

DATE : 19 june 2024

DAY : Wednesday

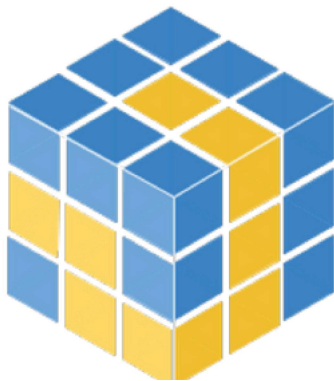
TOPICS : Library: NumPy

In Python, a library refers to a collection of modules (Python files) that contain functions, classes, and variables for a specific purpose. Libraries provide reusable functionalities that can be imported and used in various Python programs, enabling developers to avoid reinventing the wheel and focus on solving specific problems efficiently.

Here are key points about libraries in Python:

1. **Purpose:** Libraries serve to extend the capabilities of Python by providing pre-written code for common tasks, such as working with databases, handling network communication, performing mathematical computations, generating graphical user interfaces (GUIs), etc.
2. **Components:** A library typically consists of multiple modules, each addressing a specific aspect of the library's functionality. Modules within a library are organized logically based on their purpose.
3. **Importing:** To use a library in Python, you import it into your code using the `import` statement. For example, `import math` imports the `math` library, allowing access to its functions like `math.sqrt()` or constants like `math.pi`.
4. **Examples of Libraries:** Python has a vast ecosystem of libraries catering to different needs:
 - **Standard Library:** Comes with Python installation and includes modules for basic functionalities like `math`, `datetime`, `os`, `sys`, etc.
 - **Third-Party Libraries:** Developed by the Python community and external contributors, such as `numpy` for numerical computations, `requests` for HTTP requests, `matplotlib` for plotting, etc.
5. **Benefits:** Using libraries offers several advantages:
 - Saves development time by leveraging tested and optimized code.
 - Promotes code reuse and modularity.
 - Enhances functionality beyond Python's core capabilities.
 - Facilitates collaboration and community-driven development.
6. **Installation:** While many libraries are included with Python by default, others need to be installed separately. This is typically done using package managers like `pip`, which retrieves and installs libraries from the Python Package Index (PyPI) or other repositories.

In essence, libraries in Python are essential tools that empower developers to build robust applications efficiently by providing specialized functionality and reducing the complexity of coding from scratch.



NumPy

Introduction to NumPy

NumPy (Numerical Python) is a powerful, open-source library in Python used for numerical computations. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. NumPy is fundamental for scientific computing in Python and serves as the foundation for many other libraries like SciPy, Pandas, and Matplotlib.

Key Features of NumPy

1. **N-Dimensional Arrays:**
 - The core feature of NumPy is its `ndarray` object, which is a multi-dimensional array of elements, typically of the same type. These arrays allow for efficient storage and manipulation of large datasets.

2. Mathematical Functions:

- NumPy provides a wide array of mathematical functions for operations such as trigonometry, statistics, linear algebra, and more, all optimized for performance.

3. Broadcasting:

- Broadcasting allows NumPy to perform operations on arrays of different shapes without needing to copy data. It simplifies the implementation of mathematical operations.

4. Integration with C/C++ and Fortran:

- NumPy can interface with code written in C, C++, or Fortran, allowing for high-performance numerical computation.

5. Linear Algebra:

- NumPy includes functions for linear algebra operations such as matrix multiplication, eigenvalue decomposition, and singular value decomposition.

6. Random Number Generation:

- The library provides tools for generating random numbers, including random sampling from different probability distributions.

Practical Applications of NumPy

- **Data Analysis:** Efficiently handle and manipulate large datasets, performing complex calculations with ease.
- **Scientific Computing:** Solve problems in physics, chemistry, biology, engineering, and other fields requiring numerical methods.
- **Artificial Intelligence, Machine Learning & Data Science:** Form the backbone of many machine learning algorithms and frameworks, providing fast computation of matrix operations and statistical functions.
- **Image Processing:** Manipulate and analyze images, which are represented as arrays of pixel values.

✓ Let's get started

```
import numpy as np
```

✓ Creating an Array

```
l1 = [1, 2, 3, 4]
l2 = [5, 6, 7, 8]
```

```
type(l1)
```

```
list
```

```
arr1 = np.array(l1)
arr1
```

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

```
arr1.shape
```

```
(4,)
```

```
type(arr1)
```

```
numpy.ndarray
```

```
arr2 = np.array([l1, l2])
arr2
```

```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

```
arr2.shape
```

```
np.arange(1, 26)
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25])
```

Reshaping Arrays

```
np.arange(100).reshape(20, 5)
```

⇒ `array([[0, 1, 2, 3, 4],
 [5, 6, 7, 8, 9],
 [10, 11, 12, 13, 14],`

```
[15, 16, 17, 18, 19],
[20, 21, 22, 23, 24],
[25, 26, 27, 28, 29],
[30, 31, 32, 33, 34],
[35, 36, 37, 38, 39],
[40, 41, 42, 43, 44],
[45, 46, 47, 48, 49],
[50, 51, 52, 53, 54],
[55, 56, 57, 58, 59],
[60, 61, 62, 63, 64],
[65, 66, 67, 68, 69],
[70, 71, 72, 73, 74],
[75, 76, 77, 78, 79],
[80, 81, 82, 83, 84],
[85, 86, 87, 88, 89],
[90, 91, 92, 93, 94],
[95, 96, 97, 98, 99]])
```

Indexing

```
random_arr = np.random.randint(1, 100, size=(10, 10))
random_arr
```

```
array([[23,  6, 56, 99, 90, 94, 79, 42, 99, 25],
       [16, 34, 90, 28, 90, 23, 58, 23, 77, 43],
       [71, 10, 75, 15,  9, 93, 74, 83, 13, 50],
       [65, 23, 58, 16, 20, 83,  3, 41, 75, 18],
       [13, 37, 66, 31, 26, 90,  6, 65, 94, 10],
       [91, 49, 91, 37, 20, 78, 73, 13, 32, 56],
       [59, 17, 97, 11, 22, 86,  8, 64, 98, 88],
       [11, 66, 62,  6, 64, 10, 91, 86, 91, 60],
       [14, 76, 45, 99, 33, 35,  8,  8, 31, 78],
       [12, 71, 85, 14, 15, 35, 99, 33,  3, 43]])
```

```
arr1
```

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

```
arr1[2]
```

```
3
```

```
random_arr[5]
```

```
array([91, 49, 91, 37, 20, 78, 73, 13, 32, 56])
```

```
random_arr[5][2]
```

```
91
```

```
random_arr[3:8]
```

```
array([[65, 23, 58, 16, 20, 83,  3, 41, 75, 18],
       [13, 37, 66, 31, 26, 90,  6, 65, 94, 10],
       [91, 49, 91, 37, 20, 78, 73, 13, 32, 56],
       [59, 17, 97, 11, 22, 86,  8, 64, 98, 88],
       [11, 66, 62,  6, 64, 10, 91, 86, 91, 60]])
```

Sorting Arrays

```
unsorted_arr = np.array([23, 10, 25, 45, 500, 200, 50])
unsorted_arr
```

```
array([ 23,  10,  25,  45, 500, 200,  50])
```

```
sorted(unsorted_arr)
```

```
[10, 23, 25, 45, 50, 200, 500]
```

```
unsorted_arr
```

```
→ array([ 23,  10,  25,  45, 500, 200,  50])
```

```
unsorted_arr.sort()
```

```
unsorted_arr
```

```
→ array([ 10,  23,  25,  45,  50, 200, 500])
```

✓ Slicing vs Copying

```
unsorted_arr
```

```
→ array([ 10,  23,  25,  45,  50, 200, 500])
```

```
sliced_arr = unsorted_arr[2:6]  
sliced_arr
```

```
→ array([ 25,  45,  50, 200])
```

```
sliced_arr[0] = 1000
```

```
sliced_arr
```

```
→ array([1000,  45,  50, 200])
```

```
unsorted_arr
```

```
→ array([ 10,  23, 1000,  45,  50, 200, 500])
```

```
new_arr = unsorted_arr.copy()  
new_arr
```

```
→ array([ 10,  23, 1000,  45,  50, 200, 500])
```

```
new_arr[0] = -100  
new_arr
```

```
→ array([-100,  23, 1000,  45,  50, 200, 500])
```

```
unsorted_arr
```

```
→ array([ 10,  23, 1000,  45,  50, 200, 500])
```

✓ Broadcasting

```
arr = np.arange(10)  
arr
```

```
→ array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
arr[4:8] = 100
```

```
arr
```

```
→ array([ 0,  1,  2,  3, 100, 100, 100, 100,  8,  9])
```

```
arr * 10
```

```
→ array([ 0, 10, 20, 30, 1000, 1000, 1000, 1000, 80, 90])
```

✓ Conditional Statements

```
arr
↳ array([ 0,  1,  2,  3, 100, 100, 100, 100,  8,  9])

arr % 2 == 0
↳ array([ True, False,  True, False,  True,  True,  True,  True,  True,
        False])

arr[arr % 2 == 0]
↳ array([ 0,  2, 100, 100, 100, 100,  8])

(arr % 2 == 0) & (arr > 10)
↳ array([False, False, False, False,  True,  True,  True,  True, False,
        False])

arr[(arr % 2 == 0) & (arr > 10)]
↳ array([100, 100, 100, 100])
```

✓ Aggregate Functions

```
arr = np.random.randint(1, 100, size=10)
arr
↳ array([77, 39, 50, 28, 51, 55, 23, 28, 28, 74])

arr.min()
↳ 23

arr.max()
↳ 77

arr.argmin()
↳ 6

arr.argmax()
↳ 0

arr.sum()
↳ 453

np.sqrt(arr)
↳ array([8.77496439, 6.244998 , 7.07106781, 5.29150262, 7.14142843,
        7.41619849, 4.79583152, 5.29150262, 5.29150262, 8.60232527])

np.sin(arr)
↳ array([ 0.99952016,  0.96379539, -0.26237485,  0.27090579,  0.67022918,
        -0.99975517, -0.8462204 ,  0.27090579,  0.27090579, -0.98514626])

import numpy as np
id1 = np.identity(4, dtype = int)
id1
```

```
array([[1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1]])
```

```
mat2 = np.linspace(10,20, 100)
mat2
```

```
array([10.          , 10.1010101 , 10.2020202 , 10.3030303 , 10.4040404 ,
       10.50505051, 10.60606061, 10.70707071, 10.80808081, 10.90909091,
       11.01010101, 11.11111111, 11.21212121, 11.31313131, 11.41414141,
       11.51515152, 11.61616162, 11.71717172, 11.81818182, 11.91919192,
       12.02020202, 12.12121212, 12.22222222, 12.32323232, 12.42424242,
       12.52525253, 12.62626263, 12.72727273, 12.82828283, 12.92929293,
       13.03030303, 13.13131313, 13.23232323, 13.33333333, 13.43434343,
       13.53535354, 13.63636364, 13.73737374, 13.83838384, 13.93939394,
       14.04040404, 14.14141414, 14.24242424, 14.34343434, 14.44444444,
       14.54545455, 14.64646465, 14.74747475, 14.84848485, 14.94949495,
       15.05050505, 15.15151515, 15.25252525, 15.35353535, 15.45454545,
       15.55555556, 15.65656566, 15.75757576, 15.85858586, 15.95959596,
       16.06060606, 16.16161616, 16.26262626, 16.36363636, 16.46464646,
       16.56565657, 16.66666667, 16.76767677, 16.86868687, 16.96969697,
       17.07070707, 17.17171717, 17.27272727, 17.37373737, 17.47474747,
       17.57575758, 17.67676768, 17.77777778, 17.87878788, 17.97979798,
       18.08080808, 18.18181818, 18.28282828, 18.38383838, 18.48484848,
       18.58585859, 18.68686869, 18.78787879, 18.88888889, 18.98989899,
       19.09090909, 19.19191919, 19.29292929, 19.39393939, 19.49494949,
       19.5959596 , 19.6969697 , 19.7979798 , 19.8989899 , 20.          ])
```

NumPy Exercises

Import NumPy as np

```
import numpy as np
```

Create an array of 10 zeros

```
np.zeros(10, dtype = int)
```

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

```
np.zeros(10)
```

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

Create an array of 10 ones

```
np.ones(10, dtype = int)
```

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

```
np.ones(10)
```

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

Create an array of 10 fives

```
np.full(10,5)
```

```
array([5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
```

```
np.full(10,5,dtype=float)
```

```
array([5., 5., 5., 5., 5., 5., 5., 5., 5., 5.])
```

▼ Create an array of the integers from 10 to 50

```
np.arange(10,51,dtype = float)
```

```
↵ array([10., 11., 12., 13., 14., 15., 16., 17., 18., 19., 20., 21., 22.,
        23., 24., 25., 26., 27., 28., 29., 30., 31., 32., 33., 34., 35.,
        36., 37., 38., 39., 40., 41., 42., 43., 44., 45., 46., 47., 48.,
        49., 50.]
```

```
np.arange(10,51)
```

```
↵ array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
        27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43,
        44, 45, 46, 47, 48, 49, 50])
```

▼ Create an array of all the even integers from 10 to 50

```
arr = np.arange(10,51,dtype=float)
arr[arr % 2 ==0]
```

```
↵ array([10., 12., 14., 16., 18., 20., 22., 24., 26., 28., 30., 32., 34.,
        36., 38., 40., 42., 44., 46., 48., 50.]
```

```
arr = np.arange(10,51)
arr[arr % 2 ==0]
```

```
↵ array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,
        44, 46, 48, 50])
```

▼ Create a 3x3 matrix with values ranging from 0 to 8

```
np.arange(9,dtype=float).reshape(3,3)
```

```
↵ array([[0., 1., 2.],
        [3., 4., 5.],
        [6., 7., 8.]])
```

```
np.arange(9).reshape(3,3)
```

```
↵ array([[0, 1, 2],
        [3, 4, 5],
        [6, 7, 8]])
```

▼ Create a 3x3 identity matrix

```
np.identity(3, dtype = int)
```

```
↵ array([[1, 0, 0],
        [0, 1, 0],
        [0, 0, 1]])
```

```
np.identity(3)
```

```
↵ array([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]])
```

▼ Use NumPy to generate a random number between 0 and 1

```
random_arr = np.random.randint(0,1,1)
random_arr
```

```
↵ array([0])
```



```
random_arr = np.random.rand()
random_arr
```

0.37107024791712584

✓ Create the following matrix:

```
array([[ 0.01,  0.02,  0.03,  0.04,  0.05,  0.06,  0.07,  0.08,  0.09,  0.1 ],
       [ 0.11,  0.12,  0.13,  0.14,  0.15,  0.16,  0.17,  0.18,  0.19,  0.2 ],
       [ 0.21,  0.22,  0.23,  0.24,  0.25,  0.26,  0.27,  0.28,  0.29,  0.3 ],
       [ 0.31,  0.32,  0.33,  0.34,  0.35,  0.36,  0.37,  0.38,  0.39,  0.4 ],
       [ 0.41,  0.42,  0.43,  0.44,  0.45,  0.46,  0.47,  0.48,  0.49,  0.5 ],
       [ 0.51,  0.52,  0.53,  0.54,  0.55,  0.56,  0.57,  0.58,  0.59,  0.6 ],
       [ 0.61,  0.62,  0.63,  0.64,  0.65,  0.66,  0.67,  0.68,  0.69,  0.7 ],
       [ 0.71,  0.72,  0.73,  0.74,  0.75,  0.76,  0.77,  0.78,  0.79,  0.8 ],
       [ 0.81,  0.82,  0.83,  0.84,  0.85,  0.86,  0.87,  0.88,  0.89,  0.9 ],
       [ 0.91,  0.92,  0.93,  0.94,  0.95,  0.96,  0.97,  0.98,  0.99,  1. ]])
```

array([[0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1],
 [0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2],
 [0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3],
 [0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4],
 [0.41, 0.42, 0.43, 0.44, 0.45, 0.46, 0.47, 0.48, 0.49, 0.5],
 [0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.57, 0.58, 0.59, 0.6],
 [0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67, 0.68, 0.69, 0.7],
 [0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.8],
 [0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9],
 [0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1.]])

```
np.linspace(0.01,1,100).reshape(10,10)
```

array([[0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1],
 [0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2],
 [0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3],
 [0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4],
 [0.41, 0.42, 0.43, 0.44, 0.45, 0.46, 0.47, 0.48, 0.49, 0.5],
 [0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.57, 0.58, 0.59, 0.6],
 [0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67, 0.68, 0.69, 0.7],
 [0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.8],
 [0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9],
 [0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1.]])

✓ Create an array of 20 linearly spaced points between 0 and 1:

(Hint: Use linspace function)

```
np.linspace(0,1,20)
```

array([0. , 0.05263158, 0.10526316, 0.15789474, 0.21052632,
 0.26315789, 0.31578947, 0.36842105, 0.42105263, 0.47368421,
 0.52631579, 0.57894737, 0.63157895, 0.68421053, 0.73684211,
 0.78947368, 0.84210526, 0.89473684, 0.94736842, 1.])

✓ Numpy Indexing and Selection

```
mat = np.arange(1,26).reshape(5,5)
mat
```

array([[1, 2, 3, 4, 5],
 [6, 7, 8, 9, 10],
 [11, 12, 13, 14, 15],
 [16, 17, 18, 19, 20],
 [21, 22, 23, 24, 25]])

You are given this matrix named mat. Write some code to get the outputs accordingly in the cells given below


```
#Enter your code here
mat[3][4]
```

20


Start coding or [generate](#) with AI.

 20


#Enter your code here
 np.linspace(2,12,3,dtype=int).reshape(3,1)

 array([[2],
 [7],
 [12]])


Start coding or [generate](#) with AI.

 array([[2],
 [7],
 [12]])


#Enter your code here
 mat[4]

 array([21, 22, 23, 24, 25])


Start coding or [generate](#) with AI.

 array([21, 22, 23, 24, 25])

#Enter your code here
 mat[3:5]

 array([[16, 17, 18, 19, 20],
 [21, 22, 23, 24, 25]])

Start coding or [generate](#) with AI.

 array([[16, 17, 18, 19, 20],
 [21, 22, 23, 24, 25]])

✓ Get the sum of all the values in mat

mat.sum()


 325

Start coding or [generate](#) with AI.


 325

✓ Get the standard deviation of the values in mat

mat.std()


 7.211102550927978

Start coding or [generate](#) with AI.


 7.211102550927978

✓ Get the sum of all the columns in mat

mat.sum(axis= 0)

 array([55, 60, 65, 70, 75])

Start coding or [generate](#) with AI.

 `array([55, 60, 65, 70, 75])`