Problem -1:

Let A be the set {1, 2, 3, 4}. Write a program to find the ordered pairs are in the relation

I) = {(a, b) | a divides b} II) = {(a, b) | a ≤ b}

**from** itertools **import** product  
  
*# Variable initialization*result\_1 = []  
  
*# Getting Set Input From file*file\_name = input(**"Enter file name with extension : "**) *# code and input file should be in same folder  
  
# read input value from file*file = open(file\_name, **"r"**)  
Set = list(map(int, file.readlines()))  
  
*# Printing Set A*print(**"\n\nValue of the Set, A = "**,Set,**"\n"**)  
  
*# Calculating the ordered pairs*Data = list(product(Set,repeat=2))  
  
*# Calculating the ordered pair when relation following R1 ={(a,b) | a divides b}***for** i,j **in** Data:  
 **if** i%j==0 **or** j%i==0:  
 result\_1.append((i,j))  
  
*# Calculating the ordered pair when relation following R2 ={(a,b) | a ≤ b}*result\_2 = [(i,j) **for** i,j **in** Data **if** i<=j]  
  
*# printing results*print (**"The ordered pair list is for a / b : \n\t\t\t"**,result\_1)  
print (**"The ordered pair list is for a <= b : \n\t\t\t"** ,result\_2)  
file.close()

Problem-2:

Suppose that A = {1, 2, 3} and B = {1, 2}. Let R be the relation from A to B containing (a, b) if a ∈ A, b ∈ B, and a > b. Write a program to find the relation R and also represent this relation in matrix form if a1 = 1, a2 = 2, and a3 = 3, and b1 = 1 and b2 = 2

**import** numpy **as** np  
  
*# Getting Set Input From file*file\_name1 = input(**"Enter First file name with extension : "**) *# code and input file should be in same folder*file\_name2 = input(**"Enter Second file name with extension : "**) *# code and input file should be in same folder  
  
# reads input value from file*file1 = open(file\_name1,**"r"**)  
file2 = open(file\_name2,**"r"**)  
list1 = list(map(int, file1.readlines()))  
list2 = list(map(int, file2.readlines()))  
  
*# using list comprehension*output1 = [(a, b) **for** a **in** list1  
 **for** b **in** list2 **if** a > b]  
data = [1 **if** a > b **else** 0 **for** a **in** list1  
 **for** b **in** list2]  
  
*# Calculating matrix form relation of R*newArray = np.array(data)  
output2 = newArray.reshape(4, 4)  
  
*# Printing Result*print(**"Relation of R : "**,output1)  
print(**"Represent relation of R in matrix form is : "**,output2)  
*# Closing files*file1.close()  
file2.close()  
  
  
*# [(2, 1), (3, 1), (3, 2), (4, 1), (4, 2), (4, 3)]  
# [[0 0 0 0]  
# [1 0 0 0]  
# [1 1 0 0]  
# [1 1 1 0]]*

Problem-3

Write a program for graph coloring by Welch- Powell’s algorithm

*# Function Declaration***def** color\_nodes(graph):  
 color\_map = {}  
  
 *# Consider nodes in descending degree (Most neighbors .... Least neighbors)* **for** node **in** sorted(graph, key=**lambda** x: len(graph[x]), reverse=**True**):  
 neighbor\_colors = set(color\_map.get(neigh) **for** neigh **in** graph[node])  
 color\_map[node] = next(  
 color **for** color **in** range(len(graph)) **if** color **not in** neighbor\_colors  
 )  
 **return** color\_map  
  
*# Adjacent list*graph = {**'a'**:list(**'bcd'**),**'b'**: list(**'ac'**),**'c'**: list(**'abdef'**),**'d'**: list(**'ace'**),**'e'**: list(**'cdf'**),**'f'**: list(**'ce'**)}  
*# Printing result*print(**"The solution of graph coloring problem by \'Welch-Powell’s\' algorithm is : "**,color\_nodes(graph))

Problem-4

Write a program to find shortest path by Wars hall’s algorithm

*# Write a program to find shortest path by Warshall’s algorithm*INF = 1000000000  
**def** floyd\_warshall(vertex, adjacency\_matrix):  
  
 *# calculating all pair shortest path* **for** k **in** range(0, vertex):  
 **for** i **in** range(0, vertex):  
 **for** j **in** range(0, vertex):  
  
 *# relax the distance from i to j by allowing vertex k as intermediate vertex  
 # consider which one is better, going through vertex k or the previous value* adjacency\_matrix[i][j] = min(adjacency\_matrix[i][j], adjacency\_matrix[i][k] + adjacency\_matrix[k][j])  
  
 *# pretty print the graph  
 # o/d means the leftmost row is the origin vertex  
 # and the topmost column as destination vertex* print(**"Shortest path by \'Warshall’s\' algorithm is : \n"**)  
 print(**"\t\t\t\t\t\t\t\t\t\to/d"**, end=**''**)  
 **for** i **in** range(0, vertex):  
 print(**"\t\t\t{:d}"**.format(i+1), end=**'\t '**)  
 print();  
 **for** i **in** range(0, vertex):  
 print(**"\t\t\t\t\t\t\t\t\t\t{:d}"**.format(i+1), end=**''**)  
 **for** j **in** range(0,vertex):  
 **if** i==0:  
 print(end=**" \t"**)  
 **if** j>=1 **and** i<=j:  
 print(end=**"\t "**)  
 **if** i ==1 **and** j == 2 **or** i ==1 **and** j == 3 **or** i ==2 **and** j == 3:  
 print(end=**" \t"**)  
 print(**"\t\t{:d}"**.format(adjacency\_matrix[i][j]), end=**''**)  
 print();  
  
**"""  
input is given as adjacency matrix,  
input represents this undirected graph  
 A--1--B  
 | /  
 3 /  
 | 1  
 | /  
 C--2--D  
should set infinite value for each pair of vertex that has no edge  
 """**adjacency\_matrix = [  
 [ 0, 5, INF, 10],  
 [ INF, 0, 3, INF],  
 [ INF, INF, 0, 1],  
 [ INF, INF, INF, 0]  
 ]  
*# Calling function*floyd\_warshall(4, adjacency\_matrix);

Problem-5

Suppose that the relations R1 and R2 on a set A are represented by the matrices

and . Write a program to find the MR1∪R2 and MR1∩R2

*# First Function*def matrix\_intersection(mat1, mat2):  
 mat\_inter = []  
 for rows in range(len(mat1)):  
 List = []  
 print(4 \* "\t", end=" ")  
 for cols in range(len(mat1[0])):  
 List.append(mat1[rows][cols] and mat2[rows][cols])  
 print(mat1[rows][cols] and mat2[rows][cols], end=" ") *# Calculating intersection* mat\_inter.append(List)  
 print()  
  
 return mat\_inter  
  
*# second function*def matrix\_union(mat1, mat2):  
 mat\_union = []  
 for rows in range(len(mat1)):  
 List = []  
 print(4 \* "\t", end=" ")  
 for cols in range(len(mat1[0])):  
 List.append(mat1[rows][cols] or mat2[rows][cols])  
 print(mat1[rows][cols] or mat2[rows][cols], end=" ") *# Calculating union* mat\_union.append(List)  
 print()  
  
 return mat\_union  
  
*# Input declared Matrix1 and Matrix2*matrix1 = [[1, 0, 1],  
 [1, 0, 0],  
 [0, 1, 1]]  
matrix2 = [[1, 0, 1],  
 [0, 1, 1],  
 [1, 0, 1]]  
  
*# printing First Matrix*print('First Matrix M\_R1: ')  
for rows in range(len(matrix1)):  
 print(4\*"\t",end=" ")  
 for cols in range(len(matrix1[0])):  
 print(matrix1[rows][cols],end=" ")  
 print()  
  
*# printing Second Matrix*print('Second Matrix M\_R2: ')  
for rows in range(len(matrix2)):  
 print(4\*"\t",end=" ")  
 for cols in range(len(matrix2[0])):  
 print(matrix2[rows][cols],end=" ")  
 print()  
  
*# printing Rows and Columns of matrix*print("\nRows:",len(matrix1),"; Cols:",len(matrix1[0]))  
  
*# printing Matrix Intersection*print('\nMatrix Intersection:')  
mi = matrix\_intersection(matrix1, matrix2)  
  
*# printing Matrix Union*print('Matrix Union: ')  
mu = matrix\_union(matrix1, matrix2)  
  
v = ['p', 'q', 'r']  
  
r1 = []  
for i in range(len(mi)):  
 for j in range(len(mi[0])):  
 if mi[i][j] == 1:  
 r1.append((v[i], v[j]))  
  
print(r1)  
  
r2 = []  
for i in range(len(mu)):  
 for j in range(len(mu[0])):  
 if mu[i][j] == 1:  
 r2.append((v[i], v[j]))  
  
print(r2)

Problem-6

The following table gives the population of a town during the last six censuses. Write a Python program to find the population in the year of 1946 using Newton-Gregory forward interpolation formula…………………………………………………... 19 - 23

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Year: | 1911 | 1921 | 1931 | 1941 | 1951 | 1961 |
| Population: | 12 | 15 | 20 | 27 | 39 | 52 |

**from** math **import** factorial  
  
*# calculating u***def** u\_cal(u, n):  
 temp = u;  
 **for** i **in** range(1, n):  
 temp = temp \* (u - i);  
 **return** temp;  
  
*# read input value from file*file\_name = input(**"Enter file name with extension: "**) *# code and input file should be in same folder*file = open(file\_name, **"r"**)  
data = file.read()  
print(**"Values of table :"**)  
print(data)  
print()  
data = data.split()  
n = int(len(data)/2)  
x,r = [],[]  
**for** i, j **in** zip(data[0::2], data[1::2]):  
 x.append(int(i))  
 r.append(int(j))  
  
*# y[][] is used for difference table*y = [[0 **for** i **in** range(n)]  
 **for** j **in** range(n)];  
  
*# with y[][0] used for input***for** i **in** range(n):  
 y[i][0] = r[i]  
  
*# Value to interpolate at*value = float(input(**"Enter value of x for interpolation: "**));  
print(**"\nForward difference table: "**)  
print(**"Here, x = Year and f(x) = Population"**)  
  
*# Calculating the forward difference table***for** i **in** range(1, n):  
 **for** j **in** range(n - i):  
 y[j][i] = y[j + 1][i - 1] - y[j][i - 1]  
  
*# Displaying the forward difference table*formated\_table = [**"x"**, **"f(x)"**, **"∇f(x)"**]  
**for** i **in** range(2, n+1):  
 formated\_table.append(**"∇^"** + str(i) + **"f(x)"**)  
print(**" "**, end=**""**)  
**for** i **in** range(n+1):  
 print(formated\_table[i], end=**" \t"**);  
print()  
  
**for** i **in** range(n):  
 print(x[i], end=**" \t"**);  
 **for** j **in** range(n - i):  
 **if** j>1:  
 print(end=**"\t"**)  
 print(y[i][j], end=**"\t\t"**);  
 print();  
  
*# initializing u and sum*sum = y[0][0];  
u = (value - x[0]) / (x[1] - x[0]);  
  
*# Result Calculation***for** i **in** range(1, n):  
 sum = sum + (u\_cal(u, i) \* y[0][i]) / factorial(i);  
  
*# Printing Result*print(**"\nValue at"**, value,  
 **"is"**, round(sum, 6));

Problem-7

Write a Python program to find ***f(7.5)*** form the following table using Newton-Gregory backward interpolation formula……………………………………………...…. 24 - 28

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *x*: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| *f(x)*: | 1 | 8 | 27 | 64 | 125 | 216 | 343 | 512 |

**from** math **import** factorial  
  
*# read input value from file*file\_name = input(**"Enter file name with extension: "**) *# code and input file should be in same folder*file = open(file\_name, **"r"**)  
data = file.read()  
print(**"Values of table :"**)  
print(data,end=**"\n"**)  
data = data.split()  
x, y = [], []  
**for** i, j **in** zip(data[0::2], data[1::2]):  
 x.append(int(i))  
 y.append(int(j))  
  
*# Value to interpolate at*inp = float(input(**"Enter value of x for interpolation: "**))  
print(**"\nBackward difference table: "**)  
  
*# Calculating the backward difference table*table = [y]  
**for** l **in** range(len(y) - 1):  
 yn = []  
 **for** i, k **in** zip(y[1::1], y[0::1]):  
 yn.append(i - k)  
 table.append(yn)  
 y = yn  
  
*# Displaying the backward difference table*formated\_table = [**"x"**, **"f(x)"**, **"∇f(x)"**]  
**for** i **in** range(2, len(table)+1):  
 formated\_table.append(**"∇^"** + str(i) + **"f(x)"**)  
**for** i **in** range(len(table)+1):  
 print(formated\_table[i], end=**" \t"**);  
print()  
**for** i **in** range(len(table)):  
 print(x[i], end=**" \t"**);  
 **for** j **in** range(len(table) - i):  
 **if** j>1:  
 print(end=**"\t"**)  
 print(table[j][i], end=**"\t\t"**);  
 print();  
  
*# calculation of r*r = (inp - x[-1]) / (x[1] - x[0])  
  
*# result calculation*r\_component = 1  
partial\_result = 0  
**for** i **in** range(1, len(table)):  
 r\_component = r\_component \* (r + i - 1)  
 partial\_result = partial\_result + (table[i][-1] \* r\_component) / factorial(i)  
final\_result = table[0][-1] + partial\_result  
  
*# Printing Result*print(**"\nValue of"**, **"f("**,inp,**") is : "**, final\_result);

Problem-8

Write a Python program to find the value of f(15) from the following table using Newton’s divided difference formula for unequal intervals …………………… 28 - 32

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *x:* | 4 | 5 | 7 | 10 | 11 | 13 |
| *f(x):* | 48 | 100 | 294 | 900 | 1210 | 2028 |

*# function defined***def** u\_cal(i, value, x):  
 pro = 1  
 **for** j **in** range(i):  
 pro = pro \* (value - x[j])  
 **return** pro  
  
*# read input value from file*file\_name = input(**"Enter file name with extension: "**) *# code and input file should be in same folder*file = open(file\_name, **"r"**)  
data = file.read()  
print(**"Values of table :"**)  
print(data,end=**"\n"**)  
data = data.split()  
n = int(len(data)/2)  
x,r = [],[]  
**for** i, j **in** zip(data[0::2], data[1::2]):  
 x.append(int(i))  
 r.append(int(j))  
  
*# y[][] is used for divided difference table*y = [[0 **for** i **in** range(n)]  
 **for** j **in** range(n)];  
  
*# with y[][0] used for input***for** i **in** range(n):  
 y[i][0] = r[i]  
  
*# Value to interpolate at*value = float(input(**"Enter value of x for interpolation: "**));  
print()  
*# calculating divided difference table***for** i **in** range(1, n):  
 **for** j **in** range(n - i):  
 y[j][i] = ((y[j][i - 1] - y[j + 1][i - 1]) / (x[j] - x[i + j]))  
  
*# displaying divided difference table*print(**"Divided difference table:"**)  
formated\_table = [**"x"**, **"f(x)"**, **"∇f(x)"**]  
**for** i **in** range(2, n+1):  
 formated\_table.append(**"∇^"** + str(i) + **"f(x)"**)  
**for** i **in** range(n+1):  
 print(formated\_table[i], end=**"\t\t"**);  
print()  
  
**for** i **in** range(n):  
 print(x[i], end=**"\t\t "**);  
 **for** j **in** range(n - i):  
 print(y[i][j], end=**" \t\t"**);  
 print();  
  
*# initializing sum*sum = y[0][0];  
  
*# applying Newton's divided difference formula  
# Result Calculation***for** i **in** range(1, n):  
 sum = sum + (u\_cal(i,value,x) \* y[0][i]);  
  
*# Printing Result*print(**"\nValue at"**, value,  
 **"is"**, round(sum, 2));

Problem-9

Write a Python program to find the value of y when x=10 from following table using Lagrange’s interpolation formula for unequal intervals……………………..….. 33 - 35

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *x:* | 5 | 6 | 9 | 11 |
| *y:* | 12 | 13 | 14 | 16 |

*# Created class***class** Data:  
 *# Initialized Constructor* **def** \_\_init\_\_(self, x, y):  
 self.x = x  
 self.y = y  
  
**def** interpolate(f: list, xi: int, n: int) -> float:  
 result = 0.0  
 **for** i **in** range(n):  
 term = f[i].y  
 **for** j **in** range(n):  
 **if** j != i:  
 term = term \* (xi - f[j].x) / (f[i].x - f[j].x)  
 result += term  
  
 **return** result  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 f = [Data(5, 12), Data(6, 13), Data(9, 14), Data(11, 16)]  
 *# printing result* print(**"Value of f(10) is :"**, interpolate(f, 10, len(f)))

Problem-10

Write a Python program to find a real root of the equation that lies between -1 and 3 using bisection method

*# Python program for implementation of Bisection Method for solving equations  
# An example function whose solution is determined using Bisection Method.  
# The function is x^3 - 2x - 5 = 0***def** func(x):  
 **return** x \* x \* x - 2 \* x - 5  
  
*# Prints root of func(x) with error of EPSILON***def** bisection(a, b):  
 **if** func(a) \* func(b) >= 0:  
 print(**"You have not assumed right a and b\n"**)  
 **return** c = a  
 **while** (b - a) >= 0.001:  
  
 *# Find middle point* c = (a + b) / 2  
  
 *# Check if middle point is root* **if** (func(c) == 0.0):  
 **break** *# Decide the side to repeat the steps* **if** (func(c) \* func(a) < 0):  
 b = c  
 **else**:  
 a = c  
 *# printing result* print(**"The value of root is : "**, **"%.4f"** % c)  
  
*# Driver code  
# Initial values assumed*a = -1  
b = 3  
*# calling function*bisection(a, b)

Problem-11

Write a Python program to find a real root of the function in the range 1<x<3 using false position method

*# Python program for implementation of false position Method for solving equations  
# An example function whose solution is determined using false position Method.*MAX\_ITER = 1000000  
  
*# The function is x^3 - 2x - 5 = 0***def** func( x ):  
 **return** (x \* x \* x - 2 \* x -5)  
  
*# Prints root of func(x) with error of EPSILON***def** regulaFalsi( a , b):  
 **if** func(a) \* func(b) >= 0:  
 print(**"You have not assumed right a and b"**)  
 **return** c = a  
 **for** i **in** range(MAX\_ITER):  
 c = (a \* func(b) - b \* func(a))/ (func(b) - func(a))  
 **if** func(c) == 0:  
 **break** *# Decide the side to repeat the steps* **if** func(c) \* func(a) < 0:  
 b = c  
 **else**:  
 a = c  
 *# printing result* print(**"The value of root is : "** , **'%.4f'** %c)  
  
*# Driver code  
# Initial values assumed*a =-200  
b = 300  
*# calling function*regulaFalsi(a, b)