JAYPEEUNIVERSITYOFINFORMATIONTECHNOLOGY
WAKNAGHAT, P.O.–WAKNAGHAT, TEHSIL– KANDAGHAT,
DISTRICT– SOLAN (H.P.) PIN–173234(INDIA)
(Established by H.P. State Legislature vide Act No. 14 of 2002)

# 18B1WCI674: MACHINE LEARNING LAB

## Assignment-1

**Jan 05, 2026**

---

## Instructions:

1. You are advised to use Jupyter notebook/Google Colab to solve the assignmnets.
2. Notebook must be submitted to the classroom within the lab hours.

---

## Overview of Numpy array

NumPy (Numerical Python) is a fundamental library used for numerical computation in Python. It provides support for ***multi-dimensional arrays, vectorized operations, and efficient mathematical computations***, which are essential in data science, machine learning, image processing, and scientific computing.

## Part A: NumPy Basics

1. Install NumPy before starting the experiment.

```
pip install numpy
```

2. Import the NumPy library using the standard convention:

```
import numpy as np
```

3. All array creation, indexing, reshaping, and mathematical operations must be performed using NumPy functions only. Avoid using Python loops unless explicitly mentioned.

### 1. Basics & Array Creation

Array creation is the first step in numerical computing. NumPy provides various functions to initialize arrays with specific patterns such as ranges, zeros, constants, identity matrices, or random values.

1. Create a 1D array from 10 to 49:

```
np.arange(10, 50)
```

2. Create a $3 \times 3$ zero matrix:

```
np.zeros((3,3))
```

3. Create a $5 \times 5$ identity matrix:

   —

4. 10 evenly spaced values between 0 and 1:

   —

5. Random integers (size 20, range 1–100):

   —

6. $4 \times 4$ array filled with 7:

   —

7. Shape $(3,4)$ with values 0–11:

   —

8. Boolean array ($> 50$):

   —

9. Empty array of size 10:

   —

10. Convert list to NumPy array:

    —

## 2. Array Indexing & Slicing

Indexing and slicing allow selective access and modification of array elements without loops. NumPy supports boolean indexing, fancy indexing, and slicing, making data manipulation fast and readable.

1. First 5 elements:

```
arr[:5]
```

2. Reverse array:

```
arr[::-1]
```

3. Extract even numbers:

```
arr[arr % 2 == 0]
```

4. Select second row:

```
arr[1,:]
```

5. Last column:

   —

6. Replace negatives with zero:

   —

7. Elements greater than mean:

   —

8. Odd indices:

   —

9. Change index 3 to 99:

   —

10. $2 \times 2$ center submatrix:

    —

## 3. Reshaping & Manipulation

Reshaping operations modify the structure of data without changing values. This is critical when preparing data for algorithms that expect specific input dimensions.

1. Reshape size 12 to $3 \times 4$:

   ```
   arr.reshape(3,4)
   ```

2. Flatten array:

   ```
   arr.flatten()
   ```

3. Transpose $3 \times 2$ matrix:

   —

4. Vertical stacking:

   —

5. Horizontal stacking:

   —

6. Repeat elements twice:

   —

7. Tile array:

   —

8. Swap rows:

   —

9. Swap columns:

   —

10. Remove duplicates:

    —

## 4. Mathematical Operations

NumPy supports vectorized mathematical operations, eliminating the need for explicit loops. These operations are internally optimized using C, making them highly efficient.

1. Sum of elements:
   `arr.sum()`
2. Mean, median, std:
   `arr.mean(), np.median(arr), arr.std()`
3. Element-wise square:
   —
   —
4. Normalize to $[0, 1]$:
   —
5. Dot product:
   —
6. Element-wise multiplication:
   —
7. Count non-zero:
   —
8. Round to 2 decimals:
   —
9. Replace NaN with zero:
   —

—

## 5. Logical & Statistical Operations

Logical operations help in decision making, filtering, and statistical inference. These operations are often combined with boolean masks.

1. Any element $> 100$:
   `np.any(arr > 100)`
2. All positive:
   `np.all(arr > 0)`
3. Index of maximum:

4. Sort array:
5. Sort by second column:
6. Unique elements with counts:

   —
7. Cumulative sum:

   —
8. Cumulative product:

   —
9. Correlation coefficient:

   —
10. Mask $20 \leq x \leq 40$:

    —

## Part B: Advanced NumPy

Advanced operations leverage broadcasting, stride tricks, and vectorization to perform complex computations efficiently without loops.

1. Row-wise normalization:

```
arr / arr.sum(axis=1, keepdims=True)
```

2. Pairwise Euclidean distance:

```
np.sqrt(((X[:,None,:]  - X[None,:,:])**2).sum(axis=2))
```

3. Replace element with neighbor mean:

   —
4. Top-3 per column:

   —
5. Distance matrix (no loops):

   —
6. Local maxima:

   —