**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**RAMAPURAM, CHENNAI**

**COURSE CODE: 18CSS101J**

**COURSE NAME: PROGRAMMING FOR PROBLEM SOLVING**

**UNIT V**

Creating NumPy Array:Numpy Indexing-Numpy Array attributes - Slicing using Numpy -Descriptive Statistics in Numpy: Percentile - Variance in Numpy. Introduction to Pandas - Creating Series Objects, Data Frame Objects - Simple Operations with Data frames -Querying from Data Frames -Applying Functions to Data frames -Comparison between Numpy and Pandas -Speed Testing between Numpy and Pandas -Other Python Libraries

**1.Creating NumPy Array**

NumPy is an open-source Python library that facilitates efficient numerical operations on large quantities of data.

The NumPy's array class is known as ndarray or alias array. The numpy.array is not the same as the standard Python library class array.array. The array.array handles only one-dimensional arrays and provides less functionality.

**Syntax**

numpy.array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)

**Parameters:**

There are the following parameters in numpy.array() function.

1) object: array\_like

Any object, which exposes an array interface whose \_\_array\_\_ method returns any nested sequence or an array.

2) dtype : optional data-type

This parameter is used to define the desired parameter for the array element. If we do not define the data type, then it will determine the type as the minimum type which will require to hold the object in the sequence. This parameter is used only for upcasting the array.

3) copy: bool(optional)

If we set copy equals to true, the object is copied else the copy will be made when an object is a nested sequence, or a copy is needed to satisfy any of the other requirements such as dtype, order, etc.

4) order : {'K', 'A', 'C', 'F'}, optional

The order parameter specifies the memory layout of the array. When the object is not an array, the newly created array will be in C order (row head or row-major) unless 'F' is specified. When F is specified, it will be in Fortran order (column head or column-major). When the object is an array, it holds the following order.

| order | no copy | copy=True |
| --- | --- | --- |
| 'K' | Unchanged | F and C order preserved. |
| 'A' | Unchanged | When the input is F and not C then F order otherwise C order |
| 'C' | C order | C order |
| 'F' | F order | F order |

When copy=False or the copy is made for the other reason, the result will be the same as copy= True with some exceptions for A. The default order is 'K'.

5) subok : bool(optional)

When subok=True, then sub-classes will pass-through; otherwise, the returned array will force to be a base-class array (default).

6) ndmin : int(optional)

This parameter specifies the minimum number of dimensions which the resulting array should have. Users can be prepended to the shape as needed to meet this requirement.

Returns

The numpy.array() method returns an ndarray. The ndarray is an array object which satisfies the specified requirements.

Example 1: numpy.array()

import numpy as np

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

arr

Output:

array([1, 2, 3])

Example 3: More than one dimensions

import numpy as np

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

arr

**Output:**

array([[1., 2., 3.],

, 5., 7.]])

**1.2 Operations performed by numpy array:**

Many operations can be performed on NumPy arrays which makes them very helpful for manipulating data:

Selecting array elements

Slicing arrays

Reshaping arrays

Splitting arrays

Combining arrays

Numerical operations (min, max, mean, etc)

**NumPy Array Indexing**

Access Array Elements

Array indexing is the same as accessing an array element.we 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.

Example[**Get your own Python Server**](https://www.w3schools.com/spaces/)

Get the first element from the following array:

import numpy as np  
arr = np.array([1, 2, 3, 4])  
print(arr[0])

1

Example 2:

Import numpy as np  
arr = np.array([1, 2, 3, 4])  
print(arr[1])

2

**Access 2-D Arrays**

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Think of 2-D arrays like a table with rows and columns, where the dimension represents the row and the index represents the column.

Example

Access the element on the 2nd row, 5th column:

import numpy as np  
  
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])  
  
print('5th element on 2nd row: ', arr[1, 4])

output: 5th element on 2nd dim: 10

**Access 3-D Arrays**

To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

Example

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

**Negative Indexing:**

Use negative indexing to access an array from the end.

Example:

import numpy as np

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

print('Last element from 2nd dim: ', arr[1, -1])

1. **Numpy Array attributes**

We are going to learn about different array attributes in Numpy that are essential to know if you want to transform your arrays.

Array attributes in Numpy

ndarray.shape

ndarray.ndim

ndarray.itemsize

ndarray.T

Array Attributes in Numpy:Array attributes are essential to find out the shape ,dimension ,item size etc.If connected with ndarray object of numpy then we can find about these in detail.

Slicing arrays

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

Example[Get your own Python Server](https://www.w3schools.com/spaces/)

Slice elements from index 1 to index 5 from the following array:

import numpy as np  
  
arr = np.array([1, 2, 3, 4, 5, 6, 7])  
  
print(arr[1:5])

Output:

[2 3 4 5]

**4.1 Descriptive Statistics in Numpy:**

Descriptive Statistics is the building block of data science.

Advanced analytics is often incomplete without analyzing descriptive statistics of the key metrics. In simple terms, descriptive statistics can be defined as the measures that summarize a given data, and these measures can be broken down further into the measures of central tendency and the measures of dispersion.Measures of dispersion are values that describe how the data varies.

It gives us a sense of how much the data tends to diverge from the typical value, while central measures give us an idea about the typical value of the distribution.Measures of central tendency include mean,median, and the mode.On the other hand, the measures of dispersion include standard deviation, variance, and the interquartile range. It will cover the following topics in detail:

**4.2.Percentile in NumPy.**

Percentile is a measure which indicates the value below which a given percentage of points in a dataset fall. For instance, the 35th percentile(\(P\_{35}\)) is the score below which 35% of the data points may be found. IT can observe that median represents the 50th percentile. Similarly, we can have 0th percentile representing the minimum and 100th percentile representing the maximum of all data points.

There are various methods of calculation of quartiles and percentiles, but we will stick to the one below. To calculate \(k^{th}\) percentile(\(P\_{k}\)) for a data set of \(N\) observations which is arranged in increasing order, go through the following steps:

Step 1: Calculate \(\displaystyle i=\frac{k}{100}\times N\)

Step 2: If \(i\) is a whole number, then count the observations in the data set from left to right till we reach the \(i^{th}\) data point. The \(k^{th}\) percentile, in this case, is equal to the average of the value of \(i^{th}\) data point and the value of the data point that follows it.

Step 3: If \(i\) is not a whole number, then round it up to the nearest integer and count the observations in the data set from left to right till we reach the \(i^{th}\) data point. The \(k^{th}\) percentile now is just equal to the value corresponding this data point.

Example:

Suppose we want to calculate \(P\_{27}\) for the marks of students in Subject 2. Let us first arrange the data in increasing order which results in the following dataset {8,9,12,14.5,15.5,17,18}.

**Following the steps above,**

step 1: \(\displaystyle i=\frac{27}{100}\times 7 = 1.89\)

Step 2: Not applicable here as 1.89 is not a whole number, so let us move to step 3

Step 3: Rounding up 1.89 gives 2, hence 27th percentile is the value of second observation, i.e., 9

Therefore, 9 is \(27^{th}\) percentile which means that 27% of the students have scored below 9.

**4.3 Variance in Numpy.**

Variance is another measure of dispersion.It is the square of the standard deviation and the covariance of the random variable with itself.

numpy.var(a, axis=None, dtype=None, ddof=0)

Parameters are the same as numpy.mean except.

ddof : int, optional(ddof stands for delta degrees of freedom. It is the divisor used in the calculation, which is N – ddof, where N is the number of elements. The default value of ddof is 0

>>> import numpy as np

>>> A=np.array([[10,14,11,7,9.5,15,19],[8,9,17,14.5,12,18,15.5],

 [15,7.5,11.5,10,10.5,7,11],[11.5,11,9,12,14,12,7.5]])

>>> B=A.T

>>> a = np.var(B,axis=0)

>>> b = np.var(B,axis=1)

>>> print(a) [ 13.98979592 12.8877551 6.12244898 3.92857143]

>>> print(b) [ 6.546875 5.921875 8.796875 7.546875 2.875 16.5 19.0625 ]

**Importance of Variance:**

It is an important measure in descriptive statistics because it allows us to measure the spread of a data set around its mean. The observations may or may not be meaningful if observations in data sets are highly spread.

**Limitations of descriptive statistics**

Descriptive statistics measures are limited in the way that we can only make the summary about the people or objects that are actually measured. The data cannot be used to generalize to other people or objects.

For example, if we have recorded the marks of the students for the past few years and would want to predict the marks for next exam, we cannot do that only relying on descriptive statistics; inferential statistics would help. Descriptive statistics can often be difficult when we are dealing with a large dataset.

**5.Introduction to Pandas**

Pandas is an open-source library in Python that is made mainly for working with relational or labeled data . It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the [NumPy](https://www.geeksforgeeks.org/python-numpy/)library of Python. Pandas is fast and it has high performance & productivity for users.

**5.1Creating Series Object**

[Pandas Series](https://www.geeksforgeeks.org/python-pandas-series/) is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called indexes.  
Pandas Series is nothing but a column in an Excel sheet. Labels need not be unique but must be a hashable type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.

a Pandas Series will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, or an Excel file. Pandas Series can be created from lists, dictionaries, and from scalar values, etc.

import pandas as pd

import numpy as np

 # Creating empty series

ser = pd.Series()

print("Pandas Series: ", ser)

# simple array

data = np.array(['g', 'e', 'e', 'k', 's'])

ser = pd.Series(data)

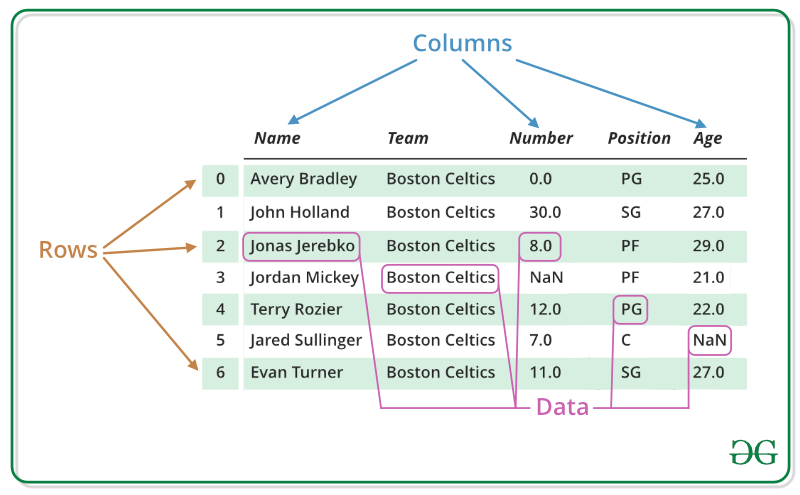
print("Pandas Series:\n", ser)

Output:

Pandas Series: Series([], dtype: float64)  
Pandas Series:  
0 g  
1 e  
2 e  
3 k  
4 s  
dtype: object

**5.2Data Frame Objects**

Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.



We will get a brief insight on all these basic operation which can be performed on Pandas.

DataFrame :

[Creating a DataFrame](https://www.geeksforgeeks.org/python-pandas-dataframe/#Basics)

[Dealing with Rows and Columns](https://www.geeksforgeeks.org/python-pandas-dataframe/#Basics1)

[Indexing and Selecting Data](https://www.geeksforgeeks.org/python-pandas-dataframe/#Basics2)

[Working with Missing Data](https://www.geeksforgeeks.org/python-pandas-dataframe/#Basics3)

[Iterating over rows and columns](https://www.geeksforgeeks.org/python-pandas-dataframe/#Basics4)

**Creating a Pandas DataFrame:**

a Pandas DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file. Pandas DataFrame can be created from the lists, dictionary, and from a list of dictionary etc. Dataframe can be created in different ways here are some ways by which we create a dataframe:

[Creating a dataframe using List](https://www.geeksforgeeks.org/create-a-pandas-dataframe-from-lists/): DataFrame can be created using a single list or a list of lists.

# import pandas as pd

import pandas as pd

 # list of strings

lst = ['Geeks', 'For', 'Geeks', 'is',

            'portal', 'for', 'Geeks']

# Calling DataFrame constructor on list

df = pd.DataFrame(lst)

print(df)

Output:  

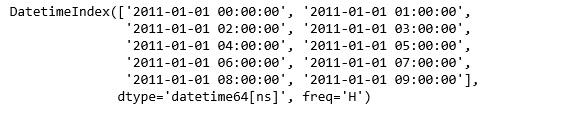

**5.3 Simple Operations with Data frames:**

Pandas Working with Dates and Times

 Using the NumPy datetime64 and timedelta64 dtypes, we have consolidated a large number of features from other Python libraries like scikits.timeseries as well as created a tremendous amount of new functionality for manipulating time series data.

Example #1: Create a dates dataframe

| import pandas as pd    # Create dates dataframe with frequency  data = pd.date\_range('1/1/2011', periods = 10, freq ='H')    data |
| --- |

Output:  


Example #2: Create range of dates and show basic features

| # Create date and time with dataframe  data = pd.date\_range('1/1/2011', periods = 10, freq ='H')    x = datetime.now()  x.month, x.year |
| --- |

Output:

(9,2018)

**5.3.Querying from Data Frame**

Definition and Usage

The query() method allows you to query the DataFrame.

The query() method takes a query expression as a string parameter, which has to evaluate to either True of False.It returns the DataFrame where the result is True according to the query expression.

**Syntax**

dataframe.query(expr, inplace)

Parameters

The inplace paramater is a [keyword argument](https://www.w3schools.com/python/gloss_python_function_keyword_arguments.asp).

| Parameter | Values | Description |
| --- | --- | --- |
| expr |  | Required. A string that represents a query expression. |
| inplace | True|False | Optional. A boolean value that specifies if the query() method should leave the original DataFrame untouched and return a copy (inplace = False). This is Default. Or: Make the changes in the original DataFrame (inplace = True) |

5.4 Return Value

A [DataFrame](https://www.w3schools.com/python/pandas/pandas_dataframes.asp) with the new result, or None if the changes were made in the original DataFrame (inplace = True)

import pandas as pd

data = {

  "name": ["Sally", "Mary", "John"],

  "age": [50, 40, 30]

}

df = pd.DataFrame(data)

print(df.query('age > 35'))

Output:

name  age

  0   Sally     50

  1   Mary    40

**5.5 Applying Functions to Data frames :**

Pandas DataFrame apply() function is used to apply a function along an axis of the DataFrame. The function syntax is:

def apply(

self,

func,

axis=0,

broadcast=None,

raw=False,

reduce=None,

result\_type=None,

args=(),

\*\*kwds

)

The important parameters are:

func: The function to apply to each row or column of the DataFrame.

axis: axis along which the function is applied. The possible values are {0 or ‘index’, 1 or ‘columns’}, default 0.

args: The positional arguments to pass to the function. This is helpful when we have to pass additional arguments to the function.

keyword: additional keyword arguments to pass to the function. This is helpful when we have to pass additional keyword arguments to the function.

1. [**pandas dataframe apply() examples**](https://www.digitalocean.com/community/tutorials/pandas-dataframe-apply-examples#pandas-dataframe-apply-examples)**:**

Let’s look at some examples of using apply() function on a DataFrame object.

[1. Applying a Function to DataFrame Elements](https://www.digitalocean.com/community/tutorials/pandas-dataframe-apply-examples#1-applying-a-function-to-dataframe-elements)

import pandas as pd

df = pd.DataFrame({'A': [1, 2], 'B': [10, 20]})

def square(x):

return x \* x

df1 = df.apply(square)

print(df)

print(df1)

Output:

A B

0 1 10

1 2 20

A B

0 1 100

1 4 400

The DataFrame on which apply() function is called remains unchanged. The apply() function returns a new DataFrame object after applying the function to its elements.

1. **Comparison between Numpy and Pandas:**

[Numpy](https://www.geeksforgeeks.org/python-numpy/) is the fundamental library of python, used to perform scientific computing. It provides high-performance multidimensional arrays and tools to deal with them. A numpy array is a grid of values (of the same type) that are indexed by a tuple of positive integers, numpy arrays are fast, easy to understand, and give users the right to perform calculations across arrays.

| # Importing Numpy package  import numpy as np    # Creating a 3-D numpy array using np.array()  org\_array = np.array([[23, 46, 85],                        [43, 56, 99],                        [11, 34, 55]])    # Printing the Numpy array  print(org\_array) |
| --- |

Output:

[[23 46 85]  
 [43 56 99]  
 [11 34 55]]

**PANDAS**

[Pandas](https://www.geeksforgeeks.org/pandas-tutorial/) is an open-source, BSD-licensed library written in Python Language. Pandas provide high performance, fast, easy-to-use data structures, and data analysis tools for manipulating numeric data and time series. Pandas is built on the numpy library and written in languages like Python, Cython, and C. In pandas, we can import data from various file formats like JSON, SQL, Microsoft Excel, etc.

| # Importing pandas library  import pandas as pd    # Creating and initializing a nested list  age = [['Aman', 95.5, "Male"], ['Sunny', 65.7, "Female"],         ['Monty', 85.1, "Male"], ['toni', 75.4, "Male"]]    # Creating a pandas dataframe  df = pd.DataFrame(age, columns=['Name', 'Marks', 'Gender'])    # Printing dataframe  df |
| --- |

Output:

Name Marks Gender  
0 Aman 95.5 Male  
1 Sunny 65.7 Female  
2 Monty 85.1 Male  
3 toni 75.4 Male

**8.Comparision between Pandas and Numpy:**

Table of Differences Between Pandas and NumPy is as follows:

| PANDAS | NUMPY |
| --- | --- |
| When we have to work on Tabular data, we prefer the pandas module. | When we have to work on Numerical data, we prefer the numpy module. |
| The powerful tools of pandas are Data frame and Series. | Whereas the powerful tool of numpy is Arrays. |
| Pandas consume more memory. | Numpy is memory efficient. |
| Pandas have a better performance when the number of rows is 500K or more. | Numpy has a better performance when number of rows is 50K or less. |
| Indexing of the pandas series is very slow as compared to numpy arrays. | Indexing of numpy arrays is very fast. |
| Pandas have 2d table object called DataFrame. | Numpy is capable of providing multi-dimensional arrays. |
| It was developed by Wes McKinney released in 2008. | It was developed by Travis Oliphant and was released in 2005. |
| It is used in a lot of organizations like Kaidee, Trivago, Abeja Inc. , and a lot more. | It is being used in organizations like Walmart Tokopedia, Instacart, and many more. |
| It has a higher industry application. | It has a lower industry application. |

**9.Speed Testing between Numpy and Pandas :**

Pandas and NumPy are both essential tools in Python.Numpy runs vector and matrix operations very efficiently, while Pandas provides the R-like data frames allowing intuitive tabular data analysis. Numpy is more optimized for arithmetic computations.Pandas has a better performance when a number of rows is 500K or more. NumPy has a better performance when number of rows is 50K or less. Indexing of the Pandas series is very slow as compared to NumPy arrays. Indexing of NumPy Arrays is very fast.

To Understand the speed test comparison between NumPy and Pandas, Lets take an example of, indexing on Pandas Series objects and NumPy .

Example:

In [1]: import numpy as np

In [2]: import pandas as pd

In [3]: np.version.version

Out[3]: '1.8.2‘

 In [4]: pd.version.version

Out[4]: '0.14.1'

In [5]: a = np.arange(100)

In [6]: aa = np.arange(100, 200)

In [7]: s = pd.Series(a)

In [8]: ss = pd.Series(aa)

In [9]: i = np.random.choice(a, size=10)

Performance Comparison of  NumPy and Pandas,

In [10]: %timeit a[i]

1000000 loops, best of 3: 998 ns per loop

In [11]: %timeit s[i]

10000 loops, best of 3: 168 µs per loop

Indexing the array is over 100 times faster than indexing the Series.

In [12]: %timeit a \* aa 1000000 loops, best of 3: 1.21 µs per loop

In [13]: %timeit s \* ss 10000 loops, best of 3: 88.5 µs per loop

Why is Pandas so much slower than NumPy?

Pandas is doing a lot of stuff when you index into a Series, and it’s doing that stuff in Python.As an illustration, here’s a visualization made by profiling s[i]:Refer the below picture.Each colored arc is a different function call in Python. There are about 100 calls there.

Pandas is fast enough most of the time, and you get the benefit of Pandas’ sophisticated indexing features. It’s only in loops that the microseconds start to add up to minutes

**10.Other Python Libraries :**

A Python library is a collection of related modules. It contains bundles of code that can be used repeatedly in different programs. It makes Python Programming simpler and convenient for the programmer. As we don’t need to write the same code again and again for different programs. Python libraries play a very vital role in fields of Machine Learning, Data Science, Data Visualization, etc.

Python standard library

The Python Standard Library contains the exact syntax, semantics, and tokens of Python. It contains built-in modules that provide access to basic system functionality like I/O and some other core modules. Most of the Python Libraries are written in the C programming language. The Python standard library consists of more than 200 core modules. All these work together to make Python a high-level programming language. Python Standard Library plays a very important role. Without it, the programmers can’t have access to the functionalities of Python. But other than this, there are several other libraries in Python that make a programmer’s life easier. Let’s have a look at some of the commonly used libraries:

TensorFlow: This library was developed by Google in collaboration with the Brain Team. It is an open-source library used for high-level computations. It is also used in machine learning and deep learning algorithms. It contains a large number of tensor operations. Researchers also use this Python library to solve complex computations in Mathematics and Physics.

Matplotlib: This library is responsible for plotting numerical data. And that’s why it is used in data analysis. It is also an open-source library and plots high-defined figures like pie charts, histograms, scatterplots, graphs, etc.

Pandas: Pandas are an important library for data scientists. It is an open-source machine learning library that provides flexible high-level data structures and a variety of analysis tools. It eases data analysis, data manipulation, and cleaning of data. Pandas support operations like Sorting, Re-indexing, Iteration, Concatenation, Conversion of data, Visualizations, Aggregations, etc.

Numpy: The name “Numpy” stands for “Numerical Python”. It is the commonly used library. It is a popular machine learning library that supports large matrices and multi-dimensional data. It consists of in-built mathematical functions for easy computations. Even libraries like TensorFlow use Numpy internally to perform several operations on tensors. Array Interface is one of the key features of this library.

SciPy: The name “SciPy” stands for “Scientific Python”. It is an open-source library used for high-level scientific computations. This library is built over an extension of Numpy. It works with Numpy to handle complex computations. While Numpy allows sorting and indexing of array data, the numerical data code is stored in SciPy. It is also widely used by application developers and engineers.

Scrapy: It is an open-source library that is used for extracting data from websites. It provides very fast web crawling and high-level screen scraping. It can also be used for data mining and automated testing of data.

Scikit-learn: It is a famous Python library to work with complex data. Scikit-learn is an open-source library that supports machine learning. It supports variously supervised and unsupervised algorithms like linear regression, classification, clustering, etc. This library works in association with Numpy and SciPy.

PyGame: This library provides an easy interface to the Standard Directmedia Library (SDL) platform-independent graphics, audio, and input libraries. It is used for developing video games using computer graphics and audio libraries along with Python programming language.

PyTorch: PyTorch is the largest machine learning library that optimizes tensor computations. It has rich APIs to perform tensor computations with strong GPU acceleration. It also helps to solve application issues related to neural networks.

PyBrain: The name “PyBrain” stands for Python Based Reinforcement Learning, Artificial Intelligence, and Neural Networks library. It is an open-source library built for beginners in the field of Machine Learning. It provides fast and easy-to-use algorithms for machine learning tasks. It is so flexible and easily understandable and that’s why is really helpful for developers that are new in research fields.

There are many more libraries in Python. We can use a suitable library for our purposes. Hence, Python libraries play a very crucial role and are very helpful to the developers.

Use of Libraries in Python Program

As we write large-size programs in Python, we want to maintain the code’s modularity. For the easy maintenance of the code, we split the code into different parts and It can use that code later ever we need it. In Python, modules play that part. Instead of using the same code in different programs and making the code complex, it define mostly used functions in modules and it can just simply import them in a program wherever there is a requirement. We don’t need to write that code but still, it can use its functionality by importing its module. Multiple interrelated modules are stored in a library. And whenever it need to use a module,it import it from its library. .