LAB TASKS

NAME: MOHAMMAD AKRAMUZZAMAN

ID:20-43047-1

SUBJECT: ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEM

SECTION: N

**ASSIGNMENT:**

import random

class Environment(object):

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

self.locationcondition={'A' : '1' , 'B' : '1'}

self.locationcondition['A']=random.randint(0,1)

self.locationcondition['B']=random.randint(0,1)

class Sreflexagent(Environment):

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

print(Environment.locationcondition)

vacuumlocation=random.randint(0,1)

if vacuumlocation==0:

print("vacuum is randomly placed at locationn A")

if Environment.locationcondition['A']==1:

print("Location A is dirt")

Environment.locationcondition['A']==0;

print("Location A has been clean")

if Environment.locationcondition['B']==1:

print("Location b is dirt")

Environment.locationcondition['B']==0;

print("Location B has been clean")

else:

print("moving to location B")

if Environment.locationcondition['B']==1:

print("Location B is dirt")

Environment.locationcondition['B']==0;

print("Location B has been clean")

elif vacuumlocation==1:

print("vacuum is randomly placed at locationn A")

if Environment.locationcondition['B']==1:

print("Location B is dirt")

Environment.locationcondition['A']==0;

print("Location A has been clean")

if Environment.locationcondition['B']==1:

print("Location A is dirt")

Environment.locationcondition['A']==0;

print("Location A has been clean")

else:

print("moving to location A")

if Environment.locationcondition['A']==1:

print("Location B is dirt")

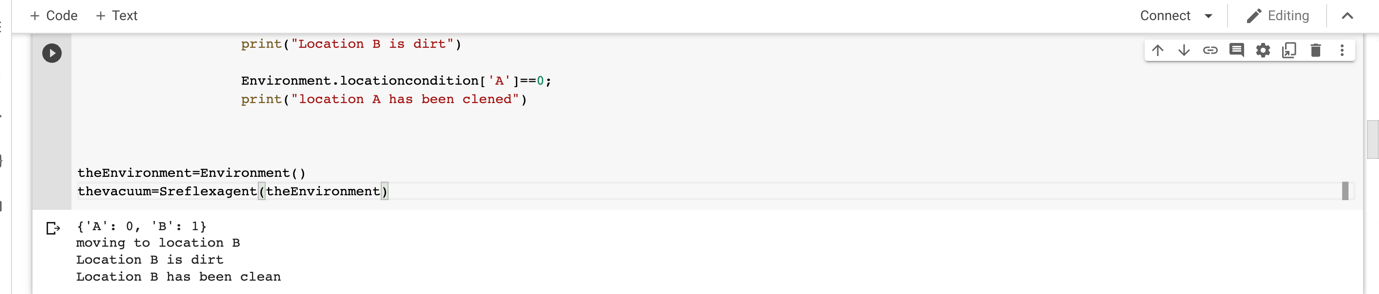
Environment.locationcondition['A']==0;

print("location A has been clened")

theEnvironment=Environment()

thevacuum=Sreflexagent(theEnvironment)

**OUTPUT:**



TASK:

a=int(input('>'))

b=int(input('>'))

c=int(input('>'))

if a>b and a>c:

print ("a is the largest")

elif b>a and b>c:

print("b is the largest")

else:

print("c is the largest")

**OUTPUT:**



TASK:

n=int(input('Add till =>'))

total=0

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

if (i%2!=0):

total=total+i

print(total)

**OUTPUT:**



TASK:

n=int(input('=>'))

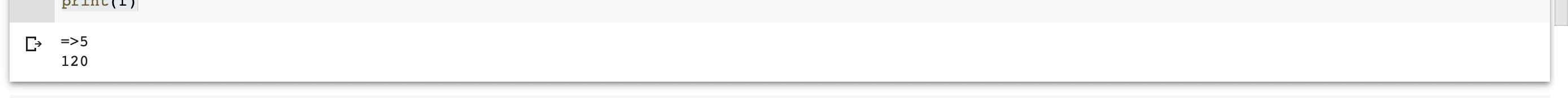
f=1

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

f=f\*i

print(f)

**OUTPUT:**



TASK:

def fibonacci\_series(n):

a=0

b=1

if n>2:

print(a)

for i in range (2,n):

c=a+b

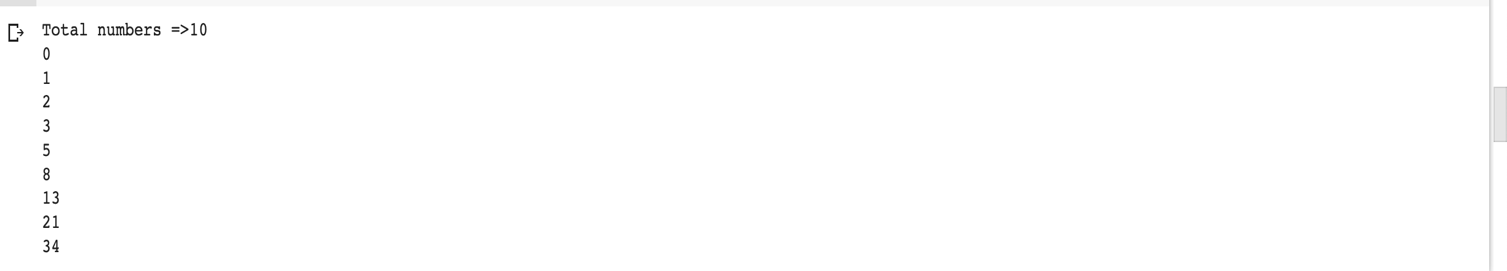
a=b

b=c

print(c)

fibonacci\_series(int(input('Total numbers =>')))

**OUTPUT:**



TASK:

import matplotlib.pyplot as pit

X = range(1, 50)

Y = [value\* 3 for value in X]

print ("Values of X:")

print (\*range (1, 50))

print ("Values of Y (thrice of X):")

print(Y)

pit.plot(X,Y)

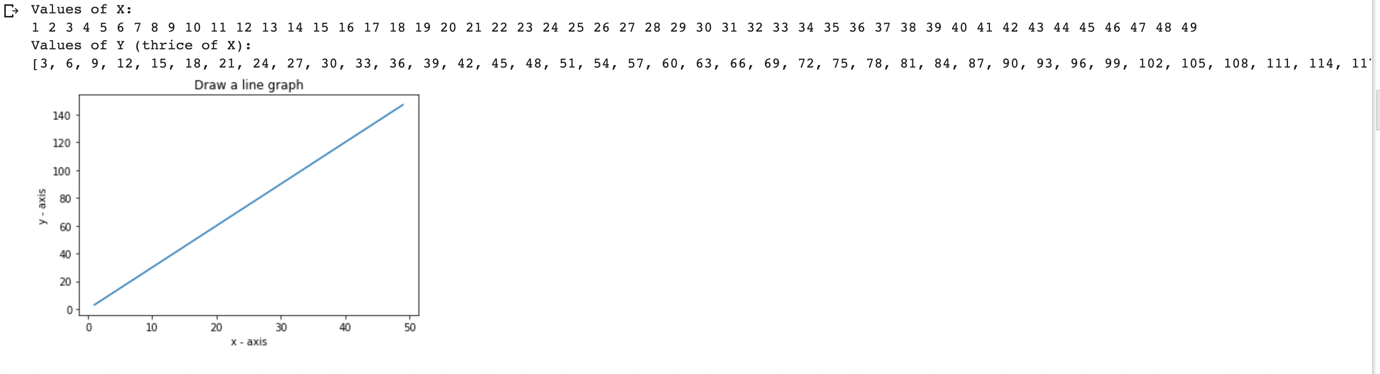
pit.xlabel('x - axis')

pit.ylabel('y - axis')

pit.title('Draw a line graph')

pit. show()

OUTPUT:



TASK:

import matplotlib.pyplot as pit

X = [1.0,1.5,2.0,2.5,3.0]

Y = [2.0,3.0,4.0,2.5,1.0]

print ("Values of X:")

print (\*range (1, 50))

print ("Values of Y (thrice of X):")

print(Y)

pit.plot(X,Y)

pit.xlabel('x - axis')

pit.ylabel('y - axis')

pit.title('Draw a line graph')

pit. show()

OUTPUT:



TASK:

n=int(input('>>'))

sum=0

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

sum+=1/i

print(str(1)+"/"+str(i),end="")

if(i!=n):

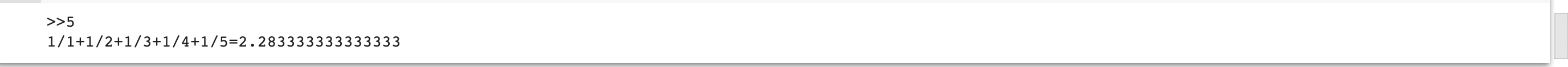
print("+",end="")

else:

print("=",end="")

print(sum)

**OUTPUT:**



TASK:

import numpy as np

list=[]

n=int(input('Number of elements>>'))

for i in range(0,n):

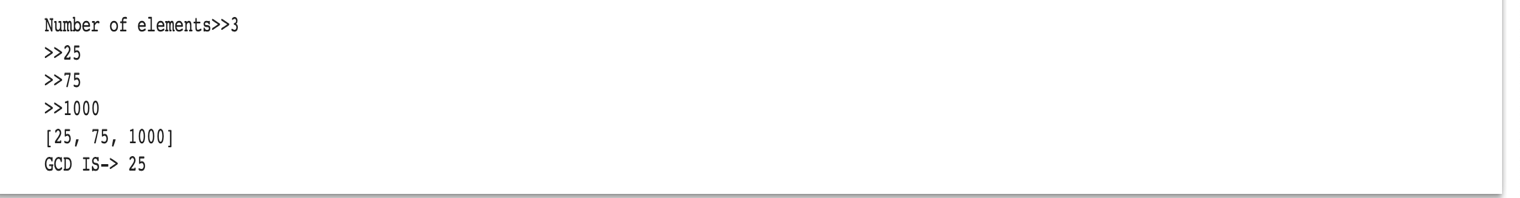
elements=int(input('>>'))

list.append(elements)

print (list)

print("GCD IS-> "+str(np.gcd.reduce(list)))

OUTPUT:



TASK:

class Person:

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

self.name = name

self.age = age

p1 = Person("John", 36)

print(p1.name)

print(p1.age)

OUTPUT:

TASK:

class Person:

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

self.name = name

self.age = age

def myfunc(self):

print("Hello my name is " + self.name)

p1 = Person("John", 36)

p1.myfunc()

OUTPUT:



TASK:

class NumPattern:

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

self.row=row

def func(self):

i=self.row

for i in range(0,i+1):

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

print('\*',end=" ")

print()

p=NumPattern(5)

p.func()

OUTPUT:



TASK:

class NumPattern:

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

self.row=row

def func(self):

i=self.row

for i in range(0,i+1):

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

print(j,end=" ")

print()

p=NumPattern(5)

p.func()

OUTPUT:



TASK:

class Pattern2:

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

self.row=row

def func(self):

for i in range(1,self.row+1):

for j in range(self.row,i-1,-1):

print (j,end=" ")

print("\n")

p1=Pattern2(5)

p1.func()

OUTPUT:



TASK:

class Pyramid2:

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

self.row=row

def func(self):

#self.row=rows

for i in range(0,self.row):

for j in range(0,self.row-i):

print (" ",end=" ")

for j in range(0,i+1):

print('\*',end=" ")

print("\n")

p2=Pyramid2(5)

p2.func()

OUTPUT:



TASK:

graph = {

'A' : ['B','C'],

'B' : ['D', 'E'],

'C' : ['F'],

'D' : [],

'E' : ['F'],

'F' : []

}

visited = set()

def dfs(visited, graph, node):

if node not in visited:

print (node)

visited.add(node)

for neighbour in graph[node]:

dfs(visited, graph, neighbour)

dfs(visited, graph, 'A')

OUTPUT:



TASK:

list1=[10,20,23,11,17]

list2=[13,43,24,36,12]

list3=[]

for num in list1:

if num%2!=0:

list3.append(num)

for num in list2:

if num%2==0:

list3.append(num)

#list3.sort()

for i in range(0,len(list3)):

for j in range(i+1,len(list3)):

if (list3[i] > list3[j]):

temp=list3[i]

list3[i]=list3[j]

list3[j]=temp

print(list3)

OUTPUT:



TASK:

class Mammal:

def walk(self):

print("Starting walking")

class Dog(Mammal):

pass

class cat(Mammal):

pass

dog=Dog()

dog.walk()

**OUTPUT:**

