**DATE :**

**EX.NO :**

**AIM:**

**FLOWCHART:**

**CODING:**

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

if num % 2 == 0:

print(f"{num} is even.")

else:

print(f"{num} is odd.")

start = int(input("Enter start of range: "))

end = int(input("Enter end of range: "))

for num in range(start, end + 1):

if num > 1:

for i in range(2, int(num \*\* 0.5) + 1):

if num % i == 0:

break

else:

print(num, end=" ")

print()

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

fact = 1

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

fact \*= i

print(f"Factorial of {num} is {fact}")

choice = input("Convert to (C)elsius or (F)ahrenheit? ").strip().lower()

if choice == 'c':

f = float(input("Enter temperature in Fahrenheit: "))

c = (f - 32) \* 5 / 9

print(f"{f}°F is {c:.2f}°C")

elif choice == 'f':

c = float(input("Enter temperature in Celsius: "))

f = (c \* 9 / 5) + 32

print(f"{c}°C is {f:.2f}°F")

else:

print("Invalid choice")

**OUTPUT:**

Enter a number: 8

8 is even.

Enter start of range: 10

Enter end of range: 20

11 13 17 19

Enter a number: 5

Factorial of 5 is 120

Convert to (C)elsius or (F)ahrenheit? c

Enter temperature in Fahrenheit: 98

98.0°F is 36.67°C

Convert to (C)elsius or (F)ahrenheit? f

Enter temperature in Celsius: 37

37.0°C is 98.60°F

**RESULT:**

**DATE :**

**EX.NO :**

**AIM:**

**FLOWCHART:**

**CODING:**

def isEven(n):

return n % 2 == 0

def isPrime(n):

for i in range(2, int(n\*\*0.5) + 1):

if n % i == 0:

return False

return True

def fact(n):

if n == 0:

return 1

else:

return n \* fact(n-1)

def ctf():

c = float(input("Enter a Celsius: "))

print(f"{c}°C is equal to {(c \* 9/5) + 32:.2f}°F")

def ftc():

f = float(input("Enter a Fahrenheit: "))

print(f"{f}°F is equal to {(f - 32) \* 5/9:.2f}°C")

while True:

print("Options:\n1. Check if a number is even\n2. Check if a number is prime\n3. Calculate factorial\n4. Convert Celsius to Fahrenheit\nothers to exit")

choice = int(input("Enter your choice: "))

if choice == 1:

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

print(f"{n} is {'Even' if isEven(n) else 'Odd'}")

elif choice == 2:

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

print(f"{n} is prime: {'Yes' if isPrime(n) else 'No'}")

elif choice == 3:

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

print(f"The factorial of {n} is: {fact(n)}")

elif choice == 4:

print("Choose conversion type:\n1. Celsius to Fahrenheit\n2. Fahrenheit to Celsius")

conversion = int(input("Enter your choice: "))

if conversion == 1:

ctf()

elif conversion == 2:

ftc()

else:

print("Invalid choice")

else:

print("Exiting...")

break

**OUTPUT:**

Options:

1. Check if a number is even

2. Check if a number is prime

3. Calculate factorial

4. Convert Celsius to Fahrenheit

others to exit

Enter your choice: 1

Enter a number: 10

10 is Even

**RESULT:**

**DATE :**

**EX.NO :**

**AIM:**

**FLOWCHART:**

**CODING:**

text = input("Enter a string: ")

print(f"\nUppercase: {text.upper()}")

print(f"Lowercase: {text.lower()}")

print(f"Capitalized: {text.capitalize()}")

print(f"Title Case: {text.title()}")

print(f"Length of String: {len(text)}")

char = input("\nEnter a character to count: ")

print(f"Occurrences of '{char}': {text.count(char)}")

word = input("\nEnter a word to find its position: ")

position = text.find(word)

if position != -1:

print(f"Position of '{word}': {position}")

else:

print(f"'{word}' not found in the string.")

old\_word = input("\nEnter a word to replace: ")

new\_word = input("Enter the new word: ")

print(f"After Replacement: {text.replace(old\_word, new\_word)}")

start\_word = input("\nEnter a word to check if the string starts with it: ")

print(f"Starts with '{start\_word}': {text.startswith(start\_word)}")

end\_word = input("\nEnter a word to check if the string ends with it: ")

print(f"Ends with '{end\_word}': {text.endswith(end\_word)}")

print(f"\nWithout leading/trailing spaces: '{text.strip()}'")

words\_list = text.split()

print(f"Words List: {words\_list}")

print(f"Joined with '-': {'-'.join(words\_list)}")

print(f"Reversed String: {text[::-1]}")

**OUTPUT:**

Enter a string: hello world

Uppercase: HELLO WORLD

Lowercase: hello world

Capitalized: Hello world

Title Case: Hello World

Length of String: 11

Enter a character to count: o

Occurrences of 'o': 2

Enter a word to find its position: world

Position of 'world': 6

Enter a word to replace: world

Enter the new word: universe

After Replacement: hello universe

Enter a word to check if the string starts with it: hello

Starts with 'hello': True

Enter a word to check if the string ends with it: world

Ends with 'world': True

Without leading/trailing spaces: 'hello world'

Words List: ['hello', 'world']

Joined with '-': hello-world

Reversed String: dlrow olleh

**RESULT:**

**DATE :**

**EX.NO :**

**AIM:**

**FLOWCHART:**

**CODING:**

values = input("Enter elements separated by spaces: ")

tpl = tuple(values.split())

print(f"\nOriginal Tuple: {tpl}")

print(f"Length of Tuple: {len(tpl)}")

print(f"First Element: {tpl[0]}")

print(f"Last Element: {tpl[-1]}")

element = input("\nEnter an element to count: ")

print(f"Occurrences of '{element}': {tpl.count(element)}")

search\_element = input("\nEnter an element to find its index: ")

if search\_element in tpl:

print(f"Index of '{search\_element}': {tpl.index(search\_element)}")

else:

print(f"'{search\_element}' not found in the tuple.")

print(f"\nTuple Slice (First 3 elements): {tpl[:3]}")

print(f"Tuple Slice (Last 3 elements): {tpl[-3:]}")

check\_element = input("\nEnter an element to check if it exists: ")

print(f"Exists in Tuple: {check\_element in tpl}")

extra\_values = input("\nEnter more elements to add (space-separated): ").split()

new\_tuple = tpl + tuple(extra\_values)

print(f"After Concatenation: {new\_tuple}")

times = int(input("\nEnter number of times to repeat the tuple: "))

print(f"Repeated Tuple: {tpl \* times}")

mutable\_list = list(tpl)

mutable\_list.append("NewItem")

modified\_tuple = tuple(mutable\_list)

print(f"\nTuple after modifying (added 'NewItem'): {modified\_tuple}")

**OUTPUT:**

Enter elements separated by spaces: apple banana cherry

Original Tuple: ('apple', 'banana', 'cherry')

Length of Tuple: 3

First Element: apple

Last Element: cherry

Enter an element to count: banana

Occurrences of 'banana': 1

Enter an element to find its index: cherry

Index of 'cherry': 2

Tuple Slice (First 3 elements): ('apple', 'banana', 'cherry')

Tuple Slice (Last 3 elements): ('apple', 'banana', 'cherry')

Enter an element to check if it exists: orange

Exists in Tuple: False

Enter more elements to add (space-separated): mango

After Concatenation: ('apple', 'banana', 'cherry', 'mango')

Enter number of times to repeat the tuple: 2

Repeated Tuple: ('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')

Tuple after modifying (added 'NewItem'): ('apple', 'banana', 'cherry', 'NewItem')

**RESULT:**

**DATE :**

**EX.NO :**

**AIM:**

**FLOWCHART:**

**CODING:**

values = input("Enter elements separated by spaces: ").split()

lst = list(values)

print(f"\nOriginal List: {lst}")

print(f"Length of List: {len(lst)}")

print(f"First Element: {lst[0]}")

print(f"Last Element: {lst[-1]}")

element = input("\nEnter an element to count: ")

print(f"Occurrences of '{element}': {lst.count(element)}")

search\_element = input("\nEnter an element to find its index: ")

if search\_element in lst:

print(f"Index of '{search\_element}': {lst.index(search\_element)}")

else:

print(f"'{search\_element}' not found in the list.")

new\_item = input("\nEnter an element to append: ")

lst.append(new\_item)

print(f"After Append: {lst}")

insert\_pos = int(input("\nEnter position to insert an element: "))

insert\_item = input("Enter element to insert: ")

lst.insert(insert\_pos, insert\_item)

print(f"After Insert: {lst}")

remove\_item = input("\nEnter an element to remove: ")

if remove\_item in lst:

lst.remove(remove\_item)

print(f"After Remove: {lst}")

else:

print(f"'{remove\_item}' not found in the list.")

pop\_pos = int(input("\nEnter position to pop element: "))

if 0 <= pop\_pos < len(lst):

lst.pop(pop\_pos)

print(f"After Pop: {lst}")

else:

print("Invalid position.")

print(f"\nList Slice (First 3 elements): {lst[:3]}")

print(f"List Slice (Last 3 elements): {lst[-3:]}")

sort\_order = input("\nSort List? (asc/desc): ").lower()

if sort\_order == "asc":

lst.sort()

elif sort\_order == "desc":

lst.sort(reverse=True)

print(f"Sorted List: {lst}")

lst.reverse()

print(f"Reversed List: {lst}")

lst.clear()

print(f"Cleared List: {lst}")

**OUTPUT:**

Enter elements separated by spaces: apple banana mango apple grape

Original List: ['apple', 'banana', 'mango', 'apple', 'grape']

Length of List: 5

First Element: apple

Last Element: grape

Enter an element to count: apple

Occurrences of 'apple': 2

Enter an element to find its index: mango

Index of 'mango': 2

Enter an element to append: orange

After Append: ['apple', 'banana', 'mango', 'apple', 'grape', 'orange']

Enter position to insert an element: 2

Enter element to insert: kiwi

After Insert: ['apple', 'banana', 'kiwi', 'mango', 'apple', 'grape', 'orange']

Enter an element to remove: banana

After Remove: ['apple', 'kiwi', 'mango', 'apple', 'grape', 'orange']

Enter position to pop element: 3

After Pop: ['apple', 'kiwi', 'mango', 'grape', 'orange']

List Slice (First 3 elements): ['apple', 'kiwi', 'mango']

List Slice (Last 3 elements): ['mango', 'grape', 'orange']

Sort List? (asc/desc): asc

Sorted List: ['apple', 'grape', 'kiwi', 'mango', 'orange']

Reversed List: ['orange', 'mango', 'kiwi', 'grape', 'apple']

Cleared List: []

**RESULT:**

**DATE :**

**EX.NO :**

**AIM:**

**FLOWCHART:**

**CODING:**

from collections import defaultdict

d = {}

size = int(input("Enter number of key-value pairs: "))

for \_ in range(size):

key = input("Enter key: ")

value = input("Enter value: ")

d[key] = value

print(f"\nOriginal Dictionary: {d}")

print(f"Keys: {list(d.keys())}")

print(f"Values: {list(d.values())}")

print(f"Items: {list(d.items())}")

search\_key = input("\nEnter a key to check existence: ")

print(f"Exists: {search\_key in d}")

get\_key = input("\nEnter a key to get value: ")

print(f"Value: {d.get(get\_key, 'Key not found')}")

remove\_key = input("\nEnter a key to remove: ")

print(f"Removed Value: {d.pop(remove\_key, 'Key not found')}")

print(f"After Removal: {d}")

default\_d = defaultdict(lambda: "Not Found")

default\_d.update(d)

search\_default = input("\nEnter a key to search in default dictionary: ")

print(f"Value: {default\_d[search\_default]}")

new\_key = input("\nEnter a new key to update: ")

new\_value = input("Enter value: ")

d.update({new\_key: new\_value})

print(f"Updated Dictionary: {d}")

copy\_dict = d.copy()

print(f"\nCopied Dictionary: {copy\_dict}")

clear\_choice = input("\nClear dictionary? (yes/no): ").lower()

if clear\_choice == "yes":

d.clear()

print(f"After Clearing: {d}")

**OUTPUT:**

Enter number of key-value pairs: 3

Enter key: name

Enter value: Sharan

Enter key: age

Enter value: 20

Enter key: country

Enter value: India

Original Dictionary: {'name': 'Sharan', 'age': '20', 'country': 'India'}

Keys: ['name', 'age', 'country']

Values: ['Sharan', '20', 'India']

Items: [('name', 'Sharan'), ('age', '20'), ('country', 'India')]

Enter a key to check existence: age

Exists: True

Enter a key to get value: country

Value: India

Enter a key to remove: age

Removed Value: 20

After Removal: {'name': 'Sharan', 'country': 'India'}

Enter a key to search in default dictionary: age

Value: Not Found

Enter a new key to update: city

Enter value: Chennai

Updated Dictionary: {'name': 'Sharan', 'country': 'India', 'city': 'Chennai'}

Copied Dictionary: {'name': 'Sharan', 'country': 'India', 'city': 'Chennai'}

Clear dictionary? (yes/no): yes

After Clearing: {}

**RESULT:**

**DATE :**

**EX.NO :**

**AIM:**

**FLOWCHART:**

**CODING:**

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def display(self):

print(f"Name: {self.name}, Age: {self.age}")

name = input("Enter name: ")

age = int(input("Enter age: "))

p = Person(name, age)

p.display()

**OUTPUT:**

Enter name: Sharan

Enter age: 20

Name: Sharan, Age: 20

**RESULT:**

**DATE :**

**EX.NO :**

**AIM:**

**FLOWCHART:**

**CODING:**

class Animal:

def \_\_init\_\_(self, name):

self.name = name

class Prey(Animal):

def flee(self):

print(f"{self.name} is fleeing!")

class Predator(Animal):

def hunt(self):

print(f"{self.name} is hunting!")

class Rabbit(Prey):

pass

class Snake(Predator):

pass

class Fish(Prey, Predator):

pass

rabbit = Rabbit("Rabbit")

snake = Snake("Snake")

fish = Fish("Fish")

rabbit.flee()

snake.hunt()

fish.flee()

fish.hunt()

**OUTPUT:**

Rabbit is fleeing!

Snake is hunting!

Fish is fleeing!

Fish is hunting!

**RESULT:**

**DATE :**

**EX.NO :**

**AIM:**

**FLOWCHART:**

**CODING:**

from multipledispatch import dispatch

@dispatch(int, int)

def product(a, b):

print(a \* b)

@dispatch(int, int, int)

def product(a, b, c):

print(a \* b \* c)

@dispatch(float, float, float)

def product(a, b, c):

print(a \* b \* c)

product(2, 3) # Output: 6

product(2, 3, 2) # Output: 12

product(2.2, 3.4, 2.3) # Output: 17.986

**OUTPUT:**

6

12

17.986

**RESULT:**

**DATE :**

**EX.NO :**

**AIM:**

**FLOWCHART:**

**CODING:**

with open("sample.txt", "w") as file:

file.write("Hello, this is a text file.\n")

file.write("Python makes file handling easy!\n")

with open("sample.txt", "r") as file:

content = file.read()

print("Text File Content:\n", content)

import pickle

data = {"Name": "Sharan", "Age": 20, "Course": "Data Science"}

with open("sample.bin", "wb") as file:

pickle.dump(data, file)

with open("sample.bin", "rb") as file:

loaded\_data = pickle.load(file)

print("Binary File Content:\n", loaded\_data)

**OUTPUT:**

Text File Content:

Hello, this is a text file.

Python makes file handling easy!

Binary File Content:

{'Name': 'Sharan', 'Age': 20, 'Course': 'Data Science'}

**RESULT:**

**DATE :**

**EX.NO :**

**AIM:**

**FLOWCHART:**

**CODING:**

import pandas as pd

data1 = {'ID': [1, 2, 3], 'Name': ['Alice', 'Bob', 'Charlie'], 'Score': [85, 90, 78]}

data2 = {'ID': [3, 4, 5], 'Name': ['Charlie', 'David', 'Eve'], 'Score': [88, 92, 80]}

df1 = pd.DataFrame(data1)

df2 = pd.DataFrame(data2)

inner\_merge = pd.merge(df1, df2, on='ID', how='inner', suffixes=('\_df1', '\_df2'))

print("INNER JOIN:\n", inner\_merge)

left\_merge = pd.merge(df1, df2, on='ID', how='left', suffixes=('\_df1', '\_df2'))

print("\nLEFT JOIN:\n", left\_merge)

right\_merge = pd.merge(df1, df2, on='ID', how='right', suffixes=('\_df1', '\_df2'))

print("\nRIGHT JOIN:\n", right\_merge)

outer\_merge = pd.merge(df1, df2, on='ID', how='outer', suffixes=('\_df1', '\_df2'))

print("\nOUTER JOIN:\n", outer\_merge)

**OUTPUT:**

INNER JOIN:

ID Name\_df1 Score\_df1 Name\_df2 Score\_df2

0 3 Charlie 78 Charlie 88

LEFT JOIN:

ID Name\_df1 Score\_df1 Name\_df2 Score\_df2

0 1 Alice 85 NaN NaN

1 2 Bob 90 NaN NaN

2 3 Charlie 78 Charlie 88

RIGHT JOIN:

ID Name\_df1 Score\_df1 Name\_df2 Score\_df2

0 3 Charlie 78 Charlie 88

1 4 NaN NaN David 92

2 5 NaN NaN Eve 80

OUTER JOIN:

ID Name\_df1 Score\_df1 Name\_df2 Score\_df2

0 1 Alice 85 NaN NaN

1 2 Bob 90 NaN NaN

2 3 Charlie 78 Charlie 88

3 4 NaN NaN David 92

4 5 NaN NaN Eve 80

**RESULT:**

**DATE :**

**EX.NO :**

**AIM:**

**FLOWCHART:**

**CODING:**

import re

text = "Hello 123, this is an example. Contact me at example@mail.com or 987-654-3210."

pattern = re.compile(r'\d+')

print("compile & findall:", pattern.findall(text))

match = re.search(r'[a-z]+@[a-z]+\.[a-z]+', text)

print("search:", match.group() if match else "Not found")

numbers = re.findall(r'\d{3}-\d{3}-\d{4}', text)

print("findall:", numbers)

censored\_text = re.sub(r'\d{3}-\d{3}-\d{4}', 'XXX-XXX-XXXX', text)

print("sub:", censored\_text)

split\_text = re.split(r'\s', text)

print("split:", split\_text)

match\_start = re.match(r'Hello', text)

print("match:", "Matched" if match\_start else "Not matched")

**OUTPUT:**

compile & findall: ['123', '987', '654', '3210']

search: example@mail.com

findall: ['987-654-3210']

sub: Hello 123, this is an example. Contact me at example@mail.com or XXX-XXX-XXXX.

split: ['Hello', '123,', 'this', 'is', 'an', 'example.', 'Contact', 'me', 'at', 'example@mail.com', 'or', '987-654-3210.']

match: Matched

**RESULT:**

**DATE :**

**EX.NO :**

**AIM:**

**FLOWCHART:**

**CODING:**

import pandas as pd

data = {

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

'Employee': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank'],

'Salary': [50000, 60000, 55000, 52000, 70000, 72000],

'Experience': [2, 3, 5, 4, 6, 7]

}

df = pd.DataFrame(data)

grouped = df.groupby('Department').agg({

'Salary': ['sum', 'mean', 'max', 'min'],

'Experience': ['mean', 'max']

})

print("Group-wise Aggregations:\n", grouped)

df['Avg\_Dept\_Salary'] = df.groupby('Department')['Salary'].transform('mean')

print("\nData with Average Department Salary:\n", df)

def experience\_category(exp):

return 'Senior' if exp > 5 else 'Junior'

df['Experience\_Level'] = df['Experience'].apply(experience\_category)

print("\nData with Experience Level:\n", df)

**OUTPUT:**

Group-wise Aggregations:

Salary Experience

sum mean max min mean max

Department

HR 107000 53500.0 55000 52000 4.500000 5

IT 142000 71000.0 72000 70000 6.500000 7

Sales 110000 55000.0 60000 50000 2.500000 3

Data with Average Department Salary:

Department Employee Salary Experience Avg\_Dept\_Salary

0 Sales Alice 50000 2 55000.0

1 Sales Bob 60000 3 55000.0

2 HR Charlie 55000 5 53500.0

3 HR David 52000 4 53500.0

4 IT Eve 70000 6 71000.0

5 IT Frank 72000 7 71000.0

Data with Experience Level:

Department Employee Salary Experience Avg\_Dept\_Salary Experience\_Level

0 Sales Alice 50000 2 55000.0 Junior

1 Sales Bob 60000 3 55000.0 Junior

2 HR Charlie 55000 5 53500.0 Junior

3 HR David 52000 4 53500.0 Junior

4 IT Eve 70000 6 71000.0 Senior

5 IT Frank 72000 7 71000.0 Senior

**RESULT:**