**CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY AND RESEARCH**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**ACADEMIC YEAR: 2020-21**

**Subject: Artificial Intelligence (CS341)**

**Assignment – 1**

**Python Programming**

1. **Ask the user for a number. Depending on whether the number is even or odd, print out an appropriate message to the user.**

**PROGRAM CODE**

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

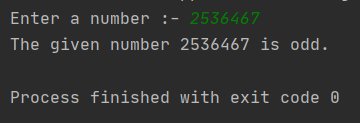
**if n%2 == 0:**

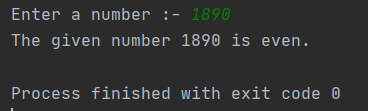
**print("The entered number is Even")**

**else:**

**print("The entered number is Odd")**

**OUTPUT**





1. **Take a list, say for example this one:**

**a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89], and write a program that prints out all the elements of the list that are less than 5.**

**PROGRAM CODE**

**a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]**

**print(a)**

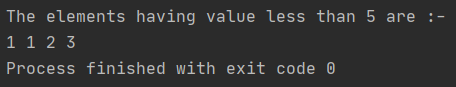
**print("Elements less than 5 are given below")**

**for number in a:**

**if number < 5:**

**print(number)**

**OUTPUT**



1. **Take two lists**

**a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]**

**b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] and write a program that returns a list that contains only the elements that are common between the lists (without duplicates). Make sure your program works on two lists of different sizes.**

**PROGRAM CODE**

**a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]**

**b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]**

**x = []**

**for i in a:**

**if i in b:**

**if i not in x:**

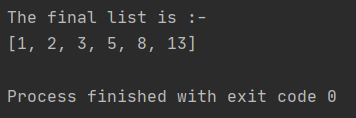
**x.append(i)**

**print("List 1 : ", a)**

**print("List 2 : ", b)**

**print("List containing** common elements (without duplicates) : ", x)

**OUTPUT**



1. **Ask the user for a string and print out whether this string is a palindrome or not.**

**PROGRAM CODE**

**x = input("Enter a string : ")**

**y = x[::-1]**

**print("Reversed string : ", y)**

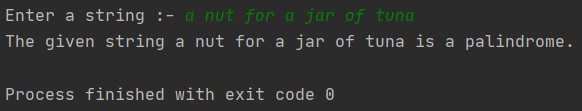
**if x == y:**

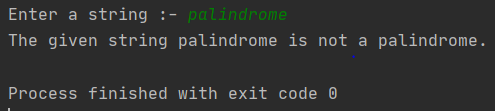
**print("It is Palindrome")**

**else:**

**print("It is not Palindrome")**

**OUTPUT**





1. **Ask the user for a number and determine whether the number is prime or not.**

**PROGRAM CODE**

**flag = 0**

**num = int(input("Enter a number (greater than 1) : "))**

**if num <= 0:**

**print("Invalid input")**

**elif num == 1:**

**print("1 is neutral (neither prime, nor composite)")**

**elif num == 2:**

**print(num, " is a Prime number")**

**else:**

**for i in range(2, (num//2)+1):**

**if num%i == 0:**

**flag = 1**

**break**

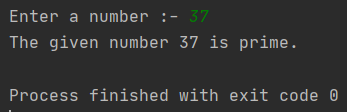
**if flag == 0:**

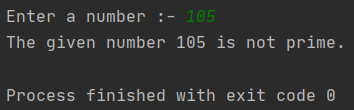
**print(num, " is a Prime number")**

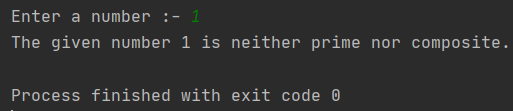
**else:**

**print(num, " is Not a Prime number")**

**OUTPUT**







1. **Write a program that asks the user how many Fibonacci numbers to generate using recursion.**

**PROGRAM CODE**

**def fibonacci(n):**

**if n <= 1:**

**return 1**

**else:**

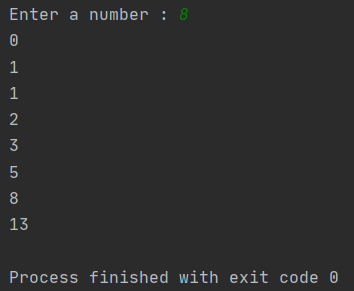
**return fibonacci(n-1) + fibonacci(n-2)**

**n = int(input("Enter the length of fibonacci series : "))**

**for i in range(n):**

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

**OUTPUT**



1. **Write a Python class named Circle constructed by a radius and two methods which will compute the area and the perimeter of a circle.**

**PROGRAM CODE**

**import math**

**class Circle:**

**radius: int**

**def \_\_init\_\_(self, r):**

**self.radius = r**

**def area(self):**

**return (math.pi)\*(self.radius)\*(self.radius)**

**def perimeter(self):**

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

**# Driver code**

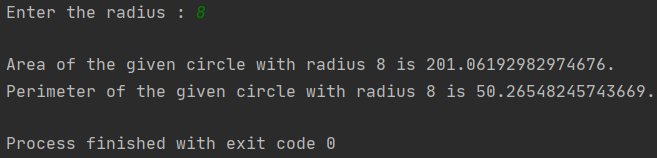
**r = float(input("Enter the radius : "))**

**c = Circle(r)**

**print("Area of the circle = ", c.area(), " sq.units")**

**print("Perimeter of the circle = ", c.perimeter(), " units")**

**OUTPUT**



1. **Develop programs for data structure algorithms using python – sorting (Bubble sort and Insertion sort).**

**PROGRAM CODE**

**# Bubble Sort**

**def bubble\_sort(x,n):**

**for round in range(n-1):**

**for j in range(n-round-1):**

**if (x[j]>x[j+1]):**

**temp = x[j]**

**x[j] = x[j+1]**

**x[j+1] = temp**

**return x**

**# Insertion Sort**

**def insertion\_sort(x,n):**

**for i in range(1,n):**

**key = x[i]**

**j = i - 1**

**while j>=0 and key<x[j]:**

**x[j+1] = x[j]**

**j -= 1**

**x[j+1] = key**

**return x**

**# Driver program**

**list1 = [5,4,3,2,1,45,37,61,23,14]**

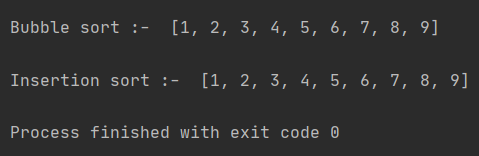
**list2 = [5,4,3,2,1,45,37,61,23,14]**

**print("List of elements : ",list1)**

**print("Bubble Sort : ", bubble\_sort(list1, len(list1)))**

**print("Insertion Sort : ", insertion\_sort(list2, len(list2)))**

**OUTPUT**



1. **Write a program that accepts a sentence and calculate the number of letters and digits.**

**PROGRAM CODE**

**count\_digits = 0**

**count\_letters = 0**

**string = input("Enter a string : ")**

**for i in string:**

**if i.isdigit():**

**count\_digits += 1**

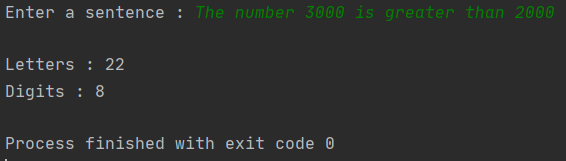
**elif i.isalpha():**

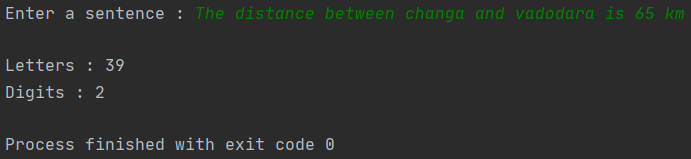
**count\_letters += 1**

**print("Number of digits = ", count\_digits)**

**print("Number of letters = ", count\_letters)**

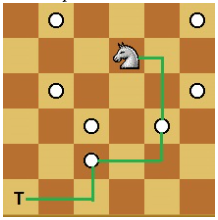
**OUTPUT**





1. **Given a square chessboard of N x N size, the position of Knight and position of a target is given. We need to find out minimum steps a Knight will take to reach the target position.**

**For example:**

****

**In above diagram Knight takes 3 step to reach from (4, 5) to (1, 1) (4, 5) 🡺 (5, 3) 🡺 (3, 2) 🡺 (1, 1) as shown in diagram.**

**PROGRAM CODE**

**class cell:**

**def \_\_init\_\_(self, x=0, y=0, dist=0):**

**self.x = x**

**self.y = y**

**self.dist = dist**

**def isInside(x, y, N):**

**if (x >= 1 and x <= N and y >= 1 and y <= N):**

**return True**

**return False**

**def minStepToReachTarget(knightpos, targetpos, N):**

**dx = [2, 2, -2, -2, 1, 1, -1, -1]**

**dy = [1, -1, 1, -1, 2, -2, 2, -2]**

**queue = []**

**queue.append(cell(knightpos[0], knightpos[1], 0))**

**visited = [[False for i in range(N + 1)]**

**for j in range(N + 1)]**

**visited[knightpos[0]][knightpos[1]] = True**

**while (len(queue) > 0):**

**t = queue[0]**

**queue.pop(0)**

**if (t.x == targetpos[0] and**

**t.y == targetpos[1]):**

**return t.dist**

**for i in range(8):**

**x = t.x + dx[i]**

**y = t.y + dy[i]**

**if (isInside(x, y, N) and not visited[x][y]):**

**visited[x][y] = True**

**queue.append(cell(x, y, t.dist + 1))**

**# Driver code**

**N = int(input("Enter the dimension of Chessboard : "))**

**kx = int(input("Enter x coordinate of Knight's position : "))**

**ky = int(input("Enter y coordinate of Knight's position : "))**

**tx = int(input("Enter x coordinate of target position : "))**

**ty = int(input("Enter y coordinate of target position : "))**

**knightpos = [kx, ky]**

**targetpos = [tx, ty]**

**print("\nMinimum steps to reach target = ", minStepToReachTarget(knightpos, targetpos, N))**

**OUTPUT**

