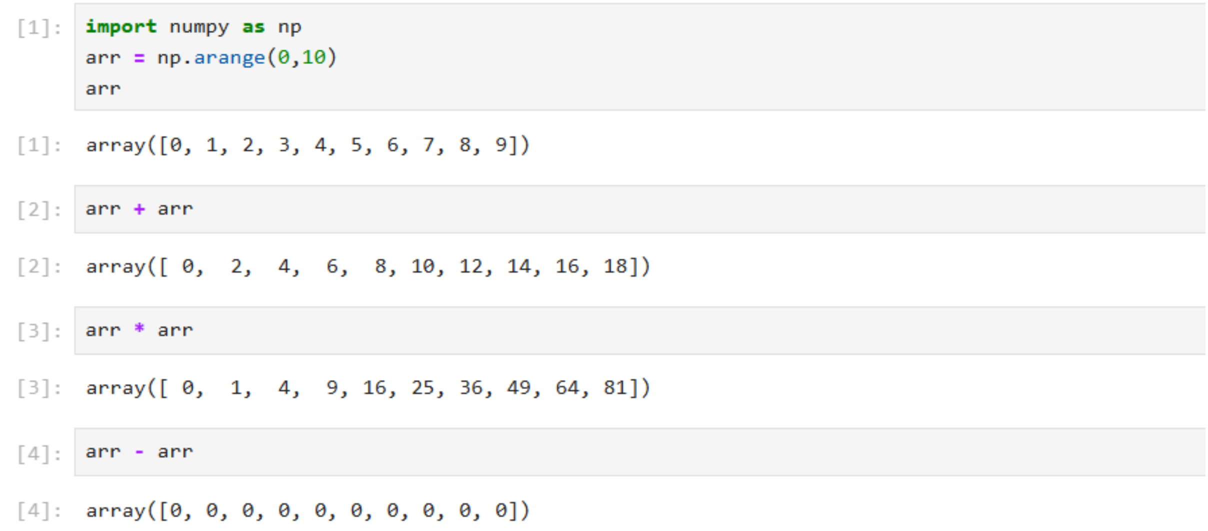
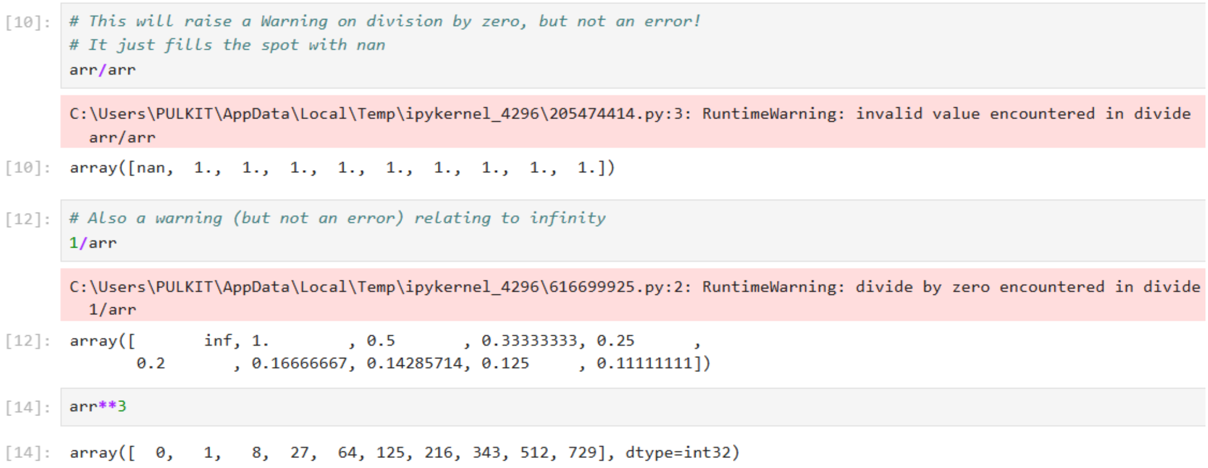
😊 NumPy Operations

1. In this lab, we are going to look at some NumPy operations in the Jupyter Notebook.
2. You can easily perform *array with array* arithmetic, or *scalar with array* arithmetic.
3. This code uses NumPy to create an array of integers from 0 to 9 and then performs various arithmetic operations on it.
4. Operations like addition, subtraction, multiplication, and exponentiation are applied element-wise.
5. When dividing the array by itself, a warning is triggered due to division by zero (at index 0), resulting in a nan (Not a Number).
6. Similarly, 1/arr causes a warning for division by zero and results in inf (infinity) at index 0.
7. NumPy handles these cases with warnings rather than throwing errors, which allows computation to continue even with invalid values.



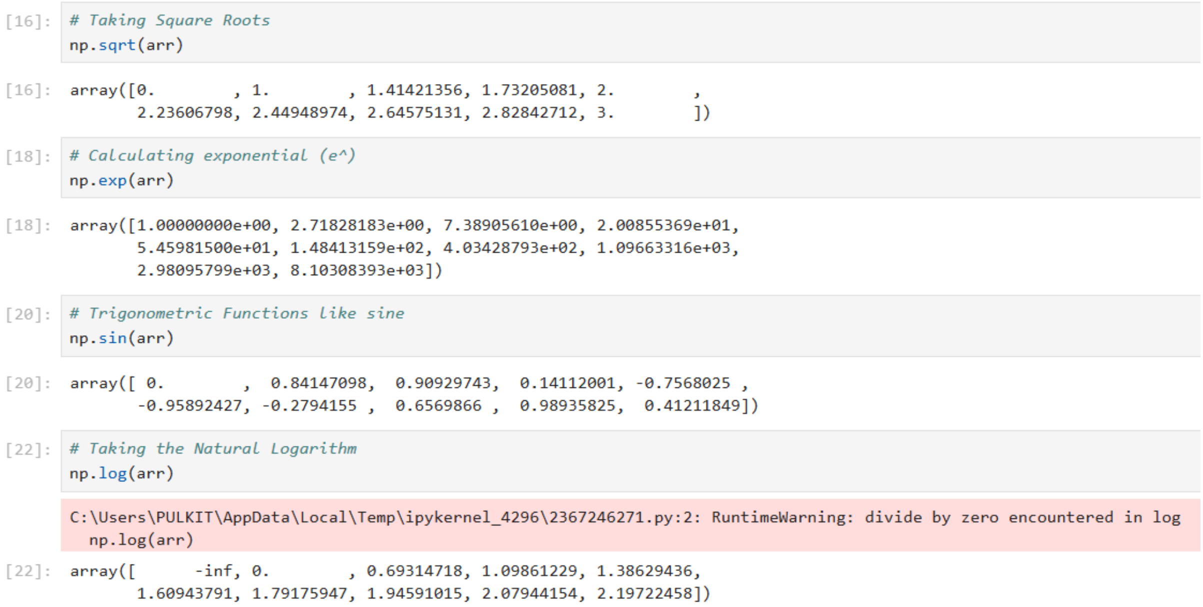
```
[1]:  import numpy as np
      arr = np.arange(0,10)
      arr

[1]:  array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

[2]:  arr + arr

[2]:  array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])

[3]:  arr * arr

[3]:  array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])

[4]:  arr - arr

[4]:  array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```



```
[10]: # This will raise a Warning on division by zero, but not an error!
      # It just fills the spot with nan
      arr/arr

      C:\Users\PULKIT\AppData\Local\Temp\ipykernel_4296\205474414.py:3: RuntimeWarning: invalid value encountered in divide
        arr/arr

[10]: array([nan,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])

[12]: # Also a warning (but not an error) relating to infinity
      1/arr

      C:\Users\PULKIT\AppData\Local\Temp\ipykernel_4296\616699925.py:2: RuntimeWarning: divide by zero encountered in divide
        1/arr

[12]: array([       inf, 1.        , 0.5       , 0.33333333, 0.25      ,
             0.2       , 0.16666667, 0.14285714, 0.125     , 0.11111111])

[14]: arr**3

[14]: array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729], dtype=int32)
```

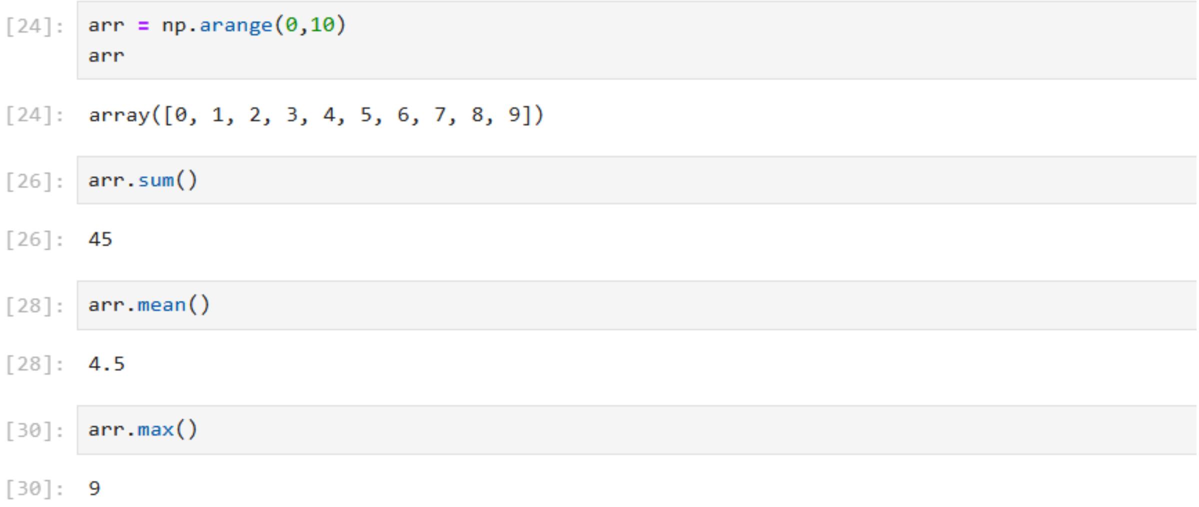
8. NumPy comes with many universal array functions, or *ufuncs*, which are essentially just mathematical operations that can be applied across the array.
9. This code demonstrates how NumPy handles various mathematical operations on arrays.

10. It calculates the square root, exponential (e^x), sine (trigonometric function), and natural logarithm (ln) of each element in the array arr.
11. These operations are applied element-wise, meaning they are performed on each item in the array individually.



```
[16]: # Taking Square Roots
      np.sqrt(arr)

[16]: array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
             2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])

[18]: # Calculating exponential (e^)
      np.exp(arr)

[18]: array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
             5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
             2.98095799e+03, 8.10308393e+03])

[20]: # Trigonometric Functions like sine
      np.sin(arr)

[20]: array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
             -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849])

[22]: # Taking the Natural Logarithm
      np.log(arr)

      C:\Users\PULKIT\AppData\Local\Temp\ipykernel_4296\2367246271.py:2: RuntimeWarning: divide by zero encountered in log
        np.log(arr)

[22]: array([      -inf, 0.        , 0.69314718, 1.09861229, 1.38629436,
             1.60943791, 1.79175947, 1.94591015, 2.07944154, 2.19722458])
```

12. NumPy also offers common summary statistics like *sum*, *mean*, and *max*. You would call these methods on an array.



```
[24]: arr = np.arange(0,10)
      arr

[24]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

[26]: arr.sum()

[26]: 45

[28]: arr.mean()

[28]: 4.5

[30]: arr.max()

[30]: 9
```

13. When working with 2-dimensional arrays (matrices) we have to consider rows and columns.
14. This becomes very important when we get to the section on pandas. In array terms, axis 0 (zero) is the vertical axis (rows), and axis 1 is the horizontal axis (columns).
15. These values (0,1) correspond to the order in which arr.shape values are returned.
16. In this code, a 2D NumPy array arr_2d is created with 3 rows and 4 columns.
17. The sum(axis=0) operation returns the sum of elements column-wise (down each column), while sum(axis=1) returns the sum row-wise (across each row).
18. The shape method confirms the array's dimensions as (3, 4).

```
[32]: arr_2d = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
      arr_2d
```

```
[32]: array([[ 1,  2,  3,  4],
             [ 5,  6,  7,  8],
             [ 9, 10, 11, 12]])
```

```
[34]: arr_2d.sum(axis=0)
```

```
[34]: array([15, 18, 21, 24])
```

```
[36]: arr_2d.shape
```

```
[36]: (3, 4)
```

```
[38]: # THINK ABOUT WHAT THIS WILL RETURN BEFORE RUNNING THE CELL!
      arr_2d.sum(axis=1)
```

```
[38]: array([10, 26, 42])
```