


```
#1
import numpy as np
a = np.array([1, 2, 3])
b = np.array([[1, 2, 3], [4, 5, 6]])
print(a+b)

[[2 4 6]
 [5 7 9]]
```

2

If the arrays differ in their number of dimensions, a "1" will be repeatedly prepended to the smaller array's shape until both shapes have the same length. Arrays with a size of 1 along a particular dimension act as if they had the size of the array with the largest shape along that dimension.

```
#3
a = np.array([1,2,3])
b = 3
print(a*b)
```

 [3 6 9]

4

Broadcasting in NumPy typically has a minimal impact on memory usage as it avoids unnecessary replication of data. Instead of creating additional copies of arrays, broadcasting enables NumPy to perform operations efficiently by reusing memory where possible, making it memory-efficient for element-wise operations on arrays of different shapes.

```
#5
a=np.arange(4).reshape(2,2)
b=[4,5]
c=a+b
print(c)
```

```
[[4 6]
 [6 8]]
```

```
#6
'''Broadcasting in NumPy allows operations between arrays with different shapes by automatically aligning their dimensions. It works by
when performing operations on arrays with differing shapes, making NumPy code more concise and readable.'''
a=np.arange(1,10).reshape(3,3)
b=np.array([1,2,3])
print(a+b)
```

```
[[ 2  4  6]
 [ 5  7  9]
 [ 8 10 12]]
```

```
# 7.
cel=np.array([[60,70,80,90,100],[10,20,30,40,50],[30, 30, 30, 40,90]])
feh = (9 * cel / 5 + 32)
print(feh)
```

```
[[140. 158. 176. 194. 212.]
 [ 50.  68.  86. 104. 122.]
 [ 86.  86.  86. 104. 194.]]
```

8 Broadcasting in NumPy offers advantages such as code simplicity, improved performance, memory efficiency, flexibility, intuitive syntax, and compatibility. It simplifies array operations, eliminates the need for explicit loops, optimizes performance through C implementation, saves memory by avoiding unnecessary copies, allows flexible handling of arrays with different shapes, provides an intuitive syntax resembling scalar operations, and seamlessly integrates with various NumPy functions for consistent behavior. Overall, broadcasting enhances the readability, efficiency, and flexibility of array operations in NumPy.

9

Broadcasting is generally more efficient than looping in NumPy for array operations due to its ability to perform element-wise operations without explicit loops, leveraging optimized C and Fortran code. Broadcasting is preferred for its speed, especially with large arrays.

In NumPy, the process by which these vectorized operations are performed is known as broadcasting. The reason for this name is that it is easy to conceptualize the above code by imagining that the single number is being "broadcast" across the larger array. In linear algebra, we would call this single number a scalar, as it is linearly scaling the vector, uniformly altering the vector's size. Crucially, NumPy broadcasts this scalar

element-wise across a larger array without actually making any copies of the scalar. Instead, the scalar is “spread” or “stretched out” so as to meet the size of any larger array of size n. The term “broadcast” is therefore only conceptual.

```
# 10.
arr_1d=np.array([10,20,30,40,50])
arr_2d=np.array([[60,70,80,90,100],[10,20,30,40,50],[30.0, 30.0, 30.0, 40,90]])
xv, yv = np.meshgrid(arr_1d, arr_2d)
print(xv)
print(yv)
```

```
[[10 20 30 40 50]
 [10 20 30 40 50]
 [10 20 30 40 50]
 [10 20 30 40 50]
 [10 20 30 40 50]
 [10 20 30 40 50]
 [10 20 30 40 50]
 [10 20 30 40 50]
 [10 20 30 40 50]
 [10 20 30 40 50]
 [10 20 30 40 50]
 [10 20 30 40 50]
 [10 20 30 40 50]
 [10 20 30 40 50]
 [10 20 30 40 50]]
[[ 60.  60.  60.  60.  60.]
 [ 70.  70.  70.  70.  70.]
 [ 80.  80.  80.  80.  80.]
 [ 90.  90.  90.  90.  90.]
 [100. 100. 100. 100. 100.]
 [ 10.  10.  10.  10.  10.]
 [ 20.  20.  20.  20.  20.]
 [ 30.  30.  30.  30.  30.]
 [ 40.  40.  40.  40.  40.]
 [ 50.  50.  50.  50.  50.]
 [ 30.  30.  30.  30.  30.]
 [ 30.  30.  30.  30.  30.]
 [ 30.  30.  30.  30.  30.]
 [ 40.  40.  40.  40.  40.]
 [ 90.  90.  90.  90.  90.]]
```