
   Key features include:

   - Fixed Size.
   - Supports Multi-Dim arrays.
   - Uses less memory

   They differ from the traditional lists as they are very optimal and use lesser memory, apply the Operations faster and are homogenous in nature. Also the operations of numpy happen to the whole array at once, as compared to the element by element followed by the traditional lists in python.

6. The performance benefits range from fast calculations to using lesser amounts of memory when it comes to large-scaled operations. Particularly because the operations in numpy are not element by element but rather on an array as whole. The vectorized Operations eliminate the need for loops leading to a more concise and a smaller code base.

7. In vstack() , the arrays are stacked vertically along the rows. A new row is always added at the bottom of the existing array.
   While hstack() is used for horizontal stacking of arrays. The new columns are added to theside of an existing array.

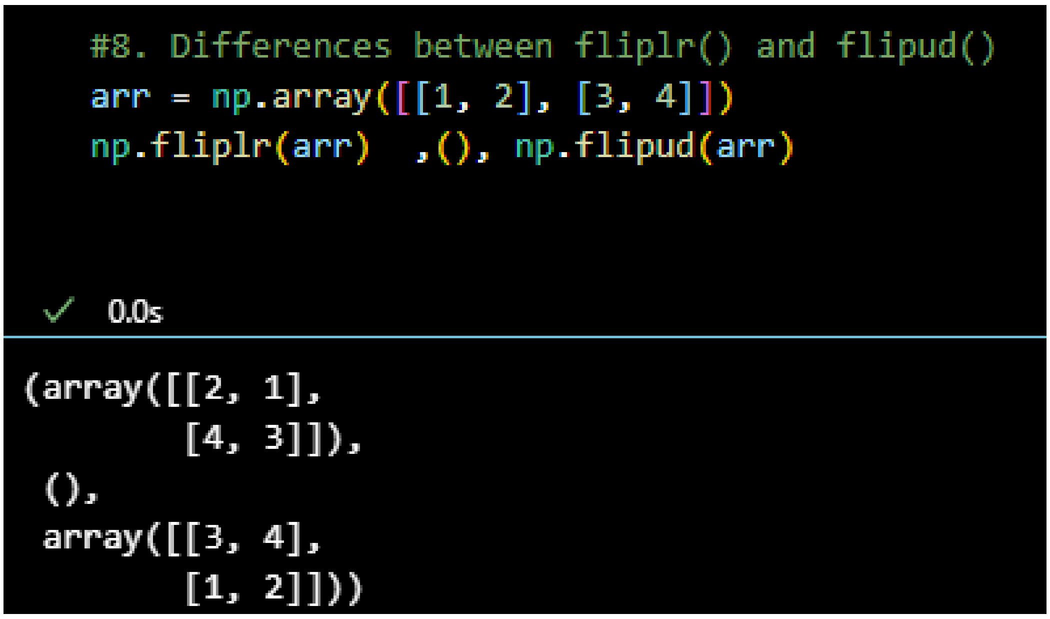


```
#7. comparing vstack and hstack
a = np.array([1, 2])
b = np.array([3, 4])
print(np.vstack((a, b)),"\n")

print(np.hstack((a, b)))
```

0.0s

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

[1 2 3 4]
```

8. Fliplr() flips an array horizontally (left to right) and this reverses the order od columns, while flipud() flips it vertically that is upside down. The order os rows are reversed this way.

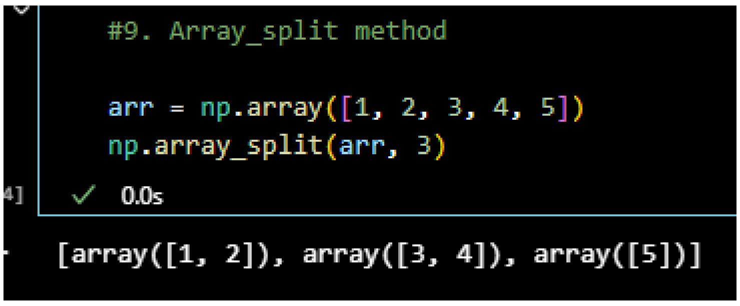


```
#8. Differences between fliplr() and flipud()
arr = np.array([[1, 2], [3, 4]])
np.fliplr(arr)  ,(), np.flipud(arr)
```

0.0s

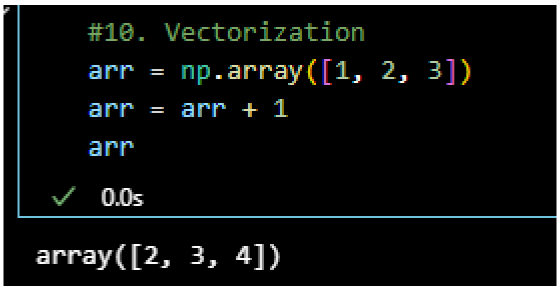
```
(array([[2, 1],
        [4, 3]]),
 (),
 array([[3, 4],
        [1, 2]]))
```

9. The array_split() method splits an array into multiple sub-arrays. If the arraycant be split equally, then it will be an uneven split.
   In these uneven splits, it ensures that each-subarray has close to equal size as possible, but allows for some sub-arrays to be smaller when necessary.



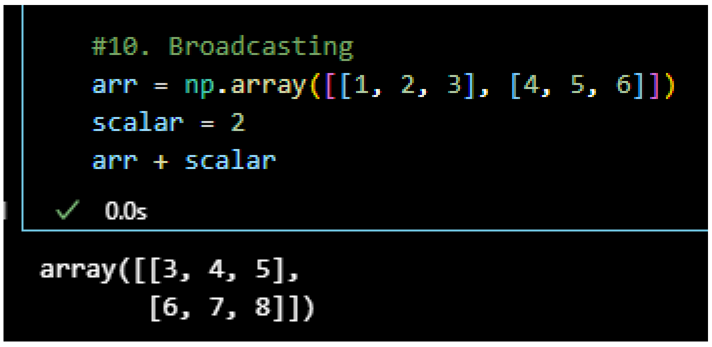
```
#9. Array_split method

arr = np.array([1, 2, 3, 4, 5])
np.array_split(arr, 3)
✓ 0.0s

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

10. **Vectorization**:
    This means applying an operation to the entire array rather than processing elements one at a time. It eliminates the need for loops, making the code faster.



```
#10. Vectorization
arr = np.array([1, 2, 3])
arr = arr + 1
arr
✓ 0.0s

array([2, 3, 4])
```

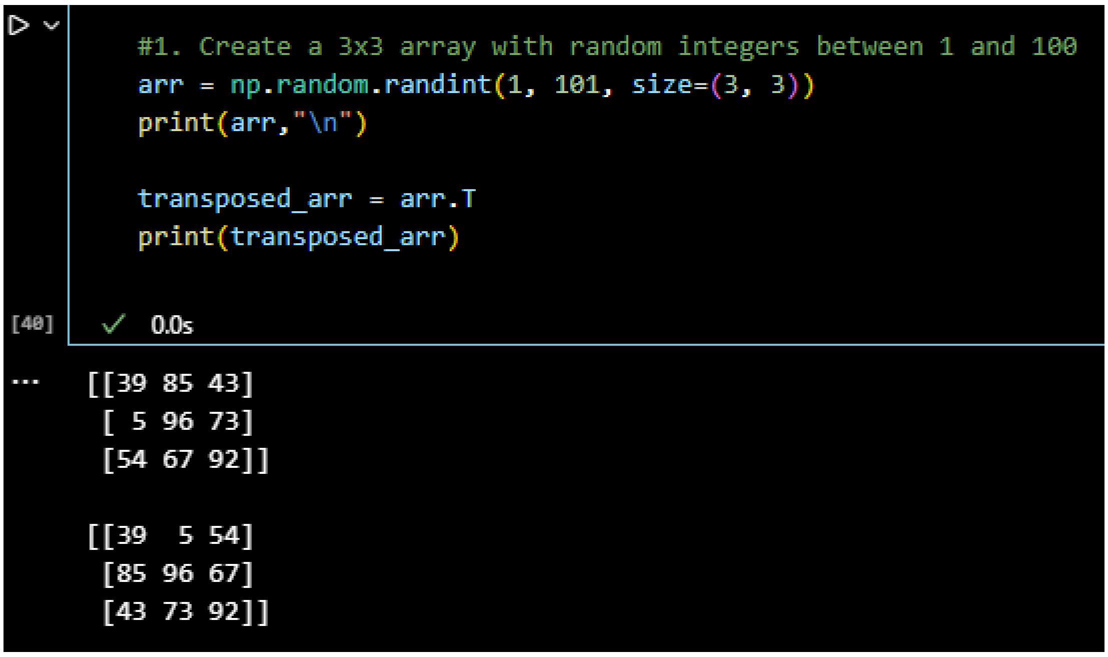
Broadcasting:
This is the process of making arrays of different shapes compatible for element-wise operations. NumPy automatically stretches smaller arrays to match the size of larger ones.



```
#10. Broadcasting
arr = np.array([[1, 2, 3], [4, 5, 6]])
scalar = 2
arr + scalar
✓ 0.0s

array([[3, 4, 5],
       [6, 7, 8]])
```

Practical Questions:

1.

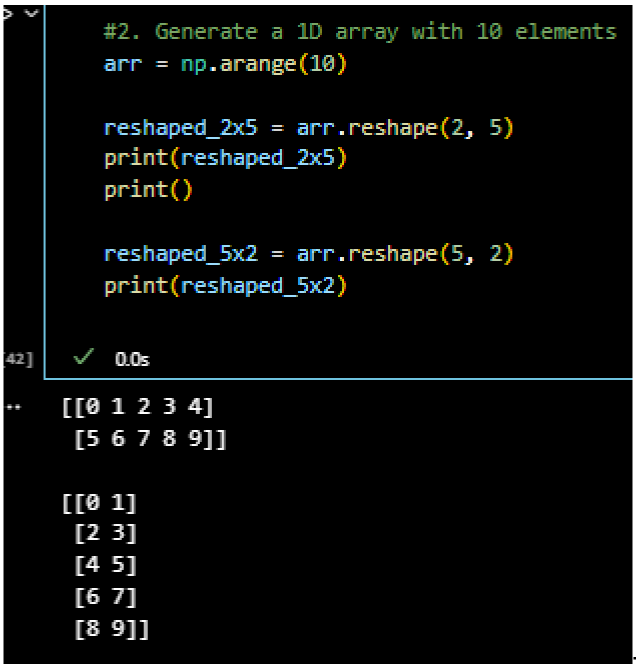


```
#1. Create a 3x3 array with random integers between 1 and 100
arr = np.random.randint(1, 101, size=(3, 3))
print(arr,"\n")

transposed_arr = arr.T
print(transposed_arr)
```

[40] ✓ 0.0s

```
[[39 85 43]
 [ 5 96 73]
 [54 67 92]]

[[39  5 54]
 [85 96 67]
 [43 73 92]]
```

2.

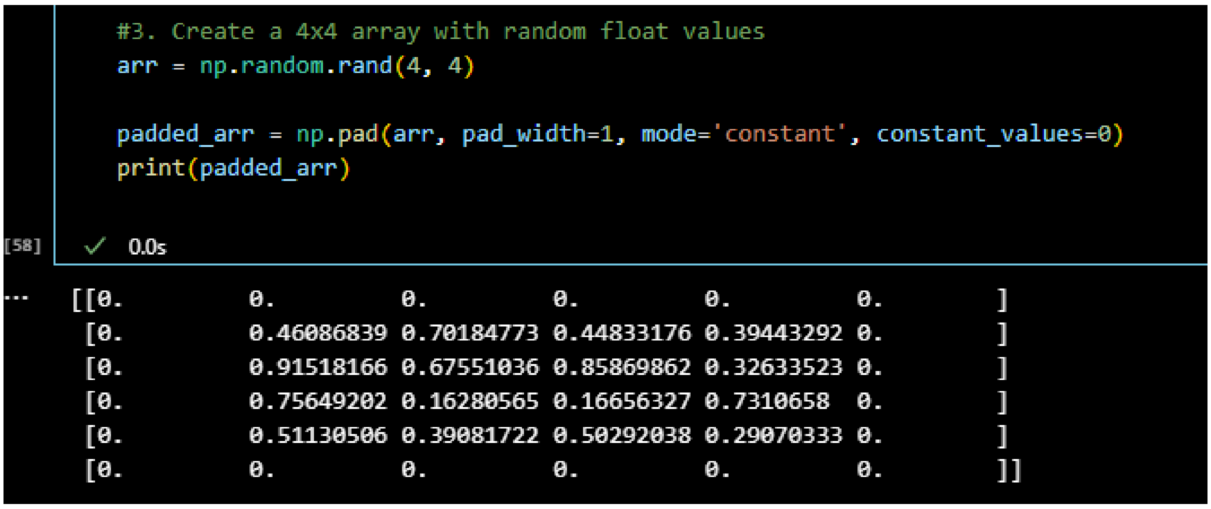


```
#2. Generate a 1D array with 10 elements
arr = np.arange(10)

reshaped_2x5 = arr.reshape(2, 5)
print(reshaped_2x5)
print()

reshaped_5x2 = arr.reshape(5, 2)
print(reshaped_5x2)
```

42] ✓ 0.0s

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

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

3.

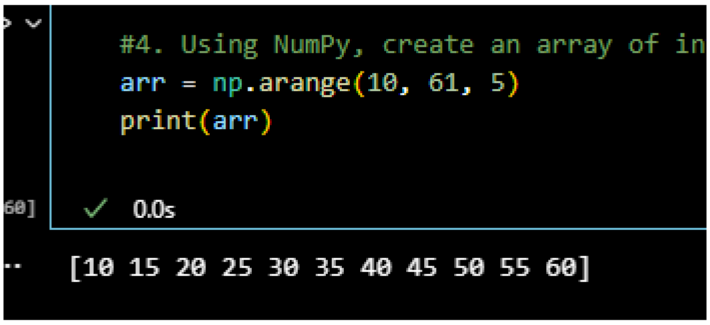


```
#3. Create a 4x4 array with random float values
arr = np.random.rand(4, 4)

padded_arr = np.pad(arr, pad_width=1, mode='constant', constant_values=0)
print(padded_arr)
```

[58] ✓ 0.0s

```
[[0.         0.         0.         0.         0.         0.        ]
 [0.         0.46086839 0.70184773 0.44833176 0.39443292 0.        ]
 [0.         0.91518166 0.67551036 0.85869862 0.32633523 0.        ]
 [0.         0.75649202 0.16280565 0.16656327 0.7310658  0.        ]
 [0.         0.51130506 0.39081722 0.50292038 0.29070333 0.        ]
 [0.         0.         0.         0.         0.         0.        ]]
```

4.

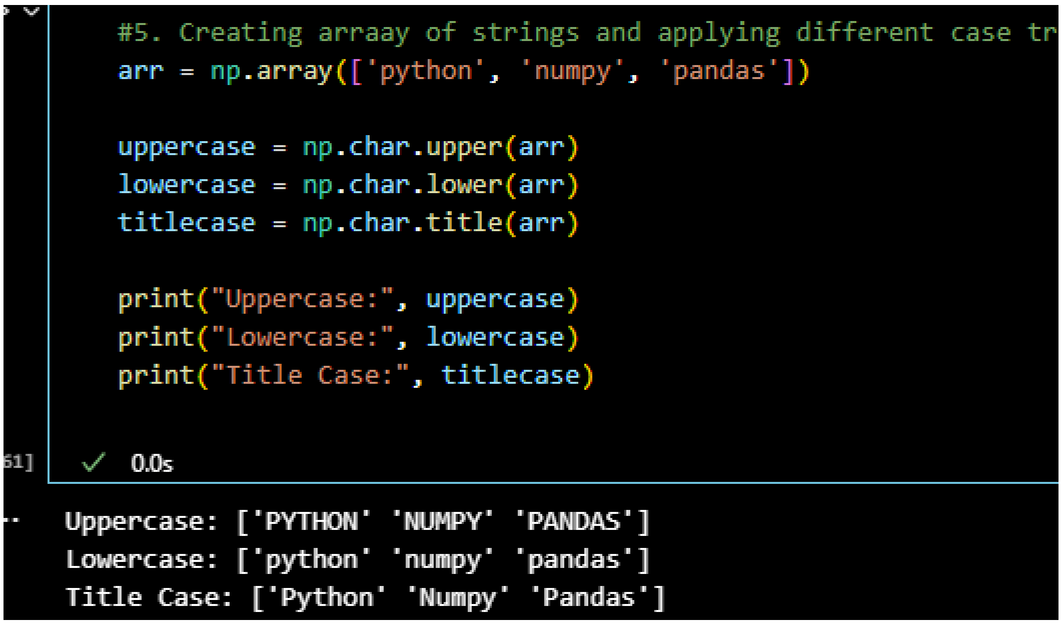


```
#4. Using NumPy, create an array of in
arr = np.arange(10, 61, 5)
print(arr)
```

60] ✓ 0.0s

```
[10 15 20 25 30 35 40 45 50 55 60]
```

5.

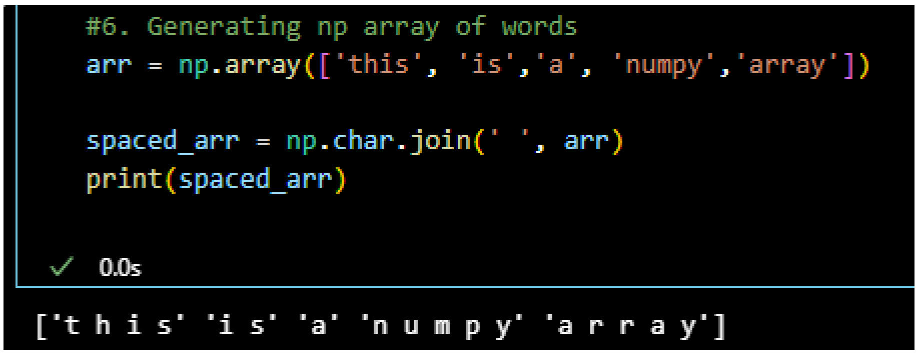


```
#5. Creating arraay of strings and applying different case tr
arr = np.array(['python', 'numpy', 'pandas'])

uppercase = np.char.upper(arr)
lowercase = np.char.lower(arr)
titlecase = np.char.title(arr)

print("Uppercase:", uppercase)
print("Lowercase:", lowercase)
print("Title Case:", titlecase)
```

61] ✓ 0.0s

```
Uppercase: ['PYTHON' 'NUMPY' 'PANDAS']
Lowercase: ['python' 'numpy' 'pandas']
Title Case: ['Python' 'Numpy' 'Pandas']
```

6.

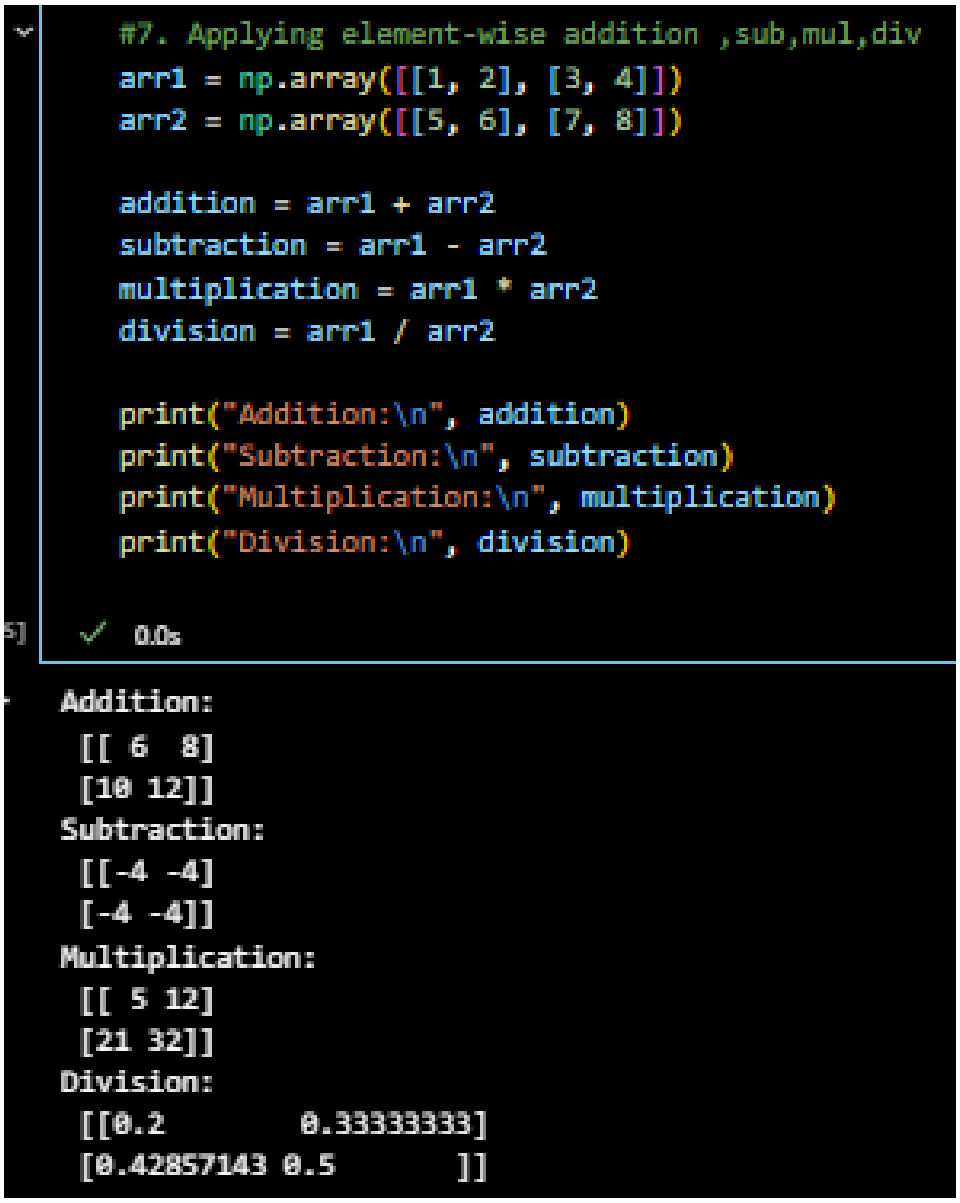


```
#6. Generating np array of words
arr = np.array(['this', 'is','a', 'numpy','array'])

spaced_arr = np.char.join(' ', arr)
print(spaced_arr)
```

✓ 0.0s

```
['t h i s' 'i s' 'a' 'n u m p y' 'a r r a y']
```

7.

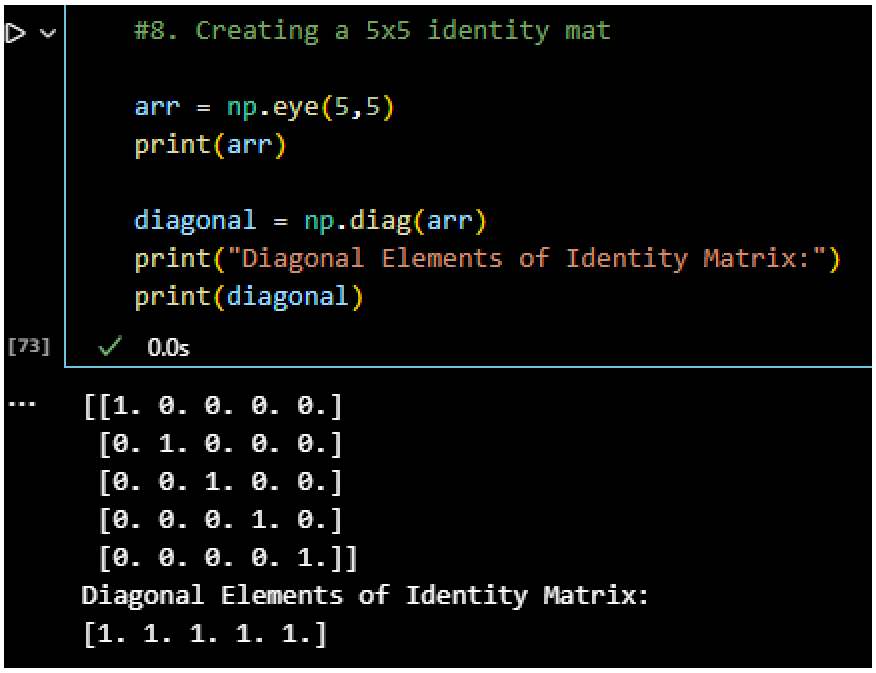


```
#7. Applying element-wise addition ,sub,mul,div
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])

addition = arr1 + arr2
subtraction = arr1 - arr2
multiplication = arr1 * arr2
division = arr1 / arr2

print("Addition:\n", addition)
print("Subtraction:\n", subtraction)
print("Multiplication:\n", multiplication)
print("Division:\n", division)
```

✓ 0.0s

```
Addition:
 [[ 6  8]
 [10 12]]
Subtraction:
 [[-4 -4]
 [-4 -4]]
Multiplication:
 [[ 5 12]
 [21 32]]
Division:
 [[0.2        0.33333333]
 [0.42857143 0.5       ]]
```

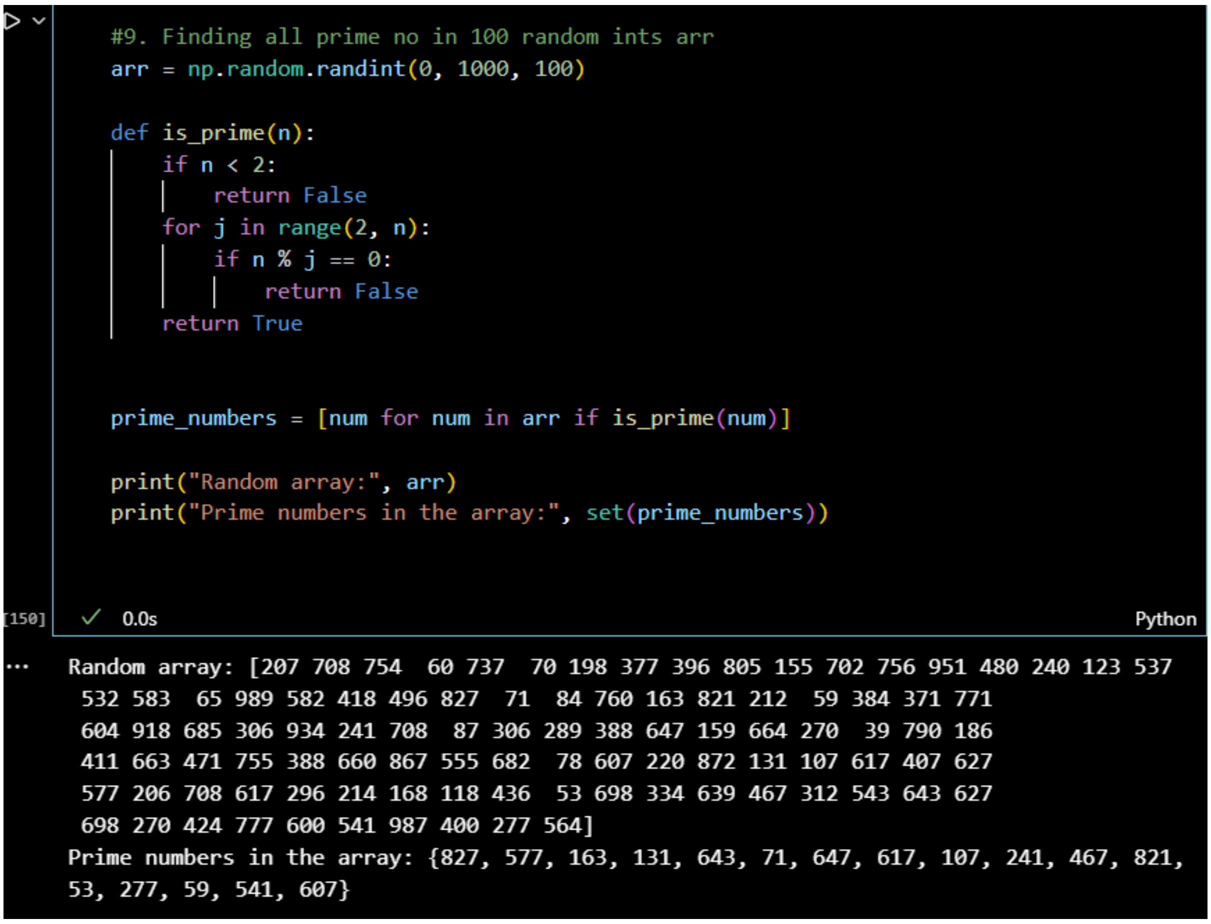
8.



```python
#8. Creating a 5x5 identity mat

arr = np.eye(5,5)
print(arr)

diagonal = np.diag(arr)
print("Diagonal Elements of Identity Matrix:")
print(diagonal)
```

```
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
Diagonal Elements of Identity Matrix:
[1. 1. 1. 1. 1.]
```

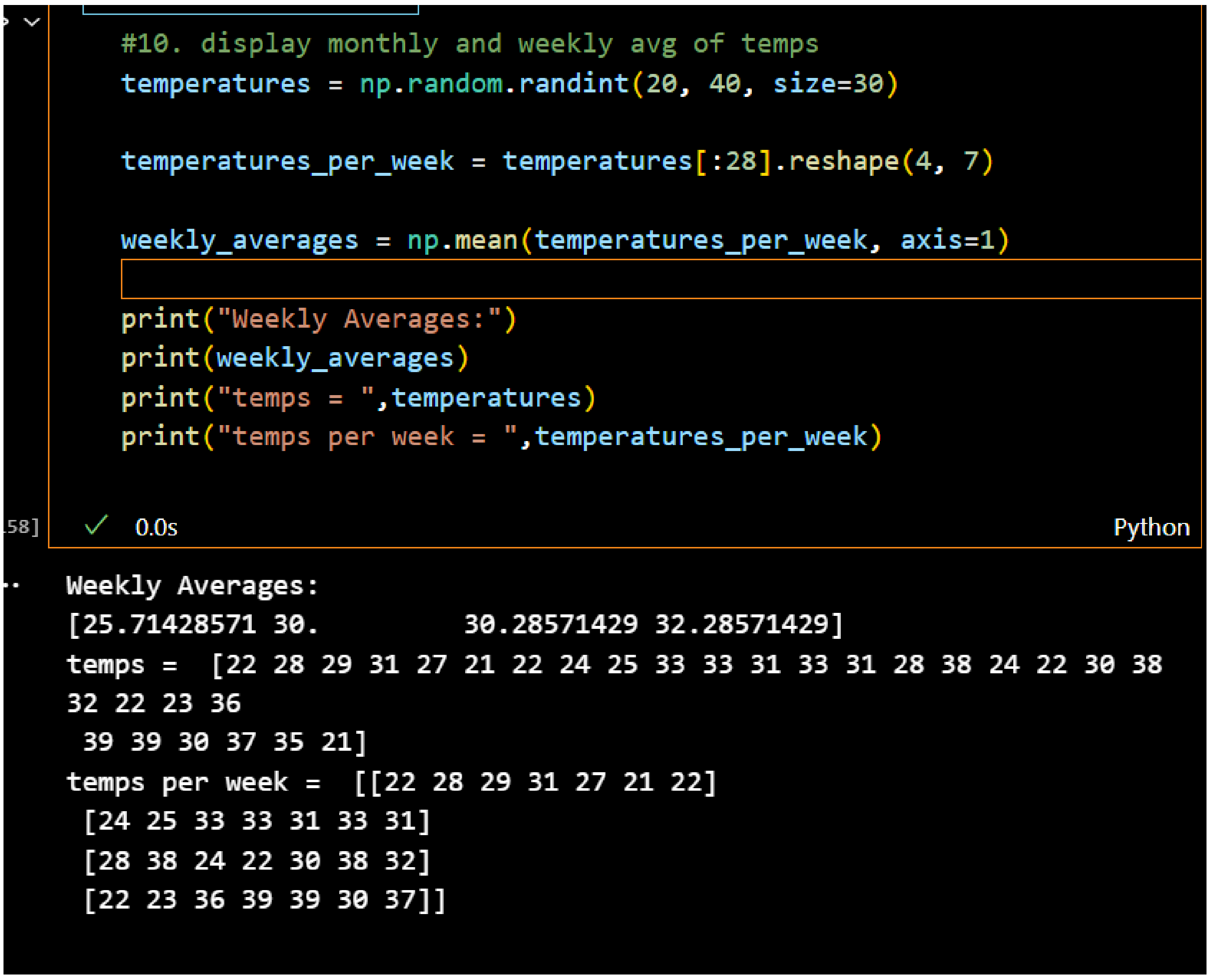
9.



```python
#9. Finding all prime no in 100 random ints arr
arr = np.random.randint(0, 1000, 100)

def is_prime(n):
    if n < 2:
        return False
    for j in range(2, n):
        if n % j == 0:
            return False
    return True


prime_numbers = [num for num in arr if is_prime(num)]

print("Random array:", arr)
print("Prime numbers in the array:", set(prime_numbers))
```

```
Random array: [207 708 754  60 737  70 198 377 396 805 155 702 756 951 480 240 123 537
 532 583  65 989 582 418 496 827  71  84 760 163 821 212  59 384 371 771
 604 918 685 306 934 241 708  87 306 289 388 647 159 664 270  39 790 186
 411 663 471 755 388 660 867 555 682  78 607 220 872 131 107 617 407 627
 577 206 708 617 296 214 168 118 436  53 698 334 639 467 312 543 643 627
 698 270 424 777 600 541 987 400 277 564]
Prime numbers in the array: {827, 577, 163, 131, 643, 71, 647, 617, 107, 241, 467, 821,
53, 277, 59, 541, 607}
```

10.



```
#10. display monthly and weekly avg of temps
temperatures = np.random.randint(20, 40, size=30)

temperatures_per_week = temperatures[:28].reshape(4, 7)

weekly_averages = np.mean(temperatures_per_week, axis=1)

print("Weekly Averages:")
print(weekly_averages)
print("temps = ",temperatures)
print("temps per week = ",temperatures_per_week)
```

```
Weekly Averages:
[25.71428571 30.         30.28571429 32.28571429]
temps =  [22 28 29 31 27 21 22 24 25 33 33 31 33 31 28 38 24 22 30 38 32 22 23 36
 39 39 30 37 35 21]
temps per week =  [[22 28 29 31 27 21 22]
 [24 25 33 33 31 33 31]
 [28 38 24 22 30 38 32]
 [22 23 36 39 39 30 37]]
```