ADS_Lab_2_Numpy_array

```
import numpy as np
```

1.Creating Arrays

Create the following arrays:

- A 1D array: [2, 4, 6, 8, 10]
- A 2D array: [[1, 2, 3], [4, 5, 6]]
- A 3D array: [[[1,2], [3,4]], [[5,6], [7,8]]]

```
array_1D = np.array([2,4,6,8,10])    # 1D array
array_1D
```

```
array([ 2,  4,  6,  8, 10])
```

```
array_2D = np.array([[1,2,3],[4,5,6]])  # 2D array
array_2D
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
array_2D.shape  # shape of 2D array
```

```
(2, 3)
```

```
array_3D = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])   # 3D array
array_3D
```

```
array([[[1, 2],
        [3, 4]],

       [[5, 6],
        [7, 8]]])
```

```
array_3D.shape  # shape of 3D array
```

```
(2, 2, 2)
```

Generate:

- An array of 10 zeros
- An array of 5 ones
- Numbers from 5 to 25 with a step of 5
- 6 equally spaced values between 0 and 1

```
array_zeros = np.zeros(10)    #array of 10 zeros
array_zeros
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
array_ones = np.ones(5)    # array of 5 ones
array_ones
```

```
array([1., 1., 1., 1., 1.])
```

```
number = np.arange(5,26,5)  #Numbers from 5 to 25 with a step of 5
number
```

```
array([ 5, 10, 15, 20, 25])
```

```
equallyPaces = np.linspace(0,1,6)  # 6 equally spaced values between 0 and 1
equallyPaces
```

```
array([0. , 0.2, 0.4, 0.6, 0.8, 1. ])
```

2.Array Join, Split, Search, Sort

- Given a = [10, 20, 30] and b = [40, 50, 60]:

1. Join them into a single array.
2. Split the resulting array into 3 equal parts.

```
a = np.array([10,20,30])
b= np.array([40,50,60])
```

```
#  Join them into a single array.
join = np.concatenate((a,b))    # concatenate function is used
join
```

```
array([10, 20, 30, 40, 50, 60])
```

```
 # Split the joined array into 3 equal parts
split_arrays = np.split(join, 3)
split_arrays
```

```
[array([10, 20]), array([30, 40]), array([50, 60])]
```

- For arr = [15, 25, 35, 25, 45]:

1. Find all indices where the element is 25.

```
arr1 = np.array([15, 25, 35, 25, 45])
indices_25 = np.where(arr1 == 25)[0]
indices_25
```

```
array([1, 3])
```

- For arr = [12, 5, 18, 7, 3]:

1. Sort the array in ascending order.

```
arr2 = np.array([12,5,18,7,3])

sorted_arr = np.sort(arr2)
sorted_arr
```

```
array([ 3,  5,  7, 12, 18])
```

3. Indexing, Slicing, Iterating

For the matrix: [[11, 12, 13], [21, 22, 23], [31, 32, 33]]

- Extract the element 22 using indexing.

```
matrix = np.array([[11,12,13],[21,22,23],[31,32,33]])
matrix
```

```
array([[11, 12, 13],
       [21, 22, 23],
       [31, 32, 33]])
```

- Slice the first two rows

```
first_two_rows = matrix[:2, :]
print("First two rows:\n", first_two_rows)
```

```
First two rows:
 [[11 12 13]
 [21 22 23]]
```

```
# Slice the last column

last_column = matrix[:, -1]
```

```
print("Last column:", last_column)
```

```
Last column: [13 23 33]
```

```
# Iterate through all elements and print them
print("All elements in matrix:")
for element in matrix.flat:
    print(element, end=' ')
print()
```

```
All elements in matrix:
11 12 13 21 22 23 31 32 33
```

4. Copying Arrays

```
arr_copy = np.array([1, 2, 3, 4, 5])

# Create a view and change the first element to 99
view_arr = arr_copy.view()
view_arr[0] = 99
print("View changed first element to 99:", view_arr)
print("Original array after view change:", arr_copy)  # also affected
```

```
View changed first element to 99: [99  2  3  4  5]
Original array after view change: [99  2  3  4  5]
```

```
# Create a deep copy and change second element to 77
deep_copy_arr = arr_copy.copy()
deep_copy_arr[1] = 77
print("Deep copy changed second element to 77:", deep_copy_arr)
print("Original array after deep copy change:", arr_copy)  # unchanged
```

```
Deep copy changed second element to 77: [99 77  3  4  5]
Original array after deep copy change: [99  2  3  4  5]
```

5. Array Shape Manipulation

```
arr_reshape = np.arange(1, 13)
```

```
# Reshape into 3x4 matrix
reshaped_matrix = arr_reshape.reshape(3, 4)
print("Reshaped 3x4 matrix:\n", reshaped_matrix)
```

```
Reshaped 3x4 matrix:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

```
# Flatten the reshaped matrix into 1D
flattened_array = reshaped_matrix.flatten()
print("Flattened array:", flattened_array)
```

```
Flattened array: [ 1  2  3  4  5  6  7  8  9 10 11 12]
```

```
# Resize into 2x8 matrix
arr_resized = np.resize(arr_reshape, (2, 8))
print("Resized 2x8 matrix:\n", arr_resized)
```

```
Resized 2x8 matrix:
 [[ 1  2  3  4  5  6  7  8]
 [ 9 10 11 12  1  2  3  4]]
```

6. Identity Array and Eye Function

- Generate a 5x5 identity matrix

```
identity_matrix = np.identity(5)
print("5x5 identity matrix:\n", identity_matrix)
```

```
5x5 identity matrix:
 [[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

```
# Generate a 4x6 matrix using eye() with diagonal offset k=2
eye_matrix = np.eye(4, 6, k=2)
print("4x6 identity with diagonal offset 2:\n", eye_matrix)
```

```
4x6 identity with diagonal offset 2:
 [[0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 1.]]
```

```
Start coding or generate with AI.
```

Lab 2b - Pandas – Series and DataFrames

1. Accessing and Slicing of Series

```
import pandas as pd
series = pd.Series([10, 20, 30, 40, 50], index=['a', 'b', 'c', 'd', 'e'])
```

```
# Access element with label 'c'
elem_c = series['c']
print("Element with label 'c':", elem_c)
```

```
Element with label 'c': 30
```

```
# Slice from index 'b' to 'd'
slice_bd = series['b':'d']
print("Slice from 'b' to 'd':\n", slice_bd)
```

```
Slice from 'b' to 'd':
 b    20
c    30
d    40
dtype: int64
```

```
# Retrieve first three elements
first_three = series.head(3)
print("First three elements:\n", first_three)
```

```
First three elements:
 a    10
b    20
c    30
dtype: int64
```

2. Accessing and Slicing of DataFrames

```
df = pd.DataFrame({
    'ID': [1, 2, 3, 4],
    'Name': ['Alex', 'Bella', 'Chris', 'Diana'],
    'Age': [22, 20, 23, 21],
    'Marks': [85, 90, 78, 92]
})
```

```
# Select the column Name
name_column = df['Name']
print("Name column:\n", name_column)
```

```
Name column:
 0     Alex
```

```
1    Bella
2    Chris
3    Diana
Name: Name, dtype: object
```

```
# Select rows with indices 1 and 2
selected_rows = df.loc[[1, 2]]
print("Rows with indices 1 and 2:\n", selected_rows)
```

```
Rows with indices 1 and 2:
    ID   Name  Age  Marks
1   2  Bella   20     90
2   3  Chris   23     78
```

```
# Slice to display first two rows and two columns
slice_rows_cols = df.iloc[:2, :2]
print("First two rows and two columns:\n", slice_rows_cols)
```

```
First two rows and two columns:
    ID   Name
0   1   Alex
1   2  Bella
```

```
# Access element at row index 2 and column Marks
elem_marks = df.at[2, 'Marks']
print("Element at row 2, column 'Marks':", elem_marks)
```

```
Element at row 2, column 'Marks': 78
```

3. Arithmetic and Logical Operations on DataFrame

```
df_ab = pd.DataFrame({
    'A': [5, 15, 25],
    'B': [10, 20, 30]
})
```

```
# Add 10 to all elements of column A
df_ab['A'] = df_ab['A'] + 10
print("Column A after adding 10:\n", df_ab['A'])
```

```
Column A after adding 10:
 0    15
1    25
2    35
Name: A, dtype: int64
```

```
# Multiply column B by 2
df_ab['B'] = df_ab['B'] * 2
print("Column B after multiplying by 2:\n", df_ab['B'])
```

```
Column B after multiplying by 2:
 0    20
1    40
2    60
Name: B, dtype: int64
```

```
# Create boolean mask where column A > 10
mask = df_ab['A'] > 10
print("Boolean mask for A > 10:\n", mask)
```

```
Boolean mask for A > 10:
 0    True
1    True
2    True
Name: A, dtype: bool
```

```
# Filter rows where column B < 25
filtered_rows = df_ab[df_ab['B'] < 25]
print("Rows where B < 25:\n", filtered_rows)
```

```
Rows where B < 25:
     A   B
0  15  20
```

4. Index Objects

```
df_index = pd.DataFrame({
    'X': [5, 7, 9],
    'Y': [6, 8, 10]
}, index=['one', 'two', 'three'])
```

```
index_obj = df_index.index  # Print index object
row_two = df_index.loc['two']  # Access row 'two'
has_four = 'four' in df_index.index  # Check if 'four' in index
```

5. Re-indexing

```
df_score = pd.DataFrame({'Score': [85, 90, 78]}, index=['a', 'b', 'c'])

reindexed = df_score.reindex(['a', 'b', 'c', 'd', 'e'])  # Re-index
filled = reindexed.fillna(0)  # Fill missing with 0
reordered = df_score.reindex(['c', 'a', 'b'])  # Reorder index
```

6. Drop Entry

```
df_drop_row = df.drop(index=2)  # Drop row index 2
df_drop_col = df.drop(columns=['Marks'])  # Drop column Marks
```

7. Selecting Entries

```
marks_above_80 = df[df['Marks'] > 80]  # Marks > 80
name_marks = df[['Name', 'Marks']]  # Name and Marks columns
age_between = df[(df['Age'] >= 20) & (df['Age'] <= 22)]  # Age between 20 and 22
```

```
Start coding or generate with AI.
```