Lecture Plan: 02 - Array Mathematics

1. Motivating the need for element-wise operations
    - Analogy between Excel Columns and Numpy Arrays
2. Element-wise Arithmetic Operations
    - +, -, /, *, **, etc. on Numpy Arrays
3. Element-wise Comparison Operations
    - Single Element Comparison: Using relational operator directly
    - Overall Comparison: `np.all` and `np.any`
4. Aggregate Functions
    - Statistical Functions: `np.sum`, `np.cumsum`, `np.min`, `np.max`, `np.mean`, `np.median`, `np.std`, `np.corrcoef`.

1. Motivating the need for element-wise operations

Explain students how calculating the simple interest in Excel was such a easy task if the sheet looked like this:

principal	rate_of_interest	num_of_years	simple_interest
12000	6	3	
18000	6.25	4	
82000	7	2	
45000	6.75	5	

You will multiply the three cells in the first row and then divide it by 100. You will then drag down the formula to the cells below to obtain the result for the remaining cells.

However, when in Python, we are required to write tedious syntax - like loops!

```
principal = [12000, 18000, 82000, 45000]
rate_of_interest = [6, 6.25, 7, 6.75]
num_of_years = [3, 4, 2, 5]
simple_interest = []

# pythonic way of finding simple interest
for i in range(0, len(principal)):
    si = principal[i]*rate_of_interest[i]*num_of_years[i]/100
    simple_interest.append(si)
print(simple_interest)
```

Can we improve upon this?? The answer is yes! And for that, we will use numpy.

2. Element-wise Arithmetic Operations

Explain Numpy Arrays differ from traditional python lists. + for lists is used to append lists and other operators like -, *, / doesn't make sense.

```
a = [1, 2, 3, 4]
b = [5, 6, 7, 8]
print(a + b)
```

The output will be `[1, 2, 3, 4, 5, 6, 7, 8]`. What if I need output as `[6, 8, 10, 12]` which is the element-wise addition? In Python, we will need to write loop for that. However, numpy arrays allow us to do this element-wise operation easily.

Mention this: To perform element-wise operations, you will need two arrays of same shape and size. Otherwise, you will get an error.

```
import numpy as np

a = np.array([1, 2, 3, 4])
b = np.array([5, 6, 7, 8])
print(a + b)   # Addition of two arrays
print(b - a)   # Subtraction of two arrays
print(a*b)     # Multiplication of two arrays
print(b/a)     # Division of two arrays
print(a**b)    # One array raised to power of another
```

Once we have the ability to perform this element-wise operation, finding the `simple_interest` becomes easy and a one line instruction.

```
principal = np.array([12000, 18000, 82000, 45000])
rate_of_interest = np.array([6, 6.25, 7, 6.75])
num_of_years = np.array([3, 4, 2, 5])

simple_interest = principal*rate_of_interest*num_of_years/100
print(simple_interest)
```

In this way, we call back to the same topic from where we started, creating a good learning loop!

3. Element-wise Comparison Operations

Explain the single element comparison and the overall comparison of two numpy arrays.

Example: Imagine you have two groups of students: Group A and Group B. Each group has a list of test scores representing the performance of the students. You want to compare the test scores of the two groups to determine which group performed better overall.

```
group_a_scores = np.array([85, 92, 78, 90, 88])
group_b_scores = np.array([80, 95, 85, 92, 87])

# Comparing scores element-wise
comparison = group_a_scores > group_b_scores
print(comparison)

# Check if all scores in Group A are greater than Group B
all_greater = np.all(group_a_scores > group_b_scores)
print(all_greater)
```

```
# Check if any score in Group A is greater than Group B
any_greater = np.any(group_a_scores > group_b_scores)
print(any_greater)
```

In this example, the all_greater variable returns False because not all scores in Group A are greater than Group B. However, the any_greater variable returns True because there is at least one score in Group A that is greater than Group B.

4. Aggregate Functions

Explain how aggregate functions in NumPy allow us to perform calculations on arrays, aggregating multiple values into a single result. These aggregate functions provide useful insights and statistical measures for analyzing data.

Note: Although all these topics have been covered in Excel, revise these statistical topics first before moving to functions.

```
expenses = np.array([100, 150, 200, 75, 125])
total_expenses = np.sum(expenses)
print(total_expenses)

sales = np.array([50, 30, 40, 20, 25])
cumulative_sales = np.cumsum(sales)
print(cumulative_sales)

temperatures = np.array([25, 30, 20, 35, 28])
min_temp = np.min(temperatures)
max_temp = np.max(temperatures)
print(min_temp, max_temp)

ages = np.array([25, 30, 35, 40, 45])
average_age = np.mean(ages)
median_age = np.median(ages)
print(average_age, median_age)

scores = np.array([80, 85, 90, 75, 95])
std_dev = np.std(scores)
print(std_dev)

x = np.array([1, 2, 3, 4, 5])
y = np.array([5, 4, 3, 2, 1])
correlation = np.corrcoef(x, y)
print(correlation)
```