# **BASIC PYTHON**

# To print Random number

b= random.randint(10,100,size=5)

**Pattern.**

num\_lines = int(input("Enter the number of lines: \n"))

for i in range(num\_lines, 0, -1):

for j in range(num\_lines - i):

print(" ", end="")

for k in range(i, 0, -1):

print(k, end=" ")

print()

7 6 5 4 3 2 1

6 5 4 3 2 1

5 4 3 2 1

4 3 2 1

3 2 1

2 1

1

**Strong Numbers**

In [2]:

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

if num < 0:

print("Please enter a non-negative integer...")

else:

original\_num = num

sum\_of\_factorials = 0

while num > 0:

digit = num % 10

factorial\_result = 1

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

factorial\_result \*= i

sum\_of\_factorials += factorial\_result

num //= 10

if original\_num == sum\_of\_factorials:

print("The number is a strong number.")

else:

print("The number is not a strong number.")

The number is not a strong number.

**Cum of odd natural numbers between the range of 1 to n both**

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

sum = 0

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

if (i % 2 != 0):

sum += i

print(f"Sum = {sum}")

Sum = 25

**Calculator using arithmetic operations**

def add(x, y):

return x + y

def subtract(x, y):

return x - y

def multiply(x, y):

return x \* y

def divide(x, y):

return x / y

num1 = float(input("Enter first number: \n"))

num2 = float(input("Enter second number: \n"))

operator = input("Enter operator: \n")

if (operator == '+'):

result = add(num1, num2)

print(f"The result is {result}")

elif (operator == '-'):

result = subtract(num1, num2)

print(f"The result is {result}")

elif (operator == '\*'):

result = multiply(num1, num2)

print(f"The result is {result}")

elif (operator == '/'):

result = divide(num1, num2)

print(f"The result is {result}")

else:

print("Invalid operator")

The result is 3.0

**Reverse a given number**

num = int(input("Enter a nuber: \n"))

rev = 0

while (num != 0):

rem = num % 10

rev = rev \* 10 + rem

num = num // 10

print(f"Reversed number is {rev}")

Reversed number is 387

**Fibonacci series**

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

if num == 0:

print("Nothing to print...")

elif num == 1:

print("0")

else:

count = 0

num1 = 0

num2 = 1

print(num1)

print(num2)

while (count != num-2):

result = num1 + num2

print(result)

count += 1

num1 = num2

num2 = result

0

1

1

2

3

**STRINGS:**

**First Character and the Last Character have been Exchanged.**

first = "sushmita"

print(first)

new=first[-1]+first[1:-1]+first[0]

print(new)

**Palindrome or not:**

name=input("enter the input: ")

print("the actual name: "+ name)

rev=name[::-1]

print("reversed string : " + rev)

if name==rev:

    print("the string is palindrome")

else :

        print("its not")

**concate uppercase letter:**

string1=input("enter the first string: ")

string2=input("enter the second string: ")

resstr= string1+string2

res=""

for i in resstr:

    if ( i.isupper()):

        res=res+i

print(res)

**Input will consist of the number of words separated by spaces and possibly commas. There will be a full stop to determine (and print) what percentage the words are more than 6 letters long.**

s=input("enter the string: ")

new=s.split()

long\_word=0

total\_word=0

for word in new:

    if len(s)>6:

        long\_word +=1

    total\_word += 1

    print(total\_word)

percentage =(long\_word/total\_word)\*100

print(f"{percentage:.2f} % of words are more than 6 letters long")

**This si an elpmaxe of tahw is tnaem.**

stringg = "hi hello how are your all"

arr = stringg.split()

new\_str = ""

for i in range(0,len(arr)):

    if i % 2 == 1:

        rev = "".join(sorted(arr[i],reverse = True))

        new\_str = new\_str + rev + " "

    else:

        new\_str= new\_str + arr[i] + " "

print(new\_str)

**number of occurrence of each individual letter in the sentence**

str1 = input("Enter a string: ")

mydict = {}

for i in str1:

    if i in mydict:

        mydict[i] = mydict.get(i) + 1

    else:

        mydict[i] = 1

print(mydict)

**only digits 0's and 1's, if it is possible to make all digits same by flipping only one digit**

n=input("enter: ")

if(n.count("1")==1 or n.count("0")==1):

    print("true")

else:

    print("false")

**Create a dictionary consisting of all the words and its frequency in the given text.**

paragraph = """In a hole in the ground there lived a hobbit.

Not a nasty, dirty, wet hole, filled with the ends of worms and an oozy smell, nor yet a dry, bare,

sandy hole with nothing in it to sit down on or to eat: it was a hobbit-hole, and that means comfort."""

p1=paragraph.split()

for x in p1:

    n=p1.count(x)

    print(x,n)

**Caesar Cipher using Python Dictionary: Caesar Cipher shift =3 means each letter will be replace by 3 letters a->d, e->h, b->e**

alphabet = 'abcdefghijklmnopqrstuvwxyz'

shift = 3

cipher\_dict = {alphabet[i] : alphabet[(i + shift) % 26] for i in range(len(alphabet))}

print(cipher\_dict)

plain\_text = "Hello world. My name is sushmita"

cipher\_text= ''.join(cipher\_dict.get(char, char) for char in plain\_text.lower())

print("Plain text: ", plain\_text)

print("cipher text: ", cipher\_text)

Create set S1 will all even numbers from 1 to 100. Create set S2 with all squares of numbers 1 to 10. Create the following sets from these two sets . Union of S1 and S2 . Intersection of S1 and S2.

Set1 = {i for i in range(2, 101, 2)}

Set2 = {x\*\*2 for x in range(1, 11)} #unordered if want to sort convert to list then sort

Set3 = Set1.union(Set2)

Set4 = Set1.intersection(Set2)

**Create a list in python for storing supermarket bill details and perform the following operations on it:**

bill\_details = ["Bread",2,200,"Milk",1,25,"Bananas",12,60,]

print(bill\_details)

#Add an item to the end of the list

bill\_details.append("egg")

bill\_details.append(12)

bill\_details.append(60)

print(bill\_details)

# Insert an item at a given position

bill\_details.insert(0,"ghee")

print(bill\_details)

#Modify an element by using the index of the element

bill\_details[4]="dal"

print(bill\_details)

#Remove an item from the list

bill\_details.remove("ghee")

print(bill\_details)

#Remove all items from the list

bill\_details.clear()

print(bill\_details)

#Slice Elements from a List

bill\_details = ["Bread",2,200,"Milk",1,25,"Bananas",12,60,60]

print(bill\_details)

print(bill\_details[2:])

#Remove the item at the given position in the list, and return it

print(bill\_details.pop(2))

#Return the number of times 'x' appear in the list

print("Bread occurrence: ",bill\_details.count("Bread"))

print("60 occurrence: ",bill\_details.count(60))

#Sort the items of the list in place

lists=[45,99,63,54,7,455]

lists.sort()

print(lists)

#Reverse the elements of the list in place

print(lists[::-1])

**to find the list in a list of lists whose sum of elements is the highest:**

lists\_lists = [[1, 2, 3], [4, 5, 6], [10, 11, 12], [7, 8, 9]]

sums\_of\_lists = [sum(lst) for lst in lists\_lists]

max\_index = sums\_of\_lists.index(max(sums\_of\_lists))

print(lists\_lists[max\_index])

**FUNCTIONS:**

**Greatest Common Divisor with gcd()**

def gcd(a, b):

if b == 0:

return a

else:

return gcd(b, a % b)

a = 60

b = 24

result = gcd(a, b)

if result:

print('GCD of', a, 'and', b, 'is', result)

else:

print('not found')

GCD of 60 and 24 is 12

function name student with his default information as his standard and school name with provision **that if student wants can change default information**

def student(name, batch='MCA', college='MPSTME'):

print(f"Name: {name}")

print(f"Batch: {batch}")

print(f"College: {college}")

print("\n")

student("Chirag")

student("Ayush","BTech")

student("Atmaja","MBA","NMIMS")

**lamda function to generate square roots**

import math

mylist = [1,2,3,4,5]

newlist = list(map(lambda x : math.sqrt(x),mylist))

print(newlist)

**M1 ka Qts**

furniture = ["SofaSet","DiningTable","TVStand","CupBoard"]

cost = [20000,8500,4599,13920]

def calBill(name,qty):

if name not in furniture:

print(f"Furniture NA")

elif (qty < 0):

print(f"Invalid Quantity")

fin = furniture.index(name)

bill = cost[fin] \* qty

return bill

what = input("Product name: \n")

howmuch = int(input("Product Units: \n"))

purchase = calBill(what,howmuch)

print(f"Total Bill: {purchase}")

**Intersection and Union**

java\_course = {"John", "Jack", "Jill", "Joe"}

python\_course = {"Jake", "John", "Eric", "Jill"}

num\_python\_students = len(python\_course)

print(f"1) Number of students enrolled for Python course: {num\_python\_students}")

python\_only\_students = python\_course - java\_course

print(f"3) **Students enrolled for Python course only**: {python\_only\_students}")

both\_courses\_students = java\_course.intersection(python\_course)

print(f"4) **Students enrolled for both Java and Python courses:** {both\_courses\_students}")

either\_course\_not\_both = (java\_course - python\_course) ^ (python\_course - java\_course)

print(f"5) **Either Java or Python courses but not both**: {either\_course\_not\_both}")

either\_course = java\_course.union(python\_course)

**longest\_consecutive\_sequence**

def longest\_consecutive\_sequence(nums):

if not nums:

return 0

nums.sort()

longest\_streak = current\_streak = 1

for i in range(1, len(nums)):

if nums[i] != nums[i-1]:

if nums[i] == nums[i-1] + 1:

current\_streak += 1

else:

longest\_streak = max(longest\_streak, current\_streak)

current\_streak = 1

return max(longest\_streak, current\_streak)

numbers = [100, 4, 200, 1, 3, 2]

result = longest\_consecutive\_sequence(numbers)

print("Length of the longest consecutive sequence:", result)

**Permutations**

from itertools import permutations

def get\_permutations(nums):

return list(permutations(nums))

numbers = [1, 2, 3]

result = get\_permutations(numbers)

print("All possible permutations:")

for permutation in result:

print(list(permutation))

**Functions EXP**

The “Variety Retail Store” sells different varieties of Furniture to the customers. The list of furniture available with its respective cost is given below:The furniture and its corresponding cost should be stored as a list. A customer can order any furniture in any quantity (the name and quantity of the furniture will be provided). If the required furniture is available in the furniture list(given above) and quantity to be purchased is greater than zero, then bill amount should be calculated. In case of invalid values for furniture required by the customer and quantity to be purchased, display appropriate error message and consider bill amount to be 0. Initialize required furniture and quantity with different values and test the results. Write a Python program to calculate and display the bill amount to be paid by the customer based on the furniture bought and quantity purchased.Hint – Create two different lists for ‘Furniture’ and ‘Cost’. Indices of two lists should be matched to retrieve the cost of a particular furniture.urniture.

cost = [20000, 8500, 4599, 13920]

[3]:

furniture

=

[

"

Sofa set

"

,

"

Dining Table

"

,

"

Tv Stand

"

,

"

Cupboard

"

]

**def**

calculate\_bill

(

name, qty

):

**if**

name

**not**

**in**

furniture:

print

(

"

Furniture unavailable

"

)

**return**

0

**else**

:

index

=

furniture

.

index(name)

**if**

qty

<

=

0

:

print

(

"

Qty undefined

"

)

**return**

0

**else**

:

bill

=

cost[index]

\*

qty

**return**

bill

furniture\_name

=

input

(

"

Enter furniture name:

"

)

quantity

=

int

(

input

(

"

Enter qty:

"

))

print

(

"

Your Bill is:

"

, calculate\_bill(furniture\_name, quantity))

Enter furniture name: Sofa set

Enter qty: 2

Your Bill is: 40000

Consider the list of courses opted by a Student “John” and available electives at ABC Training Institute: courses = (“Python Programming”, “RDBMS”, “Web Technology”, “Software Engg.”)electives = (“Business Intelligence”, “Big Data Analytics”) Write a Python Program to satisfy business requirements mentioned below: 1 List the number of courses opted by John. 2 List all the courses opted by John. 3 John is also interested in elective courses mentioned above. Print the updated tuple including electives.

[6]:

courses

=

(

"

Python Programming

"

,

"

RDBMS

"

,

"

Web Technology

"

,

"

Software Engg.

"

)

electives

=

(

"

Business Intelligence

"

,

"

Big Data Analytics

"

)

**def**

list\_courses

(

courses

):

print

(

"

The number of courses opted by John:

"

,

len

(

courses

))

print

(

"

The courses opted by John:

"

, courses)

**def**

add\_electives

(

courses, electives

):

update

=

courses

+

electives

**return**

update

list\_courses(courses)

courses

=

add\_electives(courses, electives)

list\_courses(courses)

*# Corrected syntax here*

The number of courses opted by John: 4

The courses opted by John: ('Python Programming', 'RDBMS', 'Web Technology',

'Software Engg.')

The number of courses opted by John: 6

The courses opted by John: ('Python Programming', 'RDBMS', 'Web Technology', 'Software Engg.', 'Business Intelligence', 'Big Data Analytics')

Consider a scenario from ABC Training Institute. The given table shows the marks scored by students of grade XI in Python Programming course. Write a Python program to meet the requirements mentioned below:a. Display the name and marks for every student.b. Display the top two scorers for the course.c. Display class average of this course. Hint- Implement the solution using a dictionary.

[7]:

marks\_dict

=

{

"

John

"

:

85

,

"

Mike

"

:

90

,

"

Lisa

"

:

88

,

"

David

"

:

95

,

"

Rita

"

:

89

,

␣

↪

"

Anna

"

:

81

}

**def**

display\_student\_marks

(

marks\_dict

):

**for**

name, marks

**in**

marks\_dict

.

items():

print

(

f

"

**{**

name

**}**

:

**{**

marks

**}**

"

)

**def**

display\_top\_scorers

(

marks\_dict, top

=

2

):

sorted\_marks

=

sorted

(

marks\_dict

.

items(), key

=

**lambda**

x: x[

1

]

, reverse

=

**True**

)

top\_scorers

=

sorted\_marks[:top]

**for**

name, marks

**in**

top\_scorers:

print

(

f

"

**{**

name

**}**

:

**{**

marks

**}**

"

)

**def**

calculate\_class\_average

(

marks\_dict

):

total\_marks

=

sum

(

marks\_dict

.

values())

class\_average

=

total\_marks

/

len

(

marks\_dict

)

**return**

class\_average

**def**

display\_class\_average

(

class\_average

):

print

(

f

"

Class average:

**{**

class\_average

**}**

"

)

display\_student\_marks(marks\_dict)

display\_top\_scorers(marks\_dict)

class\_average

=

calculate\_class\_average(marks\_dict)

display\_class\_average(class\_average)

John: 85

Mike: 90

Lisa: 88

David: 95

Rita: 89

Anna: 81

David: 95

Mike: 90

Class average: 88.0

Consider a scenario from ABC Training Institute. Given below are two Sets representing the names of students enrolled for a particular course:java\_course = {“John”, “Jack”, “Jill”, “Joe”}python\_course = {“Jake”, “John”, “Eric”, “Jill”} Write a Python program to list the number of students enrolled for: 1)Python course 2)Java course only 3)Python course only 4)Both Java and Python courses 5)Either Java or Python courses but not both 6)Either Java or Python courses

[8]: java\_course = {"John", "Jack", "Jill", "Joe"} python\_course = {"Jake", "John", "Eric", "Jill"}

**def** get\_enrolled\_students(course):

**return** len(course)

python\_course\_only = len(python\_course - java\_course) java\_course\_only = len(java\_course - python\_course) both\_courses = len(java\_course & python\_course) either\_but\_not\_both = len(java\_course ^ python\_course) either\_course = len(java\_course | python\_course)

print("Python course only:", python\_course\_only) print("Java course only:", java\_course\_only) print("Both Java and Python courses:", both\_courses) print("Either Java or Python courses but not both:", either\_but\_not\_both) print("Either Java or Python courses:", either\_course)

Python course only: 2

Java course only: 2

Both Java and Python courses: 2

Either Java or Python courses but not both: 4

Either Java or Python courses: 6

Given a Number in the form of a List of digits , return all possible permutations. For example

given [1,2,3] return [1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]

[10]:

**def**

perm

(

start, end

=

[]):

**if**

len

(

start

)

==

0

:

print

(

end

)

**else**

:

**for**

i

**in**

range

(

len

(

start

)):

perm(start[:i]

+

start[i

+

1

:]

, end

+

start[i:i

+

1

])

perm([

1

,

2

,

3

])

[1

, 2,

3]

[1, 3, 2]

[2, 1, 3]

[2, 3, 1]

[3, 1, 2]

[3, 2, 1]

Given an unsorted array of integers find the length of the longest consecutive sequence of elements. For example given [100,4,200,1,3,2]. The Longest consecutive sequence is [1,2,3,4] return the length as 4.

[11]:

**def**

longest\_consecutive\_sequence

(

nums

):

**if**

**not**

nums:

**return**

0

nums\_set

=

set

(

nums

)

longest\_length

=

0

**for**

num

**in**

nums\_set:

**if**

num

-

1

**not**

**in**

nums\_set:

current\_num

=

num

current\_length

=

1

**while**

current\_num

+

1

**in**

nums\_set:

current\_num

+

=

1

current\_length

+

=

1

longest\_length

=

max

(

longest\_length, current\_length

)

**return**

longest\_length

nums

=

[

100

,

4

,

200

,

1

,

3

,

2

]

result

=

longest\_consecutive\_sequence(nums)

print

(

result

)

4

1. Calculate the Greatest Common Divisor with gcd() function and by writing your own function
2. Write two functions sub and mul.
3. mul function will take any number,multiply it by 3 and return.
4. sub function will take an integer as an input, multiply that number by 3 by calling mul function, subtract 2 from it and return the value
5. Create a function name student with his default information as his standard and school name with provision that if student wants can change default information iv. Create a lamda function to generate square roots for the numbers given in the list

[12]:

**import**

**math**

**def**

gcd\_with\_builtin

(

x, y

):

**return**

math

.

gcd(x, y)

result

=

gcd\_with\_builtin(

24

,

36

)

print

(

"

GCD with builtin function:

"

, result)

GCD with builtin function: 12

[13]:

**def**

gcd\_custom

(

x, y

):

**while**

(

y

):

x, y

=

y, x

%

y

**return**

x

result

=

gcd\_custom(

24

,

36

)

print

(

"

GCD with custom function:

"

, result)

GCD with custom function: 12

[14]:

**def**

mul

(

num

):

**return**

num

\*

3

result

=

mul(

5

)

print

(

"

Multiplication:

"

, result)

Multiplication: 15

[15]:

**def**

sub

(

num

):

**return**

mul(num)

-

2

*# Example usage:*

result

=

sub(

5

)

print

(

"

Subtraction:

"

, result)

Subtraction: 13

[16]: **def** student(name, standard="10th", school="ABC School"): **return** f"Name: **{**name**}**, Standard: **{**standard**}**, School: **{**school**}**"

*# Example usage:* print("Default student information:", student("John")) print("Updated student information:", student("Alice", "12th", "XYZ School"))

Default student information: Name: John, Standard: 10th, School: ABC School

Updated student information: Name: Alice, Standard: 12th, School: XYZ School

[17]:

**import**

**math**

numbers

=

[

4

,

9

,

16

,

25

]

square\_roots

=

list

(

map

(

**lambda**

x: math

.

sqrt(x), numbers))

*# Example usage:*

print

(

"

Square roots of numbers:

"

, square\_roots)

Square roots of numbers: [2.0, 3.0, 4.0, 5.0]

[ ]:

**DICTIONARY BASICS:**#create dict mehtod 1

dict\_1={"rohan":5,"mahesh":7}#creating

dict\_2={"mahesh":7,"rohan":5}

dict\_1==dict\_2

dict\_1["mahesh"]#accessing

#creating dict using method 2

my\_class=dict({"geeta":29,"reena":30,"priya":14})

print(my\_class)

my\_fruit=dict([(1,"apple"),(2,"mango")])

print(my\_fruit)

dict\_new={1:("red","blue"),(2,3): "green","my\_list":[1,2,3,4]}

print(dict\_new)

#accessing dict

dict1={"india":30,"russia":35,"britain":25}

print(dict1)

for key in dict1.keys():

    print(key)

print(dict1.keys())

print(dict1.items())

for x in dict1.items():

    print(x)

for x in dict1.items():

    print(x[1])#print only valuees

for x in dict1.items():

    print(x)#print individual key-value pair

for x in dict1:

    print(x)

for x in dict1:

    print(x[1])

dict\_new[1][1]

#accesig list value in dict by slicing

dict\_new["my\_list"][1:]

#change delete add dict value

my\_veg={1:"onion",2:"carrot",3:"mirchi"}

print(my\_veg)

my\_veg[2]="cabbage"

print(my\_veg)

my\_veg.update({3:"pepper"})

print(my\_veg)

my\_veg.pop(2)

print(my\_veg)

my\_veg.clear()

#nested dict:

student={1:{"name":"jay","subject":"math","marks":88},2:{"name":"jash","subject":"math","marks":89}}

student[1]["marks"]

dict1={1:10,2:20}

dict2={3:30,4:40}

dict3={}

for d in (dict1,dict2):dict3.update(d)

print(dict3)

**ARRAY, TUPLES, SET BASICS:**Array

colours= ["red","green"]

colours.append("Blue")

print(colours)

colours.extend("white")

print(colours)

colours[1]= "purple"

print(colours)

stationery=["pencil", "eraser", "scale", "papers", "gluestick", "sharpner"]

print(stationery[0:3])

vowels= ["a", "e", "i"]

vowels=vowels+["o","u"]

print(vowels)

vowels = 2\*vowels

print(vowels)

Tuples

my\_tuple= ("banana", "apple", "grapes")#packing tuple

print(my\_tuple)

print(type(my\_tuple))

my\_tuple= ("banana", "apple", "grapes")#packing tuple

print(my\_tuple)

print(type(my\_tuple))

my\_tuple.count("banana")

my\_tuple.index("grapes")

(a,b,c)=my\_tuple

print(my\_tuple)

print(b)

new= ("food",("shirts", "jeans"), (("sprite", "coke"),"mangola"))

print(new[1])

print(new[1][1])

print(new[2][0][1])

# mix\_tuple=(1,"vegetables", ["yellow","banana"])

# print(mix\_tuple)

# print(type(mix\_tuple))

# mix\_tuple.index("yellow")

mix\_tuple = (1, "vegetables", ["yellow", "banana"])

print(mix\_tuple)

print(type(mix\_tuple))

mix\_list = list(mix\_tuple)

index\_of\_yellow = mix\_list.index(["yellow", "banana"])

print(index\_of\_yellow)

names=("jiya","jeet", "aehan", "Aarya", "ahaan")

for name in names:

    n= len(name)

    print(name, n)

SET

#convert string to set

#string1="good morning india"

#set1= set(string1)

#print(set1)

#print(set1[4])

#convert string to set

string1 = "good morning india"

set1 = set(string1)

print(set1)

string1 = "good morning india"

set1 = set(string1)

list1 = list(set1)

print(list1)

print(list1[4])

Dictionary

dict\_1= {"Rohan": 5, "Mahesh":7}

dict\_2= {"Mahesh": 7, "Rohan":5}

print(dict\_1)

dict\_new = {1: ("Red", "Blue"), (2, 3): "Green", "My\_list": None}

print(dict\_new)

dict1={"India":30,"Russia":35}

print(dict1)

print(dict1.keys())

print(dict1.items)

dict1["Russia"]

my\_veg={1: "onions",2: "peas", 3:"carrots"}

print(my\_veg)

my\_veg.update({3: "pepper"})

print(my\_veg)

my\_veg[2]="cabbage"

print(my\_veg)

#DIY set of symmetric difference

d ={1,2,3,9,10,11}

c ={1,2,3,4,5,6,7}

d.symmetric\_difference("c")

#check if set is subset of other

a={1,2,3,4}

b={1,2,3,4,5}

print(a.issubset(b))

DIY set Intersection

a={1,2,3}

b={1,2,3,4,5,6}

a.intersection("b")

**DATA Analysis:**

Import numpy as np

Import pandas as pd

Import matplotlib.pyplot as plt

Import seaborn as sns

**Data = pd.read\_csv(‘--------’)**

**Df = pd.DataFrame(data)**

1. **df.head()**
2. **df.tail()**
3. **df.shape**
4. **df.columns**
5. **df.dtypes**
6. **df.info() --- to print metadata**
7. **df.describe()—to print statistics**
8. **Printing numerical and categorical columns:**

cat\_col = [col **for** col **in** df.columns **if** df[col].dtype==object] print('Caregotical columns:',cat\_col)

num\_col = [col **for** col **in** df.columns **if** df[col].dtype!=object]

print('Non-caregotical',num\_col)

1. **Printing Unique Values:**



1. **df.duplicated()**
2. **df.duplicated().sum()**
3. **df.isna()**
4. **df.isnull()**
5. **df.isnull().sum()**
6. **Dropping a column: df.drop('Cabin',axis=1, inplace=True)**
7. **Filling with mean:**

mean\_age = df['Age'].mean()

mean\_age

df['Age'].fillna(mean\_age, inplace=True)

1. **Dropping:** df.dropna(subset='Embarked', inplace=True)
2. **Group by Example:**

survivors\_by\_sex = df.groupby('Sex')['Survived'].sum().reset\_index()

survivors\_by\_sex

1. **Plotting(Seaborn):**

**BarPlot:**

sns.barplot(x='Sex', y='Survived', data=survivors\_by\_sex)

plt.xlabel('Sex')

plt.ylabel('Number of Survivors')

plt.title('Number of Survivors by Gender')

plt.show()

**Pairplot:**

sns.pairplot (x='Sex', y='Survived', data=survivors\_by\_sex)

plt.xlabel('Sex')

plt.ylabel('Number of Survivors')

plt.title('Number of Survivors by Gender')

plt.show()

**Boxplot**

sns.boxplot(x='Sex', y='Age', data=df)

**ViolinPlot:**

Sns.violinplot(x=’Sex’, y=’Age’, data=df)

**Countplot:**

sns.countplot(x='Sex', data=df)

**Plotting with hue:**

survivors\_by\_sex = df.groupby(['Sex', 'Pclass'])['Survived'].sum ().reset\_index()

sns.barplot(x='Sex', hue='Pclass', y='Survived', data=survivors\_by\_sex)

**Joint Plot:**

survivors\_by\_sex = df.groupby('Sex')['Survived'].sum().reset\_index()

sns.jointplot(x='Sex', y='Survived', data=df, kind='scatter')

**lmplot:**

survivors\_by\_sex = df.groupby('Age')['Survived'].sum().reset\_index()

sns.lmplot(x='Age', y='Survived', data=survivors\_by\_sex)

**Heatmap:**

Sns.heatmap(df.corr())

**Clustermap:**

sns.clustermap(df.corr())

1. **Creating a correlation Matrix with numerical and categorical values:**

import numpy as np

numerical\_cols = df.select\_dtypes(include=np.number)

categorical\_cols = df.select\_dtypes(include='object')

numerical\_corr = numerical\_cols.corr()

encoded\_categorical = pd.get\_dummies(categorical\_cols)

concated = pd.concat([numerical\_corr, encoded\_categorical])

correlation\_matrix = concated.corr()

correlation\_matrix

1. **Correlation Positive or negative:**

A screenshot of a computer

Description automatically generated

1. **CrossTab with Bar Plot :**

pd.crosstab(df['Sex'], df['Survived']).plot.bar()

1. **CrossTab with Stacked Bar Plot**:

pd.crosstab(df['Sex'], df['Survived']).plot.bar(stacked=True)

1. **Plotting(Matplotlib):**

**Barplot:**

import matplotlib.pyplot as plt

x=df['Survived'].sum()

y=df['Sex']

plt.barh(y,x)

plt.show()

**Histogram:**

data = df['Age']

sns.histplot(data, bins=10, kde=False) # Adjust the number of bins as needed

plt.title('Histogram of Your Data')

plt.xlabel('Values')

plt.ylabel('Frequency')

plt.show()

**NUMPY:**

**Loading Image:**

from PIL import Image

image=Image.open('Image1.png')

print(image.format)

print(image.size)

print(image)

**Converting image to numpy array:**

from numpy import asarray

img\_data=asarray(image)

print(type(img\_data))

print(img\_data.shape)

print(img\_data)

**Histogram of Image Data:**

import matplotlib.pyplot as plt

plt.hist(img\_data)

A white screen with black and red text

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated



]: **for** i, part **in**␣ enumerate([blocks\_array\_divided\_part1[0],blocks\_array\_d ivided\_part1[1], ␣

blocks\_array\_divided\_part2[0],blocks\_array\_divided\_part

2[1]]): print(f"Cluster **{**i+1**}**:**\n{**part**}\n**")

**PANDAS:**

1. **Create data frames from data structures (dictionary, list, tuple)**

import pandas as pd

my\_list = ['Shrinidhi', 'Shriya', 'Pranav', 'Sanjay', 'Suman'] df\_list = pd.DataFrame(my\_list, index=[1, 2, 3, 4, 5], columns=["Names"]) print("\nDataframe using List\n", df\_list, '\n')

my\_dict = {'Name': ['Shrinidhi', 'Shriya', 'Pranav', 'Sanjay', 'Suman'], "Age": [21, 18, 15, 46,

44]} df\_dict = pd.DataFrame(my\_dict, index=[1, 2, 3, 4, 5]) print("\nDataframe using Dictionary\n", df\_dict, '\n')

my\_tuple = (85, 88, 75, 64, 94) df\_tuple = pd.DataFrame([my\_tuple], index=[1], columns=["Marks"]) print("Dataframe using Tuple\n", df\_tuple)

1. **Create data frames from unformatted text file.**

import pandas as pd

file1 = open('C:/Users/mpstme.student/Desktop/A022\_dap/faculty\_data.txt') df\_faculty\_data = pd.DataFrame()

name = []

position = [] dept = [] degree = [] exp = [] address = []

i = 0 for line in file1: if i == 0 or i % 6 == 0:

name.append(line.replace('\n', '')) elif i % 6 == 1:

position.append(line.replace('\n', '')) elif i % 6 == 2:

dept.append(line.replace('\n', '')) elif i % 6 == 3:

degree.append(line.replace('\n', '')) elif i % 6 == 4:

exp.append(line.replace('\n', '')) elif i % 6 == 5:

address.append(line.replace('\n', '')) i += 1

df\_faculty\_data['Name'] = name df\_faculty\_data['Position'] = position df\_faculty\_data['Department'] = dept df\_faculty\_data['Degree'] = degree df\_faculty\_data['Experience'] = exp df\_faculty\_data['Address'] = address

print(df\_faculty\_data) print(df\_faculty\_data.head())

A screenshot of a computer

Description automatically generated

**EDA QUESTIONS:**

1. **What is the maximum score of an employee in each of the department?**

df\_excel.groupby('dept')['sport score'].max() print("Maximum sport scores by department:")

print(max\_scores)

1. **What is name of the employee who's age is minimum in each of the cities?**

df\_excel.loc[df\_excel.groupby('city')['age'].idxmin()] print("\nYoungest employee by city:") print(minimum\_employee[['name', 'city', 'age']])

1. **Does all the departments have participation from Devashish age group between 20 to 30?**

participation = df\_excel[(df\_excel['age'].between(20, 30))]['dept'].nunique() == df\_excel['dept'].nunique() print("\nParticipation of all departments by employees aged 20-30:", participation)

1. **What is the average score of an employee in each department and city?**

df\_excel.groupby(['dept', 'city'])['sport score'].mean() print("\nAverage sport scores by department and city:") print(avg\_scores)

1. **What is the designation of the youngest player who score highest in cricket?**

df\_excel[df\_excel['sport'] == 'cricket'].groupby('designation')['age'].min() print("\nYoungest cricket player by designation:") print(youngest\_cricket\_player)

1. **What is the city and department of the eldest player who score highest in any of the sport?**

df\_excel.nsmallest(1, 'sport score')[['id', 'city', 'dept']].values[0] print("\nDetails of the person with the smallest sport score (eldest player):") print("ID:", eldest\_player[0], "City:", eldest\_player[1], "Department:", eldest\_player[2])

1. **Which department performed best on the basis of average score city wise**

best\_department = df\_excel.groupby(['dept', 'city'])['sport score'].mean().reset\_index().sort\_values('sport score', ascending=False)['dept'].values[0] print("\nBest department by average sport score:") print(best\_department)

**NORMALIZATION**

import pandas as pd

# Load the data from Excel

data = pd.read\_excel(r'Emp\_info.xlsx')

# Select only numeric columns from the dataset numeric\_columns = data.select\_dtypes(include=['number'])

# Maximum Absolute Scaling

# Divide each element by the maximum absolute value in its column max\_abs\_scaled\_data = numeric\_columns.abs().div(numeric\_columns.abs().max())

# Min-Max Scaling

# Normalizes or scales the data between 0 and 1

min\_max\_scaled\_data = (numeric\_columns - numeric\_columns.min()) / (numeric\_columns.max() - numeric\_columns.min())

# Z-Score Scaling

# Standardizes data (mean = 0 and variance = 1) z\_score\_scaled\_data = (numeric\_columns - numeric\_columns.mean()) / numeric\_columns.std()

# Save the scaled data to CSV files max\_abs\_scaled\_data.to\_csv("max\_abs\_scaled\_Emp\_info.csv", index=False) min\_max\_scaled\_data.to\_csv("min\_max\_scaled\_Emp\_info.csv", index=False) z\_score\_scaled\_data.to\_csv("z\_score\_scaled\_Emp\_info.csv", index=False)

**SQLite**

1. **Crud:**

import sqlite3

conn = sqlite3.connect('newdb')

c = conn.cursor()

***CREATE***

c.execute('''CREATE TABLE IF NOT EXISTS students\_1(

name TEXT,

roll\_no integer,

sap\_id integer,

program TEXT,

semester integer,

marks integer,

result TEXT)''')

***INSERT***

for i in range(0,1):

name = input("Name: ")

roll\_no = int(input("Roll No: "))

sap\_id = int(input('Sap Id: '))

program = input('Program: ')

semester = int(input('Semester: '))

marks = int(input('Marks: '))

result = input("Pass/Fail: ")

c.execute("INSERT INTO students\_1 VALUES(?,?,?,?,?,?,?)", (name,roll\_no,sap\_id,program,semester,marks,result))

c.execute("SELECT \* FROM students\_1")

for row in c.fetchall():

print(row)

c.execute("SELECT \* FROM students\_1 where name = 'A'")

for row in c.fetchall():

print(row)

***UPDATE***

c.execute("UPDATE students\_1 SET name='ABC' WHERE name='A'")

c.execute("SELECT \* from students\_1")

for row in c.fetchall():

print(row)

***DELETE***

c.execute("DELETE FROM students\_1 WHERE sap\_id = 4444 ")

c.execute("SELECT \* FROM students\_1")

for row in c.fetchall():

print(row)

1. **TextFile To df To db**

# Task 2 (text to df to db)

import pandas as pd

import sqlite3

# Provide the file path to the Downloads directory

fn = "C:/Users/Kshamta.M/Downloads/faculty\_data1.txt"

file1 = open(fn,"r")

data = file1.read()

print(data)

# reading entire file as list

file1 = open(fn,"r")

fd1 = file1.readlines()

print(fd1)

# Open the file and read its contents

file1 = open(fn, "r")

fd2 = file1.readlines()

# Close the file

file1.close()

# Divide the lines of the file into separate lists

f1 = fd2[0:4]

f2 = fd2[4:8]

f3 = fd2[8:12]

f4 = fd2[12:16]

f5 = fd2[16:20]

f6 = fd2[20:24]

f7=fd2[24:]

# Strip each line to remove leading and trailing whitespace

f1 = [line.strip() for line in f1]

f2 = [line.strip() for line in f2]

f3 = [line.strip() for line in f3]

f4 = [line.strip() for line in f4]

f5 = [line.strip() for line in f5]

f6 = [line.strip() for line in f6]

f7 = [line.strip() for line in f7]

# Print the divided lists

print("f1:", f1)

print("f2:", f2)

print("f3:", f3)

print("f4:", f4)

print("f5:", f5)

print("f6:", f6)

print("f7:",f7)

# converting list into dictionary

col = ["Name","Designation","Department","Qualification"]

faculty\_dict = dict(zip(col, zip(f1,f2,f3,f4,f5,f6,f7)))

print(faculty\_dict)

import pandas as pd1

faculty\_data = pd1.DataFrame(faculty\_dict)

print(faculty\_data)

import sqlite3

# Connect to SQLite database named 'faculty.db'. Creates the file if it doesn't exist

conn = sqlite3.connect("faculty.db")

# Store DataFrame into SQLite database

faculty\_data.to\_sql('faculty', conn, if\_exists='replace', index=False)

# Retrieve data from the 'faculty' table in the database and store it in a pandas DataFrame

data = pd.read\_sql\_query("SELECT \* FROM faculty", conn)

# Print the retrieved data

print(data)

# Close the database connection

conn.close()

**CRUD Alternate:**

#Task 1 - alternate

import sqlite3

# Step 1: Create a database for student with specified attributes

conn = sqlite3.connect('student.db')

c = conn.cursor()

c.execute('''CREATE TABLE IF NOT EXISTS students

(name TEXT, roll\_number TEXT, sap\_id TEXT, program TEXT, semester INTEGER, marks REAL, result TEXT)''')

conn.commit()

print("Step 1: Database created successfully.")

# Step 2: Insert data for 5 students

students\_data = [

('John Doe', 'R001', '123456', 'Computer Science', 2, 85.5, 'Pass'),

('Jane Smith', 'R002', '789012', 'Electrical Engineering', 3, 75.0, 'Pass'),

('Alice Johnson', 'R003', '345678', 'Mechanical Engineering', 4, 60.0, 'Fail'),

('Bob Brown', 'R004', '901234', 'Civil Engineering', 2, 92.0, 'Pass'),

('Eve Wilson', 'R005', '567890', 'Chemical Engineering', 3, 70.5, 'Pass')

]

c.executemany('INSERT INTO students VALUES (?,?,?,?,?,?,?)', students\_data)

conn.commit()

print("Step 2: Data inserted successfully.")

# Step 3: Search students with name and sap id

search\_name = 'John Doe'

search\_sap\_id = '567890'

c.execute('SELECT \* FROM students WHERE name=? OR sap\_id=?', (search\_name, search\_sap\_id))

search\_results = c.fetchall()

print("Step 3: Search Results:")

for result in search\_results:

print(result)

# Step 4: Update marks with Name and sap id for a student

new\_marks = 90.0

c.execute('UPDATE students SET marks=? WHERE name=? OR sap\_id=?', (new\_marks, search\_name, search\_sap\_id))

conn.commit()

print("Step 4: Marks updated successfully.")

# Step 5: Delete the data of student with sap id

delete\_sap\_id = '789012'

c.execute('DELETE FROM students WHERE sap\_id=?', (delete\_sap\_id,))

conn.commit()

print("Step 5: Data deleted successfully.")

# Close connection

conn.close()

print("Connection closed.")

**TextFile to df to db Alternate:**

#Task 2 - alternate

import sqlite3

import pandas as pd

# Step 1: Read the file to extract the data

file\_path = "faculty\_data1.txt"

with open(file\_path, "r") as file:

lines = file.readlines()

if lines:

print("Step 1: File reading completed successfully.")

# Optionally, you can print the first few lines to verify the data

print("First few lines of data:")

for line in lines[:5]:

print(line.strip())

else:

print("Step 1: Error occurred while reading the file.")

# Step 2: Create DataFrame and database table using this file

df = pd.DataFrame(columns=['Name', 'Designation', 'Department', 'Qualification'])

for i in range(0, len(lines), 4):

name = lines[i].strip()

designation = lines[i+1].strip()

department = lines[i+2].strip()

qualification = lines[i+3].strip()

df = pd.concat([df, pd.DataFrame({'Name': [name], 'Designation': [designation], 'Department': [department], 'Qualification': [qualification]})], ignore\_index=True)

# Create SQLite database and table

conn = sqlite3.connect('faculty.db')

df.to\_sql('faculty', conn, if\_exists='replace', index=False)

conn.close()

print("Step 2: Database table created successfully.")

# Step 3: Display all the contents of the database

conn = sqlite3.connect('faculty.db')

c = conn.cursor()

c.execute('SELECT \* FROM faculty')

all\_rows = c.fetchall()

conn.close()

print("Step 3: Displaying all contents of the database:")

for row in all\_rows:

print(row)