

# 5CS037 - Concepts and Technologies of AI.

## Worksheet - 1: A coding exercises on Numpy.

Prepared By: Siman Giri {Module Leader - 5CS037}

November 11, 2025

## 1 Instructions

This worksheet contains programming exercises based on the material discussed from the slides. This is a graded exercise and are to be completed on your own and is compulsory to submit.

Please answer the questions below using python in the Jupyter Notebook and follow the guidelines below:

- This worksheet must be completed individually.
- All the solutions must be written in Jupyter Notebook.
- Our Recommendation - Google Colaboratory.

————— May the Force be with You —————

## 2 Before you Start: Setting Up Your Environment.

We highly recommend the use of **Google Colab** or in option you may also use **Jupyter Notebook with Anaconda** or any other Notebook like environment.

### 2.1 Using Google Colab: What is Google Colaboratory?

- Also known as Colab, is “Jupyter notebook” like environment with live code, visualizations, and narrative text. Colab notebooks are the same as Jupyter.
- Colab notebooks are the same as Jupyter notebook, including the `.ipynb` extension and requires no setup on your computer!
- To be able to write and run code, you need to sign in with your Google credentials. This is the only step that’s required from your end. No other configuration is needed.
- Head over to `colab.research.google.com`. You’ll see the following screen.

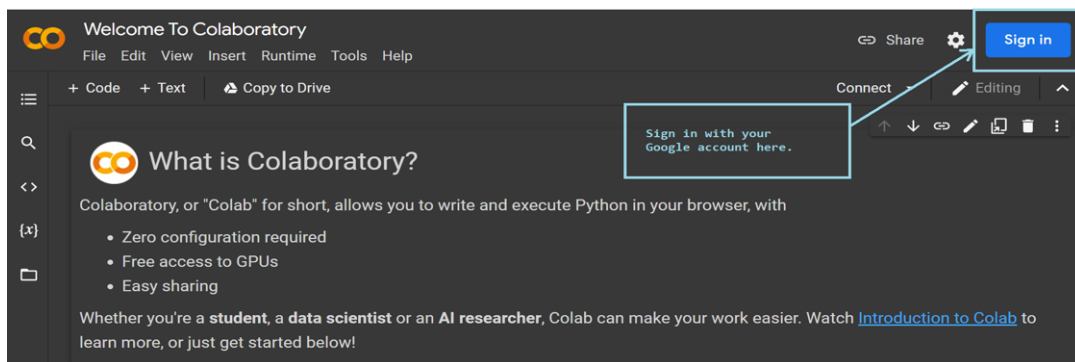


Figure 1: Google Colab: Home Screen.

### 1. Your First Google Colab Notebook.

- **Creating a new Notebook:**
  - Once you’ve signed in to Colab, you can create a new notebook by clicking on 'File' → 'New notebook', as shown below:

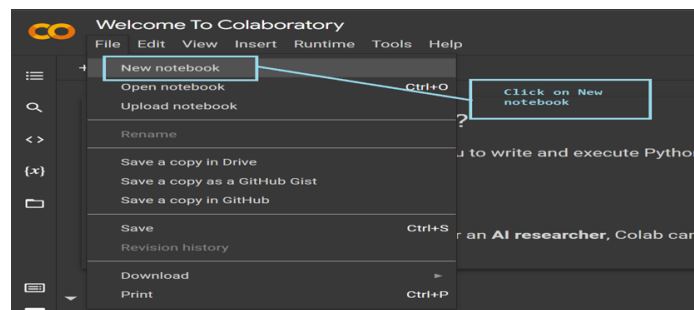


Figure 2: Google Colab: Creating a new Notebook.

- **Naming your Notebook:**

- Notebook uses the naming convention `\UntitledXX.ipynb`.
- To rename the notebook, click on this name and type in the desired name.

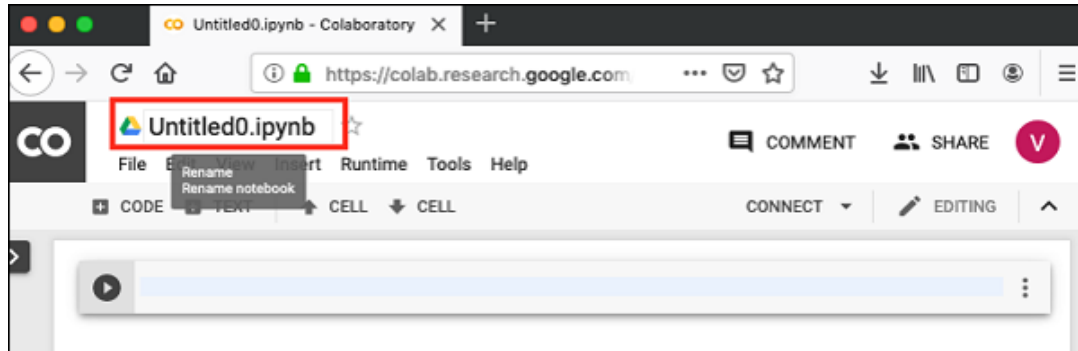


Figure 3: Google Colab: Renaming the Notebook.

- **Mounting Google Drive:**

- We can easily mount Google drive onto the current instance of Colab.
- This will enable you to access all the datasets and files that are stored in your drive.

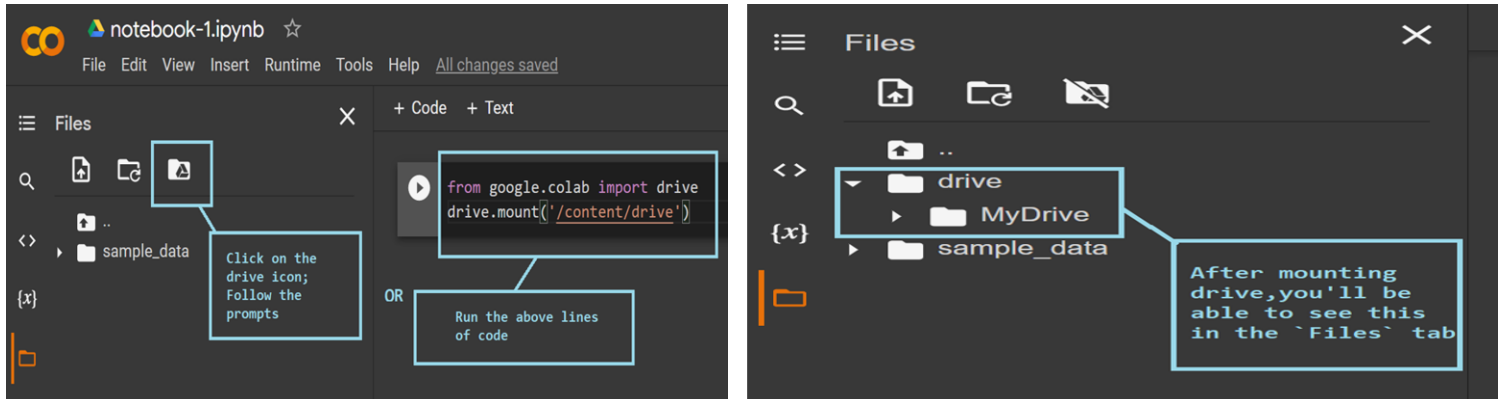


Figure 4: Google Colab: Mounting Google Drive-1.

### 3 Getting Started with Numpy.

This Section contains all the sample code from the slides and are here for your reference, you are highly recommended to run all the code with some of the input changed in order to understand the meaning of the operations and also to be able to solve all the exercises from further sections.

- **Cautions!!!:**

- This Guide does not contain sample output, as we expect you to re-write the code and observe the output.
- If found: any error or bugs, please report to your instructor and Module leader.  
{Will hugely appreciate your effort.}

#### 3.1 Array Introduction:

##### 1. Importing Numpy:

Sample Code from Slide - 14 - Importing Numpy and Array Type.

```
import numpy as np
# Create and display zero, one, and n-dimensional arrays
zero_dim_array = np.array(5)
one_dim_array = np.array([1, 2, 3])
n_dim_array = np.array([[1, 2], [3, 4]])
for arr in [zero_dim_array, one_dim_array, n_dim_array]:
    print(f"Array:\n{arr}\nDimension: {arr.ndim}\nData type: {arr.dtype}\n")
```

##### 2. Array Dimensions: Shape and Resahpe of Array:

Sample Code from Slide - 16 - Shape of an Array.

```
import numpy as np
# Create arrays of different dimensions
array_0d = np.array(5)
array_1d = np.array([1, 2, 3, 4, 5])
array_2d = np.array([[1, 2, 3], [4, 5, 6]])
array_3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
# Print arrays with shapes
for i, arr in enumerate([array_0d, array_1d, array_2d, array_3d]):
    print(f"{i}D Array:\n{arr}\nShape: {arr.shape}\n")
```

Sample Code from Slide - 17 - Reshaping an Array.

```
import numpy as np
array = np.array([[1, 2, 3], [4, 5, 6]]) # Shape (2, 3)
reshaped_array = array.reshape(3, 2) # Reshape to (3, 2), keeping 6 elements
print("Original Shape:", array.shape, "\nReshaped Shape:", reshaped_array.shape)
```

## 3.2 Array Creation:

### 1. Using In - Built Function:

Sample Code from Slide - 18 - Arrays with evenly Spaced values - arange.

```
import numpy as np
a = np.arange(1, 10)
print(a)
x = range(1, 10)
print(x) # x is an iterator
print(list(x))
# further arange examples:
x = np.arange(10.4)
print(x)
x = np.arange(0.5, 10.4, 0.8)
print(x)
```

Sample Code from Slide - 19 - Arrays with evenly Spaced values - linspace.

```
import numpy as np
# 50 values between 1 and 10:
print(np.linspace(1, 10))
# 7 values between 1 and 10:
print(np.linspace(1, 10, 7))
# excluding the endpoint:
print(np.linspace(1, 10, 7, endpoint=False))
```

### 2. Initializing Arrays with Ones, Zeros and Empty:

Sample Code form slide - 20 - Initializing an Array.

```
import numpy as np
# Create arrays of specified shapes
ones_array = np.ones((2, 3)) # Shape: (2, 3)
zeros_array = np.zeros((3, 2)) # Shape: (3, 2)
empty_array = np.empty((2, 2)) # Shape: (2, 2)
identity_matrix = np.eye(3) # Shape: (3, 3)
print(ones_array, zeros_array, empty_array, identity_matrix, sep='\n\n')
```

### 3. By Manipulating Existing Array:

#### 3.1 Using np.array:

Sample Code form slide - 21 - Converting list to array using np.array.

```
import numpy as np
array_from_list = np.array([1, 2, 3]) # [1 2 3]
array_from_tuple = np.array((4, 5, 6)) # [4 5 6]
array_from_nested_list = np.array([[1, 2, 3], [4, 5, 6]]) # [[1 2 3] [4 5 6]]
print(array_from_list, array_from_tuple, array_from_nested_list, sep='\n')
```

### 3.2 Using shape of an existing array:

Sample Code form slide - 22 - From shape of an existing array.

```
import numpy as np
# Existing Array of Shape:(2,3)
arr = np.array([[1, 2, 3], [4, 5, 6]])
# Creating Array with Shape of existing array:
zeros, ones, empty = np.zeros(arr.shape), np.ones(arr.shape), np.empty(arr.shape) # Shape: (2,3)
print(arr, zeros, ones, empty, sep='\n')
```

### 3.3 With Math, Copying or Slicing and Array:

Sample Code form slide - 23 - Manipulating existing array.

```
import numpy as np
array = np.array([1, 2, 3])
# Multiplies each element by 2, result: [2, 4, 6]
new_array = array * 2
# Creates a new array with the same elements
copied_array = np.copy(array)
# Slices elements from index 1 to 3, result:[2]
sliced_array = array[1:2]
```

Sample Code form slide - 26 - With Concatenating Operation.

```
import numpy as np
arr1 = np.array([[1, 2], [3, 4]]) # Shape: (2, 2)
arr2 = np.array([[5, 6], [7, 8]]) # Shape: (2, 2)
concat_axis0 = np.concatenate((arr1, arr2), axis=0)
# Concatenate along axis 0 (vertical) Shape: (4, 2)
concat_axis1 = np.concatenate((arr1, arr2), axis=1)
# Concatenate along axis 1 (horizontal) Shape: (2, 4)
print(concat_axis0, concat_axis1, sep='\n')
```

### 3.4 Array: Indexing and Slicing:

Sample Code form slide - 34 - Sample Code for Indexing and Slicing.

```
import numpy as np
# Arrays
arr1d = np.array([0, 1, 2, 3, 4, 5])
arr2d = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
arr3d = np.array([[[0, 1], [2, 3]], [[4, 5], [6, 7]], [[8, 9], [10, 11]]])
# Slicing examples
print("Basic:", arr1d[1:4], arr2d[1:3, 0:2], arr3d[1:2, 0:2, 1:2])
# Output Basic: arr1d:[1 2 3] arr2d:[[3 4] [6 7]] arr3d:[[[5] [7]]]
print("Step:", arr1d[1:5:2], arr2d[:, 1::2, 1::2], arr3d[:, :2, ::2, 1::2])
# Output: Step: [1 3] [[1] [7]] [[[1] [9]]]
```

### 3.3 Array Mathematics:

#### 1. Elementary Mathematical Operation with universal functions.

Sample Code form slide - 37 - Intriduction to "ufuncs".

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
print("Add:", np.add(arr1, arr2)) # Output Add: [5 7 9]
print("Subtract:", np.subtract(arr1, arr2))
# Output Subtract: [-3 -3 -3]
print("Divide:", np.divide(arr1, arr2))
# Output Divide: [0.25 0.4 0.5 ]
print("Multiply:", np.multiply(arr1, arr2))
# Output Multiply: [ 4 10 18]
print("Power:", np.power(arr1, arr2))
# Output Power: [ 1 32 729]
print("Exp:", np.exp(arr1))
# Output Exp: [ 2.71828183 7.3890561 20.08553692]
```

#### 2. Matrix Multiplications.

Sample Code form slide - 44 - Using "np.dot".

```
import numpy as np
# Define two matrices
A = np.array([[1, 2, 3], [4, 5, 6]])
B = np.array([[7, 8], [9, 10], [11, 12]])
# Matrix multiplication using np.dot
result_dot = np.dot(A, B)
print("Result with np.dot:\n", result_dot) # Output shape: (2, 2)
```

Sample Code form slide - 45 - Using "np.matmul".

```
import numpy as np
# Define two matrices
A = np.array([[1, 2, 3], [4, 5, 6]])
B = np.array([[7, 8], [9, 10], [11, 12]])
# Matrix multiplication using @ operator
result_at = A @ B
print("Result with @ operator:\n", result_at) # Output shape: (2, 2)
# Matrix multiplication using np.matmul
result_matmul = np.matmul(A, B)
print("Result with np.matmul:\n", result_matmul) # Output shape: (2, 2)
```

### 3.4 Linear Algebra with Numpy - np.linalg:

Sample Code form slide - 48 - Common linear algebra operation with np.linalg.

```
import numpy as np
A = np.array([[1, 2], [3, 4]])
B = np.array([5, 6])
print("Inverse:\n", np.linalg.inv(A))
# Output: Inverse: [[-2.  1. ], [ 1.5 -0.5]]
print("Determinant:", np.linalg.det(A))
# Output: Determinant: -2.0
print("Frobenius Norm:", np.linalg.norm(A, 'fro'))
# Output: Frobenius Norm: 5.4772
print("2-Norm (Euclidean):", np.linalg.norm(A))
# Output: 2-Norm(Euclidean): 5.4772
print("Solution x:", np.linalg.solve(A, B))
#Output: Solution x: [-4.  4.5]
```



## 4 TO - DO - Task

Please complete all the problem listed below.

### 4.1 Warming Up Exercise: Basic Vector and Matrix Operation with Numpy.

#### Problem - 1: Array Creation:

Complete the following Tasks:

1. Initialize an empty array with size  $2 \times 2$ .
2. Initialize an all one array with size  $4 \times 2$ .
3. Return a new array of given shape and type, filled with fill\_value. {Hint: np.full}
4. Return a new array of zeros with same shape and type as a given array. {Hint: np.zeros\_like}
5. Return a new array of ones with same shape and type as a given array. {Hint: np.ones\_like}
6. For an existing list `new_list = [1,2,3,4]` convert to an numpy array. {Hint: np.array()}

#### Problem - 2: Array Manipulation: Numerical Ranges and Array indexing:

Complete the following tasks:

1. Create an array with values ranging from 10 to 49. {Hint: np.arange()}.
2. Create a  $3 \times 3$  matrix with values ranging from 0 to 8.  
{Hint: look for np.reshape() }
3. Create a  $3 \times 3$  identity matrix. {Hint: np.eye() }
4. Create a random array of size 30 and find the mean of the array.  
{Hint: check for np.random.random() and array.mean() function}
5. Create a  $10 \times 10$  array with random values and find the minimum and maximum values.
6. Create a zero array of size 10 and replace 5<sup>th</sup> element with 1.
7. Reverse an array `arr = [1,2,0,0,4,0]`.
8. Create a 2d array with 1 on border and 0 inside.
9. Create a  $8 \times 8$  matrix and fill it with a checkerboard pattern.

**Problem - 3: Array Operations:**

For the following arrays:

`x = np.array([[1,2],[3,5]])` and `y = np.array([[5,6],[7,8]])`;

`v = np.array([9,10])` and `w = np.array([11,12])`;

Complete all the task using numpy:

1. Add the two array.
2. Subtract the two array.
3. Multiply the array with any integers of your choice.
4. Find the square of each element of the array.
5. Find the dot product between: `v` and `w` ; `x` and `v` ; `x` and `y`.
6. Concatenate `x` and `y` along row and Concatenate `v` and `w` along column.  
{Hint: try `np.concatenate()` or `np.vstack()` functions.
7. Concatenate `x` and `v`; if you get an error, observe and explain why did you get the error?

**Problem - 4: Matrix Operations:**

- For the following arrays:

`A = np.array([[3,4],[7,8]])` and `B = np.array([[5,3],[2,1]])`;

Prove following with Numpy:

1. Prove  $A.A^{-1} = I$ .
  2. Prove  $AB \neq BA$ .
  3. Prove  $(AB)^T = B^T A^T$ .
- Solve the following system of Linear equation using Inverse Methods.

$$2x - 3y + z = -1$$

$$x - y + 2z = -3$$

$$3x + y - z = 9$$

{Hint: First use Numpy array to represent the equation in Matrix form. Then Solve for:  $AX = B$ }

- Now: solve the above equation using `np.linalg.inv` function. {Explore more about "linalg" function of Numpy}

## 4.2 Experiment: How Fast is Numpy?

In this exercise, you will compare the performance and implementation of operations using plain Python lists (arrays) and NumPy arrays. Follow the instructions:

### 1. Element-wise Addition:

- Using **Python Lists**, perform element-wise addition of two lists of size 1,000,000. Measure and Print the time taken for this operation.
- Using **Numpy Arrays**, Repeat the calculation and measure and print the time taken for this operation.

### 2. Element-wise Multiplication

- Using **Python Lists**, perform element-wise multiplication of two lists of size 1,000,000. Measure and Print the time taken for this operation.
- Using **Numpy Arrays**, Repeat the calculation and measure and print the time taken for this operation.

### 3. Dot Product

- Using **Python Lists**, compute the dot product of two lists of size 1,000,000. Measure and Print the time taken for this operation.
- Using **Numpy Arrays**, Repeat the calculation and measure and print the time taken for this operation.

### 4. Matrix Multiplication

- Using **Python lists**, perform matrix multiplication of two matrices of size 1000x1000. Measure and print the time taken for this operation.
- Using **NumPy arrays**, perform matrix multiplication of two matrices of size 1000x1000. Measure and print the time taken for this operation.

————— The - End —————