**Assignment :- 01**

**Question 1)** Write a Python program to check whether the string is Symmetrical or Palindrome.

**Ans:**

**Code :-**

|  |
| --- |
| def palindrome(a):    mid = (len(a)-1)//2  start = 0  last = len(a)-1  flag = 0    while(start <= mid):    if (a[start]== a[last]):    start += 1  last -= 1    else:  flag = 1  break;    if flag == 0:  print("The entered string is palindrome")  else:  print("The entered string is not palindrome")    def symmetry(a):    n = len(a)  flag = 0    if n%2:  mid = n//2 +1  else:  mid = n//2    start1 = 0  start2 = mid    while(start1 < mid and start2 < n):    if (a[start1]== a[start2]):  start1 = start1 + 1  start2 = start2 + 1  else:  flag = 1  break    if flag == 0:  print("The entered string is symmetrical")  else:  print("The entered string is not symmetrical")    string = 'amaama'  palindrome(string)  symmetry(string) |

**Output :-**



**Question 2)** Write a Python Program to Accept the Strings Which Contains all Vowels. Given a string, the task is to check if every vowel is present or not. We consider a vowel to be present if it is present in upper case or lower case. i.e., ‘a’, ‘e’, ‘i’.’o’, ‘u’ or ‘A’, ‘E’, ‘I’, ‘O’, ‘U’

**Ans:**

**Code :-**

|  |
| --- |
| def check(string) :    string = string.lower()    vowels = set("aeiou")    s = set({})    for char in string :    if char in vowels :  s.add(char)  else:  pass    if len(s) == len(vowels) :  print("Accepted")  else :  print("Not Accepted")    if \_\_name\_\_ == "\_\_main\_\_" :    string = "SEEquoiaL"    check(string) |

**Output :-**



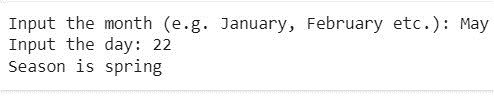
**Question 3)**Write a Python program that reads two integers representing a month and day and prints the season for that month and day.

**Ans:**

**Code :-**

|  |
| --- |
| month = input("Input the month (e.g. January, February etc.): ")  day = int(input("Input the day: "))  if month in ('January', 'February', 'March'):  season = 'winter'  elif month in ('April', 'May', 'June'):  season = 'spring'  elif month in ('July', 'August', 'September'):  season = 'summer'  else:  season = 'autumn'  if (month == 'March') and (day > 19):  season = 'spring'  elif (month == 'June') and (day > 20):  season = 'summer'  elif (month == 'September') and (day > 21):  season = 'autumn'  elif (month == 'December') and (day > 20):  season = 'winter'  print("Season is",season) |

**Output :-**



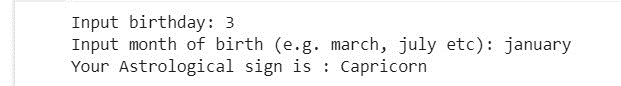
**Question 4)** Write a Python program to display the astrological sign for a given date of birth.

**Ans:**

**Code :-**

|  |
| --- |
| day = int(input("Input birthday: "))  month = input("Input month of birth (e.g. march, july etc): ")  if month == 'december':  astro\_sign = 'Sagittarius' if (day < 22) else 'capricorn'  elif month == 'january':  astro\_sign = 'Capricorn' if (day < 20) else 'aquarius'  elif month == 'february':  astro\_sign = 'Aquarius' if (day < 19) else 'pisces'  elif month == 'march':  astro\_sign = 'Pisces' if (day < 21) else 'aries'  elif month == 'april':  astro\_sign = 'Aries' if (day < 20) else 'taurus'  elif month == 'may':  astro\_sign = 'Taurus' if (day < 21) else 'gemini'  elif month == 'june':  astro\_sign = 'Gemini' if (day < 21) else 'cancer'  elif month == 'july':  astro\_sign = 'Cancer' if (day < 23) else 'leo'  elif month == 'august':  astro\_sign = 'Leo' if (day < 23) else 'virgo'  elif month == 'september':  astro\_sign = 'Virgo' if (day < 23) else 'libra'  elif month == 'october':  astro\_sign = 'Libra' if (day < 23) else 'scorpio'  elif month == 'november':  astro\_sign = 'scorpio' if (day < 22) else 'sagittarius'  print("Your Astrological sign is :",astro\_sign) |

**Output :-**



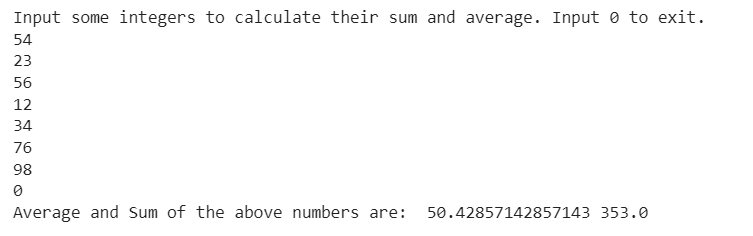
**Question 5)** Write a Python program to calculate the sum and average of n integer numbers (input from the user). Input 0 to finish.

**Ans:**

**Code :-**

|  |
| --- |
| print("Input some integers to calculate their sum and average. Input 0 to exit.")  count = 0  sum = 0.0  number = 1  while number != 0:  number = int(input(""))  sum = sum + number  count += 1  if count == 0:  print("Input some numbers")  else:  print("Average and Sum of the above numbers are: ", sum / (count-1), sum) |

**Output :-**



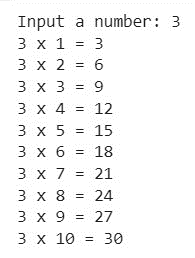
**Question 6)** Write a Python program to create the multiplication table (from 1 to 10) of a number.

**Ans:**

**Code :-**

|  |
| --- |
| n = int(input("Input a number: "))  # use for loop to iterate 10 times  for i in range(1,11):  print(n,'x',i,'=',n\*i) |

**Output :-**



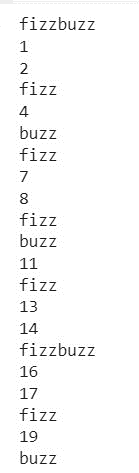
**Question 7)** Write a Python program that iterates the integers from 1 to 20. For multiples of three print "Fizz" instead of the number and for multiples of five print "Buzz". For numbers that are multiples of three and five, print "Fizz Buzz".

**Ans:**

**Code :-**

|  |
| --- |
| for fizzbuzz in range(21):  if fizzbuzz % 3 == 0 and fizzbuzz % 5 == 0:  print("fizzbuzz")  continue  elif fizzbuzz % 3 == 0:  print("fizz")  continue  elif fizzbuzz % 5 == 0:  print("buzz")  continue  print(fizzbuzz) |

**Output :-**



**Assignment :- 02**

**Question 1)** Find the number of rows and columns of a given matrix using NumPy.

(a)Using .shape Attribute.

(b)Using Indexing.

**Ans:**

1. **Using .shape Attribute.**

**Code :-**

|  |
| --- |
| import numpy as np  matrix = np.array([[6, 9, 8], [0, 9, 8]])    dimensions = np.shape(matrix)  rows, columns = dimensions    print("Rows:", rows)  print("Columns:", columns) |

**Output :-**



1. **Using Indexing.**

**Code :-**

|  |
| --- |
| import numpy as np    matrix = np.array([[6, 3, 1], [9, 7, 2]])  rows = matrix.shape[0]  columns = matrix.shape[1]    print("Rows:", rows)  print("Columns:", columns) |

**Output :-**



**Question 2)** Finding the sum of diagonal elementsof a NumPy array.

(a)For 3X3 Numpy matrix.

(b)For 5X5 Numpy Matrix.

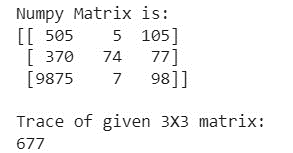
**Ans:**

**(a)For 3X3 Numpy matrix.**

**Code :-**

|  |
| --- |
| import numpy as np    # creating a 3X3 Numpy matrix  n\_array = np.array([[505, 5, 105],  [370, 74, 77],  [9875, 7,98]])    # Displaying the Matrix  print("Numpy Matrix is:")  print(n\_array)    # calculating the Trace of a matrix  trace = np.trace(n\_array)      print("\nTrace of given 3X3 matrix:")  print(trace) |

**Output :-**

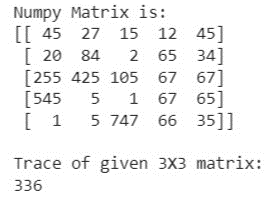


**(b)For 5X5 Numpy Matrix.**

**Code :-**

|  |
| --- |
| import numpy as np    # creating a 3X3 Numpy matrix  n\_array = np.array([[45, 27, 15,12,45],  [20, 84, 2,65,34],  [255, 425, 105,67,67],  [545, 5, 1,67,65],  [1, 5, 747,66,35]])    # Displaying the Matrix  print("Numpy Matrix is:")  print(n\_array)    # calculating the Trace of a matrix  trace = np.trace(n\_array)      print("\nTrace of given 3X3 matrix:")  print(trace) |

**Output :-**



**Question 3)** Write a python program to Calculate inner, outer, and cross products of matrices and vectors using NumPy.

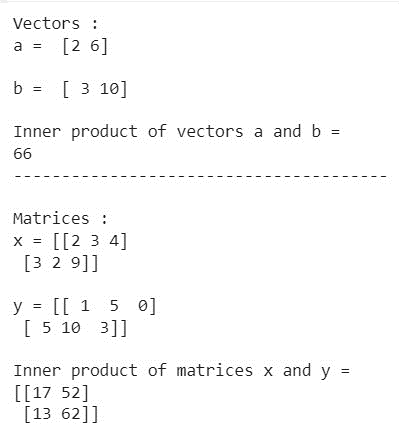
**Ans:**

**(a)Using Inner Method**

**Code :-**

|  |
| --- |
| import numpy as np  # Vectors  a = np.array([2, 6])  b = np.array([3, 10])  print("Vectors :")  print("a = ", a)  print("\nb = ", b)  # Inner Product of Vectors  print("\nInner product of vectors a and b =")  print(np.inner(a, b))  print("---------------------------------------")  # Matrices  x = np.array([[2, 3, 4], [3, 2, 9]])  y = np.array([[1, 5, 0], [5, 10, 3]])  print("\nMatrices :")  print("x =", x)  print("\ny =", y)  # Inner product of matrices  print("\nInner product of matrices x and y =")  print(np.inner(x, y)) |

**Output :-**

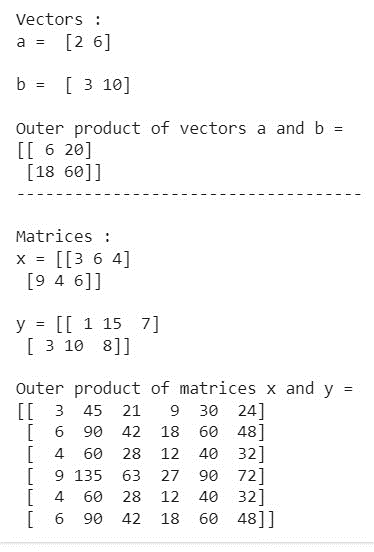


**(b)Using Outer Method**

**Code :-**

|  |
| --- |
| import numpy as np  # Vectors  a = np.array([2, 6])  b = np.array([3, 10])  print("Vectors :")  print("a = ", a)  print("\nb = ", b)  # Outer product of vectors  print("\nOuter product of vectors a and b =")  print(np.outer(a, b))  print("------------------------------------")  # Matrices  x = np.array([[3, 6, 4], [9, 4, 6]])  y = np.array([[1, 15, 7], [3, 10, 8]])  print("\nMatrices :")  print("x =", x)  print("\ny =", y)  # Outer product of matrices  print("\nOuter product of matrices x and y =")  print(np.outer(x, y)) |

**Output :-**

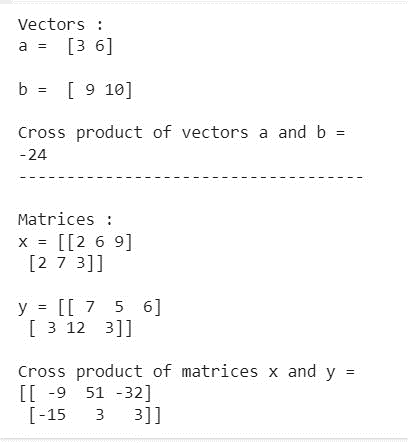


**(c) Using Cross Method**

**Code :-**

|  |
| --- |
| import numpy as np  # Vectors  a = np.array([3, 6])  b = np.array([9, 10])  print("Vectors :")  print("a = ", a)  print("\nb = ", b)  # Cross product of vectors  print("\nCross product of vectors a and b =")  print(np.cross(a, b))  print("------------------------------------")  # Matrices  x = np.array([[2, 6, 9], [2, 7, 3]])  y = np.array([[7, 5, 6], [3, 12, 3]])  print("\nMatrices :")  print("x =", x)  print("\ny =", y)  # Cross product of matrices  print("\nCross product of matrices x and y =")  print(np.cross(x, y)) |

**Output :-**



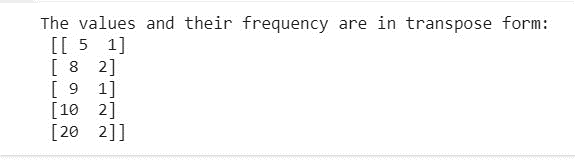
**Question 4)** How to count the frequency of unique values in NumPy array.

**Ans:**

**Code :-**

|  |
| --- |
| import numpy as np  # create a 1d-array  ini\_array = np.array([10, 20, 5,  10, 8, 20,  8, 9])  # Get a tuple of unique values  # and their frequency in  # numpy array  unique, frequency = np.unique(ini\_array,  return\_counts = True)  # convert both into one numpy array  # and then transpose it  count = np.asarray((unique,frequency )).T  print("The values and their frequency are in transpose form:\n",  count) |

**Output :-**



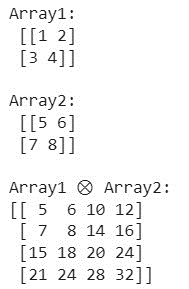
**Question 5)** Write a program Compute the Kronecker product of two multi dimension NumPy arrays.

**Ans:**

**Code :-**

|  |
| --- |
| import numpy  # Creating arrays  array1 = numpy.array([[1, 2], [3, 4]])  print('Array1:\n', array1)  array2 = numpy.array([[5, 6], [7, 8]])  print('\nArray2:\n', array2)  # Computing the Kronecker Product  kroneckerProduct = numpy.kron(array1, array2)  print('\nArray1 ⊗ Array2:')  print(kroneckerProduct) |

**Output :-**



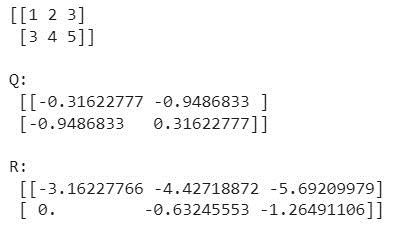
**Question 6)** Write a program Calculate the QR decomposition of a given matrix using NumPy.

**Ans:**

**Code :-**

|  |
| --- |
| import numpy as np  # Original matrix  matrix1 = np.array([[1, 2, 3], [3, 4, 5]])  print(matrix1)  # Decomposition of the said matrix  q, r = np.linalg.qr(matrix1)  print('\nQ:\n', q)  print('\nR:\n', r) |

**Output :-**



**Question 7)** How to use numpy.random.choice() method to choose elements from the list with different probability.

**Ans:**

**Code :-**

|  |
| --- |
| import numpy as np  # create a list  num\_list = [10, 20, 30, 40, 50]  # uniformly select any element  # from the list  number = np.random.choice(num\_list)  print(number) |

**Output :-**



**Assignment :- 03**

**Question 1)**

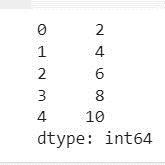
(a) Write a Pandas program to create and display a one-dimensional array-like object containing an array of data.

**Ans:**

**Code :-**

|  |
| --- |
| import pandas as pd  ds = pd.Series([2,4,6,8,10])  print(ds) |

**Output :-**

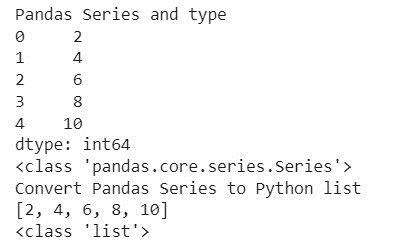


(b) Write a Pandas program to convert a Panda module Series to Python list and it's type.

**Code :-**

|  |
| --- |
| import pandas as pd  ds = pd.Series([2, 4, 6, 8, 10])  print("Pandas Series and type")  print(ds)  print(type(ds))  print("Convert Pandas Series to Python list")  print(ds.tolist())  print(type(ds.tolist())) |

**Output :-**

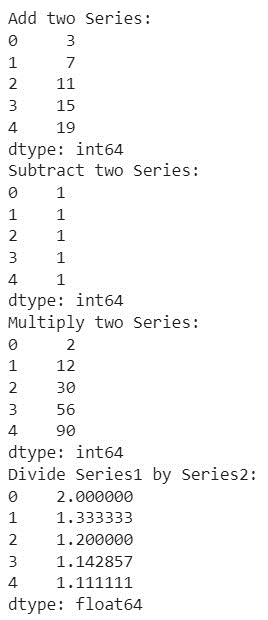


c) Write a Pandas program to add, subtract, multiple and divide two Pandas Series. Series [2, 4, 6, 8, 10], [1, 3, 5, 7, 9]

**Code :-**

|  |
| --- |
| import pandas as pd  ds1 = pd.Series([2, 4, 6, 8, 10])  ds2 = pd.Series([1, 3, 5, 7, 9])  ds = ds1 + ds2  print("Add two Series:")  print(ds)  print("Subtract two Series:")  ds = ds1 - ds2  print(ds)  print("Multiply two Series:")  ds = ds1 \* ds2  print(ds)  print("Divide Series1 by Series2:")  ds = ds1 / ds2  print(ds) |

**Output :-**



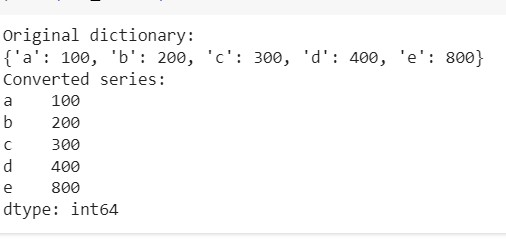
(d) Write a Pandas program to convert a dictionary to a Pandas series. dictionary:

d1 = {'a': 100, 'b': 200, 'c':300, 'd':400, 'e':800}

**Code :-**

|  |
| --- |
| import pandas as pd  d1 = {'a': 100, 'b': 200, 'c':300, 'd':400, 'e':800}  print("Original dictionary:")  print(d1)  new\_series = pd.Series(d1)  print("Converted series:")  print(new\_series) |

**Output :-**



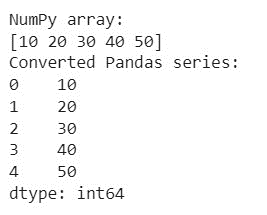
(e) Write a Pandas program to convert a NumPy array to a Pandas series.

NumPy array: d1 = [10, 20, 30, 40, 50].

**Code :-**

|  |
| --- |
| import numpy as np  import pandas as pd  np\_array = np.array([10, 20, 30, 40, 50])  print("NumPy array:")  print(np\_array)  new\_series = pd.Series(np\_array)  print("Converted Pandas series:")  print(new\_series) |

**Output :-**



**Question 2)**

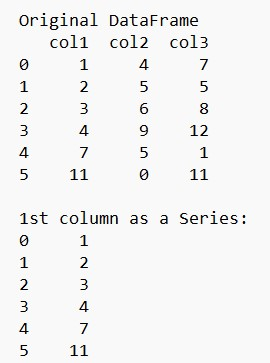
(a) Write a Pandas program to convert the first column of a DataFrame as a Series. The items are:-- {'col1': [1, 2, 3, 4, 7, 11], 'col2': [4, 5, 6, 9, 5, 0], 'col3': [7, 5, 8, 12, 1,11]}

**Ans:**

**Code :-**

|  |
| --- |
| import pandas as pd  d = {'col1': [1, 2, 3, 4, 7, 11], 'col2': [4, 5, 6, 9, 5, 0], 'col3': [7, 5, 8, 12, 1,11]}  df = pd.DataFrame(data=d)  print("Original DataFrame")  print(df)  s1 = df.ix[:,0]  print("\n1st column as a Series:")  print(s1)  print(type(s1)) |

**Output :-**



(b) Write a Pandas program to convert a given Series to an array.

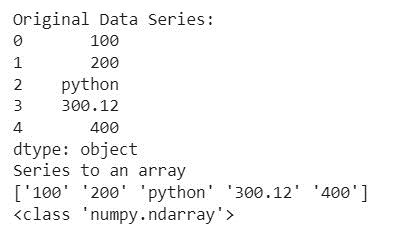
List is ['100', '200', 'python', '300.12', '400']

**Ans:**

**Code :-**

|  |
| --- |
| import pandas as pd  import numpy as np  s1 = pd.Series(['100', '200', 'python', '300.12', '400'])  print("Original Data Series:")  print(s1)  print("Series to an array")  a = s1.values  print(a)  print(type(a)) |

**Output :-**



**Question 3)**

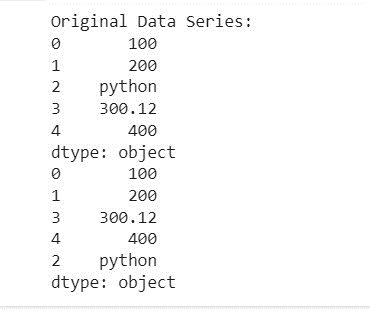
(a) Write a Pandas program to sort a given Series. ['100', '200', 'python', '300.12', '400']

**Ans:**

**Code :-**

|  |
| --- |
| import pandas as pd  s = pd.Series(['100', '200', 'python', '300.12', '400'])  print("Original Data Series:")  print(s)  new\_s = pd.Series(s).sort\_values()  print(new\_s) |

**Output :-**



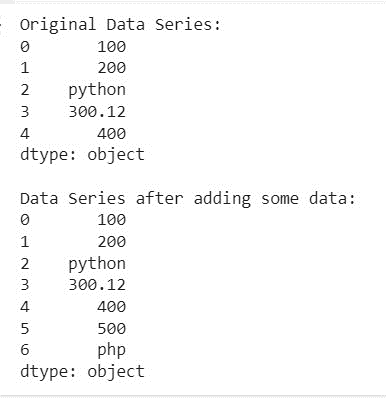
(b) Write a Pandas program to add some data to an existing Series. ['100', '200', 'python', '300.12', '400']

**Ans:**

**Code :-**

|  |
| --- |
| import pandas as pd  s = pd.Series(['100', '200', 'python', '300.12', '400'])  print("Original Data Series:")  print(s)  print("\nData Series after adding some data:")  new\_s = pd.concat([s, pd.Series([500, "php"])], ignore\_index=True)  print(new\_s) |

**Output :-**



**Assignment :- 04**

**Question 1)** Write a Pandas program to select rows from a given Data Frame based on values in some columns.

Data Frame

col1 col2 col3

0 1 4 7

1 4 5 8

2 3 6 9

3 4 7 0

4 5 8 1

Rows for colum1 value == 4

col1 col2 col3

1 4 5 8

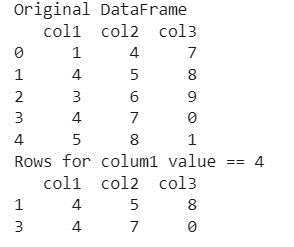
3 4 7 0

**Ans:**

**Code :-**

|  |
| --- |
| import pandas as pd  import numpy as np  d = {'col1': [1, 4, 3, 4, 5], 'col2': [4, 5, 6, 7, 8], 'col3': [7, 8, 9, 0, 1]}  df = pd.DataFrame(data=d)  print("Original DataFrame")  print(df)  print('Rows for colum1 value == 4')  print(df.loc[df['col1'] == 4]) |

**Output :-**



**Question 2)** Write a Pandas program to delete Data Frame row(s) based on given column value.

Original Data Frame

col1 col2 col3

0 1 4 7

1 4 5 8

2 3 6 9

3 4 7 0

4 5 8 1

New Data Frame

col1 col2 col3

0 1 4 7

2 3 6 9

3 4 7 0

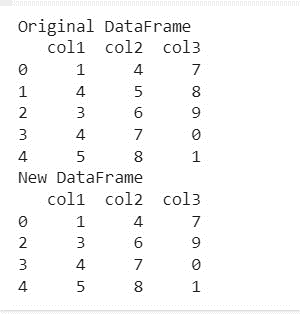
4 5 8 1

**Ans:**

**Code :-**

|  |
| --- |
| import pandas as pd  import numpy as np  d = {'col1': [1, 4, 3, 4, 5], 'col2': [4, 5, 6, 7, 8], 'col3': [7, 8, 9, 0, 1]}  df = pd.DataFrame(data=d)  print("Original DataFrame")  print(df)  df = df[df.col2 != 5]  print("New DataFrame")  print(df) |

**Output :-**



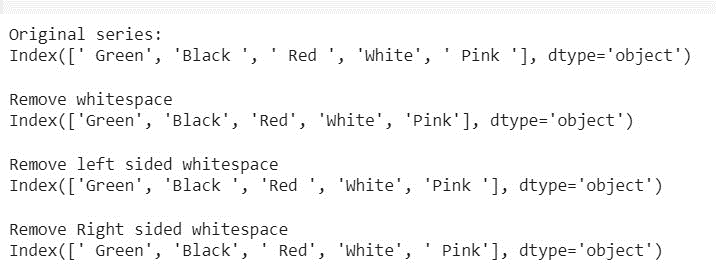
**Question 3)** Write a Pandas program to remove white spaces, left sided whites paces and right sided white spaces of the string values of a given pandas series.

**Ans:**

**Code :-**

|  |
| --- |
| import pandas as pd  color1 = pd.Index([' Green', 'Black ', ' Red ', 'White', ' Pink '])  print("Original series:")  print(color1)  print("\nRemove whitespace")  print(color1.str.strip())  print("\nRemove left sided whitespace")  print(color1.str.lstrip())  print("\nRemove Right sided whitespace")  print(color1.str.rstrip()) |

**Output :-**



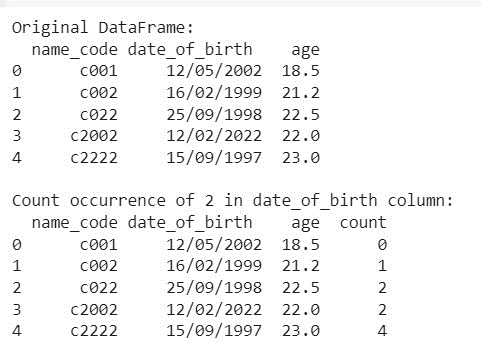
**Question 4)** Write a Pandas program to count of occurrence of a specified sub string in a Data Frame column..Write a Pandas program to count of occurrence of a specified sub string in a Data Frame column.

**Ans:**

**Code :-**

|  |
| --- |
| import pandas as pd  df = pd.DataFrame({  'name\_code': ['c001','c002','c022', 'c2002', 'c2222'],  'date\_of\_birth ': ['12/05/2002','16/02/1999','25/09/1998','12/02/2022','15/09/1997'],  'age': [18.5, 21.2, 22.5, 22, 23]  })  print("Original DataFrame:")  print(df)  print("\nCount occurrence of 2 in date\_of\_birth column:")  df['count'] = list(map(lambda x: x.count("2"), df['name\_code']))  print(df) |

**Output :-**



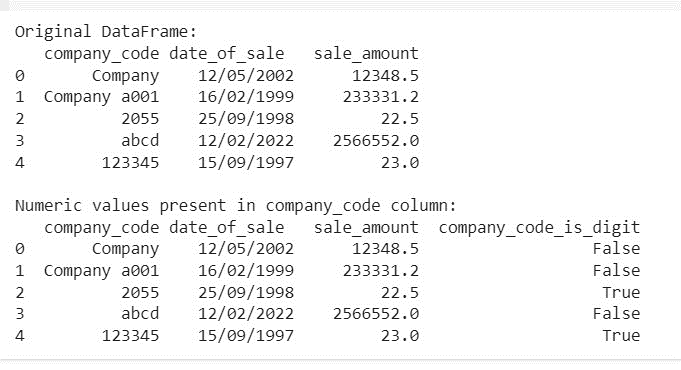
**Question 5)** Write a Pandas program to check whether only numeric values present in a given column of a Data Frame.

**Ans:**

**Code :-**

|  |
| --- |
| import pandas as pd  df = pd.DataFrame({  'company\_code': ['Company','Company a001', '2055', 'abcd', '123345'],  'date\_of\_sale ': ['12/05/2002','16/02/1999','25/09/1998','12/02/2022','15/09/1997'],  'sale\_amount': [12348.5, 233331.2, 22.5, 2566552.0, 23.0]})    print("Original DataFrame:")  print(df)  print("\nNumeric values present in company\_code column:")  df['company\_code\_is\_digit'] = list(map(lambda x: x.isdigit(), df['company\_code']))  print(df) |

**Output :-**



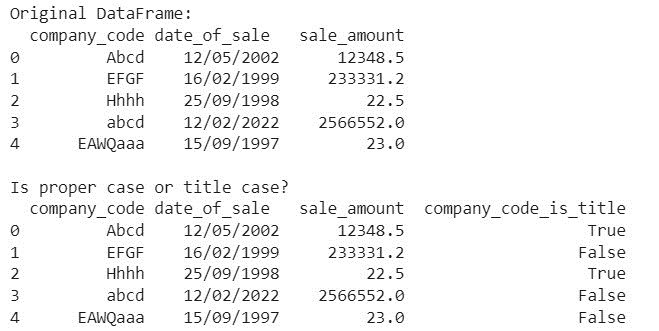
**Question 6)** Write a Pandas program to check whether only proper case or title case is present in a given column of a Data Frame.

**Ans:**

**Code :-**

|  |
| --- |
| import pandas as pd  df = pd.DataFrame({  'company\_code': ['Abcd','EFGF', 'Hhhh', 'abcd', 'EAWQaaa'],  'date\_of\_sale ': ['12/05/2002','16/02/1999','25/09/1998','12/02/2022','15/09/1997'],  'sale\_amount': [12348.5, 233331.2, 22.5, 2566552.0, 23.0]})  print("Original DataFrame:")  print(df)  print("\nIs proper case or title case?")  df['company\_code\_is\_title'] = list(map(lambda x: x.istitle(), df['company\_code']))  print(df) |

**Output :-**



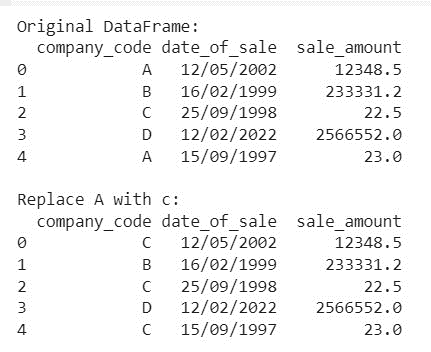
**Question 7)** Write a Pandas program to replace arbitrary values with other values in a given Data Frame.

**Ans:**

**Code :-**

|  |
| --- |
| import pandas as pd  df = pd.DataFrame({  'company\_code': ['A','B', 'C', 'D', 'A'],  'date\_of\_sale': ['12/05/2002','16/02/1999','25/09/1998','12/02/2022','15/09/1997'],  'sale\_amount': [12348.5, 233331.2, 22.5, 2566552.0, 23.0]  })  print("Original DataFrame:")  print(df)  print("\nReplace A with c:")  df = df.replace("A", "C")  print(df) |

**Output :-**



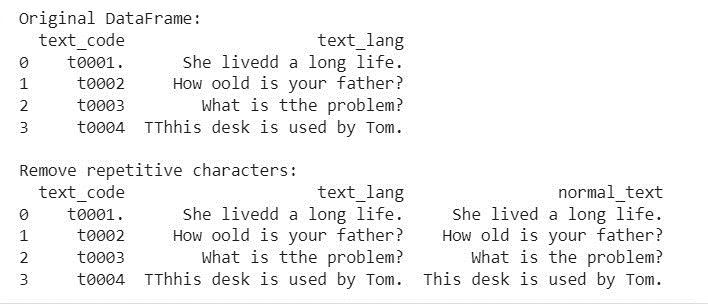
**Question 8)** Write a Pandas program to remove repetitive characters from the specified column of a given Data Frame.

**Ans:**

**Code :-**

|  |
| --- |
| import pandas as pd  import re as re  pd.set\_option('display.max\_columns', 10)  df = pd.DataFrame({  'text\_code': ['t0001.','t0002','t0003', 't0004'],  'text\_lang': ['She livedd a long life.', 'How oold is your father?', 'What is tthe problem?','TThhis desk is used by Tom.']  })  print("Original DataFrame:")  print(df)  def rep\_char(str1):  tchr = str1.group(0)  if len(tchr) > 1:  return tchr[0:1] # can change the value here on repetition  def unique\_char(rep, sent\_text):  convert = re.sub(r'(\w)\1+', rep, sent\_text)  return convert  df['normal\_text']=df['text\_lang'].apply(lambda x : unique\_char(rep\_char,x))  print("\nRemove repetitive characters:")  print(df) |

**Output :-**



**Question 9)** Write a Pandas program to extract date (format: mm-dd-yyyy) from a given column of a given Data

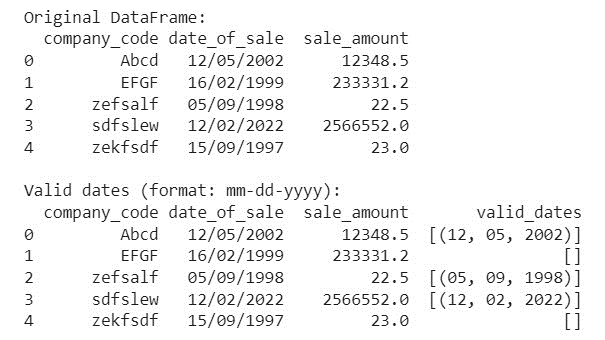
Frame.

**Ans:**

**Code :-**

|  |
| --- |
| import pandas as pd  import re as re  df = pd.DataFrame({  'company\_code': ['Abcd','EFGF', 'zefsalf', 'sdfslew', 'zekfsdf'],  'date\_of\_sale': ['12/05/2002','16/02/1999','05/09/1998','12/02/2022','15/09/1997'],  'sale\_amount': [12348.5, 233331.2, 22.5, 2566552.0, 23.0]  })  print("Original DataFrame:")  print(df)  def find\_valid\_dates(dt):  #format: mm-dd-yyyy  result = re.findall(r'\b(1[0-2]|0[1-9])/(3[01]|[12][0-9]|0[1-9])/([0-9]{4})\b',dt)  return result  df['valid\_dates']=df['date\_of\_sale'].apply(lambda dt : find\_valid\_dates(dt))  print("\nValid dates (format: mm-dd-yyyy):")  print(df) |

**Output :-**



**Question 10)** Write a Pandas program to extract numbers less than 100 from the specified column of a given Data

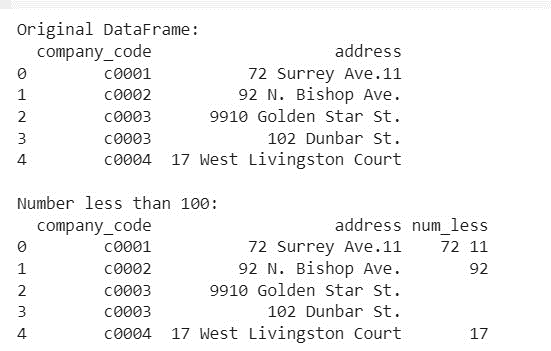
Frame.

**Ans:**

**Code :-**

|  |
| --- |
| import pandas as pd  import re as re  pd.set\_option('display.max\_columns', 10)  df = pd.DataFrame({  'company\_code': ['c0001','c0002','c0003', 'c0003', 'c0004'],  'address': ['72 Surrey Ave.11','92 N. Bishop Ave.','9910 Golden Star St.', '102 Dunbar St.', '17 West Livingston Court']  })  print("Original DataFrame:")  print(df)  def test\_num\_less(n):  nums = []  for i in n.split():  result = re.findall(r'\b(0\*(?:[1-9][0-9]?|100))\b',i)  nums.append(result)  all\_num=[",".join(x) for x in nums if x != []]  return " ".join(all\_num)  df['num\_less'] = df['address'].apply(lambda x : test\_num\_less(x))  print("\nNumber less than 100:")  print(df) |

**Output :-**



**Assignment :- 05**

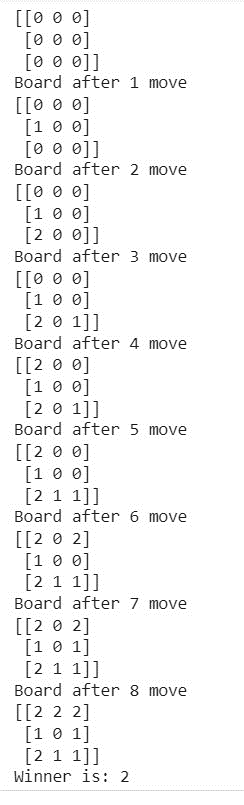
**Question 1)** write a program to implement tic-tac-toe game problem.

**Ans:**

**Code :-**

|  |
| --- |
| import numpy as np  import random  from time import sleep    def create\_board():  return(np.array([[0, 0, 0],  [0, 0, 0],  [0, 0, 0]]))    def possibilities(board):  l = []    for i in range(len(board)):  for j in range(len(board)):    if board[i][j] == 0:  l.append((i, j))  return(l)  def random\_place(board, player):  selection = possibilities(board)  current\_loc = random.choice(selection)  board[current\_loc] = player  return(board)    def row\_win(board, player):  for x in range(len(board)):  win = True    for y in range(len(board)):  if board[x, y] != player:  win = False  continue    if win == True:  return(win)  return(win)    def col\_win(board, player):  for x in range(len(board)):  win = True    for y in range(len(board)):  if board[y][x] != player:  win = False  continue    if win == True:  return(win)  return(win)    def diag\_win(board, player):  win = True  y = 0  for x in range(len(board)):  if board[x, x] != player:  win = False  if win:  return win  win = True  if win:  for x in range(len(board)):  y = len(board) - 1 - x  if board[x, y] != player:  win = False  return win    def evaluate(board):  winner = 0    for player in [1, 2]:  if (row\_win(board, player) or  col\_win(board, player) or  diag\_win(board, player)):    winner = player    if np.all(board != 0) and winner == 0:  winner = -1  return winner    def play\_game():  board, winner, counter = create\_board(), 0, 1  print(board)  sleep(2)    while winner == 0:  for player in [1, 2]:  board = random\_place(board, player)  print("Board after " + str(counter) + " move")  print(board)  sleep(2)  counter += 1  winner = evaluate(board)  if winner != 0:  break  return(winner)    print("Winner is: " + str(play\_game())) |

**Output :-**



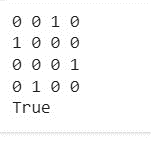
**Question 2)** Write a program to simulate 4-Queen problem.

**Ans:**

**Code :-**

|  |
| --- |
| global N  N = 4    def printSolution(board):  for i in range(N):  for j in range(N):  print (board[i][j],end=' ')  print()  def isSafe(board, row, col):    for i in range(col):  if board[row][i] == 1:  return False    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):  if board[i][j] == 1:  return False    for i, j in zip(range(row, N, 1), range(col, -1, -1)):  if board[i][j] == 1:  return False    return True    def solveNQUtil(board, col):  if col >= N:  return True    for i in range(N):    if isSafe(board, i, col):  board[i][col] = 1  if solveNQUtil(board, col + 1) == True:  return True    board[i][col] = 0  return False    def solveNQ():  board = [ [0, 0, 0, 0],  [0, 0, 0, 0],  [0, 0, 0, 0],  [0, 0, 0, 0]  ]    if solveNQUtil(board, 0) == False:  print ("Solution does not exist")  return False    printSolution(board)  return True  solveNQ() |

**Output :-**



**Question 3)** Write a program to implement breadth first search algorithm.

**Ans:**

**Code :-**

|  |
| --- |
| from collections import defaultdict    class Graph:    def \_\_init\_\_(self):    self.graph = defaultdict(list)    def addEdge(self, u, v):  self.graph[u].append(v)    def BFS(self, s):    visited = [False] \* (max(self.graph) + 1)    queue = []  queue.append(s)  visited[s] = True    while queue:  s = queue.pop(0)  print(s, end=" ")    for i in self.graph[s]:  if visited[i] == False:  queue.append(i)  visited[i] = True    if \_\_name\_\_ == '\_\_main\_\_':  g = Graph()  g.addEdge(0, 1)  g.addEdge(0, 2)  g.addEdge(1, 2)  g.addEdge(2, 0)  g.addEdge(2, 3)  g.addEdge(3, 3)    print("Following is Breadth First Traversal"  " (starting from vertex 2)")  g.BFS(2) |

**Output :-**



**Question 4)** Write a program to implement depth first search algorithm.

**Ans:**

**Code :-**

|  |
| --- |
| from collections import defaultdict  class Graph:  def \_\_init\_\_(self):    self.graph = defaultdict(list)    def addEdge(self, u, v):  self.graph[u].append(v)    def DFSUtil(self, v, visited):  visited.add(v)  print(v, end=' ')    for neighbour in self.graph[v]:  if neighbour not in visited:  self.DFSUtil(neighbour, visited)  def DFS(self, v):  visited = set()  self.DFSUtil(v, visited)  if \_\_name\_\_ == "\_\_main\_\_":  g = Graph()  g.addEdge(0, 1)  g.addEdge(0, 2)  g.addEdge(1, 2)  g.addEdge(2, 0)  g.addEdge(2, 3)  g.addEdge(3, 3)    print("Following is Depth First Traversal (starting from vertex 2)")    g.DFS(2) |

**Output :-**



**Assignment :- 06**

**Question 1)** Write a program to implement A\* search algorithm using python.

**Ans:**

**Code :-**

|  |
| --- |
| import heapq  class Node:  def \_\_init\_\_(self, x, y):  self.x = x  self.y = y  self.g = float('inf')  self.h = 0  self.parent = None  def \_\_lt\_\_(self, other):  return (self.g + self.h) < (other.g + other.h)  def heuristic(node, goal):  return abs(node.x - goal.x) + abs(node.y - goal.y)  def astar(grid, start, end):  open\_list = []  closed\_list = set()  start\_node = Node(start[0], start[1])  goal\_node = Node(end[0], end[1])  start\_node.g = 0  start\_node.h = heuristic(start\_node, goal\_node)  heapq.heappush(open\_list, start\_node)  while open\_list:  current\_node = heapq.heappop(open\_list)  if current\_node.x == goal\_node.x and current\_node.y == goal\_node.y:  path = []  while current\_node:  path.insert(0, (current\_node.x, current\_node.y))  current\_node = current\_node.parent  return path  closed\_list.add((current\_node.x, current\_node.y))  neighbors = []  for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:  neighbor\_x, neighbor\_y = current\_node.x + dx, current\_node.y + dy  if 0 <= neighbor\_x < len(grid) and 0 <= neighbor\_y < len(grid[0]) and grid[neighbor\_x][neighbor\_y] == 0:  neighbors.append((neighbor\_x, neighbor\_y))  for neighbor in neighbors:  neighbor\_node = Node(neighbor[0], neighbor[1])  if (neighbor\_node.x, neighbor\_node.y) in closed\_list:  continue  tentative\_g = current\_node.g + 1  if tentative\_g < neighbor\_node.g:  neighbor\_node.parent = current\_node  neighbor\_node.g = tentative\_g  neighbor\_node.h = heuristic(neighbor\_node, goal\_node)  if neighbor\_node not in open\_list:  heapq.heappush(open\_list, neighbor\_node)  return None  grid = [  [0, 0, 0, 1, 0],  [1, 1, 0, 1, 0],  [0, 0, 0, 0, 0],  [0, 1, 1, 1, 0],  [0, 0, 0, 0, 0]  ]  start = (0, 0)  end = (4, 4)  path = astar(grid, start, end)  if path:  print("Shortest path:", path)  else:  print("No path found.") |

**Output :-**



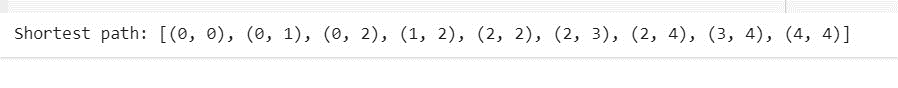
**Question 2)** Write a program to implement Ao\* search algorithm using python.

**Ans:**

**Code :-**

|  |
| --- |
| import heapq  class Node:  def \_\_init\_\_(self, x, y):  self.x = x  self.y = y  self.g = float('inf')  self.h = 0  self.f = None  self.parent = None  def \_\_lt\_\_(self, other):  return self.f < other.f  def heuristic(node, goal):  return abs(node.x - goal.x) + abs(node.y - goal.y)  def ao\_star(grid, start, end):  open\_list = []  closed\_list = set()  start\_node = Node(start[0], start[1])  goal\_node = Node(end[0], end[1])  start\_node.g = 0  start\_node.h = heuristic(start\_node, goal\_node)  start\_node.f = start\_node.g + start\_node.h  heapq.heappush(open\_list, start\_node)  while open\_list:  current\_node = heapq.heappop(open\_list)  if current\_node.x == goal\_node.x and current\_node.y == goal\_node.y:  return reconstruct\_path(current\_node)  closed\_list.add((current\_node.x, current\_node.y))  neighbors = []  for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:  neighbor\_x, neighbor\_y = current\_node.x + dx, current\_node.y + dy  if 0 <= neighbor\_x < len(grid) and 0 <= neighbor\_y < len(grid[0]) and grid[neighbor\_x][neighbor\_y] == 0:  neighbors.append((neighbor\_x, neighbor\_y))  for neighbor in neighbors:  neighbor\_node = Node(neighbor[0], neighbor[1])  if (neighbor\_node.x, neighbor\_node.y) in closed\_list:  continue  tentative\_g = current\_node.g + 1  if tentative\_g < neighbor\_node.g:  neighbor\_node.parent = current\_node  neighbor\_node.g = tentative\_g  neighbor\_node.h = heuristic(neighbor\_node, goal\_node)  neighbor\_node.f = neighbor\_node.g + neighbor\_node.h  if neighbor\_node not in open\_list:  heapq.heappush(open\_list, neighbor\_node)  return None  def reconstruct\_path(node):  path = []  while node:  path.insert(0, (node.x, node.y))  node = node.parent  return path  grid = [  [0, 0, 0, 1, 0],  [1, 1, 0, 1, 0],  [0, 0, 0, 0, 0],  [0, 1, 1, 1, 0],  [0, 0, 0, 0, 0]  ]  start = (0, 0)  end = (4, 4)  path = ao\_star(grid, start, end)  if path:  print("Shortest path:", path)  else:  print("No path found.") |

**Output :-**



**Question 3)** Write a program to implement Hill Climbing algorithm using python.

**Ans:**

**Code :-**

|  |
| --- |
| import random  def objective\_function(x):  return -x\*\*2  def hill\_climbing(max\_iterations, step\_size, initial\_solution):  current\_solution = initial\_solution  current\_value = objective\_function(current\_solution)  for \_ in range(max\_iterations):  neighbor = current\_solution + random.uniform(-step\_size, step\_size)  neighbor\_value = objective\_function(neighbor)  if neighbor\_value > current\_value:  current\_solution = neighbor  current\_value = neighbor\_value  return current\_solution, current\_value  if \_\_name\_\_ == "\_\_main\_\_":  max\_iterations = 1000  step\_size = 0.1  initial\_solution = random.uniform(-10, 10)  best\_solution, best\_value = hill\_climbing(max\_iterations, step\_size, initial\_solution)  print(f"Best Solution: {best\_solution}")  print(f"Best Value: {best\_value}") |

**Output :-**



**Assignment :- 07**

**Question 1)** Write a program to Minimax and Alpha-Beta Pruning Using Python.

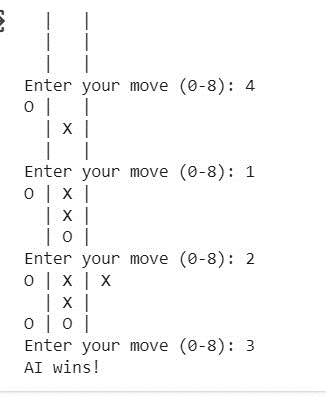
**Ans:**

**(a) Minimax**

**Code :-**

|  |
| --- |
| # Tic-Tac-Toe board representation  board = [' ' for \_ in range(9)]  def print\_board(board):  for i in range(0, 9, 3):  print(board[i], '|', board[i + 1], '|', board[i + 2])  def check\_winner(board):  for i in range(0, 9, 3):  if board[i] == board[i + 1] == board[i + 2] and board[i] != ' ':  return board[i]    for i in range(3):  if board[i] == board[i + 3] == board[i + 6] and board[i] != ' ':  return board[i]    if board[0] == board[4] == board[8] and board[0] != ' ':  return board[0]  if board[2] == board[4] == board[6] and board[2] != ' ':  return board[2]    if ' ' not in board:  return 'Draw'    return None  # Minimax algorithm  def minimax(board, depth, is\_maximizing):  winner = check\_winner(board)    if winner is not None:  if winner == 'X':  return -1  elif winner == 'O':  return 1  else:  return 0  if is\_maximizing:  max\_eval = float('-inf')  for i in range(9):  if board[i] == ' ':  board[i] = 'O'  eval = minimax(board, depth + 1, False)  board[i] = ' '  max\_eval = max(max\_eval, eval)  return max\_eval  else:  min\_eval = float('inf')  for i in range(9):  if board[i] == ' ':  board[i] = 'X'  eval = minimax(board, depth + 1, True)  board[i] = ' '  min\_eval = min(min\_eval, eval)  return min\_eval  def best\_move(board):  best\_eval = float('-inf')  best\_move = -1  for i in range(9):  if board[i] == ' ':  board[i] = 'O'  eval = minimax(board, 0, False)  board[i] = ' '  if eval > best\_eval:  best\_eval = eval  best\_move = i    return best\_move  while True:  print\_board(board)  move = int(input("Enter your move (0-8): "))  if board[move] == ' ':  board[move] = 'X'  if check\_winner(board) is not None:  print("You win!")  break  if ' ' not in board:  print("It's a draw!")  break  ai\_move = best\_move(board)  board[ai\_move] = 'O'  if check\_winner(board) is not None:  print("AI wins!")  break  else:  print("Invalid move. Try again.") |

**Output :-**



**(b) Alpha-Beta Pruning**

**Code :-**

|  |
| --- |
| class Node:  def \_\_init\_\_(self, value, children=[]):  self.value = value  self.children = children  # Alpha-beta pruning function  def alpha\_beta(node, depth, alpha, beta, is\_maximizing\_player):  if depth == 0 or len(node.children) == 0:  return node.value # Leaf node, return its value    if is\_maximizing\_player:  max\_eval = float('-inf')  for child in node.children:  eval = alpha\_beta(child, depth - 1, alpha, beta, False)  max\_eval = max(max\_eval, eval)  alpha = max(alpha, eval)  if beta <= alpha:  break # Prune the remaining branches  return max\_eval  else:  min\_eval = float('inf')  for child in node.children:  eval = alpha\_beta(child, depth - 1, alpha, beta, True)  min\_eval = min(min\_eval, eval)  beta = min(beta, eval)  if beta <= alpha:  break # Prune the remaining branches  return min\_eval  if \_\_name\_\_ == "\_\_main\_\_":  # Create a sample game tree  leaf1 = Node(3)  leaf2 = Node(6)  leaf3 = Node(2)  leaf4 = Node(8)  node1 = Node(10, [leaf1, leaf2])  node2 = Node(12, [leaf3, leaf4])  root = Node(0, [node1, node2])  # Perform alpha-beta pruning  result = alpha\_beta(root, 3, float('-inf'), float('inf'), True)  print("Optimal value:", result) |

**Output :-**



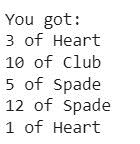
**Question 2)** Write a program to shuffle Deck of cards.

**Ans:**

**Code :-**

|  |
| --- |
| import itertools, random  deck = list(itertools.product(range(1,14),['Spade','Heart','Diamond','Club']))  random.shuffle(deck)  print("You got:")  for i in range(5):  print(deck[i][0], "of", deck[i][1]) |

**Output :-**



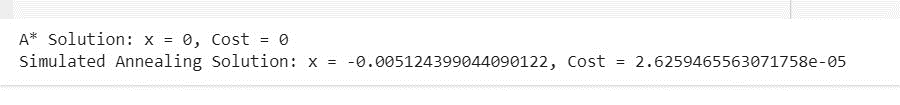
**Question 3)** Write a program to A star Using simulated annealing.

**Ans:**

**Code :-**

|  |
| --- |
| import random  import math  def objective\_function(x):  return x\*\*2  initial\_state = 10.0  initial\_temperature = 100.0  cooling\_rate = 0.95  def a\_star\_search():  best\_state = initial\_state  best\_cost = objective\_function(best\_state)  for state in range(-100, 101):  cost = objective\_function(state)  if cost < best\_cost:  best\_state = state  best\_cost = cost  return best\_state, best\_cost  def simulated\_annealing():  current\_state = initial\_state  current\_cost = objective\_function(current\_state)  best\_state = current\_state  best\_cost = current\_cost  temperature = initial\_temperature  while temperature > 0.1:  neighbor\_state = current\_state + random.uniform(-1, 1)  neighbor\_cost = objective\_function(neighbor\_state)  if neighbor\_cost < current\_cost:  current\_state = neighbor\_state  current\_cost = neighbor\_cost  if current\_cost < best\_cost:  best\_state = current\_state  best\_cost = current\_cost  else:  acceptance\_probability = math.exp(-(neighbor\_cost - current\_cost) / temperature)  if random.random() < acceptance\_probability:  current\_state = neighbor\_state  current\_cost = neighbor\_cost  temperature \*= cooling\_rate  return best\_state, best\_cost  if \_\_name\_\_ == "\_\_main\_\_":  a\_star\_solution, a\_star\_cost = a\_star\_search()  print(f"A\* Solution: x = {a\_star\_solution}, Cost = {a\_star\_cost}")  # Simulated Annealing  sa\_solution, sa\_cost = simulated\_annealing()  print(f"Simulated Annealing Solution: x = {sa\_solution}, Cost = {sa\_cost}") |

**Output :-**



**Assignment :- 08**

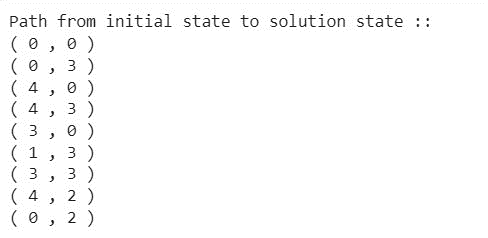
**Question 1)** Write a program to solve water jug problem using python.

**Ans:**

**Code :-**

|  |
| --- |
| from collections import deque  def Solution(a, b, target):  m = {}  isSolvable = False  path = []  q = deque()  q.append((0, 0))  while (len(q) > 0):  # Current state  u = q.popleft()  if ((u[0], u[1]) in m):  continue  if ((u[0] > a or u[1] > b or  u[0] < 0 or u[1] < 0)):  continue  path.append([u[0], u[1]])  m[(u[0], u[1])] = 1  if (u[0] == target or u[1] == target):  isSolvable = True  if (u[0] == target):  if (u[1] != 0):  path.append([u[0], 0])  else:  if (u[0] != 0):  path.append([0, u[1]])  sz = len(path)  for i in range(sz):  print("(", path[i][0], ",",  path[i][1], ")")  break  q.append([u[0], b])  q.append([a, u[1]])  for ap in range(max(a, b) + 1):  c = u[0] + ap  d = u[1] - ap  if (c == a or (d == 0 and d >= 0)):  q.append([c, d])  c = u[0] - ap  d = u[1] + ap  if ((c == 0 and c >= 0) or d == b):  q.append([c, d])  q.append([a, 0])  q.append([0, b])  if (not isSolvable):  print("Solution not possible")  if \_\_name\_\_ == '\_\_main\_\_':  Jug1, Jug2, target = 4, 3, 2  print("Path from initial state "  "to solution state ::")  Solution(Jug1, Jug2, target) |

**Output :-**



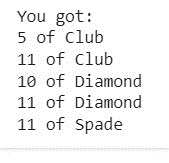
**Question 2)** Write a program to shuffle Deck of cards using python .

**Ans:**

**Code :-**

|  |
| --- |
| import itertools, random  deck = list(itertools.product(range(1,14),['Spade','Heart','Diamond','Club']))  random.shuffle(deck)  print("You got:")  for i in range(5):  print(deck[i][0], "of", deck[i][1]) |

**Output :-**



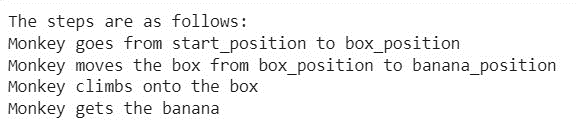
**Question 3)** Write a program to solve the Monkey Banana problem using python .

**Ans:**

**Code :-**

|  |
| --- |
| def Monkey\_go\_box(monkey, box):  print(f"Monkey goes from {monkey} to {box}")  def Monkey\_move\_box(box, banana):  print(f"Monkey moves the box from {box} to {banana}")  def Monkey\_climb\_box():  print("Monkey climbs onto the box")  def Monkey\_get\_banana():  print("Monkey gets the banana")  if \_\_name\_\_ == "\_\_main\_\_":  monkey = "start\_position"  banana = "banana\_position"  box = "box\_position"  print("The steps are as follows:")  Monkey\_go\_box(monkey, box)  Monkey\_move\_box(box, banana)  Monkey\_climb\_box()  Monkey\_get\_banana() |

**Output :-**



**Question 4)** Write a program to solve traveling salesman problem using python .

**Ans:**

**Code :-**

|  |
| --- |
| from sys import maxsize  from itertools import permutations  V = 4  def travellingSalesmanProblem(graph, s):  vertex = []  for i in range(V):  if i != s:  vertex.append(i)  min\_path = maxsize  next\_permutation=permutations(vertex)  for i in next\_permutation:  current\_pathweight = 0  k = s  for j in i:  current\_pathweight += graph[k][j]  k = j  current\_pathweight += graph[k][s]    min\_path = min(min\_path, current\_pathweight)  return min\_path  if \_\_name\_\_ == "\_\_main\_\_":  graph = [[0, 10, 15, 20], [10, 0, 35, 25],  [15, 35, 0, 30], [20, 25, 30, 0]]  s = 0  print(travellingSalesmanProblem(graph, s)) |

**Output :-**



**Assignment :- 09**

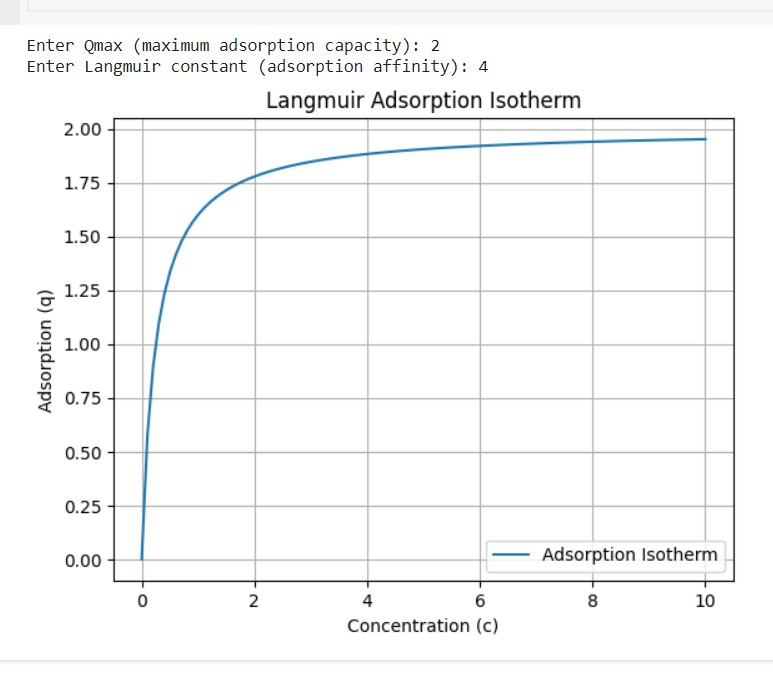
**Question 1)** Develop a Python script to model adsorption using a simple adsorption isotherm equation, allowing users to input values and plot the adsorption curve.

**Ans:**

**Code :-**

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  # Define the Langmuir isotherm equation  def langmuir\_adsorption(q\_max, K, c):  return (q\_max \* K \* c) / (1 + K \* c)  # Input values from the user  q\_max = float(input("Enter Qmax (maximum adsorption capacity): "))  K = float(input("Enter Langmuir constant (adsorption affinity): "))  c\_values = np.linspace(0, 10, 100) # Concentration values for the curve  # Calculate the adsorption data  adsorption\_data = [langmuir\_adsorption(q\_max, K, c) for c in c\_values]  plt.figure()  plt.plot(c\_values, adsorption\_data, label='Adsorption Isotherm')  plt.xlabel('Concentration (c)')  plt.ylabel('Adsorption (q)')  plt.title('Langmuir Adsorption Isotherm')  plt.legend()  plt.grid(True)  plt.show() |

**Output :-**



**Question 2)** Write a Python function to apply double negation to a logical expression represented as a string and return the simplified expression.

**Ans:**

**Code :-**

|  |
| --- |
| def simplify\_double\_negation(expression):  while "not not" in expression:  expression = expression.replace("not not", "")  return expression  input\_expression = "not not (A and (B or not C)) or not not (D and E)"  simplified\_expression = simplify\_double\_negation(input\_expression)  print("Original Expression:", input\_expression)  print("Simplified Expression:", simplified\_expression) |

**Output :-**



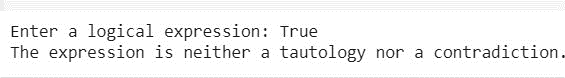
**Question 3)** Create a Python program that checks if a given logical expression violates the law of contradiction and provides feedback on the contradiction if found.

**Ans:**

**Code :-**

|  |
| --- |
| import sympy  def check\_contradiction(expression):  try:  parsed\_expression = sympy.sympify(expression)  simplified = sympy.simplify(parsed\_expression)  if simplified is True:  print("The expression is a tautology (always true). It does not violate the law of contradiction.")  elif simplified is False:  print("The expression is a contradiction (always false) and violates the law of contradiction.")  else:  print("The expression is neither a tautology nor a contradiction.")  except sympy.SympifyError:  print("Invalid expression. Please check your logical expression.")  if \_\_name\_\_ == "\_\_main\_\_":  expression = input("Enter a logical expression: ")  check\_contradiction(expression) |

**Output :-**



**Question 4)** Implement a Python function that represents a Modus Ponens inference rule. Given a knowledge base with facts and an implication, the function should determine if the conclusion can be derived.

**Ans:**

**Code :-**

|  |
| --- |
| def modus\_ponens(knowledge\_base, antecedent, conclusion):  # Check if the antecedent is in the knowledge base  if antecedent in knowledge\_base:  # Check if the implication (antecedent -> conclusion) is also in the knowledge base  if (antecedent, conclusion) in knowledge\_base:  return True, conclusion # Modus Ponens applies, return True and the conclusion  return False, None # Modus Ponens does not apply  # Example usage:  if \_\_name\_\_ == "\_\_main\_\_":  # Define a knowledge base with facts and implications  knowledge\_base = {  "A": True,  "B": True,  ("A", "C"): True,  ("B", "D"): True  }  antecedent = "A"  conclusion = "C"  result, derived\_conclusion = modus\_ponens(knowledge\_base, antecedent, conclusion)  if result:  print(f"Modus Ponens applies. Conclusion '{conclusion}' is derived.")  else:  print("Modus Ponens does not apply or the knowledge base is incomplete.") |

**Output :-**



**Question 5)** Create a Python program that demonstrates Modus Tollens. Given a conditional statement and the negation of the consequent, the program should check if it implies the negation of the antecedent.

**Ans:**

**Code :-**

|  |
| --- |
| def modus\_tollens(conditional, negation\_of\_consequent, negation\_of\_antecedent):  # Check if the conditional statement is in the form of "if P, then Q"  if conditional.count("->") == 1:  antecedent, consequent = conditional.split("->")  antecedent = antecedent.strip()  consequent = consequent.strip()    # Check if the given conditional statement matches the expected form  if conditional == f"{antecedent} -> {consequent}":  # Check if the negation of the consequent matches the consequent of the conditional  if negation\_of\_consequent == f"not {consequent}":  # Check if the negation of the antecedent matches the antecedent of the conditional  if negation\_of\_antecedent == f"not {antecedent}":  return True, f"not {antecedent}" # Modus Tollens applies, return True and the negation of the antecedent  return False, None # Modus Tollens does not apply  # Example usage:  if \_\_name\_\_ == "\_\_main\_\_":  conditional = "P -> Q"  negation\_of\_consequent = "not Q"  negation\_of\_antecedent = "not P"  result, derived\_negation = modus\_tollens(conditional, negation\_of\_consequent, negation\_of\_antecedent)  if result:  print(f"Modus Tollens applies. Negation of antecedent '{derived\_negation}' is derived.")  else:  print("Modus Tollens does not apply or the given statements do not match the expected form.") |

**Output :-**



**Assignment :- 10**

**Question 1)** Write a Python function to demonstrate the commutative law for addition and multiplication using user input.

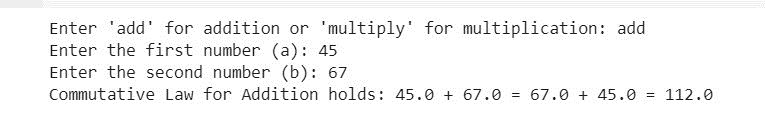
**Ans:**

**Code :-**

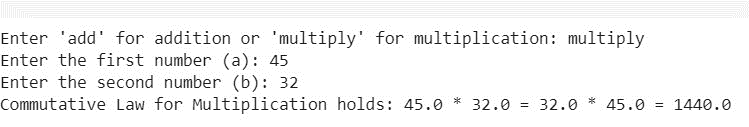
|  |
| --- |
| def commutative\_law():  operation = input("Enter 'add' for addition or 'multiply' for multiplication: ").strip().lower()  a = float(input("Enter the first number (a): "))  b = float(input("Enter the second number (b): "))  if operation == "add":  result1 = a + b  result2 = b + a  if result1 == result2:  print(f"Commutative Law for Addition holds: {a} + {b} = {b} + {a} = {result1}")  else:  print("Commutative Law for Addition does not hold.")  elif operation == "multiply":  result1 = a \* b  result2 = b \* a  if result1 == result2:  print(f"Commutative Law for Multiplication holds: {a} \* {b} = {b} \* {a} = {result1}")  else:  print("Commutative Law for Multiplication does not hold.")  else:  print("Invalid operation. Please enter 'add' or 'multiply'.")  if \_\_name\_\_ == "\_\_main\_\_":  commutative\_law() |

**Output :-**

**For Addition:-**



**For Multiplication:-**



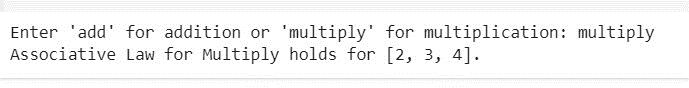
**Question 2)** Create a Python program that verifies the associative law for addition and multiplication with a list of numbers.

**Ans:**

**Code :-**

|  |
| --- |
| def verify\_associative\_law(numbers, operation):  if operation == "add":  # Associative Law for Addition  left\_associative\_result = sum(numbers)  right\_associative\_result = numbers[0]  for num in numbers[1:]:  right\_associative\_result += num  return left\_associative\_result == right\_associative\_result  elif operation == "multiply":  # Associative Law for Multiplication  left\_associative\_result = 1  right\_associative\_result = numbers[0]  for num in numbers:  left\_associative\_result \*= num  for num in numbers[1:]:  right\_associative\_result \*= num  return left\_associative\_result == right\_associative\_result  else:  return False  if \_\_name\_\_ == "\_\_main\_\_":  numbers = [2, 3, 4]  operation = input("Enter 'add' for addition or 'multiply' for multiplication: ").strip().lower()  result = verify\_associative\_law(numbers, operation)  if result:  print(f"Associative Law for {operation.capitalize()} holds for {numbers}.")  else:  print(f"Associative Law for {operation.capitalize()} does not hold for {numbers}.") |

**Output :-**



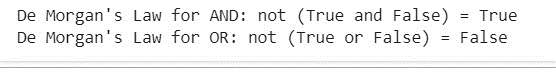
**Question 3)** Write a Python function that implements De Morgan's Laws for two Boolean variables and returns the results for both AND and OR operations.

**Ans:**

**Code :-**

|  |
| --- |
| def demorgans\_laws(a, b):  # Apply De Morgan's Laws  not\_a\_and\_b = (not a) or (not b)  not\_a\_or\_b = (not a) and (not b)  return not\_a\_and\_b, not\_a\_or\_b  if \_\_name\_\_ == "\_\_main\_\_":  a = True # Change to True or False as needed  b = False # Change to True or False as needed  result\_and, result\_or = demorgans\_laws(a, b)  print(f"De Morgan's Law for AND: not ({a} and {b}) = {result\_and}")  print(f"De Morgan's Law for OR: not ({a} or {b}) = {result\_or}") |

**Output :-**



**Question 4)** Implement a Python function that illustrates the law of excluded middle by determining if a given proposition is either true or false.

**Ans:**

**Code :-**

|  |
| --- |
| def law\_of\_excluded\_middle(proposition):  if proposition:  return "The proposition is true"  else:  return "The proposition is false"  if \_\_name\_\_ == "\_\_main\_\_":  # You can change the value of the proposition as needed  proposition = True  result = law\_of\_excluded\_middle(proposition)  print(result) |

**Output :-**



**Assignment :- 11**

**Question 1)**Implement a Python-based resolution theorem prover for predicate logic. Given a set of clauses, the program should determine if a query can be inferred.

**Ans:**

**Code :-**

|  |
| --- |
| def resolution(clauses):  """  Resolve the given set of clauses using the resolution algorithm.  :param clauses: List of clauses in CNF (Conjunctive Normal Form)  :return: True if the query can be inferred, False otherwise  """  new\_clause = set()  while True:  # Generate all possible pairs of clauses  for i in range(len(clauses)):  for j in range(i + 1, len(clauses)):  resolvents = resolve\_pair(clauses[i], clauses[j])  # Check if resolvent is empty (contradiction)  if not resolvents:  return True  # Add unique resolvents to the new\_clause set  new\_clause.update(resolvents)  # If new\_clause is a subset of clauses, no new resolvents can be generated  if new\_clause.issubset(set(clauses)):  return False  # Update clauses with the new resolvents  clauses.extend(new\_clause)  def resolve\_pair(c1, c2):  """  Resolve two clauses and return the set of resolvents.  :param c1: First clause  :param c2: Second clause  :return: Set of resolvents  """  resolvents = set()  for literal in c1:  if ('-' + literal) in c2:  # Resolve the two clauses by removing the resolved literals  resolvent = (c1 - {literal}) | (c2 - {'-' + literal})  resolvents.add(tuple(resolvent))  return resolvents  # Example usage:  clauses = [  {'A', '-B', 'C'},  {'B', '-C'},  {'-A', 'B'}  ]  query = ['-C']  result = resolution(clauses + [set(query)])  print("Query can be inferred:", result) |

**Output :-**



**Question 2)** Develop a Python function that identifies and counts the unique predicates and variables in a predicate logic expression.

**Ans:**

**Code :-**

|  |
| --- |
| import re  def extract\_predicates\_variables(expression):  # Define regular expressions for predicates and variables  predicate\_pattern = re.compile(r'([a-zA-Z\_]\w\*)\(')  variable\_pattern = re.compile(r'([a-zA-Z\_]\w\*)')  # Find all matches for predicates and variables  predicates = set(predicate\_pattern.findall(expression))  variables = set(variable\_pattern.findall(expression))  return predicates, variables  def count\_predicates\_variables(expression):  predicates, variables = extract\_predicates\_variables(expression)  # Count the number of unique predicates and variables  predicate\_count = len(predicates)  variable\_count = len(variables)  return predicate\_count, variable\_count  # Example usage:  predicate\_logic\_expression = "parent(john, mary) && ancestor(mary, ann) || sibling(john, ann)"  predicate\_count, variable\_count = count\_predicates\_variables(predicate\_logic\_expression)  print("Predicates:", predicate\_count)  print("Variables:", variable\_count) |

**Output :-**



**Question 3)** Write a Python program that uses Modus Ponens to perform inference on a set of predicate logic statements.

**Ans:**

**Code :-**

|  |
| --- |
| def modus\_ponens(rule, statement):  """  Apply Modus Ponens inference rule.  :param rule: The implication rule in the form (P, Q), meaning if P then Q.  :param statement: The given statement P.  :return: The inferred statement Q if the rule is applicable, otherwise None.  """  antecedent, consequent = rule  if antecedent == statement:  return consequent  else:  return None  # Example usage:  if \_\_name\_\_ == "\_\_main\_\_":  # Example Modus Ponens rule: If it's raining, then the ground is wet.  modus\_ponens\_rule = ("raining", "ground is wet")  # Given statement: It's raining.  given\_statement = "raining"  # Apply Modus Ponens  result = modus\_ponens(modus\_ponens\_rule, given\_statement)  # Output the result  if result is not None:  print(f"Using Modus Ponens: If {given\_statement}, then {result}.")  else:  print("Modus Ponens is not applicable for the given statements.") |

**Output :-**



THANK YOU