1. Flow controls, Functions, String Manupulation
   1. Students marks and grade

marks = []

for i in range(5):

marks.append(float(input(f"Enter mark {i+1}: ")))

total = sum(marks)

average = total / len(marks)

if average >= 90:

grade = 'A'

elif average >= 80:

grade = 'B'

elif average >= 70:

grade = 'C'

elif average >= 60:

grade = 'D'

else:

grade = 'F'

print(f"Total: {total}, Average: {average}, Grade: {grade}")

* 1. Prime number using function

def is\_prime(num):

if num <= 1:

return False

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

if num % i == 0:

return False

return True

n = int(input('Enter a number to check if it is prime: '))

if is\_prime(n):

print('The number is a prime number')

else:

print('The number is not a prime number')

* 1. String Manupulation

string = "Hello, World!"

print(string.upper())

print(string.lower())

print(string.capitalize())

print(string.count('l'))

print(string.replace('Hello', 'Hi'))

print(string.startswith('Hello'))

print(string.endswith('World!'))

print(string.split(','))

print(string.find('World'))

print(string.isalpha())

1. Operation on Typles and Lists
   1. Tuple operations

my\_tuple = (1, 2, 3, 4, 5)

print("Tuple element at index 2:", my\_tuple[2])

print("Sliced tuple:", my\_tuple[1:4])

concatenated\_tuple = my\_tuple + (6, 7)

print("Concatenated tuple:", concatenated\_tuple)

repeated\_tuple = my\_tuple \* 2

print("Repeated tuple:", repeated\_tuple)

print("Is 3 in tuple?", 3 in my\_tuple)

print("Length of tuple:", len(my\_tuple))

print("Index of 3 in tuple:", my\_tuple.index(3))

print("Count of 2 in tuple:", my\_tuple.count(2))

* 1. Operations on lists

my\_list = [6, 7, 8, 9, 10]

print("List element at index 3:", my\_list[3])

print("Sliced list:", my\_list[2:])

concatenated\_list = my\_list + [11, 12]

print("Concatenated list:", concatenated\_list)

repeated\_list = my\_list \* 3

print("Repeated list:", repeated\_list)

print("Is 12 in list?", 12 in my\_list)

print("Length of list:", len(my\_list))

my\_list.append(11)

print("List after append:", my\_list)

my\_list.extend([12, 13])

print("List after extend:", my\_list)

my\_list.insert(0, 0)

print("List after insert:", my\_list)

my\_list.remove(11)

print("List after remove:", my\_list)

popped\_element = my\_list.pop(3)

print("Popped element:", popped\_element)

print("List after pop:", my\_list)

print("Index of 9 in list:", my\_list.index(9))

print("Count of 7 in list:", my\_list.count(7))

my\_list.sort()

print("Sorted list:", my\_list)

my\_list.reverse()

print("Reversed list:", my\_list)

my\_list.clear()

print("Cleared list:", my\_list)

1. Operations on sets

# Predefined sets

set1 = {1, 2, 3, 4}

set2 = {2, 3, 5, 6}

# Initial sets

print("Initial sets:")

print("Set 1:", set1)

print("Set 2:", set2)

# Set operations

print("\nSet operations and results:")

print("Union (|):", set1 | set2)

print("Intersection (&):", set1 & set2)

print("Difference (-):", set1 - set2)

print("Symmetric difference (^):", set1 ^ set2)

print("Subset (<):", set1 < set2)

print("Superset (>):", set1 > set2)

print("Isdisjoint (disjoint):", set1.isdisjoint(set2))

print("Membership (in):", 3 in set1)

print("Length (len):", len(set1))

print("Copy (copy):", set1.copy())

set1.discard(5)

print("discard (discard):", set1)

set1.add(7)

print("add (add):", set1)

set1.remove(2)

print("remove (remove):", set1)

print("issubset (issubset):", set1.issubset(set2))

print("issuperset (issuperset):", set1.issuperset(set2))

1. Operation on dictionaries

# Predefined dictionaries

dict1 = {'a': 1, 'b': 2, 'c': 3}

dict2 = {'b': 3, 'c': 4, 'd': 5}

# Initial dictionaries

print("Initial dictionaries:")

print("Dictionary 1:", dict1)

print("Dictionary 2:", dict2)

# Dictionary operations

print("\nDictionary operations and results:")

print("Keys of Dictionary 1:", dict1.keys())

print("Values of Dictionary 1:", dict1.values())

print("Items of Dictionary 1:", dict1.items())

dict1.update({'d': 4})

print("Updated Dictionary 1:", dict1)

dict1.pop('b')

print("Dictionary 1 after removing 'b':", dict1)

print("Value of 'c' in Dictionary 2:", dict2.get('c'))

dict2['e'] = 6

print("Dictionary 2 after adding 'e':", dict2)

print("Pop item from Dictionary 2:", dict2.popitem())

print("Length of Dictionary 2:", len(dict2))

print("Copy of Dictionary 2:", dict2.copy())

dict2.clear()

print("Cleared Dictionary 2:", dict2)

# Additional considerations

print("\nAdditional considerations:")

print("- Dictionaries are unordered collections of key-value pairs.")

print("- Keys are unique within a dictionary.")

print("- Use keys to access values in a dictionary.")

1. Looping and Recursive Functions
   1. Remove duplicates from an list

input\_list = [1, 2, 3, 4, 2, 3, 5]

unique\_list = []

for item in input\_list:

if item not in unique\_list:

unique\_list.append(item)

print("Original list:", input\_list)

print("List with duplicates removed:", unique\_list)

* 1. Fibonacci Series using recusion

def fibonacci(n):

if n <= 1:

return n

return fibonacci(n-1) + fibonacci(n-2)

num\_terms = int(input('Enter the number of terms to print fibonacci sequence : '))

print("Fibonacci sequence:")

for i in range(num\_terms):

print(fibonacci(i), end=" ")

* 1. Factorial using recursion

def factorial(n):

if n == 0:

return 1

return n \* factorial(n - 1)

number = int(input("Enter the value to find the Factorial : "))

print(f"The factorial of {number} is:", factorial(number))

1. Classes and Objects

class Student:

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

self.name = data['Name']

self.id = data['Id']

self.age = data['Age']

self.cgpa = data['cgpa']

self.phone = data['Phone']

self.address = data['Address']

def display\_info(self):

print("Name : ", self.name)

print("Register No. : ", self.id)

print("Age : ", self.age)

print("CGPA : ", self.cgpa)

print("Contact No. : ", self.phone)

def update\_std\_detaild(self,prototype, value):

setattr(self, prototype, value)

std\_data = {

'Name' : "Tamil",

'Id' : "22UCS626",

'Age' : 18,

'cgpa' : 77.77,

'Phone' : "+91 99431 12938",

}

# Creating an instance of the Student class

student1 = Student(std\_data)

# Accessing attributes and calling method

student1.display\_info()

student1.update\_std\_detaild('name',"Tamilarasan N")

student1.update\_std\_detaild('cgpa',67.19)

student1.display\_info()

1. Method Overridding in Python

import math

class Shape:

def area(self):

pass

class Rectangle(Shape):

def \_\_init\_\_(self, length, width):

self.length = length

self.width = width

def area(self):

return self.length \* self.width

class Circle(Shape):

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

self.radius = radius

def area(self):

return math.pi \* self.radius \*\* 2

class Triangle(Shape):

def \_\_init\_\_(self, base, height):

self.base = base

self.height = height

def area(self):

return 0.5 \* self.base \* self.height

class Square(Shape):

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

self.side = side

def area(self):

return self.side \*\* 2

# Usage

rectangle = Rectangle(5, 4)

print("Area of Rectangle:", rectangle.area()) # Output: 20

circle = Circle(3)

print("Area of Circle:", circle.area()) # Output: 28.274333882308138

triangle = Triangle(4, 3)

print("Area of Triangle:", triangle.area()) # Output: 6.0

square = Square(5)

print("Area of Square:", square.area())

1. Inheritance in Python

class Vehicle:

def \_\_init\_\_(self, make, model, year):

self.make = make

self.model = model

self.year = year

def display\_info(self):

print(f"Make: {self.make}, Model: {self.model}, Year: {self.year}")

class Car(Vehicle):

def \_\_init\_\_(self, make, model, year, num\_doors):

super().\_\_init\_\_(make, model, year)

self.num\_doors = num\_doors

def display\_info(self):

super().display\_info()

print(f"Number of doors: {self.num\_doors}")

class Motorcycle(Vehicle):

def \_\_init\_\_(self, make, model, year, engine\_size):

super().\_\_init\_\_(make, model, year)

self.engine\_size = engine\_size

def display\_info(self):

super().display\_info()

print(f"Engine size: {self.engine\_size} cc")

# Usage

car = Car("Toyota", "Camry", 2022, 4)

car.display\_info()

print()

motorcycle = Motorcycle("Harley-Davidson", "Sportster", 2020, 1200)

motorcycle.display\_info()

#Program 2

class Person:

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

self.name = name

self.age = age

def display\_info(self):

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

class Scholar:

def \_\_init\_\_(self, scholarship\_amount):

self.scholarship\_amount = scholarship\_amount

def display\_scholarship\_info(self):

print(f"Scholarship Amount: ${self.scholarship\_amount}")

class Student(Person, Scholar):

def \_\_init\_\_(self, name, age, scholarship\_amount, grade):

Person.\_\_init\_\_(self, name, age)

Scholar.\_\_init\_\_(self, scholarship\_amount)

self.grade = grade

def display\_info(self):

super().display\_info()

super().display\_scholarship\_info()

print(f"Grade: {self.grade}")

person = Person("Ravi",25)

print("Normal person : ")

person.display\_info()

student = Student("Tamil", 19, 29000, "A")

print("Student : ")

student.display\_info()

file handling

def create\_file(file\_name):

with open(file\_name, 'w') as file:

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

file.write("We can perform read and write operations on this file.")

print(f"File '{file\_name}' created successfully.")

def read\_file(file\_name):

with open(file\_name, 'r') as file:

content = file.read()

print(f"Content of file '{file\_name}':\n{content}")

def write\_file(file\_name, new\_content):

with open(file\_name, 'a') as file:

file.write("\n" + new\_content)

print("Content appended to file successfully.")

file\_name = "sample.txt"

create\_file(file\_name)

read\_file(file\_name)

new\_content = "This is new content appended to the file."

write\_file(file\_name, new\_content)

read\_file(file\_name)

10. exception handling

#Program 1 built in defined exception

try:

a=int(input("First Number:"))

b=int(input("Second Number:"))

result=a/b

print(result)

except ZeroDivisionError:

print("Division by Zero")

else:

print("Successful Division")

#Program 2 user defined exception

class ValidMark(Exception):

def \_\_init\_\_(self, message="Invalid mark. Please enter a valid mark."):

self.message = message

super().\_\_init\_\_(self.message)

marks = []

i = 0

while len(marks) < 5:

try:

mark = int(input(f"Enter Mark {i+1}: "))

if mark < 0 or mark > 100:

raise ValidMark()

marks.append(mark)

i += 1

except ValidMark as er:

print(er)

print("The marks are",marks)

11. regular expression

# program 1:

import re

# Sample text

text = "The cat sat on the mat."

pattern = r'cat'

match\_result = re.match(pattern, text)

if match\_result:

print("Match found with match():", match\_result.group())

else:

print("No match found with match()")

search\_result = re.search(pattern, text)

if search\_result:

print("Match found with search():", search\_result.group())

else:

print("No match found with search()")

#Program 2

import re

text = "Hilo, my name is Tamil. You can reach me at tamil.tj.1967@gmail.com"

email\_pattern = r'([\w\.-]+)@([\w\.-]+)' # email validation pattern

matches = re.findall(email\_pattern, text)

print("Email addresses found:")

for match in matches:

print(f"Username: {match[0]}, Mail-Domain: {match[1]}")

replaced\_text = re.sub(email\_pattern, r'<22ucs626@mail.sjctni.edu>', text) # replasing the exisisting email

print("\nNew replaced Email : ")

print(replaced\_text)