# Numpy

In [3]:
```
import numpy as np
```

1. Create an array with zeros and ones and print the output

In [5]:
```
print(np.zeros(3,dtype=np.int32))
print(np.ones(4,dtype=np.int64))
```

```
[0 0 0]
[1 1 1 1]
```

2.Create an array and print the output

In [6]:
```
arr=np.array([44,3,45,87,43])
print(arr)
```

```
[44  3 45 87 43]
```

3. Create an array whose initial content is random and print the output

In [8]:
```
rand=np.empty(5)
print(rand)
```

```
[2.12199579e-314 9.79101081e-307 5.25685847e-321 6.95208261e-310
 2.55894528e-307]
```

4. Create an array with the range of values with even intervals

In [9]:
```
even=np.arange(0,50,2)
print(even)
```

```
[ 0  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46
 48]
```

5. create an array with values that are spaced linearly in a specified interval

In [14]:
```
arr2=np.linspace(0,50,num=22,dtype=np.int64)
print(arr2)
```

```
[ 0  2  4  7  9 11 14 16 19 21 23 26 28 30 33 35 38 40 42 45 47 50]
```

6. Access and manipulate elements in the array

In [15]:
```
print(arr[0:3])
arr[4]=99
print(arr)
```

```
[44  3 45]
[44  3 45 87 99]
```

7. Create a 2-dimensional array and check the shape of the array

In [22]:
```
arr2d=np.array([[1,2,3],
                [36,7,8]])
print(arr2d.shape)
```

```
(2, 3)
```

8. Using the arange() and linspace() function to evenly space values in a specified interval

In [23]:
```
b=np.arange(0,10,2)
c=np.linspace(0,10,5,dtype=np.int32)
print(b)
print(c)
```

```
[0 2 4 6 8]
[ 0  2  5  7 10]
```

9. Create an array of random values between 0 and 1 in a given shape

In [40]:
```
arr3=np.linspace(0,1,5)
print(arr3)
```

```
[0.   0.25 0.5  0.75 1.  ]
```

10. Repeat each element of an array by a specified number of times using repeat() and tile() functions

In [41]:
```
arr4=np.repeat(b,3)
print(arr4)
```

```
[0 0 0 2 2 2 4 4 4 6 6 6 8 8 8]
```

In [43]:
```
arr5=np.tile(b,3)
print(arr5)
```

```
[0 2 4 6 8 0 2 4 6 8 0 2 4 6 8]
```

11. How do you know the shape and size of an array?

Using shape() and size()

12. Create an array that indicates the total number of elements in an array

In [53]:
```
ar6=np.array([np.size(b)])
print(ar6)
```

```
[5]
```

13. To find the number of dimensions of the array

In [48]:
```
np.ndim(arr2d)
```

Out[48]:
```
2
```

14. Create an array and reshape into a new array

In [49]:
```
print(arr2d)
print(arr2d.reshape(6,1))
```

```
[[ 1  2  3]
 [36  7  8]]
[[ 1]
 [ 2]
 [ 3]
 [36]
 [ 7]
 [ 8]]
```

15. Create a null array of size 10

In [50]:
```
print(np.zeros(10))
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

16. Create any array with values ranging from 10 to 49 and print the numbers whose remainders are zero when divided by 7

In [54]:
```
arr5=np.arange(10,50)
cond=arr5[arr5%7==0]
print(cond)
```

```
[14 21 28 35 42 49]
```

17. Create an array and check any two conditions and print the output

In [59]:
```
cond1=arr5[(arr5>10) & (arr5<35)]
print(cond1)
```

```
[11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34]
```

18. Use Arithmetic operator and print the output using array

In [67]:
```
a1=np.arange(1,7)
a2=np.arange(7,13)
print(a1+a2)
print(f"{a1-a2}\n({a2/a1})\n{a1*a2}")
```

```
[ 8 10 12 14 16 18]
[-6 -6 -6 -6 -6 -6]
([7.  4.  3.  2.5 2.2 2. ])
[ 7 16 27 40 55 72]
```

19. Use Relational operators and print the results using array

In [71]:
```
a5=np.arange(0,6)
print(a5[(a5>2) & (a5<4)])
```

```
[3]
```

20. Difference between python and ipython

Interactive Shell Features:

Python: Python has a basic interactive shell, where you can run one command at a time. Ipython/Jupyter: Ipython provides an enhanced interactive shell with features like tab-completion, history browsing, syntax highlighting, and support for inline plotting and rich media display (such as images and videos). Notebook Interface:

Python: Python itself does not have a notebook interface. Jupyter: Jupyter provides a notebook interface, allowing you to create and share documents (notebooks) containing code, visualizations, explanatory text, and media elements. Jupyter notebooks are popular for data analysis, visualization, machine learning, and educational purposes. Rich Output:

Python: In the standard Python shell, the output is usually plain text. Jupyter: Jupyter notebooks support rich outputs, including HTML, images, LaTeX equations, and interactive visualizations. This feature makes it ideal for presenting data analysis or sharing interactive code demonstrations. Extensions and Magic Commands:

Python: The standard Python shell has limited features beyond running Python code. Jupyter: Jupyter supports magic commands, which are special commands prefixed with '%'. These magic commands provide shortcuts for various actions and system interactions, such as loading files, debugging, or timing code execution. Support for Multiple Kernels:

Python: Python is a programming language, and its interactive shell only supports Python code execution. Jupyter: Jupyter notebooks can support multiple programming languages by using different kernels. This allows you to work with languages like Julia, R, and others within

In [ ]: