# numpy

July 6, 2025

[1]:
```
#numpy
import numpy as np
arr = np.array([[10, 20, 30],[10, 20, 40]])
print(arr)
print(np.shape(arr))
print(arr.ndim)
print(np.size(arr))
```

```
[[10 20 30]
 [10 20 40]]
(2, 3)
2
6
```

[2]:
```
print(arr * 2)
```

```
[20 40 60]
```

[3]:
```
sum_arr = np.sum(arr)
print(sum_arr)
```

```
60
```

[19]:
```
arr2 = np.ones((2, 3))
print(arr2)
```

```
[[1. 1. 1.]
 [1. 1. 1.]]
```

[9]:
```
arr3 = np.zeros((2, 3))
print(arr3)
```

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

[51]:
```
#range like 0 to 10 and space between 1
arr4 = np.arange(0, 10, 1)
print(arr4)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

[33]:
```
arr5 = np.linspace(1, 5, 5)
print(arr5)
```

```
[1. 2. 3. 4. 5.]
```

[37]:
```
# Get dimensions
print(arr2)
arr6 = np.shape(arr2)
print(arr6)
```

```
[[1. 1. 1.]
 [1. 1. 1.]]
(2, 3)
```

[40]:
```
# Total elements
print(arr)
arr7 = np.size(arr)
print(arr7)
```

```
[10 20 30]
3
```

[66]:
```
# Data type
#str1 = np.dtype(arr2)
print(type(np.dtype))
```

```
<class 'numpy._DTypeMeta'>
```

[44]:
```
np.dtype
```

[44]:
```
numpy.dtype
```

[11]:
```
mat = np.full((3, 3), 7)
print(mat)
print(np.min(mat))
```

```
[[7 7 7]
 [7 7 7]
 [7 7 7]]
7
```

[5]:
```
a = np.array([[1, 2, 3], [4, 5, 6]])

print("Shape:", a.shape)
print("Sum:", np.sum(a))
print("Mean:", np.mean(a))
```

```
print("Square root:\n", np.sqrt(a))
```

```
Shape: (2, 3)
Sum: 21
Mean: 3.5
Square root:
 [[1.         1.41421356 1.73205081]
 [2.         2.23606798 2.44948974]]
```

[36]:
```
import numpy as np

data = np.array([12, 15, 21, 18, 19, 21, 25, 30])

print("Sum:", np.sum(data))

print("Mean:", np.mean(data)) #sum all values and divided by total number of␣
 ↪values(sum / n)

# first Ascending order of values If odd number of elements: pick the middle␣
 ↪one-- If even number: average the two middle values
print("Median:", np.median(data))

#Mean = 20.125
# ² = [(12-20.125)² + (15-20.125)² + (21-20.125)² + ... + (30-20.125)²] / 8 = 5.
 ↪25 -like  ² = (mean - value)² / n (Total no. of values)
print("Standard Deviation:", np.std(data))

#It's simply the square of the standard deviation -- std_dev = np.std(data) =>␣
 ↪std_dev ** 2
print("Variance:", np.var(data))

#Mode The value 21 appears twice, more than any other number. If repeat␣
 ↪multiple values (by default) returns the smallest mode
from scipy import stats

print("Mode:", stats.mode(data))


print("Min:", np.min(data))
print("Max:", np.max(data))
```

```
Sum: 161
Mean: 20.125
Median: 20.0
Standard Deviation: 5.2544623892459255
Variance: 27.609375
Mode: ModeResult(mode=21, count=2)
```

```
Min: 12
Max: 30
```

Types of Quartiles:

Q1 (First Quartile) The 25th percentile 25% of the data is below this value

Q2 (Second Quartile) The 50th percentile = Median Divides the data in half

Q3 (Third Quartile) The 75th percentile 75% of the data is below this value

Step 1: Sort the data [3, 5, 7, 8, 12, 13, 14, 18, 21] Step 2: Find Q2 (Median) Middle value = 12

Step 3: Find Q1 Lower half: [3, 5, 7, 8] → Q1 = (5 + 7)/2 = 6

Step 4: Find Q3 Upper half: [13, 14, 18, 21] → Q3 = (14 + 18)/2 = 16

#For even [12, 15, 18, 19, 21, 21, 25, 30] Formula: Percentile rank = (P / 100) × (n - 1) For Q1 (25th percentile): Rank=0.25 × (8 − 1)=1.75 Now find the value at 1.75th position (between index 1 and 2):

Index 1 = 15 Index 2 = 18 Interpolate between them: Q1 = 15 + 0.75 × (18 − 15) = 15 + 2.25 = 17.25

Q3 (75th Percentile): Rank=0.75×(8−1)=5.25 Interpolate between index 5 and 6:

Index 5 = 21 Index 6 = 25 Q3 = 21 + 0.25 × (25 − 21)= 21 + 1 = 22.0

[31]:
```
#Now Quartile with even values
data = np.array([12, 15, 18, 19, 21, 21, 25, 30])

q1 = np.percentile(data, 25)   # First quartile
q2 = np.percentile(data, 50)   # Median (50th percentile) 19 +21 / 2 median =
 ↪20.0
q3 = np.percentile(data, 75)   # Third quartile
print(q1)
print(q2)
print(q3)
```

```
17.25
20.0
22.0
```

[34]:
```
da = np.array([2, 4, 6, 8])

print("Sum:", np.sum(da))
print("Mean:", np.mean(da))
print("Median:", np.median(da))
print("Standard Deviation:", np.std(da))
print("Variance:", np.var(da))
print("Min:", np.min(da))
print("Max:", np.max(da))
```

```
Sum: 20
Mean: 5.0
Median: 5.0
Standard Deviation: 2.23606797749979
Variance: 5.0
Min: 2
Max: 8
```

[18]:
```python
# Dimensions array
ary = np.array(9)
ary2 = np.array([10, 20, 30])

print("\nArray Dimention: ", ary.ndim)
print(ary)

print("\nArray Dimention: ", ary2.ndim)
print(ary2)

ary3 = np.array([[1, 2, 3], [4, 5, 6]])
print("\nArray Dimension (Matrix): ",ary3.ndim)  # Output: 2
print(ary3)

ary4 = np.array([[1, 2, 3], [4, 5, 6], [8, 5, 6]])
print("\nArray Dimension (Matrix): ", ary4.ndim)
print(ary4)

ary5 = np.array([
    [[1, 2], [3, 4]], [[5, 6], [7, 8]]
])
print("\nArray Dimension (Matrix): ", ary5.ndim)  # Output: 3
print(ary5)
print(np.shape(ary5))
```

```
Array Dimention:  0
9

Array Dimention:  1
[10 20 30]

Array Dimension (Matrix):  2
[[1 2 3]
 [4 5 6]]

Array Dimension (Matrix):  2
[[1 2 3]
 [4 5 6]
 [8 5 6]]
```

```
Array Dimension (Matrix):  3
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
(2, 2, 2)
```

[80]:
```python
arr = np.array([
    [[1, 2, 3],
     [4, 5, 6]],

    [[7, 8, 9],
     [10, 11, 12]]
])

print(arr)
print("Shape:", arr.shape)
print("Dimensions:", arr.ndim)
#This means:
#2 matrices
#Each matrix has 2 rows
#Each row has 3 columns
#(number_of_matrices, number_of_rows, number_of_columns)
```

```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
Shape: (2, 2, 3)
Dimensions: 3
```

[79]:
```python
arr_3d = np.array([
    [[1, 2, 3],
     [4, 5, 6]],

    [[7, 8, 9],
     [10, 11, 12]],

    [[7, 8, 9],
     [10, 11, 12]]
])

print(arr_3d)
print("Shape:", arr_3d.shape)
```

```
print("Dimensions:", arr_3d.ndim)
```

```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]

 [[ 7  8  9]
  [10 11 12]]]
Shape: (3, 2, 3)
Dimensions: 3
```

[2]:
```
#Calculator
# Title: Python CLI Calculator with Clean Structure

print("Welcome to Python CLI Calculator")

# Infinite loop for menu
while True:
    # 1. Input/Output Function + Variables
    print("\nSelect From Menu")
    print("1. Addition")
    print("2. Subtraction")
    print("3. Multiplication")
    print("4. Division")
    print("5. Exit")

    choice = input("Enter your choice (1-5): ")   # Input is str by default

    # 2. Type Casting + 4. Conditional Statements
    if choice == "5":
        print("Thank you for using the calculator!")
        break  # 5. Control Flow: Exit the loop

    if choice not in ["1", "2", "3", "4"]:
        print("Invalid input. Please try again.")
        continue

    # 3. Using Data Types + 7. Type Casting
    try:
        num1 = float(input("Enter first number: "))
        num2 = float(input("Enter second number: "))
    except ValueError:
        print("Please enter valid numbers.")
        continue
```

```
    # 6. Operators + 4. Conditional Logic
    if choice == "1":
        result = num1 + num2
        operation = "+"
    elif choice == "2":
        result = num1 - num2
        operation = "-"
    elif choice == "3":
        result = num1 * num2
        operation = "*"
    elif choice == "4":
        if num2 == 0:
            print("Error: Division by zero.")
            continue
        result = num1 / num2
        operation = "/"

    # 8. Output Function
    print(f"Result: {num1} {operation} {num2} = {result}")
```

```
Welcome to Python CLI Calculator

Select From Menu
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Exit

Enter your choice (1-5):  0

Invalid input. Please try again.

Select From Menu
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Exit

Enter your choice (1-5):  5

Thank you for using the calculator!
```

[6]:
```
#Calculator
arr15 = np.random.rand(3, 2, 1)*2
arr15
```

[6]:
```
array([[[0.06864168],
        [0.33367047]],

       [[0.81849621],
        [1.38223509]],

       [[0.68296453],
        [0.37106168]]])
```

1. Using np.zeros( ) method
2. Using np.ones( ) method
3. Using np.empty( ) method
4. Using np.full( ) method
5. Using np.eye( ) method
6. Using np.arange( ) method
7. Using np.linspace( ) method
8. Using random.rand( ) method
9. Using random.randint( ) method
10. Basics Operations on Numpy Array

[22]:
```
arr1 = np.zeros(5, dtype = np.int16)
print("\nZeros:", arr1)
arr2 = np.zeros((3, 2), dtype = np.int16)
print("\nZeros natrix:",arr2)
arr3 = np.zeros((3, 2))
print("\nZeros natrix:",arr3)
```

```
Zeros: [0 0 0 0 0]

Zeros natrix: [[0 0]
 [0 0]
 [0 0]]

Zeros natrix: [[0. 0.]
 [0. 0.]
 [0. 0.]]
```

[42]:
```
#ones(shape, dtype, order, like)
arr4 = np.ones([2, 3], dtype=np.int16, order = 'F')
print(arr4)
print(np.shape(arr4))
print(np.ndim(arr4))
print(np.size(arr4))
```

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

```
2
6
```

[40]:
```
g = np.array([[1, 2], [3, 4]], dtype=np.int16)
h = np.ones((2, 2), like=g)  # returns int16 type array like g
print("Like g (int16):\n", h)
print("dtype:", h.dtype)
```

```
Like g (int16):
 [[1. 1.]
 [1. 1.]]
dtype: float64
```

[52]:
```
#numpy.empty(shape, dtype=float, order='C', *, like=None)
arr7 = np.empty(3, dtype=np.int16, order = 'C')
arr7
```

[52]:
```
array([ -496, 15363, 32331], dtype=int16)
```

[63]:
```
#numpy.full(shape, fill_value, dtype=None, order='C', *, like=None)
arr8 = np.full(4, 5)
print(arr8)

arr11 = np.full((2, 4),5)
arr11
```

```
[5 5 5 5]
```

[63]:
```
array([[5, 5, 5, 5],
       [5, 5, 5, 5]])
```

[11]:
```
#numpy.eye(N (rows), M=None (columns), k=0 (Index of the diagonal. Default is␣
 ↪0), dtype=<class 'float'>, order='C')

arr9 = np.eye(4)
arr9
arr10 = np.eye(4, 4, k = -1)
print(arr10)
```

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

[29]:
```
#numpy.arange([start,] stop[, step], dtype=None, *, like=None)
arr11 = np.arange(2, 20, 2)
arr11
```

[29]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18])
```

[28]:
```
print(np.arange(2, 10, 2, dtype = np.float32))
```

```
[2. 4. 6. 8.]
```

[38]:
```
arr_2d = np.arange(2, 20, 2).reshape(3, 3)
print(arr_2d)
```

```
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
```

[52]:
```
# Create a 3D array with shape (2 blocks, 3 rows, 2 columns)
arr_3d = np.arange(18).reshape(2, 3, 3)
print("3D Array:\n", arr_3d)
```

```
3D Array:
 [[[ 0  1  2]
  [ 3  4  5]
  [ 6  7  8]]

 [[ 9 10 11]
  [12 13 14]
  [15 16 17]]]
```

[13]:
```
rad = np.random.rand(3, 3, 3)
print(rad)
print(np.ndim(rad))
```

```
[[[0.36096062 0.9163431  0.26108641]
  [0.7086184  0.84006773 0.72013752]
  [0.69288172 0.0793261  0.91567152]]

 [[0.07340347 0.79143381 0.16493285]
  [0.83644269 0.10463797 0.39409597]
  [0.55528338 0.21417341 0.42370176]]

 [[0.51583334 0.48147461 0.42337983]
  [0.02460274 0.88156839 0.34494623]
  [0.91685143 0.24727662 0.25916133]]]
3
```

[10]:
```
import numpy as np
arr_opr = np.random.randint(1, 50, size = (5, 3))
print(arr_opr)
```

```
[[ 7 48 40]
```

```
 [21 33  7]
 [28 28 15]
 [37  2 46]
 [47 37 13]]
```

[18]:
```
#Basic Arithmetic Operations
print("Added by 2", arr_opr + 2)
print("Substraction by 2", arr_opr - 2)
print("Multiple by 2", arr_opr * 2)
print("Divide by 2", arr_opr / 2)
#Power and Modulo
print("Mod by 2", arr_opr % 2)
print("Power by 2", np.power(arr_opr, 2))
#Aggregate Functions
print("Sum", np.sum(arr_opr))
print("Mean", np.mean(arr_opr))
print("Median", np.median(arr_opr))
print("Standardivation", np.std(arr_opr))
print("Variance", np.var(arr_opr))
```

```
Added by 2 [[ 9 50 42]
 [23 35  9]
 [30 30 17]
 [39  4 48]
 [49 39 15]]
Substraction by 2 [[ 5 46 38]
 [19 31  5]
 [26 26 13]
 [35  0 44]
 [45 35 11]]
Multiple by 2 [[14 96 80]
 [42 66 14]
 [56 56 30]
 [74  4 92]
 [94 74 26]]
Divide by 2 [[ 3.5 24.  20. ]
 [10.5 16.5  3.5]
 [14.  14.   7.5]
 [18.5  1.  23. ]
 [23.5 18.5  6.5]]
Mod by 2 [[1 0 0]
 [1 1 1]
 [0 0 1]
 [1 0 0]
 [1 1 1]]
Power by 2 [[  49 2304 1600]
 [ 441 1089   49]
 [ 784  784  225]
```

```
 [1369    4 2116]
 [2209 1369  169]]
Sum 409
Mean 27.266666666666666
Median 28.0
Standardivation 15.07521881175269
Variance 227.2622222222222
```

[25]:
```
#Dot Product
#The number of columns in the first matrix must equal the number of rows in the second matrix
a = np.array([[1, 2],
              [3, 4]])
b = np.array([[5, 6],
              [7, 8]])
print("dimension:", np.ndim(a))
print("\nDot Product:\n", np.dot(a, b))

#Result[0][0] = 1×5 + 2×7 = 5 + 14 = 19
#Result[0][1] = 1×6 + 2×8 = 6 + 16 = 22
#Result[1][0] = 3×5 + 4×7 = 15 + 28 = 43
#Result[1][1] = 3×6 + 4×8 = 18 + 32 = 50
```

```
dimension: 2

Dot Product:
 [[19 22]
 [43 50]]
```

[ ]: