METHODIST
COLLEGE OF ENGINEERING & TECHNOLOGY
(An UGC Autonomous Institution)
Approved by AICTE, New Delhi & Affiliated to Osmania University
Accredited by NBA & NAAC with A+ Grade
Estd : 2008

DEPARTMENT OF CSE

Branch : AIDS – IV Sem

SAC Lab

Program No.	Program Name	Date	Status
1	Program or script which demonstrate the use of different data types	17-02-25	Executed
2	PROGRAM TO MAKE USE OF FOLLOWING OPERATORS A.ARITHMETIC OPERATIONS B.LOGICAL OPERATORS C.RELATIONAL OPERATIONS	17-02-25	Executed
3	PROGRAM TO MAKE USE OF IF-ELSE,E-IF AND NESTED IF ELSE LOOPS a.Finding biggest of 3 numbers b.Even or odd c.Prime number d.Finding the grades of student	17-02-25	Executed
4	Program to make use of while loop .Print the series from 1 to n B.print the even and odd series C.sum of natural numbers d.Armstrong number e.Palindrome	18-02-25	Executed
5	PROGRAM TO MAKE USE OF FOR LOOP	18-02-25	

	a.Print the series from 1 to n b.Print the even and odd series C.sum of natural numbers d.Armstrong number e.palindrome		Executed
6	Program which demonstrate the use of function following a.No values passing and no parameters return B.passing values and no parameters returns c.No passing valuesbut return types d.Parameters passing and return types	18-02-25	Executed
7	Program to make use of lists and their operations or methods	18-02-25	Executed
8	Program to make use of dictionaries and their methods	18-02-25	Executed
9	Program to make use of tuples and their methods	18-02-25	Executed
10	Program to work with sets and their operations	18-02-25	Executed

11	CREATE A SINGLE DIMENSIONAL ARRAY AND WORK WITH FOLLOWING FUNCTIONS SUM MEAN MIN MAX SHAPE SIZE DTYPE STD RESHAPE REVERSE TRANSPOSE SQUARE	10-03-25	Executed
12	CREATE A TWO DIMENSIONAL ARRAY AND WORK WITH FOLLOWING FUNCTIONS SUM MEAN MIN MAX SHAPE SIZE DTYPE STD RESHAPE REVERSE TRANSPOSE SQUARE DOT HSTACK VSTACK	10-03-25	Executed
13	TRIGNOMETRIC FUNCTIONS SIN COS TAN	10-03-25	Executed
14	ARITHMETIC FUNCTIONS ADD SUB AND MUL	17-03-25	Executed
15	BITWISE OPERATORS	17-03-25	Executed
16	ROUNDING FUNCTIONS	02-04-25	Executed
17	COMPARISION FUNCTIONS	02-04-25	Executed
18	EXPONENTIAL AND LOGARITHMIC FUNCTIONS	02-04-25	Executed

19	STATISTICS AND AGGREGATION	02-04-25	Executed
20	LINEAR ALGEBRA FUNCTIONS	21-04-25	Executed

21	PERFORM THE FOLLOWING BOOLEAN FUNCTIONS
	np.all(arr1)
	np.any(arr1)
	np.where()
		21-04-25	Executed

22	PERFORM THE FOLLOWING SORTING METHODS - np.sort(arr1) -np.argsort(arr1) np.unique(arr1)	21-04-25	Executed
23	FILE READING & WRITING	09-06-25	Executed
24	AGGREGATION AND UNIVERSAL FUNCTIONS	17-05-25	Executed
25	IDENTITY MATRIX	17-05-25	Executed
26	PANDAS PANDAS BASICS	17-05-25	Executed

27	Perform the following Operations on Multiple arrays a. Stack two arrays vertically b. Stack two arrays horizontally c. Get the common items between two python numpy arrays d. Remove from one array those items that exist in another e. Get the positions where elements of two arrays match	17-05-25	Executed
28	Work with following functions in Series and Data frames using Pandas Function Description df.sort_values('col', ascending=True) Sorts by a column df.sort_values(['col1', 'col2']) Sorts by multiple columns df.set_index('col') Sets a column as index df.reset_index() Resets index	17-05-25	Executed
29	df.groupby('col').mean() Groups by column and calculates mean	17-05-25	Executed
30	WORK WITH FILTERING AND CONDITIONAL SELECTION	09-06-25	Executed

31	SORTING AND RECORDING DATA	09-06-25	Executed
32	Statical analysis Compute the mean,median,standard deviation of a numpy array. Find the percentile scores of a numpy array Compute the Euclidean distance between two arrays	09-06-25	Executed
33	Implement the following Web API’s methods for any simple applications A.Get b.Put c.Post d.Update	09-06-25	Executed
34	Implement sqlite3	09-06-25	Executed
35	Implement the data transformation in python using Pandas ü Removing duplicates ü Adding a column ü Replacing values ü Renaming axis/index ü Discretization and binning ü Detecting and filtering outliers ü Permutation and random sampling	16-06-25	Executed

36	**Implement String Manipulation Functions in Python**	**16-06-25**	**Executed**
37	**Data Wrangling in python:Join,Combine,and Reshape**	**16-06-25**	**Executed**
38	**Data Aggregation and Group Operations in Pandas**	**16-06-25**	**Executed**

NAME:G.DEEKSHITHA RAJ
ROLL NO. :160723747132

17-02-25

5.PROGRAM TO MAKE USE OF FOR LOOP

```
n=int(input("enter the range of sum\n"))

sum=0

for i in range(n+1):

        sum+=i

        print('sum of',n,'numbers is:',sum)
```

OUTPUT:

```
enter the range of sum

15

sum of 15 numbers is: 0

sum of 15 numbers is: 1

sum of 15 numbers is: 3

sum of 15 numbers is: 6

sum of 15 numbers is: 10

sum of 15 numbers is: 15
sum of 15 numbers is: 21

sum of 15 numbers is: 28

sum of 15 numbers is: 36

sum of 15 numbers is: 45

sum of 15 numbers is: 55

sum of 15 numbers is: 66

sum of 15 numbers is: 78

sum of 15 numbers is: 91

sum of 15 numbers is: 105

sum of 15 numbers is: 120
```

```
# a. Print the series from 1 to n

n = int(input("Enter a number (n): "))

print("Series from 1 to", n)

for i in range(1, n + 1):

    print(i, end=' ')

print("\n")
```

```
OUTPUT

Enter a number (n): 5

Series from 1 to 5
1 2 3 4 5
```

```
# b. Print even and odd series
```

```
print("Even numbers from 1 to", n)
for i in range(1, n + 1):
    if i % 2 == 0:
        print(i, end=' ')
print("\n")
```

OUTPUT

```
Even numbers from 1 to 5
2 4
```

```
print("Odd numbers from 1 to", n)
for i in range(1, n + 1):
    if i % 2 != 0:
        print(i, end=' ')
print("\n")
```

OUTPUT:

```
Odd numbers from 1 to 5
1 3 5
```

```
# c. Sum of natural numbers
sum_natural = 0
for i in range(1, n + 1):
    sum_natural += i
print("Sum of natural numbers from 1 to", n, "is", sum_natural)
```

OUTPUT:

```
Sum of natural numbers from 1 to 5 is 15
```

```
# d. Check Armstrong number
num = int(input("\nEnter a number to check for Armstrong: "))
order = len(str(num))
sum_armstrong = 0
for digit in str(num):
    sum_armstrong += int(digit) ** order
if num == sum_armstrong:
    print(num, "is an Armstrong number.")
else:
    print(num, "is not an Armstrong number.")
```

OUTPUT:

```
Enter a number to check for Armstrong: 21
21 is not an Armstrong number.
```

```
# e. Check Palindrome number
num = input("\nEnter a number to check for Palindrome: ")
is_palindrome = True
for i in range(len(num) // 2):
    if num[i] != num[-(i + 1)]:
        is_palindrome = False
        break
if is_palindrome:
    print(num, "is a Palindrome number.")
else:
    print(num, "is not a Palindrome number.")
```

OUTPUT

```
Enter a number to check for Palindrome: 5
5 is a Palindrome number.
```

1.PROGRAM TO DEMONSTRATE THE USE OF DIFFERENT DATATYPES

```
#INTEGER

a=1

print(a)

print(type(a))

#float

b=2.8

print(b)

print(type(b))

#complex

c=1j

print(c)
print(type(c))

#string

d='hi'

print(d)

print(type(d))
```

output:

```
1
<class 'int'>
2.8
<class 'float'>
1j
<class 'complex'>
hi
<class 'str'>
```

2.PROGRAM TO MAKE USE OF FOLLOWING OPERATORS ARITHMETIC OPERATIONS,LOGICAL OPERATIONS AND RELATIONAL OPERATIONS

```
#ARITHMETIC
a=7
b=2
#addition
print('sum:',a+b)
#subtraction
print('subtraction:',a-b)
#multiplication
print('multiplication:',a*b)
#division
print('division:',a/b)
#modulo
print('modulo:',a%b)
```

OUTPUT:

```
sum: 9
subtraction: 5
```

```
multiplication: 14
division: 3.5
modulo: 1
```

```
#logical
a=5
b=6
print((a>2)and(b>=6))
```

OUTPUT:

```
TRUE
```

```
#relational
a=10
b=5
if a>b:
    print("a is greater than b")
if a<=b:
    print("a is less than or equal to b")
if a==b:
    print("a is equal to b")
if a!=b:
    print("a is not equal to b")
```

OUTPUT:

```
a is greater than b
a is not equal to b
```

3. Program to make use of If-else, el-if and nested if else loops

a. Finding biggest of 3 numbers

b. Even or odd

c. Prime number

d. Finding the grades of student

```
# a. Finding biggest of 3 numbers
print("a. Find the biggest of 3 numbers")
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
c = int(input("Enter third number: "))

if a >= b and a >= c:
    biggest = a
elif b >= a and b >= c:
    biggest = b
else:
    biggest = c

print("The biggest number is:", biggest)
```

OUTPUT:

```
a. Find the biggest of 3 numbers
Enter first number: 20
Enter second number: 30
Enter third number: 15
The biggest number is: 30
```

```
# b. Even or odd
print("\nb. Check if a number is Even or Odd")
num = int(input("Enter a number: "))
if num % 2 == 0:
    print(num, "is Even")
else:
    print(num, "is Odd")
```

OUTPUT:

```
b. Check if a number is Even or Odd
Enter a number: 55
55 is Odd
```

```
# c. Prime number check
print("\nc. Prime Number Check")
num = int(input("Enter a number: "))

if num <= 1:
    print(num, "is not a Prime number")
else:
    is_prime = True
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            is_prime = False
            break
    if is_prime:
        print(num, "is a Prime number")
    else:
        print(num, "is not a Prime number")
```

OUTPUT:

```
c. Prime Number Check
Enter a number: 56
56 is not a Prime number
```

```
# d. Finding the grade of a student
print("\nd. Student Grade Calculator")
marks = float(input("Enter the student's marks (0-100): "))

if marks > 100 or marks < 0:
    print("Invalid marks entered.")
else:
    if marks >= 90:
        grade = 'A'
    elif marks >= 80:
        grade = 'B'
    elif marks >= 70:
        grade = 'C'
    elif marks >= 60:
        grade = 'D'
    elif marks >= 40:
        grade = 'E'
    else:
        grade = 'F'

    print("The student's grade is:", grade)
```

OUTPUT:

```
d. Student Grade Calculator
Enter the student's marks (0-100): 56
The student's grade is: E
```

4. Program to make use of while loop

a. Print the series from 1 to n

b. Print the even and odd series

c. Sum of natural numbers

d. Armstrong number

e. Palindrome

```
# a. Print the series from 1 to n
n = int(input("Enter a number (for series 1 to n): "))
i = 1
print("Series from 1 to", n, ":")
```

```
while i <= n:
    print(i, end=" ")
    i += 1
print("\n")
```

OUTPUT

```
Enter a number (for series 1 to n): 5
Series from 1 to 5 :
1 2 3 4 5
```

```
# b. Print even and odd series
i = 1
print("Even numbers up to", n, ":")
while i <= n:
    if i % 2 == 0:
        print(i, end=" ")
    i += 1
print()

i = 1
print("Odd numbers up to", n, ":")
while i <= n:
    if i % 2 != 0:
        print(i, end=" ")
    i += 1
print("\n")
```

OUTPUT:

```
Even numbers up to 5 :
2 4
Odd numbers up to 5 :
1 3 5
```

```
# c. Sum of natural numbers
i = 1
total = 0
while i <= n:
    total += i
    i += 1
print("Sum of natural numbers from 1 to", n, "is:", total)
print()
```

OUTPUT:

```
Sum of natural numbers from 1 to 5 is: 15
```

```
# d. Armstrong number check
num = int(input("Enter a number to check Armstrong: "))
temp = num
sum = 0
digits = len(str(num))
while temp > 0:
    digit = temp % 10
    sum += digit ** digits
    temp //= 10
if sum == num:
    print(num, "is an Armstrong number.")
else:
    print(num, "is not an Armstrong number.")
print()
```

OUTPUT:

```
Enter a number to check Armstrong: 24
24 is not an Armstrong number.
```

```
# e. Palindrome check
num = int(input("Enter a number to check Palindrome: "))
temp = num
rev = 0
while temp > 0:
    digit = temp % 10
    rev = rev * 10 + digit
    temp //= 10
if rev == num:
    print(num, "is a Palindrome.")
else:
    print(num, "is not a Palindrome.")
```

OUTPUT:

```
Enter a number to check Palindrome: 67
67 is not a Palindrome.
```

6. program which demonstartes the use of function following

1.no values passing and no parameters return

2.passing values and no parameters returns

3. no passing values but return types

4.parameters passing and return types

```
# 1. No values passing, no parameters, no return type
def greet():
   print("Hello! Welcome to the function demonstration.")

# 2. Passing values, no parameters, and no return type
def greet_person(name):
   print(f"Hello, {name}! How are you today?")
# 3. No values passing, but return type
def get_favorite_color():
   return "Blue"
```

```
# 4. Passing values and return types
def add_numbers(a, b):
    return a + b
# Function calls
# 1. Calling function with no parameters and no return value
greet()
# 2. Calling function with a parameter and no return value
name = "Alice"
greet_person(name)
# 3. Calling function with no parameters but with a return value
color = get_favorite_color()
print(f"My favorite color is {color}.")
# 4. Calling function with parameters and getting a return value
num1 = 10
num2 = 20
result = add_numbers(num1, num2)
print(f"The sum of {num1} and {num2} is {result}.")
```

outputs:

```
Hello! Welcome to the function demonstration.
Hello, Alice! How are you today?
My favorite color is Blue.
The sum of 10 and 20 is 30.
```

7.PROGRAM TO MAKE USE OF CLASSES AND OBJECTS

```
Define a class
class Dog:
    def __init__(self, name, breed):
        self.name = name
        self.breed = breed
    def bark(self):
        return "Woof!"


my_dog = Dog("Buddy","Golden Retriever")
print(my_dog.name)
print(my_dog.breed)
print(my_dog.bark())
```

OUTPUT:

```
Buddy
Golden Retriever
```

8.PROGRAM TO MAKE USE OF LISTS AND THEIR METHODS

```
#INITIALIZING A LIST
details=['abc','18','deek']
print(details)
#list length
print(len(details))
#APPENDING TO LIST
app=input("enter the item to append")
details.append(app)
print(details)
#return index of value
print(details)
print(details.index('deek'))
```

OUTPUT:

```
['abc', '18', 'deek']
3
enter the item to appendboor
['abc', '18', 'deek', 'boor']
['abc', '18', 'deek', 'boor']
2
```

PROGRAM TO MAKE USE OF DICTIONARIES AND THEIR METHODS

```
a='10': 'Deekshitha', '02': 'Akshay'

print(a)

print(type(a))

#length of Dictionary

print(len(a))

print(a['02'])

print(a.get('02'))

k-a.keys() #list of all keys

print(k)

va.values() #list of all values
```

```
print(v)

x=items #list of all items

print(x)

if '02' in a

print('yes')

a.update(['03':'asha’)

#adding to dictionary

print(x)
```

OUTPUT:

```
{'10': 'Deekshitha', '02': 'Akshay'}

<class 'dict'>

2

Akshay

Akshay

dict_keys(['10', '02'])

dict_values(['Deekshitha', ‘Akshay'])

dict_items([('10', 'Deekshitha'), ('02', 'Akshay')])
```

```
yes

dict_items([('10', 'Deekshitha'), ('02', 'Akshay'), ('03', 'asha')])
```

PROGRAM TO MAKE USE OF TUPLES AND THEIR METHODS

```
Tuple1 = (0, 1, 2, 3, 2, 3, 1, 3, 2)
Tuple2 = ('deek', 'sai', 'babbi',
        'java', 'python')

# count the appearance of 3
res = Tuple1.count(3)
print('Count of 3 in Tuple1 is:', res)

# count the appearance of python
```

```
res = Tuple2.count('python')
print('Count of Python in Tuple2 is:', res)
```

PROGRAM TO MAKE USE OF SETS AND THEIR METHODS

```
#SET OPERATIONS
#Creating a Set
s1 = {1, 2, 3, 4, 5}
s2 = {3,4, 5, 6, 7,}

#Union Operation
s3 = s1.union(s2)
print(s3)
```

```
#Intersection Operation
s3 = s1.intersection(s2)
print(s3)

#Difference Operation
s3 = s1.difference(s2)
print(s3)

#Symmetric Difference Operation
s3 = s1.symmetric_difference(s2)
print(s3)
```

OUTPUT:

OUTPUT:

```
{1, 2, 3, 4, 5, 6, 7}
{3,4, 5}
{1, 2}
```

24.02.2025

12.CREATE A SINGLE DIMENSIONAL ARRAY AND WORK WITH FOLLOWING FUNCTIONS SUM MEAN MIN MAX SHAPE SIZE DTYPE STD RESHAPE REVERSE TRANSPOSE SQUARE

```
import numpy as np
#1D Array Operations
print("1D ARRAY OPERATIONS")
arr1D=np.array([1,2,3,4,5])
print("original 1D array:",arr1D)
print("shape:",arr1D.shape)
print("size:",arr1D.size)
print("Data Type:",arr1D.dtype)
print("sum:",np.sum(arr1D))
print("mean:",np.mean(arr1D))
print("max:",np.max(arr1D))
print("min:",np.min(arr1D))
print("standard deviation",np.std(arr1D))
print("reshaped(5,1):\n",arr1D.reshape(5,1))
print("Flatenning of arr1=",arr1D.flatten())
print(“transpose”,arr1D.T)
print()
```

OUTPUT:

```
1D ARRAY OPERATIONS
original 1D array: [1 2 3 4 5]
shape: (5,)
size: 5
Data Type: int32
sum: 15
mean: 3.0
max: 5
min: 1
standard deviation 1.4142135623730951
reshaped(5,1):
 [[1]
 [2]
 [3]
 [4]
 [5]]
Flatenning of arr1= [1 2 3 4 5]


transpose [1 2 3 4 5]
```

C.CREATE A TWO DIMENSIONAL ARRAY AND WORK WITH FOLLOWING FUNCTIONS SUM MEAN MIN MAX SHAPE SIZE DTYPE STD RESHAPE REVERSE TRANSPOSE SQUARE DOT HSTACK VSTACK

```
import numpy as np
arr1=np.array([[1,2,3],[2,3,4],[2,3,1]])
arr2=np.array([[1,2,3],[1,2,3],[1,2,3]])
arr3=np.array([[1,2,3],[2,3,4],[2,3,1],[1,1,1],[1,1,2]])
print("array1 elements=",arr1)
print("array2 elements=",arr2)
print("type of array1",type(arr1))
print("type of array2",type(arr2))
print("data type of array1",(arr1.dtype))
print("data type of array2",(arr2.dtype))
print("sum of two arrays",arr1+arr2)
print("shape of arrays",arr1.shape)
print("reshape the array",arr1.reshape(1*9))
print("reshape the array",arr1.reshape(9*1))
print("max of array1",np.max(arr1))
print("transpose",arr1.T)
print("matrix mul=",np.dot(arr1,arr2))
```

OUTPUT

```
array1 elements= [[1 2 3]
 [2 3 4]
 [2 3 1]]
array2 elements= [[1 2 3]
 [1 2 3]
 [1 2 3]]
type of array1 <class 'numpy.ndarray'>
type of array2 <class 'numpy.ndarray'>
data type of array1 int32
data type of array2 int32
sum of two arrays [[2 4 6]
 [3 5 7]
 [3 5 4]]
shape of arrays (3, 3)
reshape the array [1 2 3 2 3 4 2 3 1]
reshape the array [1 2 3 2 3 4 2 3 1]
max of array1 4
transpose [[1 2 2]
 [2 3 3]
 [3 4 1]]
matrix mul= [[ 6 12 18]
 [ 9 18 27]
 [ 6 12 18]]
```

E.TRIGNOMETRIC FUNCTIONS SIN COS TAN

```
import numpy as np
# define angles in degrees
angles_degrees = np.array([30, 20, 5])
# convert to radians
angles_radians = np.radians(angles_degrees)
# compute trigonometric functions
print("sine:", np.sin(angles_radians))
print("cosine:", np.cos(angles_radians))
print("Tangent:", np.tan(angles_radians))
```

OUTPUT:

```
sine: [0.5        0.34202014 0.08715574]
cosine: [0.8660254  0.93969262 0.9961947 ]
Tangent: [0.57735027 0.36397023 0.08748866]
```

D.ARITHMETIC FUNCTIONS ADD SUB AND MUL

```
import numpy as np
arr1=np.array([1,2,3,4])
arr2=np.array([5,6,7,8])
#AADN
print("addition:",np.add(arr1,arr2))
#subtraction
print("subtraction:",np.subtract(arr1,arr2))
#MULTIPLICATION
print("multiplication:",np.multiply(arr1,arr2))
```

OUTPUT:

```
addition: [ 6  8 10 12]
subtraction: [-4 -4 -4 -4]
multiplication: [ 5 12 21 32]
```

03/03/2025

BITWISE OPERATORS :

```
import numpy as np
arr1 = np.array([2, 3])
arr2 = np.array([1, 1])
print("Bitwise AND:", np.bitwise_and(arr1, arr2))
print("Bitwise OR:", np.bitwise_or(arr1, arr2))
print("Bitwise XOR:", np.bitwise_xor(arr1, arr2))
print("Bitwise NOT (arr1):", np.bitwise_not(arr1))
print("Bitwise NOT (arr2):", np.bitwise_not(arr2))
```

OUTPUT:

```
Bitwise AND: [0 1]
Bitwise OR: [3 3]
Bitwise XOR: [3 2]
Bitwise NOT (arr1): [-3 -4]
Bitwise NOT (arr2): [-2 -2]
```

ROUNDING FUNCTIONS:

```
import numpy as np
arr = np.array([1.1, 2.5, 6.6])
print("Floor:", np.floor(arr))
print("Ceil:", np.ceil(arr))
print("Round:", np.round(arr))
```

OUTPUT:

```
Floor: [1. 2. 6.]
Ceil: [2. 3. 7.]
Round: [1. 2. 7.]
```

COMPARISION FUNCTIONS:

```
import numpy as np
arr1=np.array([1,2,3])
arr2=np.array([3,2,1])
print("Greater than:",np.greater(arr1,arr2))
print("Less than:",np.less(arr1,arr2))
print("Equal:",np.equal(arr1,arr2))
```

OUTPUT:

```
Greater than: [False False  True]
Less than: [ True False False]
Equal: [False  True False]
```

EXPONENTIAL AND LOGARITHMIC FUNCTIONS:

```
import numpy as np
arr=np.array([1,2,3])
print("Exponential(e^x):",np.exp(arr))
print("Natural log(lnx):",np.log(arr))
print("log base 10:",np.log10(arr))
```

OUTPUT:

```
Exponential(e^x): [ 2.71828183  7.3890561  20.08553692]
Natral log(lnx): [0.         0.69314718 1.09861229]
log base 10: [0.         0.30103    0.47712125]
```

STATISTICS AND AGGREGATION:

```
import numpy as np
arr=np.array([1,2,3,4,5])
print("Sum:",np.sum(arr))
print("Mean:",np.mean(arr))
print("Median:",np.median(arr))
print("Standard deviation:",np.std(arr))
print("Variance:",np.var(arr))
print("Min:",np.min(arr))
print("Max:",np.max(arr))
print("Product:",np.prod(arr))
print("Percentile:",np.percentile(arr,))
print("Mode:",np.mode(arr))
```

OUTPUT:

```
Sum: 15
Mean: 3.0
Median: 3.0
Standard deviation: 1.4142135623730951
Variance: 2.0
Min: 1
Max: 5
Product: 120
```

LINEAR ALGEBRA FUNCTIONS

```
import numpy as np
A=np.array([[1,2],[3,4]])
B=np.array([[2,0],[1,2]])
print("Matrix Multiplication:",np.dot(A,B))
print("Determinant:",np.linalg.det(A))
print("Inverse:",np.linalg.inv(A))
print("Rank:",np.linalg.matrix_rank(A))
print("Eigen:",np.linalg.eig(A))
print("Singular value decomposition:",np.linalg.svd(A))
```

OUTPUT:

```
Matrix Multiplication: [[ 4  4]
 [10  8]]
Determinant: -2.0000000000000004
Inverse: [[-2.   1. ]
 [ 1.5 -0.5]]
Rank: 2
Eigen: EigResult(eigenvalues=array([-0.37228132,  5.37228132]),
eigenvectors=array([[-0.82456484, -0.41597356],
       [ 0.56576746, -0.90937671]]))
Singular value decomposition: SVDResult(U=array([[-0.40455358, -0.9145143
],
       [-0.9145143 ,  0.40455358]]), S=array([5.4649857 , 0.36596619]),
Vh=array([[-0.57604844, -0.81741556],
       [ 0.81741556, -0.57604844]]))
```

10/03/2025

12.PERFORM THE FOLLOWING BOOLEAN FUNCTIONS

np.all(arr1)
np.any(arr1)
np.where()

```
import numpy as np
arr1=np.array([1,2,3,4])
result_all=np.all(arr1)
print(result_all)
result_any=np.any(arr1)
print(result_any)
result_where=np.where(arr1==0)
print(result_where)
```

```
import numpy as np
arr=np.array([2,4,6,9])
d=np.where(arr>5,'high','low')
print(d)
```

OUTPUT:

```
True
True
True
['low' 'low' 'high' 'high']
```

13.PERFORM THE FOLLOWING SORTING METHODS

- np.sort(arr1)
-np.argsort(arr1)
np.unique(arr1)

```
a=np.array([5,4,3,2,1])
a1=np.array ([1,2,3,4,5])
a2=np.array([5,4,1,2,3])
print(np.sort(a))
print(np.argsort(a1))
print(np.unique(a2))
```

OUTPUT:

```
[1 2 3 4 5]
[0 1 2 3 4]
[1 2 3 4 5]
```

11.FILE READING & WRITING

np.loadtxt(filename,delimiter=”,”) - loads data from a text file
np.savetxt(filename,arr,delimiter=”,”) - saves array to a text file
np.save(filename,arr) - saves array to a binary .npy file
np.load(filename) - loads an array from a .npy file

```
arr=([[20,60,30,40,50,],[20,60,30,40,50,]])

filename=r"d:\array.txt";

np.savetxt(filename, arr, delimiter=",")

print("file saved")
```

O/P=

```
file saved
```

```
arr=([[20,30,40,50,],[20,30,40,50,]])

filename=r&quot;d:\array.txt&quot;

np.loadtxt(filename, delimiter=&quot;,&quot;)
```

O/P=

```
array([[20., 30., 40., 50.],

[20.,30., 40., 50.]])
```

AGGREGATION AND UNIVERSAL FUNCTIONS :

```
import numpy as np

data_list = [1, 2, 3, 4, 5]
```

```
data_array = np.array(data_list)

# Aggregation functions
sum_value = np.sum(data_array)  #SUM
mean_value = np.mean(data_array)  # Average
max_value = np.max(data_array)  # Maximum value

# Universal function
squared = np.square(data_array)  # Squares

# Output results
print("Sum:", sum_value)
print("Mean:", mean_value)
print("Max:", max_value)
print("Squared values:", squared)
```

OUTPUT:

```
Sum: 15
Mean: 3.0
Max: 5
Squared values: [ 1  4  9 16 25]
```

IDENTITY MATRIX

(`np.identity`, `np.eye`)

```
import numpy as np
matrix=np.eye(3)
```

```
random_mask=np.random.randint(10,20,size=(3,3))
result=np.where(matrix==0,random_mask,matrix)
print(result)
```

OUTPUT

```
[[ 1. 13. 18.]
 [14.  1. 19.]
 [16. 17.  1.]]
```

Arr+value
Arr-value
Arr*value

```
import numpy as np
arr=np.random.randint(1,10,size=(3,4))
scalar=5
add=arr+scalar
sub=arr-scalar
mul=arr*scalar
print("Addition:\n",add)
print("Subtraction:\n",sub)
print("Multiplication:\n",mul)
```

OUTPUT:

```
Addition:
 [[ 8 11 11  6]
 [11  9  6  6]
 [13 11  7  8]]
Subtraction:
 [[-2  1  1 -4]
 [ 1 -1 -4 -4]
 [ 3  1 -3 -2]]
Multiplication:
 [[15 30 30  5]
 [30 20  5  5]
 [40 30 10 15]]
```

PANDAS

1.PANDAS BASICS

A.INSTALLING PANDAS

a.IMPORT PANDAS AND CHECK THE VERSION

b.CREATE A SERIES FROM A LIST,NUMPY ARRAY AND DICT

c.CONVERT THE INDEX OF _A SERIES INTO A COLUMN OF A DATA FRAME

d.COMBINE MANY SERIES TO FORM A DATA FRAME

#INITIALIZATION

```
conda install pandas
```

CREATING A DATA FRAME

```
import pandas as pd
data = {'name': ['alice', 'bob', 'charlie'], 'age': [16, 17, 18], 'city':
['florida', 'austin', 'dallas']}
df = pd.DataFrame(data)
print(df)
```

Output:

```
     name  age     city
0    alice   16  florida
1      bob   17   austin
2  charlie   18   dallas
```

Grouping & Aggregation

Function	Description
df.groupby('col').mean()	Groups by column and calculates mean
df.groupby('col').agg(['mean', 'sum'])	Performs multiple aggregations
df.pivot_table(values='col1', index='col2')	Creates a pivot table

```
import pandas as pd
# Create a DataFrame
data = {'col': ['Akshi', 'Bubbles', 'Akshi', 'Bubbles', 'Akshi'],
        'value': [10, 20, 30, 40, 50],
        'category': ['X', 'Y', 'X', 'Y', 'X']}
df = pd.DataFrame(data)

# 1. Group by 'col' and calculate the mean of 'value'
grouped_mean = df.groupby('col').mean()

# 2. Group by 'col' and perform multiple aggregations (mean and sum)
grouped_agg = df.groupby('col').agg(['mean', 'sum'])

# 3. Create a pivot table using 'category' as the index and 'value' as the values
pivot_table = df.pivot_table(values='value', index='category')

print("Grouped by 'col' and Mean of 'value':")
print(grouped_mean)

print("\nGrouped by 'col' with Mean and Sum of 'value':")
print(grouped_agg)

print("\nPivot Table with 'category' as index:")
print(pivot_table)
```

OUTPUT:

```
Grouped by 'col' and Mean of 'value':
        value
```

```
col
Akshi     30.0
Bubbles   30.0

Grouped by 'col' with Mean and Sum of 'value':
        value
         mean sum
col
Akshi    30.0  90
Bubbles  30.0  60

Pivot Table with 'category' as index:
          value
category
X            30
Y            30
```

BASIC INFORMATION

```
import pandas as pd

data = {
    'name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'age': [25, 30, 40, 28, 22],  # Adding a missing age value
    'salary': [50000, 60000, 75000, 80000, 62000],
    'city': ['Newyork', 'LOS', 'Chicago', 'Haustin', 'Miami']
}
```

```
# Create DataFrame
df = pd.DataFrame(data)

# Display basic information
print("First 3 rows:\n", df.head(3))  # First 3 rows

print("\nLast 2 rows:\n", df.tail(2))  # Last 2 rows

print("\nShape of DataFrame:", df.shape)  # Shape (rows, columns)

print("\nColumn Names:", df.columns)  # Column names

print("\nIndex of the DataFrame:")
print(df.index)  # Index of the DataFrame
```

OUTPUT

```
First 3 rows:
      name  age  salary     city
0    Alice   25   50000  Newyork
1      Bob   30   60000      LOS
2  Charlie   40   75000  Chicago


Last 2 rows:
    name  age  salary     city
3  David   28   80000  Haustin
4    Eve   22   62000    Miami

Shape of DataFrame: (5, 4)

Column Names: Index(['name', 'age', 'salary', 'city'], dtype='object')

Index of the DataFrame:
RangeIndex(start=0, stop=5, step=1)
```

16. Perform the following Operations on Multiple arrays

a. Stack two arrays vertically
b. Stack two arrays horizontally
c. Get the common items between two python numpy arrays
d. Remove from one array those items that exist in another
e. Get the positions where elements of two arrays match

```
import numpy as np

# Create two little arrays (like student IDs from two classes)
class_a = np.array([3, 4, 5, 6, 7])
class_b = np.array([1, 2, 3, 4, 5])

# Show what we're starting with
print("Class A IDs:", class_a)
print("Class B IDs:", class_b)
print("\n")
```

OUTPUT:

```
Class A IDs: [3 4 5 6 7]
Class B IDs: [1 2 3 4 5]
```

```
# a. Stack them vertically (like piling lists on top of each other)
vertical_stack = np.vstack((class_a, class_b))
print("Stacked vertically (rows):")
print(vertical_stack)
print("\n")
```

OUTPUT:

```
Stacked vertically (rows):
```

```
[[3 4 5 6 7]
 [1 2 3 4 5]]
```

```
# b. Stack them horizontally (like side-by-side columns)
horizontal_stack = np.hstack((class_a, class_b))
print("Stacked horizontally (one long row):")
print(horizontal_stack)
print("\n")
```

OUTPUT:

```
Stacked horizontally (one long row):
[3 4 5 6 7 1 2 3 4 5]
```

```
# c. Find common IDs between the two classes
common_items = np.intersect1d(class_a, class_b)
print("IDs in both classes:")
print(common_items)
print("\n")
```

```
IDs in both classes:
[3 4 5]
```

```
# d. Remove Class B’s IDs from Class A
unique_to_a = np.setdiff1d(class_a, class_b)
print("IDs only in Class A (not in B):")
print(unique_to_a)
print("\n")
```

```
IDs only in Class A (not in B):
[6 7]
```

14. **Work with following functions in Series and Data frames using Pandas**

Function	Description
df.sort_values('col', ascending=True)	Sorts by a column
df.sort_values(['col1', 'col2'])	Sorts by multiple columns
df.set_index('col')	Sets a column as index
df.reset_index()	Resets index

```
import pandas as pd


# Create a sample dataset of students

students_data = {

        'Name': ['sai', 'deek', 'vaish', 'shri'],

        'Grade': [85, 95, 79, 92],

        'Age': [17, 18, 19, 18]

        }

 # Make it into a DataFrame (like a nice table)

df = pd.DataFrame(students_data)


# Let's see what we have

print("Original table:")

print(df)

print("\n")
```

OUTPUT:

```
Original table:
        Name  Grade  Age
0       sai     85   17
1       deek    95   18
2      vaish    79   19
3      shri     92   18
```

```
# Sort by Grade from highest to lowest
df_sorted_by_grade = df.sort_values('Grade', ascending=False)
print("Sorted by Grade (highest to lowest):")
print(df_sorted_by_grade)
print("\n")
```

OUTPUT:

```
Sorted by Grade (highest to lowest):
        Name  Grade  Age
1       deek    95   18
3       shri    92   18
0       sai     85   17
2        vaish  79   19
```

```
# Sort by Age first, then Grade
df_multi_sort = df.sort_values(['Age', 'Grade'], ascending=[True, False])
```

```
print("Sorted by Age, then Grade:")

print(df_multi_sort)

print("\n")
```

```
OUTPUT:
Sorted by Age, then Grade:
         Name  Grade  Age
0        sai     85   17
1        deek    95   18
3        shri    92   18
2        vaish   79   19
```

```
# Make Name the index (like a label for each row)

df_with_index = df.set_index('Name')

print("With Name as index:")

print(df_with_index)

print("\n")
```

```
OUTPUT:
With Name as index:
         Grade  Age
Name
sai      85   17
deek     95   18
vaish    79   19
```

```
shri       92   18
```

```
# Put the index back to numbers
df_reset = df_with_index.reset_index()
print("Back to normal numbering:")
print(df_reset)
```

OUTPUT:

```
Back to normal numbering:
          Name  Grade  Age
0         sai     85   17
1         deek    95   18
2         vaish   79   19
3          shri   92   18
```

15.df.groupby('col').mean()   Groups by column and calculates mean

```
import pandas as pd
           data={
                'name':['shri','vaish','aksh','deek','sai','lal'],
                 'age':[27,33,27,19,23,35],
                 'pin':[1,2,3,4,5,6]
               }
           df=pd.DataFrame(data)
           print(df.groupby('name').mean())
```

OUTPUT:

```
       age    pin
name
shri   27  1
vaish    33  2
aksh    27  3
```

2→df.groupby('col').agg(['mean', 'sum']) Performs multiple aggregations

```
import pandas as pd
data={
     'name':['shri','vaish','aksh','deek','sai','lal'],
    'age':[27,33,27,19,23,35],
     'pin':[1,2,3,4,5,6]
   }
df=pd.DataFrame(data)
print(df.groupby('name').agg(['mean', 'sum']))
```

OUTPUT:

```
              age              pin
     mean sum  mean sum
name
```

```
shri  27  54  1   2
vaish   33  66   2   4
aksh  47  94  3   6
```

3. df.pivot_table(values='col1', index='col2') Creates a pivot table

```
import pandas as pd
data={
           'name':['shri','vaish','aksh','deek','sai','lal'],
          'age':[27,33,27,19,23,35],
          'pin':[1,2,3,4,5,6]
     }
df=pd.DataFrame(data)
print(df.pivot_table(values='age', index='pin'))
OUTPUT:
       age
pin
1      27
2      33
3      27
```

9-06-25

WORK WITH FILTERING AND CONDITIONAL SELECTION

```
import pandas as pd

data={'Name':['deek','bubbu','rose','hri'],

     'Age':[20,19,19,20],

    'Salary':[20000,30000,40000,50000],

    'City':['Chennai','Chennai','Delhi','Banglore']
}
df=pd.DataFrame(data)
print("Original data:",df)
print("Age\n",df[df['Age']>28])
print("Salary\n",df[df['Salary']>15000])
```

Output:

```
Original data:    Name  Age  Salary      City
0  deek   20   20000   Chennai
1  bubbu   19   30000   Chennai
2  rose   19   40000     Delhi
3   hri   20   50000  Banglore
Age
 Empty DataFrame
Columns: [Name, Age, Salary, City]
Index: []
Salary
    Name  Age  Salary      City
0   deek   20   20000   Chennai
1  bubbu   19   30000   Chennai
2  rose   19   40000     Delhi
3   hri   20   50000  Banglore
```

SORTING AND RECORDING DATA

```
import pandas as pd
```

```
data={'Name':['deek','bubbu','rose','hri'],
     'Age':[20,19,19,20],
    'Salary':[20000,30000,40000,50000],
    'City':['Chennai','Chennai','Delhi','Banglore']
}
df=pd.DataFrame(data)
print("Original data:",df)
print("sorted values:",df.sort_values("Salary",ascending=True))
print(df.set_index('Name'))
print(df.reset_index())
```

Output:

```
Original data:    Name  Age  Salary      City
0   deek   20   20000   Chennai
1  bubbu   19   30000   Chennai
2  rose   19   40000     Delhi
3   hri   20   50000  Banglore
sorted values:    Name  Age  Salary      City
0   deek   20   20000   Chennai
1  bubbu   19   30000   Chennai
2  rose   19   40000     Delhi
3   hri   20   50000  Banglore
      Age  Salary      City
Name
deek    20   20000   Chennai
```

```
bubbu   19   30000   Chennai

rose   19   40000     Delhi

hri    20   50000  Banglore

   index  Name  Age  Salary      City

0      0   deek   20   20000   Chennai

1      1  bubbu   19   30000   Chennai

2      2  rose   19   40000     Delhi

3      3   hri   20   50000  Banglore
```

GROUPING AND AGGREGATIONS

```
import pandas as pd

data={'Name':['deek','bubbu','rose','hri'],

     'Age':[20,19,19,20],

    'Salary':[20000,30000,40000,50000],

    'City':['Chennai','Chennai','Delhi','Banglore']

}

df=pd.DataFrame(data)

print("Original data:",df)

print("group by column and mean:",df.groupby('Name').mean())

print("Multiple Aggregations:",df.groupby('Age').agg(['mean','sum']))

print("Pivot table:",df.pivot_table(values='Age',index='Name'))
```

Output:

```
Original data:    Name  Age  Salary      City
```

```
0   deek   20   20000   Chennai

1  bubbu   19   30000   Chennai

2  rose   19   40000     Delhi

3   hri   20   50000  Banglore

group by column and mean:        Age   Salary

Name

deek   20.0  20000.0

rose  19.0  40000.0

hri   20.0  50000.0

bubbu  19.0  30000.0

Multiple Aggregations:       Salary

        mean    sum

Age

19   35000.0  70000

20   35000.0  70000

Pivot table:       Age

Name

deek    20

rose   19

hri    20

bubbu   19
```

Statical analysis

1) Compute the mean,median,standard deviation of a numpy array.

```
import numpy as np

a=np.array([12,23,34,45])

print('mean\n',np.mean(a))

print('median\n',np.median(a))

print('standard deviation\n',np.std(a))
```

Output:

```
mean

 28.5

median

 28.5

standard deviation

 12.298373876248844
```

2) Find the percentile scores of a numpy array

```
import numpy as np

a=np.random.rand(1,10)

# b=np.array([12,23,34,45])

# matching_rows = np.where(a == b)[0]

print('percentile\n',np.percentile(a,9))
```

OUTPUT:

```
percentile

0.24034489065728132
```

3) Compute the Euclidean distance between two arrays

```
import numpy as np
```

```
a=np.array([1,3])

b=np.array([4,7])

distance = np.linalg.norm(a-b)

print(distance)

Output:

5.0
```

4) Compute the Euclidean distance between two arrays

```
import numpy as np

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

c=np.corrcoef(data[:,0],data[:,1])

print(c)
```

.Implement the following Web API’s methods for any simple applications

A.Get b.Put c.Post d.Update

```
Import requests

url="https://jsonplaceholder.typicode.com/posts/1"

response=requests.get(url)

if response.status_code==200:

   print(response.json())

else:

   print*("Failed to retrieve data:",response.status_code)


#OUTPUT:
```

```
{'userId': 1, 'id': 1, 'title': 'sunt aut facere repellat provident occaecati excepturi optio reprehenderit', 'body': 'quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto'}
```

B.Making a Put request

```
import requests

url="https://jsonplaceholder.typicode.com/posts/1"

update_data={"title":"Update Title"}

response=requests.put(url,json=update_data)

print(response.status_code)

print(response.json())


#OUTPUT:

200

{'title': 'Update Title', 'id': 1}
```

C.Making a Post request

```
import requests

url="https://jsonplaceholder.typicode.com/posts"

data={

  "title":"Interacting with API's",

  "body":"This is a sample API request",

  "user id":1

}
```

```
response=requests.post(url,json=data)
print(response.status_code)
print(response.json())
#OUTPUT:
201
{'title': "Interacting with API's", 'body': 'This is a sample API request', 'user id': 1, 'id': 101}
```

D.Making a Delete request

```
import requests
url="https://jsonplaceholder.typicode.com/posts/1"
response=requests.delete(url)
print(response.status_code)

#OUTPUT:
200
```

Implement sqlite3

```
#step1--> importing library
import sqlite3
print("step1:sqlite3 library imported:\n");
#step2--> connecting to database
conn=sqlite3.connect(r'D:\student_new.db')
cursor=conn.cursor()
print("step2:connected to database'student_new.db'in D drive.\n")
```

```
#step3--> creating a table

cursor.execute("create table if not exsits student(id INTEGER PRIMARY KEY,name TEXT,age INTEGER)")

conn.commit()

print("step3:Table'studens'created successfully(if not already exists)\n")

#step4--> inserting records

cursor.execute("insert into student(name,age)VALUES(?,?)",('bubbu',20))

cursor.execute("insert into student(name,age)VALUES(?,?)",('deek',20))

print("step4:two records inserted into'student'table.\n")

#step5--> displaying records

cursor.execute("SELECT*FROM student")

rows=cursor.fetchall()

print("step5:displaying all records in 'student'table.\n")

for row in rows:

    print(row)

print()

#step6--> adding a new column

cursor.execute("alter table student add column grades TEXT")

conn.commit()

print("step6:column 'grade' added to 'student'table.\n")

#step7--> updating records

cursor.execute("UPDATE student SET age=23 WHERE name='bubbu'")

conn.commit()

print("step7:Updated 'bubbu's age to 21.\n")
```

```
#step8--> deleting a record

cursor.execute("DELETE from student WHERE add name='bubbu'")

conn.commit()

print("step8:Deleted record where name is 'bubbu'\n")

#step9--> Displaying records after deletion

cursor.execute("select * from student")

rows=cursor.fetchall()

print("step9: Displaying records after deletion:")

for row in rows:

    print(row)

print()

#step10--> Drop the table

cursor.execute("DROP TABLE IF EXISTS students")

conn.commit()

print("step10: Dropped the 'student' table.\n")

#step11--> close the connection

conn.close()
```

OUTPUT:

```
step1:sqlite3 library imported:

step2:connected to database'student_new.db'in D drive.

step3:Table'studens'created successfully(if not already exists)

step4:two records inserted into'student'table.

step5:displaying all records in 'student'table.

(1,'bubbu',20)
```

```
(2,’deek’,20)

step6:column 'grade' added to 'student'table.

step7:Updated 'bubbu's age to 21.

step8:Deleted record where name is 'bubbu'

step9: Displaying records after deletion:

(1,’bubbu’,21,none)

(2,’deek’,20,none)

step10: Dropped the 'student' table
```

21.IMPLEMENT THE DATA TRANSFORMATION IN PYTHON

a) Removing duplicates

b) Adding a column

c) Replacing values

d)Renaming axis/index

e) Discretization and binning

f)Detecting and filtering outliers

g) Permutation and random sampling

```
import pandas as pd

import numpy as np

# Sample dataset

data = {'Name': ['Alice', 'Bob', 'Charlie', 'Alice', 'Eve', 'Frank', 'Grace', 'Heidi'],'Age': [25, 30, 35, 25, 29, 45, 50, 22],'Score': [88, 90, 85, 88, 76, 95, 67, 70],'Gender': ['F', 'M', 'M', 'F', 'F', 'M', 'F', 'F']

}

df = pd.DataFrame(data)

print("Original DataFrame:")

print(df)
```

```
# 1. Removing Duplicates
df = df.drop_duplicates()
print("\nAfter Removing Duplicates:")
print(df)
# 2. Adding a Column
df['Pass'] = df['Score'] >= 75
print("\nAfter Adding 'Pass' Column:")
print(df)
# 3. Replacing Values
df['Gender'] = df['Gender'].replace({'F': 'Female', 'M': 'Male'})
print("\nAfter Replacing Gender Values:")
print(df)
# 4. Renaming Axis / Index
df.rename(index={0: 'ID001', 1: 'ID002'}, inplace=True)
print("\nAfter Renaming Index:")
print(df)
# 5. Discretization / Binning Age
df['Age_Group'] = pd.cut(df['Age'], bins=[0, 25, 35, 50], labels=['Young', 'Middle-aged', 'Senior'])
print("\nAfter Binning 'Age' into Age_Group:")
print(df)
# 6. Detecting and Filtering Outliers in Score (IQR Method)
Q1 = df['Score'].quantile(0.25)
Q3 = df['Score'].quantile(0.75)
IQR = Q3 - Q1
filtered_df = df[(df['Score'] >= Q1 - 1.5 * IQR) & (df['Score'] <= Q3 + 1.5 * IQR)]
```

```
print("\nAfter Filtering Outliers in 'Score':")

print(filtered_df)

# 7. Permutation and Random Sampling

N

sample_df = df.sample(n=3, random_state=1)

print("\nShuffled DataFrame:")

print(shuffled_df)

print("\nRandom Sample of 3 Rows:")

print(sample_df)

# 8. Computing Indicator/Dummy Variables

df_with_dummies = pd.get_dummies(df, columns=['Gender'])

print("\nAfter Computing Dummy Variables for 'Gender':")

print(df_with_dummies)
```

OUTPUT:

```
Original DataFrame:

      Name  Age  Score Gender

0    Alice   25     88      F

1      Bob   30     90      M

2  Charlie   35     85      M

3    Alice   25     88      F

4      Eve   29     76      F

5    Frank   45     95      M

6    Grace   50     67      F

7    Heidi   22     70      F
```

```
After Removing Duplicates:

      Name  Age  Score Gender

0    Alice   25     88      F

1      Bob   30     90      M

2  Charlie   35     85      M

4      Eve   29     76      F

5    Frank   45     95      M

6    Grace   50     67      F

7    Heidi   22     70      F


After Adding 'Pass' Column:

      Name  Age  Score Gender   Pass

0    Alice   25     88      F   True

1      Bob   30     90      M   True

2  Charlie   35     85      M   True

4      Eve   29     76      F   True

5    Frank   45     95      M   True

6    Grace   50     67      F  False

7    Heidi   22     70      F  False


After Replacing Gender Values:

      Name  Age  Score  Gender   Pass

0    Alice   25     88  Female   True

1      Bob   30     90    Male   True

2  Charlie   35     85    Male   True
```

```
4     Eve   29    76  Female   True
5    Frank   45    95    Male   True
6    Grace   50    67  Female  False
7    Heidi   22    70  Female  False

After Renaming Index:
         Name  Age  Score  Gender   Pass
ID001    Alice   25    88  Female   True
ID002      Bob   30    90    Male   True
2     Charlie   35    85    Male   True
4         Eve   29    76  Female   True
5       Frank   45    95    Male   True
6       Grace   50    67  Female  False
7       Heidi   22    70  Female  False

After Binning 'Age' into Age_Group:
         Name  Age  Score  Gender   Pass    Age_Group
ID001    Alice   25    88  Female   True        Young
ID002      Bob   30    90    Male   True  Middle-aged
2     Charlie   35    85    Male   True  Middle-aged
4         Eve   29    76  Female   True  Middle-aged
5       Frank   45    95    Male   True       Senior
6       Grace   50    67  Female  False       Senior
7       Heidi   22    70  Female  False        Young
```

```
After Filtering Outliers in 'Score':

         Name  Age  Score  Gender   Pass    Age_Group
ID001    Alice   25     88  Female   True       Young
ID002      Bob   30     90    Male   True  Middle-aged
2      Charlie   35     85    Male   True  Middle-aged
4          Eve   29     76  Female   True  Middle-aged
5        Frank   45     95    Male   True       Senior
6        Grace   50     67  Female  False       Senior
7        Heidi   22     70  Female  False        Young
```

22.Implement String Manipulation Functions in python

```
#1.Case Conversion Methods
text='hello world'
print(text.upper())
print(text.lower())
print(text.title())
print(text.capitalize())
#2.strip(),lstrip(),rstrip()
s=' Python '
print(s.strip())
print(s.lstrip())
print(s.rstrip())
#3.replace()
text='data science'
print(text.replace('data','AI'))
```

```
#split() and join()

sentence='AI,ML,DL'

print(sentence.split(','))

words=['Data','Science']

print(".join(words))

#5.finds() and index()

msg='machine learning'

print(msg.find('learn'))

print(msg.index('learn'))

#6.Startswith() and endswith()

'hello.py'.endswith('.py')

'notebook.ipynb'.startswith('note')

#7.count()

text='banana'

print(text.count('a'))

#8.isalpha(),isdigit(),isalnum(),isspace()

print('abc'.isalpha())

print('123'.isdigit())

print('abc12'.isalnum())

print(".isspace())

#9.zfill()

num='42'

print(num.zfill(5))

#10.format() and f-strings

name='Alice'
```

```
score=95
"Name:{},Score:{}".format(name,score)
f"Name:{name},Score:{score}"
#11.casefold()
"Straße".casefold()=="strasse"
```

output:

```
HELLO WORLD
hello world
Hello World
Hello world
Python
Python
 Python
AI science
['AI', 'ML', 'DL']
DataScience
8
8
3
True
True
True
False
00042
```

23.DATA WRANGLING IN PYTHON:Join,Combine,and Reshape

Hierarchical Indexing Combining and Merging Datasets Reshape

→Hierarchical Indexing

```
import pandas as pd

index=pd.MultiIndex.from_tuples([('USA','NY'),('USA','CA'),('INDIA','DELHI')],names=['Country','city']
)

df=pd.DataFrame({'Population':[8.6,39.5,18.9]},index=index)

print(df)
```

Output:

```
Population

Country city

USA     NY          8.6

       CA         39.5

INDIA   DELHI       18.9
```

→Combing and merging Datasets

```
import pandas as pd

# DataFrames

df1 = pd.DataFrame({'Emp_ID': [1, 2, 3], 'Name': ['Alice', 'Bob', 'Charlie']})

df2 = pd.DataFrame({'Emp_ID': [1, 2, 4], 'Salary': [50000, 60000, 70000]})

print("Original DF1",df1)

print("Original DF2",df2)

#Inner join

result=pd.merge(df1,df2,on='Emp_ID',how='inner')

print("Inner join:\n",result)
```

```
#left join

left_result=pd.merge(df1,df2,on='Emp_ID',how='left')

print("Left join:\n",left_result)

#Right join

right_result=pd.merge(df1,df2,on='Emp_ID',how='right')

print("Right join:\n",right_result)

#Full join

full_join=pd.merge(df1,df2,on='Emp_ID',how='outer')

print("Full join:\n",full_join)
```

Output:

```
Original DF1    Emp_ID     Name

0       1    Alice

1       2      Bob

2       3  Charlie

Original DF2    Emp_ID  Salary

0       1   50000

1       2   60000

2       4   70000

Inner join:

    Emp_ID   Name  Salary

0       1  Alice   50000

1       2    Bob   60000

Left join:

    Emp_ID     Name   Salary

0       1    Alice  50000.0
```

```
1       2      Bob  60000.0
2       3  Charlie      NaN
Right join:
   Emp_ID   Name  Salary
0       1  Alice   50000
1       2    Bob   60000
2       4    NaN   70000
Full join:
   Emp_ID     Name   Salary
0       1    Alice  50000.0
1       2      Bob  60000.0
2       3  Charlie      NaN
3       4      NaN  70000.0
```

```
#1.Dataset Creation
import pandas as pd
data={
'Department':['Sales','Sales','HR','HR','IT','IT','Sales','IT'],
'Employee':['Alice','Bob','Charlie','David','Eve','Frank','Grace','Hank'],
'Salary':[50000,60000,45000,47000,70000,72000,52000,73000],
'Bonus':[5000,6000,4500,4700,7000,7200,5200,7300]}
df=pd.DataFrame(data)
print("Original Data frame\n",df)
#2.Selecting subset of columns
```

```
subset=df[['Department','Salary']]

print("Selecting the group of columns\n",subset)

#3.Grouping Operations

→grouping with dictionary

dept_mapping={'Sales':'Business','HR':'Support','IT':'Technical'}

df['Category']=df['Department'].map(dept_mapping)

grouped_by_category=df.groupby('Category').sum()

print("Category by Group\n",grouped_by_category)

 →#grouping with series

dept_series=pd.Series(dept_mapping)

grouped_by_series=df.groupby(df['Department'].map(dept_series)).sum()

print("Series by Group\n",grouped_by_series)

→#Grouping with Function

import pandas as pd

# Sample DataFrame

data = {

    'Name': ['Alice', 'Bob', 'Charlie', 'David'],

    'Department': ['Sales', 'HR', 'IT', 'Sales'],

    'Salary': [50000, 60000, 55000, 52000],

    'Bonus': [5000, 6000, 4500, 5200]

}

df = pd.DataFrame(data)

→# Define a function to map department to category

def map_department_to_category(dept):
    if dept == 'Sales':
        return 'Business'
    elif dept == 'HR':
        return 'Support'
    elif dept == 'IT':
        return 'Technical'
    else:
        return 'Other'
# Grouping by applying function
grouped_by_function = df.groupby(df['Department'].map(map_department_to_category)).sum()
# Print the result
print("Grouping by Function:\n", grouped_by_function)
#4.Aggregation Functions
#Sample DataFrame
import pandas as pd
data={
'Department':['Sales','Sales','HR','HR','IT','IT','Sales','IT'],
'Employee':['Alice','Bob','Charlie','David','Eve','Frank','Grace','Hank'],
'Salary':[50000,60000,45000,47000,70000,72000,52000,73000],
'Bonus':[5000,6000,4500,4700,7000,7200,5200,7300]}
df=pd.DataFrame(data)
print("Original Data frame\n",df)
print(df.groupby('Department').any())
print(df.groupby('Department').all())
print(df.groupby('Department')['Salary'].cumprod())
print(df.groupby('Department')['Bonus'].cumsum())

print(df.groupby('Department').first())

print(df.groupby('Department').last())

print(df.groupby('Department')['Salary'].mean())

print(df.groupby('Department')['Bonus'].median())

print(df.groupby('Department')['Salary'].min())

print(df.groupby('Department').nth(1))

print(df.groupby('Department')['Salary'].ohlc())

print(df.groupby('Department')['Salary'].prod())

print(df.groupby('Department')['Salary'].quantile(0.25))

print(df.groupby('Department')['Salary'].rank())

print(df.groupby('Department').size())
```

Output:

Original Dta frame

```
  Department Employee  Salary  Bonus
0      Sales    Alice   50000   5000
1      Sales      Bob   60000   6000
2         HR  Charlie   45000   4500
3         HR    David   47000   4700
4         IT      Eve   70000   7000
5         IT    Frank   72000   7200
6      Sales    Grace   52000   5200
7         IT     Hank   73000   7300
```

Selecting the group of columns

```
  Department  Salary
0      Sales   50000
1      Sales   60000
2         HR   45000
3         HR   47000
4         IT   70000
5         IT   72000
6      Sales   52000
7         IT   73000
Category by Group
              Department        Employee  Salary  Bonus
Category
Business   SalesSalesSales  AliceBobGrace  162000  16200
Support             HRHR   CharlieDavid   92000   9200
Technical          ITITIT   EveFrankHank  215000  21500
Series by Group
              Department        Employee  ...  Bonus                  Category
Department                                ...
Business   SalesSalesSales  AliceBobGrace  ...  16200    BusinessBusinessBusiness
Support             HRHR   CharlieDavid  ...   9200            SupportSupport
Technical          ITITIT   EveFrankHank  ...  21500  TechnicalTechnicalTechnical

[3 rows x 5 columns]
Grouping by Function:
             Name  Department  Salary  Bonus
```

```
Department
Business    AliceDavid  SalesSales  102000  10200
Support           Bob        HR   60000   6000
Technical      Charlie         IT   55000   4500
Original Data frame
  Department Employee  Salary  Bonus
0      Sales    Alice   50000   5000
1      Sales      Bob   60000   6000
2         HR  Charlie   45000   4500
3         HR    David   47000   4700
4         IT      Eve   70000   7000
5         IT    Frank   72000   7200
6      Sales    Grace   52000   5200
7         IT     Hank   73000   7300
           Employee  Salary  Bonus
Department
HR             True    True   True
IT             True    True   True
Sales          True    True   True
           Employee  Salary  Bonus
Department
HR             True    True   True
IT             True    True   True
Sales          True    True   True
0           50000
```

```
1         3000000000
2              45000
3         2115000000
4              70000
5         5040000000
6    156000000000000
7    367920000000000
Name: Salary, dtype: int64
0     5000
1    11000
2     4500
3     9200
4     7000
5    14200
6    16200
7    21500
Name: Bonus, dtype: int64
           Employee  Salary  Bonus
Department
HR          Charlie   45000   4500
IT              Eve   70000   7000
Sales         Alice   50000   5000
           Employee  Salary  Bonus
Department
HR            David   47000   4700
```

```
IT           Hank   73000   7300
Sales        Grace   52000   5200
Department
HR      46000.000000
IT      71666.666667
Sales    54000.000000
Name: Salary, dtype: float64
Department
HR      4600.0
IT      7200.0
Sales    5200.0
Name: Bonus, dtype: float64
Department
HR      45000
IT      70000
Sales    50000
Name: Salary, dtype: int64
  Department Employee  Salary  Bonus
1     Sales      Bob   60000   6000
3        HR    David   47000   4700
5        IT    Frank   72000   7200
            open   high    low  close
Department
HR          45000  47000  45000  47000
IT          70000  73000  70000  73000
```

```
Sales       50000  60000  50000  52000

Department

HR           2115000000

IT      367920000000000

Sales    156000000000000

Name: Salary, dtype: int64

Department

HR       45500.0

IT       71000.0

Sales    51000.0

Name: Salary, dtype: float64

0    1.0

1    3.0

2    1.0

3    2.0

4    1.0

5    2.0

6    2.0

7    3.0

Name: Salary, dtype: float64

Department

HR       2

IT       3

Sales    3

dtype: int64
```