**Introduction to NumPy**

**Explain Python Arrays with suitable example.**

Python does not have built-in support for Arrays, but [**Python Lists**](https://www.w3schools.com/python/python_lists.asp) can be used instead.

Arrays are used to store multiple values in one single variable.

Create an array containing car names:

cars = ["Ford", "Volvo", "BMW"]

print(cars)

output: [‘Ford’, ‘Volvo’, ‘BMW’]

**Access the Elements of an Array**

You refer to an array element by referring to the index number.

cars = ["Ford", "Volvo", "BMW"]

x = cars[0]

print(x)

**output:** Ford

**The Length of an Array**

Use the len() method to return the length of an array (the number of elements in an array).

cars = ["Ford", "Volvo", "BMW"]

x = len(cars)

print(x)

**output:** 3

**Looping Array Elements**

You can use the for in loop to loop through all the elements of an array.

cars = ["Ford", "Volvo", "BMW"]

for x in cars:  
  print(x)

**output:** Ford

Volvo

BMW

**Adding Array Elements**

You can use the append() method to add an element to an array.

cars = ["Ford", "Volvo", "BMW"]

cars.append("Honda")

print(cars)

**output:** ['Ford', 'Volvo', 'BMW', 'Honda']

**Removing Array Elements**

You can use the pop() method to remove an element from the array.

cars = ["Ford", "Volvo", "BMW"]

cars.pop(1)

print(cars)

**output:** ['Ford', 'BMW']

You can also use the **remove()** method to remove an element from the array.

cars = ["Ford", "Volvo", "BMW"]

cars.remove("Volvo")

print(cars)

**output:** ['Ford', 'BMW']

***Explain NumPyArray in detail.***

* NumPy is a Python library.
* NumPy is used for working with arrays.
* NumPy is short for "Numerical Python".
* In Python we have lists that serve the purpose of arrays, but they are slow to process.
* NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
* NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.
* This behavior is called locality of reference in computer science.
* This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.
* The array object in NumPy is called ndarray.
* We can create a NumPy ndarray object by using the array() function.

………………………………………………………………..

**0-D arrays**, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

import numpy as np

arr = np.array(42)

print(arr)

output: 42

**uni-dimensional or 1-D array** has 0-D arrays as its elements.

import numpy as np

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

print(arr)

print(type(**arr))**

**output**: [1 2 3 4 5]

<class 'numpy.ndarray'>

**2-D Array** has 1-D arrays as its elements.

These are often used to represent matrix.

NumPy has a whole sub module dedicated towards matrix operations called **numpy.mat**

import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

print(arr)

**output:** [[1 2 3]

[4 5 6]]

**3-D Array** has 2-D arrays (matrices) as its elements .

import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)

**Output:** [[[1 2 3]

[4 5 6]]

[[1 2 3]

[4 5 6]]]

……………………………………………………………………………………………………………………………………………………

import numpy as np  
  
a = np.array(42) # 0-D array  
b = np.array([1, 2, 3, 4, 5]) # 1-D array  
c = np.array([[1, 2, 3], [4, 5, 6]]) # 2-D array  
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]) # 3-D array

print(a)

print(b)

print(c)

print(d)

print(a.ndim) # to display no.of dimensions  
print(b.ndim)  
print(c.ndim)  
print(d.ndim)

***Explain Indexing and Slicing?***

**NumPy Array Indexing:**

Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

**Accessing 1-D Array:**

import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[0])

Output : 1

**Accessing 2-D Array:**

import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st row: ', arr[0, 1])

Output: 2nd element on 1st dim: 2

**Accessing 3-D Array:**

#Access the third element of the second array of the first array

import numpy as np

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

print(arr[0, 1, 2])

Output: 6

**NumPy Slicing:**

Slicing in python means taking elements from one given index to another given index.

We pass slice instead of index like this: [*start*:*end*].

We can also define the step, like this: [*start*:*end*:*step*].

If we don't pass start its considered 0

If we don't pass end its considered length of array in that dimension

If we don't pass step its considered 1

import numpy as np  
arr = np.array([1, 2, 3, 4, 5, 6, 7])  
  
print(arr[1:5]) #Slice elements from index 1 to index 5

print(arr[4:]) #Slice elements from index 4 to the end of the array

print(arr[:4]) #Slice elements from the beginning to index 4

print(arr[-3:-1]) #Slice from the index 3 from the end to index 1 from the end:

print(arr[1:5:2]) #Return every other element from index 1 to index 5

print(arr[::2]) #Return every other element from the entire array

Output:

[2 3 4 5]

[5 6 7]

[1 2 3 4]

[5 6]

[2 4]

[1 3 5 7]

**Slicing 2-D Array:**

import numpy as np

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[1, 1:4 #From the second element, slice elements from index 1 to index 4

print(arr[0:2, 2]) #From both elements, return index 2

print(arr[0:2, 1:4]) #From both elements, slice index 1 to index 4

Output:

[7 8 9]

[3 8]

[[2 3 4]

[7 8 9]]

***Explain Operations on Arrays?***

**NumPy Array:** Numpy array is a powerful N-dimensional array object which is in the form of rows and columns. We can initialize **NumPy arrays** from nested Python lists and access it elements.

NumPy is an open source mathematical and scientific computing library for Python programming tasks. The name NumPy is shorthand for Numerical Python. The NumPy library offers a collection of high-level mathematical functions including support for multi-dimensional arrays, masked arrays and matrices.

A Numpy array on a structural level is made up of a combination of:

* The **Data**pointer indicates the memory address of the first byte in the array.
* The **Data type** or **dtype** pointer describes the kind of elements that are contained within the array.
* The **shape** indicates the shape of the array.
* The **strides** are the number of bytes that should be skipped in memory to go to the next element.

import numpy as np

arr1 = np.array([1,2,3,4,5])

arr2 = np.array([6,7,8,9,10])

newarr = np.add(arr1, arr2)

print(newarr)

newarr = np.subtract(arr1, arr2)

print(newarr)

newarr = np.multiply(arr1, arr2)

print(newarr)

newarr = np.divide(arr1, arr2)

print(newarr)

newarr = np.power(arr1, arr2)

print(newarr)

print(min(arr1))

print(max(arr1))

Output:

[ 7 9 11 13 15]

[-5 -5 -5 -5 -5]

[ 6 14 24 36 50]

[0.16666667 0.28571429 0.375 0.44444444 0.5 ]

[ 1 128 6561 262144 9765625]

**Few NumPy Methods:**

 Min, Max, Unique, Mean, Expand Dimensions, Median, Squeeze, Abs, Round,

Reshape, Retrieve Common Elements, Copy To, Difference, Union, Splitting….

***Describe how to concatenate Arrays?***

Joining means putting contents of two or more arrays in a single array.

In SQL we join tables based on a key, whereas in NumPy we join arrays by axes.

We pass a sequence of arrays that we want to join to the **concatenate() function**, along with the axis. If axis is not explicitly passed, it is taken as 0.

Concatenating 1-D Array:

import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.concatenate((arr1, arr2))

print(arr)

Output: [1 2 3 4 5 6]

Concatenating 2-D Array:

Output: [[1 2 5 6]

[3 4 7 8]]

***Explain Reshaping NumPy Arrays ?***

The **shape of an array** is the number of elements in each dimension.

import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print(arr.shape)

Output:- (2, 4)

**Reshaping** means changing the shape of an array.

The shape of an array is the number of elements in each dimension.

By reshaping we can add or remove dimensions or change number of elements in each dimension.

Converting 1-D to 2-D:

# Converting the 1-D array with 12 elements into a 2-D array.

#The outermost dimension will have 4 arrays, each with 3 elements:

import numpy as np

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

newarr = arr.reshape(4, 3)

print(newarr)

Output:-

[[ 1 2 3]

[ 4 5 6]

[ 7 8 9]

[10 11 12]]

Converting 1-D to 3-D:

#Converting 1-D array with 12 elements into a 3-D array.

#The outermost dimension will have 2 arrays that contains 3 arrays, each with 2 elements:

import numpy as np

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

newarr = arr.reshape(2, 3, 2)

print(newarr)

Output:

[[[ 1 2]

[ 3 4]

[ 5 6]]

[[ 7 8]

[ 9 10]

[11 12]]]

**Flattening array** means converting a multidimensional array into a 1D array.

We can use reshape(-1) to do this.

import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

newarr = arr.reshape(-1)

print(newarr)

Output: [1 2 3 4 5 6]

***Explain Splitting NumPy Array?***

Splitting is reverse operation of Joining.

Joining merges multiple arrays into one and Splitting breaks one array into multiple.

We use array\_split() for splitting arrays, we pass it the array we want to split and the number of splits.

import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array\_split(arr, 3)

print(newarr)

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

**Split into Arrays:**

import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array\_split(arr, 3)

print(newarr[0])

print(newarr[1])

print(newarr[2])

Output:

[1 2]

[3 4]

[5 6]

***Discuss Statistical Operations on Numpy Arrays?***

|  |  |
| --- | --- |
| Functions | Descriptions |
| median() | return the median of an array |
| mean() | return the mean of an array |
| std() | return the standard deviation of an array |
| percentile() | return the nth percentile of elements in an array |
| min() | return the minimum element of an array |
| max() | return the maximum element of an array |

import numpy as np

array1 = np.array([2,6,9,15,17,22,65,1,62])

min\_val = np.min(array1)

max\_val = np.max(array1)

print("Minimum value:", min\_val)

print("Maximum value:", max\_val)

Minimum value: 1

Maximum value: 65

**Data Handling**

***Give an introduction to Python Libraries?***

* Python libraries are collections of pre-written code and functions that extend the capabilities of the Python programming language.
* They provide a wide range of tools and modules for various tasks.
* It makes easier for developers to work on specific tasks.
* Python libraries are commonly used by data scientists and those interested in data science, insights, and data visualizations.

**Few Python libraries:**

**Pandas** : Pandas is a powerful library for data manipulation and analysis.

It provides data structures like DataFrames and Series, which are useful for working with structured data.

**Matplotlib** : Matplotlib is a popular library for creating data visualizations, including line plots, scatter plots, bar charts, and more.

**Seaborn** : Seaborn is built on top of Matplotlib and offers a higher-level interface for creating informative and attractive statistical graphics.

**Scikit-learn** : Scikit-learn is widely used for machine learning tasks.

It includes a wide variety of algorithms for classification, regression, clustering, and more.

**TensorFlow and PyTorch** : These deep learning libraries are used for building and training neural networks and are essential for tasks like image recognition and natural language processing.

**Statsmodels** : Statsmodels provides tools for estimating and interpreting models for various statistical analysis tasks, including regression analysis.

**SciPy** : SciPy is built on top of NumPy and provides additional scientific computing functions, including optimization, integration, and signal processing.

**Jupyter Notebook** : While not a traditional library, Jupyter Notebook is a popular tool for creating and sharing documents that contain live code, equations, visualizations, and narrative text. It's commonly used for data exploration and analysis.

**NumPy :** NumPy is essential for numerical computations in Python. It provides support for arrays and matrices, along with a large collection of mathematical functions to operate on them.

# Importing math library

import math

A = 16

print(math.sqrt(A))

Output: 4.0

***Explain Series in Python?***

A Pandas Series is like a column in a table.

It is a one-dimensional array holding data of any type.

import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)

Output: 0 1

1 7

2 2

**Labels:** label can be used to access a specified value.

The values are labeled with their index number. First value has index 0, second value has index 1 etc.

print(myvar[0])

Output: 2

**Create Labels:** With the index argument, we can name our import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)own labels.

Output: x 1

y 7

z 2

When you have created labels, you can access an item by referring to the label.

print(myvar["y"])

**Key-Value objects as series**

We can also use a key/value object, like a dictionary, when creating a Series.

import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories)

print(myvar)

Output: day1 420

day2 380

day3 390

To select only some of the items in the dictionary, use the index argument and specify only the items you want to include in the Series.

#Create a Series using only data from "day1" and "day2"

import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories, index = ["day1", "day2"])

print(myvar)

Output: day1 420

day2 380

**DataFrames:** Data sets in Pandas are usually multi-dimensional tables, called DataFrames.

Series is like a column, a DataFrame is the whole table.

#Create a DataFrame from two Series

import pandas as pd

data = {

"calories": [420, 380, 390],

"duration": [50, 40, 45]

}

myvar = pd.DataFrame(data)

print(myvar)

Output: calories duration

0 420 50

1 380 40

2 390 45

***Explain DataFrames in Python?***

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

import pandas as pd  
data = {  
  "calories": [420, 380, 390],  
  "duration": [50, 40, 45]  
}  
  
#load data into a DataFrame object:  
df = pd.DataFrame(data)  
print(df)

Output: calories duration

0 420 50

1 380 40

2 390 45

#refer to the row index:  
print(df.loc[0])

The DataFrame is like a table with rows and columns.

Pandas use the loc attribute to return one or more specified row(s)

#refer to the row index:  
print(df.loc[0])

Output:calories 420

duration 50

#use a list of indexes:  
print(df.loc[[0, 1]])

Output: calories duration

0 420 50

1 380 40

**Named Index**: With the index argument, we can name our own indexes.

#Add a list of names to give each row a name

import pandas as pd  
data = {  
  "calories": [420, 380, 390],  
  "duration": [50, 40, 45]  
}  
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])  
print(df)

Output : calories duration

day1 420 50

day2 380 40

day3 390 45

**Locate named Indexes:** Use the named index in the loc attribute to return the specified row(s).

#refer to the named index to return day2  
print(df.loc["day2"])

Output: calories 380

duration 40

Name: day2

***Export and Import data between CSV Files and DataFrames?***

A simple way to store big data sets is to use CSV files (comma separated files).

CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

import pandas as pd

df = pd.read\_csv('data.csv')

print(df.to\_string()) #use to\_string() to print the entire DataFrame.

If we have a large DataFrame with many rows, Pandas will only return the **first 5 rows, and the last 5 rows.**

import pandas as pd  
df = pd.read\_csv('data.csv')  
print(df)  #return the first 5 rows, and the last 5 rows

**Max\_rows:** The number of rows returned is defined in Pandas option settings.

import pandas as pd  
print(pd.options.display.max\_rows)

**Import CSV file in Pandas using csv module:**

One can directly import the csv files using [csv module](https://www.geeksforgeeks.org/how-to-create-multiple-csv-files-from-existing-csv-file-using-pandas/). In this code example the below code reads a CSV file (“nba.csv”) into a Pandas DataFrame using Python’s `csv` and `pandas` modules. It then prints the values in the first column of the DataFrame.

# import the module csv

import csv

import pandas as pd

# open the csv file

with open(r"D:\SDHR\python\_2025\datanew.csv") as csv\_file:

# read the csv file

csv\_reader = csv.reader(csv\_file, delimiter=',')

# now we can use this csv files into the pandas

df = pd.DataFrame([csv\_reader], index=None)

df.head()

# iterating values of first column

for val in list(df[1]):

print(val)

**Loading CSV Data into a NumPy Array:**

import numpy as np

# Specify the path to the CSV file

csv\_file\_path = ' D:\SDHR\python\_2025\datanew.csv'

# Use genfromtxt to read the CSV file into a NumPy array

data\_array = np.genfromtxt('D:\SDHR\python\_2025\datanew.csv', delimiter=',')

# Now, data\_array contains the data from the CSV file

print(data\_array)