Q. 2
(a)

```
import pandas as pd
from datetime import datetime
class Student():

   def init(self,EMPLOYEE_ID,FIRST_NAME,
                LAST_NAME,EMAIL,PHONE_NUMBER,
                HIRE_DATE,JOB_ID,SALARY,COMMISSION_PCT,
                MANAGER_ID,DEPARTMENT_ID):
       self.empID=EMPLOYEE_ID
       self.f_name=FIRST_NAME
       self.l_name=LAST_NAME
       self.DeptID=DEPARTMENT_ID
       self.salary=SALARY
       self.email=EMAIL
       self.phone=PHONE_NUMBER
       self.hire=HIRE_DATE
       self.jobID=JOB_ID
       self.comm=COMMISSION_PCT
       self.mgrID=MANAGER_ID

   def changeSalary(self, x):
       self.salary=self.salary+x

   def changeDept(self, y):
       self.id=y

   def str(self):
       return "EMPLOYEE_ID: %s, FIRST_NAME: %s, LAST_NAME: %s, EMAIL: %s,
PHONE_NUMBER: %s, HIRE_DATE: %s, JOB_ID: %s, SALARY: %s, COMMISSION_PCT:
%s, MANAGER_ID: %s, DEPARTMENT_ID: %s"%(self.empID,
self.f_name,self.l_name,self.email,self.phone,self.hire,self.jobID,self.sa
lary,self.comm, self.mgrID,self.DeptID)
df =pd.read_csv('/home/deltaplus/employees.csv')
df.head()
```

Q. 2
(b)

```
obj=[]
for i in range(len(df)):

obj.append(Student(df["EMPLOYEE_ID"][i],df["FIRST_NAME"][i],df["LAST_NAME"][i],df["EMAIL"][i],df["PHONE_NUMBER"][i],df["HIRE_DATE"][i],df["JOB_ID"][i],df["SALARY"][i],df["COMMISSION_PCT"][i],df["MANAGER_ID"][i],df["DEPARTMENT_ID"][i]))
Obj
```

Q. 2

(c)

```
date_time_str0 ='1-JAN-03'
date_time_obj0 = datetime.strptime(date_time_str0, '%d-%b-%y' )

for i in range(len(obj)):
   if datetime.strptime(obj[i].hire , '%d-%b-%y' )< date_time_obj0:
       obj[i].changeSalary(60)
       print(obj[i])
```

Q. 2

(d+e)

```
L=[]
for i in range(len(obj)):
   L.append([obj[i].empID,
obj[i].f_name,obj[i].l_name,obj[i].email,obj[i].phone,obj[i].hire,obj[i].jobID,obj[i].salary,obj[i].comm, obj[i].mgrID,obj[i].DeptID])

df1=pd.DataFrame(L)
df1.columns=df.columns
df1.to_csv("employee_updated.csv",index=None)

M=[]
N=[]
mgr=dict()
for i in range(len(obj)):
   M.append(obj[i].mgrID)

for i in range(len(obj)):
   mgr[obj[i].mgrID]=M.count(obj[i].mgrID)
```

```
for i in mgr.keys():
   if mgr[i]>3:
       N.append(int(i))

for i in range(len(obj)):
   if obj[i].empID in N:
       print(obj[i])
```

Q. 3

```
class NewComplex(object):
   def init(self, real, imag=0.0):
       self.real = real
       self.imag = imag

   def add(self, other):
       return NewComplex(self.real + other.real,
                      self.imag + other.imag)

   def mul(self, other):
       return NewComplex(self.real*other.real - self.imag*other.imag,
                      self.imag*other.real + self.real*other.imag)


   def lt(self,other):
       return 'Operation is illegal'
   def le(self,other):
       return 'Operation is illegal'
   def gt(self,other):
       return 'Operation is illegal'
   def ge(self,other):
       return 'Operation is illegal'

   def eq(self, other):
       return self.real == other.real and self.imag == other.imag

   def str(self):
       out=''
```

```
        if self.imag<0:
            out='%g-j%g' % (self.real, abs(self.imag))
        else:
            out='%g+j%g' % (self.real, self.imag)
        return out

    def repr(self):
        return  str(self)

u=NewComplex(2,-1)
v=NewComplex(1)
w=u+v
print(w)
x=u*v
print(x)
```

Q. 4

(a)

```
import numpy as np
import matplotlib.pyplot as plt
from itertools import zip_longest
class Polynomial:

    def init(self, *coefficients):

        self.coefficients = list(coefficients)

    def call(self, x):
        res = 0
        for index, coeff in enumerate(self.coefficients[::-1]):
            res += coeff * x** index
        return res

    def add(self, other):
        c1 = self.coefficients[::-1]
        c2 = other.coefficients[::-1]
        res = [sum(t) for t in zip_longest(c1, c2, fillvalue=0)]
        return Polynomial(*res[::-1])
```

```
   def roots(self):

       return np.roots(self.coefficients)

   def repr(self):

      return "Polynomial" + str(tuple(self.coefficients))

   def str(self):

       def x_expr(degree):
           if degree == 0:
               res = ""
           elif degree == 1:
               res = "x"
           else:
               res = "x^"+str(degree)
           return res

       degree = len(self.coefficients) - 1
       res = ""

       for i in range(0, degree+1):
           coeff = self.coefficients[i]
           if abs(coeff) == 1 and i < degree:
               res += f"{'+' if coeff>0 else '-'}{x_expr(degree-i)}"
           elif coeff != 0:
               res += f"{coeff:+g}{x_expr(degree-i)}"

       return res.lstrip('+')
p=Polynomial(3,2,1)
q=Polynomial(7,5,3)
p
p=Polynomial(3,2,1)
q=Polynomial(7,5,3)
q
```

Q. 4

(b)

```
p=Polynomial(3,2,1)
q=Polynomial(7,5,3)
r=p+q
print(r)
X = np.linspace(-3, 3, 50, endpoint=True)
F1 = p(X)
F2 = q(X)
F_add = r(X)
plt.plot(X, F1, label="Poly 1")
plt.plot(X, F2, label="Poly 2")
plt.plot(X, F_add, label="Poly_add")
plt.legend()
plt.show()
```